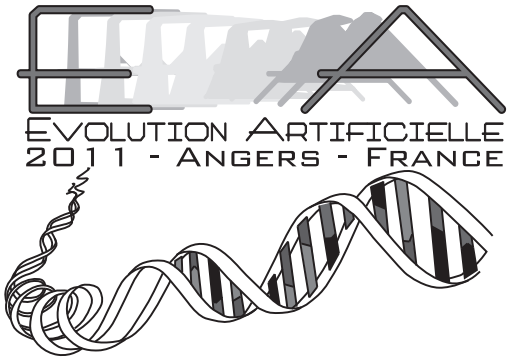
Jin-Kao Hao
Pierrick Legrand
Pierre Collet
Nicolas Monmarché
Evelyne Lutton
Marc Schoenauer



Artificial Evolution 2011 (Evolution Artificielle 2011)

10th Biennal International Conference on Artificial Evolution

Proceedings

ISBN 978-2-9539267-1-2

24-26 October 2011, Angers, France

Foreword

This proceedings includes the papers presented at the 10th Biennial International Conference on Artificial Evolution, EA[1] 2011, held in Angers (France). Previous EA editions took place in Strasbourg (2009), Tours (2007), Lille (2005), Marseille (2003), Le Creusot (2001), Dunkerque (1999), Nimes (1997), Brest (1995), and Toulouse (1994).

Authors had been invited to present original work relevant to Artificial Evolution, including, but not limited to: Evolutionary Computation, Evolutionary Optimization, Co-evolution, Artificial Life, Population Dynamics, Theory, Algorithmics and Modeling, Implementations, Application of Evolutionary Paradigms to the Real World (industry, biosciences, ...), other Biologically-Inspired Paradigms (Swarm, Artificial Ants, Artificial Immune Systems, Cultural Algorithms...), Memetic Algorithms, Multi-Objective Optimization, Constraint Handling, Parallel Algorithms, Dynamic Optimization, Machine Learning and hybridization with other soft computing techniques.

Each submitted paper has been reviewed by three members of the International Program Committee. Among the 64 submissions received, 33 papers have been selected for oral presentation and 10 other papers for poster presentation. As for the previous editions (see LNCS volumes 1063, 1363, 1829,2310, 2936, 3871, 4926 and 5975), a selection of the best papers which are presented at the Conference and further revised will be published as a volume of Springer's LNCS series.

To celebrate the tenth anniversary of EA, we are grateful to two of the co-founders of the EA series, Evelyne Lutton and Marc Schoenauer both from INRIA (France) who accepted to give a talk on "Twenty Years of Artificial Evolution in France" (Jean-Marc Alliot from ENAC Toulouse is the other co-founder). We would also like to express our sincere gratitude to our invited speaker: René Doursat from Institut des Systèmes Complexes – Paris who gives the talk "Artificial Evo-Devo: Bringing Back Self-Organized Multi-Agent Systems into Evolutionary Computation".

The success of the conference resulted from the input of many people to whom I would like to express my appreciation: The members of Program Committee and the secondary reviewers for their careful reviews that ensure the quality of the selected papers and the Conference. The members of the Organizing Committee for their efficient work and dedication assisted by Catherine Pawlonski-Boisseau and Christine Bardaine and others from the Computer Science Department of University of Angers. The members of the Steering Committee for their valuable assistance. Sylvie Reverdy from Centre de Congrès d'Angers for her very efficient actions. Marc Schoenauer for his support with the MyReview system.

I take this opportunity to thank the different partners whose financial and material support contributed to the organization of the Conference: Université d'Angers, Conseil Général "Maine et Loire", Ville d'Angers, Angers Loire Métropole, Région "Pays de La Loire", INRIA, Ministère de l'Enseignement Supérieur et de la Recherche, Laboratoire LERIA, Centre de Congrès d'Angers.

Last but not least, I thank all the authors who have submitted their research papers to the Conference, and the authors of accepted papers who attend the Conference to present their work. Thank you all.

Jin-Kao Hao

EA 2011 Chair
Université d'Angers, France

[1] As for previous editions of the conference, the EA acronym is based on the original French name "Évolution Artificielle".

Contents

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Sponsoring Institutions

- Université d'Angers
- Conseil Général "Maine et Loire"
- Ville d'Angers
- Angers Loire Métropole
- Région "Pays de La Loire"
- INRIA
- Ministère de l'Enseignement Supérieur et de la Recherche
- Association Evolution Artificielle `http://www.lifl.fr/EA/`

Steering Committee

Pierre Collet (Université Louis Pasteur de Strasbourg)
Evelyne Lutton (INRIA)
Nicolas Monmarché (Université François Rabelais de Tours)
Marc Schoenauer (INRIA)

Organizing Committee

Anne Auger (Submissions)
Béatrice Duval (Sponsors)
Jin-Kao Hao (Chair)
Laetitia Jourdan (Publicity)
Pierrick Legrand (Proceedings / LNCS publication)
Jean-Michel Richer (Webmaster)
Sébastien Vérel (Treasurer)

Program Committee

Enrique Alba, Universidad de Málaga, Spain
Anne Auger, INRIA Saclay, France
Wolfgang Banzhaf, University of Newfoundland, Canada
Hans-Georg Beyer, Vorarlberg University of Applied Sciences, Austria
Amine Boumaza, Université du Littoral Côte d'Opale, France
Nicolas Bredeche, Université Paris-Sud XI, France
Edmund Burke, University of Nottingham, UK
Stefano Cagnoni, Università di Parma, Italy
Alexandre Caminada, Université de Technologie de Belfort-Montbéliard, France
Carlos A. Coello Coello, Instituto Politécnico Nacional, Mexico
Philippe Collard, Université de Nice - Sophia Antipolis, France
Pierre Collet, Université de Strasbourg, France
David Corne, Heriot Watt University, UK
Ernesto Costa, University of Coimbra, Portugal
Daniel Delahaye, Ecole Nationale Aviation Civile, France
Benjamin Doerr, Universität des Saarlandes, Germany
Nicolas Durand, Institut de Recherche en Informatique de Toulouse, France
Marc Ebner, Eberhard Karls Universität Tübingen, Germany
Ágoston E. Eiben, Vrije Universiteit Amsterdam, The Nertherlands
Aniko Ekart, Aston University, UK
Cyril Fonlupt, Université du Littoral, France
Christian Gagné, Université Laval, Canada
Mario Giacobini, University of Torino, Italy
Fred Glover, OptTek, USA
Jens Gottlieb, SAP AG, Germany
Nikolaus Hansen, INRIA Saclay, France
Jin-Kao Hao, Université d'Angers, France
Matthew Hyde, University of Nottingham, UK
Yaochu Jin, University of Surrey, UK
Colin Johnson, University of Kent, UK
Laetitia Jourdan, INRIA Lille, France
Gary Kochenberger, University of Colorado, USA
Natalio Krasnogor, University of Nottingham, UK
Pier Luca Lanzi, Politecnico di Milano, Italy
Claude Lattaud, Université René Descartes, France
Pierrick Legrand, Université de Bordeaux, France
Jean Louchet, Inria Saclay, France
Evelyne Lutton, Inria Saclay, France
Bob McKay, Seoul National University, South Korea
Nicolas Monmarché, Université de Tours, France
Pablo Moscato, University of Newcastle, Australia
Jean-Baptiste Mouret, Université de Paris 6, France
Yuichi Nagata, Japan Advanced Institute of Science and Technology, Japan
Miguel Nicolau, University College Dublin, Ireland
Giuseppe Nicosia, University of Catania, Italy
Gabriela Ochoa, University of Nottingham, UK
Yew-Soon Ong, Nanyang Technological University, Singapore
Martin Pelikan, University of Missouri, USA
Denis Robilliard, Université du Littoral, France
Eduardo Rodriguez-Tello, CINVESTAV Tamaulipas, Mexico
Frédéric Saubion, Université d'Angers, France
Marc Schoenauer, INRIA Saclay - Île-de-France, France
Patrick Siarry, Université Paris Val de Marne, France
Moshe Sipper, Ben-Gurion University of the Negev, Israel
Stephen Smith, University of York, UK
Christine Solnon, Université Lyon 1, France
Terence Soule, University of Idaho, USA
Thomas Stuetzle, Université Libre de Bruxelles, Belgique
El-Ghazali Talbi, INRIA Lille, France
Kay Chen Tan, National University of Singapore, Singapore
Ke Tang, University of Science and Technology of China, China
Olivier Teytaud, INRIA Saclay, France
José Torres-Jiménez, CINVESTAV Tamaulipas, Mexico
Shigeyoshi Tsutsui, Hannan University, Japan
Gilles Venturini, Université de Tours, France
Sebastien Verel, Université de Nice-Sophia Antipolis, France
Darrell Whitley, Colorado State University, USA
Jun Zhang, Sun Yat-sen University, China
Qingfu Zhang, University of Essex, UK

Invited Talks

René Doursat, Institut des Systèmes Complexes — Paris (France)

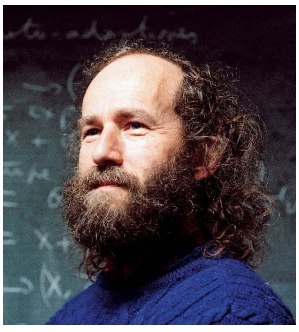
Artificial Evo-Devo: Bringing Back Self-Organized Multi-Agent Systems into Evolutionary Computation

Multicellular organisms and social insect constructions are rather unique examples of naturally evolved systems that exhibit both self-organization and a strong architecture. Can we export their precise self-formation capabilities to technological systems? I have proposed a new research field called "Morphogenetic Engineering" [1], which explores the artificial design and implementation of complex, heterogeneous morphologies capable of developing without central planning or external lead. Particular emphasis is set on the programmability and controllability of self-organization, properties that are often underappreciated in complex systems science-while, conversely, the benefits of multi-agent self-organization are often underappreciated in engineering methodologies, including evolutionary computation. In this talk I will present various examples of morphogenetic engineering, in particular multi-agent systems inspired by biological development based on gene regulation networks, and self-construction of graph topologies based on "programmed attachment" rules. Potential applications range from robotic swarms to autonomic networks and socio-technical systems. In all cases, the challenge is to "meta-design", especially through an evolutionary search, the proper set of rules followed by each agent of a complex system on how to interact with the other agents and the environment. Whether "offline" (slow time scale), where agents always share the same genotype, or "online" (fast time scale), where agent types may diverge and specialize dynamically, it constitutes an inherently massively parallel "artificial evo-devo" (evolutionary developmental) problem.

[1].Doursat, R., Sayama, H. and Michel, O., eds. (2011) Morphogenetic Engineering: Toward Programmable Complex Systems, in NECSI "Studies on Complexity" Series, Springer-Verlag, to appear.

Evelyne Lutton and Marc Schoenauer, INRIA (France)

Twenty Years of Artificial Evolution in France



Accepted Papers

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Ant Colony Optimization for the Identification of Nonlinear Functions with Unknown Structures

Bianca Minodora Heiman[1,2], Guillaume Sandou[1]

[1] SUPELEC Systems Sciences (E3S), Automatic Control Department
3, rue Joliot Curie, 91192 Gif-sur-Yvette, FRANCE
[2] Universitatea Polytechnica Bucuresti,
Department of Automatic Control and Computers Science
Guillaume.Sandou@supelec.fr

Abstract. The identification of systems is a key feature to get representative models and so to design efficient control laws. Numerous methods exist to identify the parameters of nonlinear systems when the global structure of the model is given. The problem appears to be much more difficult when this structure is unknown. This paper introduces ant colony optimization (ACO) in solving the problem of non linear systems identification in the case of an unknown structure of the model. Numerical results are much than satisfactory and prove the viability of the approach.

Keywords: Ant colony optimization, identification, nonlinear systems, unknown structure, symbolic regression.

1 Introduction

The modeling of a system is a fundamental step to understand its behavior and to develop control laws to manage it. Two main approaches are usually used. The first one consists in deriving some equations from physical laws. The second one is concerned with a behavioral representation of input / output relations. In this study, such a “black box” approach is considered, that is to say that inputs/outputs data have been collected from the plant and should now be used to define the model.

In such a context, a traditional procedure is to postulate a generic expression for the model. Then, optimization techniques are used to identify the parameters of the model. Such an approach uses the a priori knowledge of the plant to define the general expression of the model. A deterministic nonlinear method such as gradient based or Newton based ones can be used for that purpose. Classical approaches deal for example with nonlinear least square methods [1].

Another possibility is to identify not only the coefficients of the function but also the general shape of the function. This kind of problem often refers to “symbolic regression”. This is the case for example with the identification methods based on neural networks [2]-[3]. With this kind of method, no a priori knowledge is required to define an input-output model. However, the corresponding model may be difficult to use, as the analytic formulae of the model is often intractable for the design of

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

nonlinear controllers. Finally, these methods are preferably used for classification purposes [4].

More recently, in the field of symbolic regression, the use of genetic algorithm has allowed the identification of the model structure together with its parameters [5]-[11]. The idea is to use the tree representation of mathematical functions and to design some crossing over and mutation operators applying on trees. Currently, this kind of approach appears to be the most efficient and the most used one. However, it is difficult with these methods to add some a priori knowledge on the identification procedure. Further, some studies ([5] for instance) limits itself to polynomial regressions.

In this paper, the goal is to solve the same generic problem with an ant colony algorithm. That is to say that a plant is given, but we have no idea about the input / output relation. Thus, a function has to be found, mapping the inputs / outputs. Using as for the genetic algorithm approach, the tree description of function, the optimization problem consists in finding the best values associated to each node so as to minimize a global least square criterion. Finally, the optimization problem can be seen as a graph exploration problem which can be solved by ant colony optimization.

Indeed, ant colony optimization was introduced by Marco Dorigo in the beginning of the 90s [12]. This algorithm is based on the social behavior of ants and allows solving complex optimization problems. First examples of application deal with the solution to the Travelling Salesman Problem [13]. The developed algorithm can be easily extended for the solution to graph exploration problems. Finally, such an algorithm is a good candidate for the solution of the identification of nonlinear functions with unknown structures, as it can be reformulated into a binary tree construction.

The use of ant colony algorithms has already been proposed for the definition of approximation algorithms. In particular, in [14]-[15], an ant algorithm is used to search for a program approximating a function from a set of given instructions. These studies refer to a more generic domain called “automatic programming”. Results are promising, but they do not exploit the possibility of avoiding “infeasible programs” inherent to the constructive ant algorithm procedure. In [16], an ant algorithm is also used to approximate a plane curve with a polygon. Finally, in [17] an ant colony is used to solve the symbolic regression problem but it exhibits little success.

The paper is organized as follows. In section 2 the optimization problem is defined, together with its modeling as a graph exploration problem using the tree representation of functions. Ant colony optimization is called up in section 3. Section 4 provides some numerical results, exhibiting much than satisfactory results and proving the viability of the approach. Discussions about the method and forthcoming works are given in section 5. Finally, conclusions are drawn in section 6.

2 Problem statement and modeling

2.1 Problem statement

In this study, it is considered that an unknown function $y = f(x)$ has to be identified from N measured data $(x_i, y_i), i = 1, \ldots, N$. Note that in this study only static plants are considered, but the extension to dynamical systems is straightforward, et least for mono-variable problems.

We look for not only the parameters of this given function f but for the function itself. Using a least square criterion, this problem can be stated as:

$$\min_{f} \sum_{i=1}^{N} (y_i - f(x_i))^2 \quad \textbf{(1)}$$

2.2 Problem modelling

Mathematical functions can be easily represented by trees. For instance, consider the function given by:

$$f(x) = \sin(x+3) \times \exp(x) \quad \textbf{(2)}$$

This function can be represented by the tree given in figure 1.

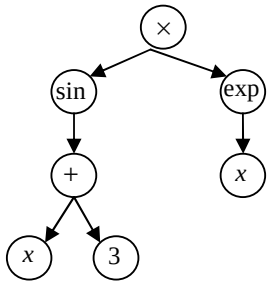


Fig. 1. Representation of $f(x) = \sin(x+3) \times \exp(x)$.

Indeed, a function is composed of binary operators (such as the addition operation), unary operators (such as 'sin' function) and zero operator (such as constants or variables).

It is possible to formulate this tree as a full binary tree, considering for example that a unary operator is a binary operator with a null right child. For instance, the function of equation (2) and represented in figure 1 corresponds to the full binary tree in figure 2, where the symbol $\varnothing$ is used to denote a null child.

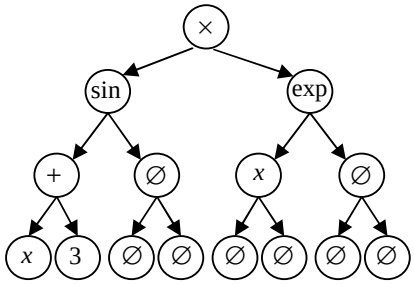


Fig. 2. Full binary tree representation of $f(x) = \sin(x+3) \times \exp(x)$.

The advantages of using a full binary tree are the facts that:

- For a given depth *p* of the tree (the depth is 4 for the tree of figure 2), the number of nodes is $2^p - 1$;
- The children of the node number i are the nodes $2i$ and $2i+1$.

Finally, given a set of possible operators Ω (for instance $\Omega = \{1,2,3,x,\varnothing,\cos,\sin\}$) and a given depth *p*, the problem can be reformulated as the following optimization problem:

$$\min_{\theta_1,\ldots,\theta_{2^p-1}} \sum_{i=1}^{N} (y_i - f_{\theta_1,\ldots,\theta_{2^p-1}}(x_i))^2 \quad \text{s.t } \theta_i \in \Omega,\ i = 1,\ldots,2^p - 1 \tag{3}$$

where $f_{\theta_1,\ldots,\theta_{2^p-1}}$ corresponds to the function encoded as a full binary tree by the parameters $\theta_1,\ldots,\theta_{2^p-1}$. Numerous constraints have to be added to that optimization problem. Constraints refer to the construction of viable expressions. For instance, if the value in node i is ' cos ', then the value in node $2i+1$ has to be $\varnothing$. Finally, the problem can be seen as a minimization problem and its modeling refers to a graph exploration problem.

The introduction of the symbol $\varnothing$ to deal with full binary trees could be seen as a bad idea as it deals to an increase of the number of optimization variables, and so to an increase of the combinatorial explosion. However, it will be shown in the next section that this problem is easily overcome by ant colony.

This optimization problem is hard to solve with classical and exact methods such as "Branch and Bound". Indeed, the cost criterion (3) has no analytical expression in function of the optimization variables, and the constraints are expressed through Boolean formulations. Such a problem can be solved by ant colony optimization as it will be shown in the following section.

3 Ant colony optimization

3.1 Ant colony social behavior

Ant colony optimisation was firstly introduced by Marco Dorigo [12]. This algorithm is based on the social behaviour of ants when they are looking for food. Consider figure 3a, where the ants have managed to find food. Each ant does not know where to go. It only chooses its path depending on the pheromone trail which has been laid on the ground by previous ants. In figure 3b, the path of pheromone is broken because of an obstacle. Thus, the first ant does not know whether to turn left or right. First ants randomly chose their path. But, the ants which have chosen the shortest path will come first to destination: the trail of pheromone in the shortest path is increasing faster than in the longest path, and because of a positive close loop structure, all ants chose the shortest path at the end of the experience (see figure 3c).

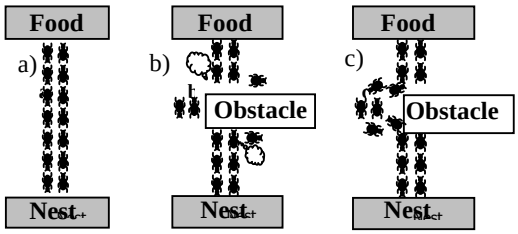


Fig. 3. Ants looking for food

3.2 Ant colony optimization

The first application of ant colony optimization was firstly to solve the Travelling Salesman Problem [13]. In this well-known problem, the goal is to find, for finitely many towns whose pair wise distances are known, the shortest path connecting them. When solving the problem with ant colony optimization, ants walk in the corresponding graph trying to find interesting connecting routes. These paths are built using a stochastic procedure. If an ant f is arrived in town i at iteration k of the algorithm, then the probability that it chooses town j at the next step is given by:

$$p_k^f(i \to j) = \frac{\eta^\alpha(i \to j)\tau_k^\beta(i \to j)}{\sum_{l \in J_k^f(i)} \eta^\alpha(i \to l)\tau_k^\beta(i \to l)} \quad \textbf{(4)}$$

Where:

- $\eta(i \to j)$ is the attractiveness and refers to the "local choice"; in the case of the Travelling Salesman Problem, $\eta(i \to j) = 1/d_{ij}$, with d_{ij} the distance between town i and j.
- $\tau_k(i \to j)$ is the trail of pheromone which has been deposited on the edge $i \to j$ by the ants during previous iterations.
- α, β are weighting factors.
- $J_k^f(i)$ denotes the set of possible towns for the ant f when arrived in town i at iteration k.

At the end of iteration k, the pheromone trail is updated:

$$\tau_{k+1}(i \to j) = (1-\rho)\tau_k(i \to j) + \Delta\tau_k(i \to j) \quad \textbf{(5)}$$

With:

- ρ the evaporation factor;
- $\Delta\tau_k(i \to j)$ the pheromone added by the ants in iteration k. In practise only few ants deposit some pheromone depending on the quality of the path they have performed.

3.3 Ant colony for the identification of non linear functions with unknown structure

The ant colony algorithm will now be adapted to the identification of non linear functions. The problem is modeled such as in section 2 with full binary trees. In that case, ants walk on the graph defined by the tree. The transition rule (4) has to be modified to take into account the specificities of the problem.

If the ant f has chosen the symbol a_1 for the node i at iteration k, then it has to chose the symbol for the left child $2i$ and the right child $2i+1$. The probability that it chooses a_2 and a_3 is given by:

$$\begin{aligned} &p_k^f(\theta_i = a_1 \to \theta_{2i} = a_2) \\ &= \frac{(\eta_{i\to 2i}(a_1 \to a_2))^\alpha (\tau_{i\to 2i,k}(a_1 \to a_2))^\beta}{\sum_{\omega\in\Omega} (\eta_{i\to 2i}(a_1 \to \omega))^\alpha (\tau_{i\to 2i,k}(a_1 \to \omega))^\beta} \\ &p_k^f(\theta_i = a_1 \to \theta_{2i+1} = a_3) \\ &= \frac{(\eta_{i\to 2i+1}(a_1 \to a_3))^\alpha (\tau_{i\to 2i+1,k}(a_1 \to a_3))^\beta}{\sum_{\omega\in\Omega} (\eta_{i\to 2i+1}(a_1 \to \omega))^\alpha (\tau_{i\to 2i+1,k}(a_1 \to \omega))^\beta} \end{aligned} \quad \textbf{(6)}$$

where:

- $\eta_{i\to 2i}(a_1 \to a_2)$ (resp. $\eta_{i\to 2i+1}(a_1 \to a_3)$) is the attractiveness for an ant in node i which has chosen symbol a_1 to choose symbol a_2 (resp. a_3) in node $2i$ (resp. $2i+1$);

- $\tau_{i\to 2i,k}(a_1 \to a_2)$ (rsp. $\tau_{i\to 2i+1,k}(a_1 \to a_3)$) is the trail of pheromone which has been deposited on the edge $a_1 \to a_2$ (rsp. $a_1 \to a_3$) by previous ants in node i in its transition to node $2i$ (resp. $2i+1$) until the iteration k.

Note that the value of the attractiveness and the trail of pheromone do not only depend on the transition (in other words the next choice), but also on the place in the tree, that is to say of i. As a result, the space complexity of this structure is $O(p \cdot m^2)$ where p is the depth of the tree and m the number of symbols. It may seem that this choice is not relevant as the complexity grows quadratically with respect to the number of symbols. However, in automatic control, we look for relative simple expressions and so this number should remain relatively low.

It is possible to take into account some a priori knowledge on the model by choosing the value of η to favor some edges. For instance, the attractiveness to choose constants and variables can be increased to favor simple mathematical expressions. This possibility to incorporate some a priori knowledge is an advantage of ACO compared with genetic algorithms.

However, in this study the goal is to prove the viability of the approach and it is considered that no a priori knowledge on the model is available. Thus, the value of the attractiveness is only chosen as a Boolean value to limit the search to viable trees. For instance:

- if node i has been set to 'cos', then the right child is necessarily $\varnothing$:

$$\begin{aligned} &\eta_{i\to 2i+1}(\cos \to \varnothing) = 1 \\ &\eta_{i\to 2i+1}(\cos \to \omega \neq \varnothing) = 0 \end{aligned} \quad \textbf{(7)}$$

- Nodes of the last level of the tree can only be constants or the variables x;
- ...

It is important to note that ant colony can explicitly take into account these constraints. This is also an advantage of ACO compared with other stochastic algorithms. Finally, the introduction of the symbol $\varnothing$ does not lead to an increase of the complexity of the problem as the ant colony is a constructive algorithm.

4 Numerical results

4.1 Parameter settings

In this study, the following set of symbols is used:

$$\Omega = (1,2,3,4,5,6,7,8,9, x, \sin, \cos, \exp, +, -, /, *, \varnothing) \quad \textbf{(8)}$$

The parameters of the algorithm are:

- $\alpha = 1$ (as the attractiveness is a Boolean value there is no need to tune this value);
- $\beta = 2$;
- $\rho = 0.2$;
- Number of ants: 10.

In this study, a MAX-MIN ant colony algorithm is used [18]. In this algorithm, the value of the pheromone trail is bounded:

$$\sigma_{\min} \leq \tau_{i \to j,k} \leq \sigma_{\max} \quad \textbf{(9)}$$

Doing so, every viable transition has a non zero probability to be chosen. This allows escaping from local minima. Chosen value are $\sigma_{\min} = 1, \sigma_{\max} = 5$. The pheromone trails are initialized to $\sigma_{\max}$.

An elitist algorithm is used. It means that only the best ant is allowed to add some pheromone on the path it has built. This value for the extra amount of pheromone is equal to 1. Once again, to avoid local minima, the pheromone trail is reinforced only if a new better solution is found by the ants.

Our goal is to provide an algorithm which could be used in the automatic control field. In that domain, people are not much used to metaheuristics algorithms. Thus, it is strongly important to have an algorithm which does not require a fine tuning of parameters. That is why the value of parameters has not be optimized in this paper.

4.2 Experimental results

Table 1 presents the results obtained when the depth of the tree is set to 3. As the algorithm is a stochastic one, it can be only validated by statistical results. In table 1, the following results are given:

- The function to be identified. It is used to create the data $(x_i, y_i), i = 1, \ldots, N$, with $N = 100$.
- The threshold. A test is successful when the least square criterion (3) is less than this threshold.
- The number of success. For each experiment, 100 tests are performed.
- Mean number of iterations to compute the solution.

The maximum allowed number of iterations is set to 500. For some functions, noise is added. For that purpose, a random value, comprising in the $\pm 10\%$ of the maximum of the considered function for each sample.

Table 1. Numerical results for a depth of 3

Function	Threshold	Number of success	Mean nb of iterations
$y = \exp(x)$	2	100	130
$y = \sin(x+1)$	1	100	121
$y = \sin(x+1) + noise$	2	100	118

As can be seen from this table, results are very satisfactory for these low dept functions as the initial function is always found by the algorithm. The average number of iterations required for the result to be found is about 125 iterations. The use of a threshold as a stopping criterion is motivated by two points:

- The approach is a "black box" one. The interest is to find a global input / output relation which fits with the data.
- In the real world, data are measured and so measurement noise has to be taken into account: in that case, the cost can never be zero.

The value of the threshold is not so crucial for "pure data" that is to say without any noise. The last case is a more realistic one, as noise is added to the data. In such a situation, the value of the threshold should be adapted to the level of the noise.

Table 2 presents some results obtained for higher depths in the search space. The same kind of results is given, exhibiting once again satisfactory results.

As can be easily seen and guessed, the number of required iterations increase with the depth of trees (the size of the search space increases exponentially with the depth) and with the decrease of the stopping threshold.

Table 2. Numerical results for higher depths

Function	Depth	Threshold	Nb of success	Mean nb of iterations
$y = \exp(x)$	4	1	100	97
$y = \sin(x+1)$	4	1	100	45
$y = \sin(x+1) + noise$	4	3	70	377
$y = \cos(3x) + \sin(1/x)$	4	3	100	170
	4	2	100	183
	5	2	50	368
	5	3	97	286
$y = \exp(x+1) + \sin(\cos(\frac{x+1}{3}))$	6	2	93	314

For the last case, the algorithm almost never finds the initial function $y = \exp(x+1) + \sin(\cos((x+1)/3))$, but "only" the main component $y = \exp(x+1)$. The initial function can be found when decreasing the threshold. Of course, this leads to an increase in the number of iterations.

The results have been obtained with Matlab 2007b on an Intel Core Duo CPU 2.5 GHz. Mean computations time are given in table 3, depending on the depth.

Table 3. Computation times

Depth	3	4	5	6
CPU time (s)	3	6	12	20

5 Discussion

5.1 Considering real variables

In the previous experiments, only a finite set of constants are used (the integers from 1 to 9), which can be seen as a strong restriction of the proposed algorithm. In this section the interest is in the identification of functions which may depend on real variables. For that purpose, a post-processing step is added. In that stage, a classical nonlinear least square method is used, keeping the structure found by the ant colony.

In other words, suppose that the ant colony has found the function $y = 2\sin(x+1)$. Then the following optimization problem will be solved:

$$\min_{a,b} \sum_{i=1}^{N} (y_i - a\sin(x_i + b))^2 \qquad \textbf{(10)}$$

The constant values found by the ant colony are used as the initial point for the optimization algorithm. This method leads to a two step identification procedure, where the ant colony finds the structure of the model, and the non linear least squares method give the fine values of parameters.

Another possibility is of course to use an extension of ant colony algorithms to continuous problems, see for instance [19]-[20]. This approach has not been tested yet.

5.2 Local minima

Ant Colony Optimization belongs to the class of metaheuristics methods. Thus, there is no guarantee on the global optimality of the solution. However, the goal of the identification procedure is to get a representative and usable model of the plant and this goal is achieved even if the solution is a local minimum.

5.3 Identification of nonlinear dynamical systems

The natural extension of this paper deals with the identification of nonlinear dynamical systems modeled by a state space representation:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) \\ y_k &= h(x_k, u_k) \end{aligned} \qquad \textbf{(11)}$$

If some experimental data (u_k, y_k) are available, there is no theoretical difficulties to extend the results obtained in this paper, as the problem refers to the identification of nonlinear functions f and h. The method can be used to define reduced order models as the number of state variables has to be chosen. However, if the system is MIMO (Multi-Inputs Multi-Outputs) and / or has several state variables the number of functions to identify increase, and so the complexity of the problem. Forthcoming works deal with that problem.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

6 Conclusions

In this paper, an identification method has been presented to tackle the problem of the identification of nonlinear functions with unknown structures. The technique is based on Ant Colony Optimization, which is a metaheuristic optimization method introduced by Marco Dorigo and based on the social behaviour of real ants.

The problem is firstly reformulated into a graph exploration problem, using the tree representation of functions. This kind of problem is particularly well suited to Ant Colony Optimization. Results obtained with different benchmark functions are much than satisfactory and prove the viability of the approach.

Forthcoming works deal with:

- the incorporation of the a priori knowledge available on the plant. In that case, the use of adapted values for the attractiveness seems to be a promising idea. For that point, the use of a classical method such as genetic algorithm outperforms our algorithm. However, it will be much easier to add this a priori knowledge with the pheromone process.
- the extension of the method to the identification of nonlinear dynamical systems. Indeed, in Automatic Control, plants are modeled by differential equations and we need such a description of the system.

References

1. Bates, D. M. and Watts, D. G.: Nonlinear Regression Analysis and Its Applications. New York: Wiley (1988).
2. Chen, S., Billings, S. A. and Grant, P. M.: Non-linear system identification using neural networks. In: International Journal of Control, vol. 51(6), 1191--1214 (1990).
3. Su, Y.-T. and Sheen, Y. T.: Neural network for system identification. In: International Journal of Systems Science, vol. 23(12), 2171--2186 (1992).
4. Zhang, G. P.: Neural networks for classification: a survey. In: IEEE Transactions on systems, man and cybernetics, part C: applications and reviews, vol. 30(4), 451--462 (2000).
5. Davidson, J. W., Savic, D. A., Walters, G. A.: Symbolic and numerical regressions: experiments and applications. In: Information Sciences, vol. 150, 95—117 (2003).
6. Gray, J. G., Murray-Smith, D. J., Li, Y., Sharman, K. C. and Weinbrenner, T.: Nonlinear model structure identification using genetic programming. In: Control Engineering Practice, vol. 6, 1341-1352 (1998).
7. Hoai, N.X.: Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: the preliminary results. In: Proceedings of the 5th Australia-Japan co-joint Workshop on Evolutionary Computation, Ontago New Zealand (2001).
8. Madar, J., Abonyi, J. and Szeifert, F.: Genetic programming for the identification of nonlinear input-output models. In: Industrial and Engineering Chemistry Research, vol. 44, 3178--3186 (2005).
9. Raidl, G.R.: A hybrid GP approach for numerically robust symbolic regression. In: Proceedings of the Third Annual Conference on Genetic Programming, pp. 323-28. University of Wisconsin, USA (1998).
10. Salhi, A., Glaser, H., De Roure, D.: Parallel implementation of a genetic-programming based tool for symbolic regression. In: Information Processing Letters, vol. 66, 299—307 (1998).

11. Winkler, S., Affenzeller, M. and Wagner, S.: New methods for the identification of nonlinear model structures based upon genetic programming techniques. In: Proceedings of the 15th International Conference on Systems Science, vol. 1, 386--393, Wroclaw, Poland (2004).
12. Dorigo M. and Gambardella, L. M.: Ant Colony System: a Cooperative Learning Approach to the Traveling Salesman Problem. In: IEEE Transactions on Evolutionary Computation, vol. 1, 53--66 (1997).
13. Dorigo M., Maniezzo V., and Colorni A.: The Ant System: Optimization by a Colony of Cooperating Agents. In: IEEE Transactions on Systems, Man and Cybernetics-Part B, vol. 26(1), 1--13 (1996).
14. Boryczka, M. and Czech, Z.J.: Solving approximation problems by ant colony programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA (2002).
15. Boryczka, M. and Czech, Z.J. and Wieczorek, W.: Ant colony programming for approximation problems. In: Proceedings of the Intelligent Information Systems 2002 Symposium, Sopot, Poland, (2002).
16. Yin, P.-Y.: Ant colony search algorithms for optimal polygonal approximation of plane curves. In: Pattern recognition, vol. 36, 1783—1797 (2003).
17. Green, J., Whalley, J. L. and Johnson, C. G.: Automatic programming with ant colony optimization. In: Proceedings of the 2004 UK Workshop on Computational Intelligence, 70—77, Loughborough University, united Kingdom (2004).
18. Stützle T. and Hoos, H. H.: MAX-MIN Ant System. In: Future Generation Computer Systems, vol. 16, p 889—914 (2000).
19. Bilchev G., Parmee I., The ant colony metaphor for searching continuous spaces. In: Lecture Notes in Computer Science, Vol. 993, 25-39 (1995).
20. Socha K., Dorigo M.: Ant colony optimization for continuous domains. In: European Journal of Operational Research, vol. 185(3), 1155—1173 (2008).

An Immigrants Scheme Based on Environmental Information for Ant Colony Optimization for the Dynamic Travelling Salesman Problem

Michalis Mavrovouniotis[1] and Shengxiang Yang[2]

[1] Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, United Kingdom
mm251@mcs.le.ac.uk
[2] Department of Information Systems and Computing, Brunel University
Uxbridge, Middlesex UB8 3PH, United Kingdom
shengxiang.yang@brunel.ac.uk

Abstract. Ant colony optimization (ACO) algorithms have proved to be powerful methods to address dynamic optimization problems (DOPs). However, once the population converges to a solution and a dynamic change occurs, it is difficult for the population to adapt to the new environment since high levels of pheromone will be generated to a single trail and force the ants to follow it even after a dynamic change. A good solution is to maintain the diversity via transferring knowledge to the pheromone trails. Hence, we propose an immigrants scheme based on environmental information for ACO to address the dynamic travelling salesman problem (DTSP) with traffic factor. The immigrants are generated using a probabilistic distribution based on the frequency of cities, constructed from a number of ants of the previous iteration, and replace the worst ants in the current population. Experimental results based on different DTSP test cases show that the proposed immigrants scheme enhances the performance of ACO by the knowledge transferred from the previous environment and the generation of guided diversity.

1 Introduction

Ant colony optimization (ACO) algorithms have proved to be powerful metaheuristics for solving difficult real-world optimization problems under static environments [2, 12]. ACO is inspired from real ant colonies, where a population of ants searches for food from their nest. Ants communicate via their pheromone trails and they are able to track shortest paths between their nest and food sources. Inspired from this behaviour, ACO algorithms are able to locate the optimum, or a near optimum solution, in stationary optimization problems, efficiently [10]. However, in many real-world problems we have to deal with dynamic environments, where the optimum is moving, and the objective is not only to locate the optimum but to track it also [7]. ACO algorithms face a serious challenge in dynamic optimization problems (DOPs) because they lose their adaptation capabilities once the population has converged [1].

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Over the years, several approaches have been developed for ACO algorithms to enhance their performance for DOPs, such as local and global restart strategies [6], memory-based approaches [5], pheromone update schemes to maintain diversity [3], and immigrants schemes to increase diversity [8, 9]. Most of these approaches have been applied in different variations of the dynamic travelling salesman problem (DTSP), since it has many similarities with many real-world applications [11]. Among these approaches, immigrants schemes have been found beneficial to deal with DTSPs, where immigrant ants are generated and used to replace other ants of the current population [8, 9]. The most important concern when applying immigrants schemes is how to generate immigrants.

In this paper, we propose an immigrants scheme which is based on environmental information for ACO for the DTSP. The environmental information-based immigrants ACO (EIIACO) selects the first n best ants from the previous environment and creates a probabilistic distribution based on the frequency of cities that appear next to each other. Using the information obtained from the previous population, immigrants are generated to replace the worst ants in the current population. The introduced immigrants transfer knowledge and can guide the population toward the promising regions after a change occurs. It is expected that the environmental information-based immigrants scheme may enhance the performance of ACO for DTSPs, especially when the environments before and after a change are similar.

The remaining of this paper is organized as follows. Section 2 describes the DTSP used in the experimental study in this paper. Section 3 describes existing ACO algorithms for the DTSP, which are also investigated as peer algorithms in the experiments. Section 4 describes the proposed EIIACO algorithm. The experimental results and analysis are presented in Section 5. Finally, the conclusions and relevant future work are presented in Section 6.

2 DTSP with the Traffic Jam

The TSP is the most fundamental and well-known *NP*-hard combinatorial optimization problem. It can be described as follows: Given a collection of cities, we need to find the shortest path that starts from one city and visits each of the other cities once and only once before returning to the starting city.

In this paper, we generate a DTSP via introducing the traffic factor. We assume that the cost of the link between cities i and j is $D_{ij} = D_{ij} \times F_{ij}$, where D_{ij} is the normal travelled distance and F_{ij} is the traffic factor between cities i and j. Every f iterations a random number R in $[F_L, F_U]$ is generated probabilistically to represent traffic between cities, where F_L and F_U are the lower and upper bounds of the traffic factor, respectively. Each link has a probability m to add traffic such that $F_{ij} = 1 + R$, where the traffic factor of the remaining links is set to 1 (indicates no traffic). Note that f and m denote the frequency and magnitude of the changes in the dynamic environment, respectively. The TSP becomes more challenging and realistic when it is subject to a dynamic environment. For example, a traffic factor closer to the upper bound F_U repre-

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

sents rush hour periods which increases the travelled distance significantly. On the other hand, a traffic factor closer to the lower bound F_L represents normal hour periods which increases the travelled distance slightly.

3 ACO for the DTSP

3.1 Standard ACO

The standard ACO (S-ACO) algorithm, i.e., Max-Min AS (MMAS), consists of a population of μ ants [12]. Initially, all ants are placed to a randomly selected city and all pheromone trails are initialized with an equal amount of pheromone. With a probability $1 - q_0$, where $0 \leq q_0 \leq 1$ is a parameter of the decision rule, ant k chooses the next city j, while its current city is i, probabilistically, as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{if } j \in N_i^k, \quad (1)$$

where τ_{ij} and $\eta_{ij} = 1/D_{ij}$ are the existing pheromone trails and heuristic information available a priori between cities i and j, respectively, N_i^k denotes the neighbourhood of cities of ant k that have not yet been visited when its current city is i, and α and β are the two parameters that determine the relative influence of pheromone trail and heuristic information, respectively. With the probability q_0, ant k chooses the next city with the maximum probability, i.e., $[\tau]^\alpha[\eta]^\beta$, and not probabilistically as in Eq. (1).

Later on, the best ant retraces the solution and deposits pheromone according to its solution quality on the corresponding trails as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best}, \forall\ (i,j) \in T^{best}, \quad (2)$$

where $\Delta\tau_{ij}^{best} = 1/C^{best}$ is the amount of pheromone that the best ant deposits and C^{best} is the tour cost of T^{best}. However, before adding any pheromone, a constant amount of pheromone is deduced from all trails due to the pheromone evaporation such that, $\tau_{ij} \leftarrow (1-\rho)\,\tau_{ij}$, $\forall\ (i,j)$, where $0 < \rho \leq 1$ is the rate of evaporation. The pheromone trail values are kept to the interval $[\tau_{min}, \tau_{max}]$ and they are re-initialized to τ_{max} every time the algorithm shows a stagnation behaviour, where all ants follow the same path, or when no improved tour has been found for several iterations [12].

The S-ACO algorithm faces a serious challenge when it is applied to DTSPs. The pheromone trails of the previous environment will not make sense to the new one. From the initial iterations the population of ants will eventually converge to a solution, and, thus, high intensity of pheromone trails will be generated into a single path. The pheromone trails will influence ants to the current path even after a dynamic change. The pheromone evaporation is the only mechanism used to eliminate the pheromone trails generated previously, and help ants to adapt to the new environment. Therefore, S-ACO requires a sufficient amount of time in order to recover when a dynamic change occurs.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

3.2 Population-Based ACO

The population-based ACO (P-ACO) algorithm is the memory based version of ACO [5]. It differs from the S-ACO algorithm described above since it follows a different framework. The algorithm maintains a population of ants (solutions), called population-list (memory), which is used to update pheromone trails without any evaporation.

The initial phase and the first iterations of the P-ACO algorithm work in the same way as with the S-ACO algorithm. The pheromone trails are initialized with an equal amount of pheromone and the population-list of size K is empty. For the first K iterations, the iteration-best ant deposits a constant amount of pheromone, using Eq. (2) with $\Delta\tau_{ij}^{best} = (\tau_{max} - \tau_{init})/K$. Here, τ_{max} and τ_{init} denote the maximum and initial pheromone amount, respectively. This positive update procedure is performed whenever an ant enters the population-list. On iteration $K + 1$, the ant that has entered the population-list first, i.e., the oldest ant, needs to be removed in order to make room for the new one, and its pheromone trails are reduced by $\Delta\tau_{ij}^{best}$, which equals to the amount added when it entered the population-list before.

The population-list is a long-term memory, denoted as k_{long}, since it may contain ants from previous environments that survive in more than one iteration. Therefore, when a change occurs, the ants stored in the population-list are re-evaluated in order to be consistent with the new environment where different traffic factors F_{ij} are introduced. The pheromone trails are updated accordingly using the ants currently stored in the population-list.

The P-ACO algorithm has a more aggressive mechanism to eliminate previously generated trails since the corresponding pheromone of the ants that are replaced by other ants in the population-list are removed directly. However, they face the same challenge with S-ACO because identical ants may be stored in the population-list and dominate the search space with a high intensity of pheromone to a single trail. However, the P-ACO algorithm may be beneficial when a new environment is similar to an old one because the solutions stored in the population-list from the previous environments will guide ants towards promising areas in the search space.

4 Immigrants Based on Environmental Information

Many immigrants schemes have been found beneficial in genetic algorithms (GAs) for binary-encoded DOPs [4, 13, 14]. Therefore, to handle the problems described in S-ACO and P-ACO algorithms when addressing DTSPs, immigrants schemes are integrated with ACO since they maintain a certain level of diversity during the execution and transfer knowledge from previous environments [8, 9]. The immigrants are integrated within P-ACO as follows. A short-term memory is used, denoted as k_{short}, where the ants survive only for one iteration. All the ants of the current iteration replace the old ones, instead of only replacing the oldest one as in P-ACO. Then, a predefined portion of the worst ants are

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Algorithm 1 Generate Frequency Of Cities

```
1: input k_short      // short term memory
2: input n            // size of k_short
3: for k = 1 to n do
4:    for i = 1 to l do
5:       city_one_id = ant[k].tour[i]
6:       city_two_id = ant[k].tour[i + 1]
7:       frequency_of_cities[city_one_id][city_two_id]+=1
8:       frequency_of_cities[city_two_id][city_one_id]+=1
9:    end for
10: end for
11: return frequency_of_cities
```

replaced by immigrant ants in k_{short}. When ants are removed from k_{short}, a negative update is made to their pheromone trails and when new ants are added to k_{short}, a positive update is made to their pheromone trails as in P-ACO.

Different immigrants schemes were integrated with P-ACO, such as the traditional immigrants [8], where immigrant ants are generated randomly, the elitism-based immigrants [8], where immigrant ants are generate using the best ant from k_{short}, and the memory-based immigrants [9], where immigrant ants are generated using the best ant from k_{long}. The information obtained from the elitism- and memory-based immigrants to transfer knowledge is based on individual information (one ant). The proposed EIIACO algorithm generates immigrants using environmental information (population of ants) to transfer knowledge from the previous environment to a new one. EIIACO follows the same framework with other ACO algorithms based on immigrants schemes, as described above, but differs in the way immigrant ants are generated.

Environmental information-based immigrants are generated using all the ants stored in k_{short} of the previous environment. Within EIIACO, a probabilistic distribution based on the frequency of cities is extracted, representing information of the previous environment, which is used as the base to generate immigrant ants. The frequency vector of each city i, i.e, $\boldsymbol{D}_{c_i}$, is constructed by taking the ants of k_{short} as a dataset and locating city c_i from them. The successor and predecessor cities, i.e., c_{i-1} and c_{i+1}, respectively, of city c_i are obtained and update $\boldsymbol{D}_{c_i}$ accordingly. For example, one is added to the corresponding position $i-1$ and $i+1$ in $\boldsymbol{D}_{c_i}$. The process is repeated for all cities and a table $S = (\boldsymbol{D}_{c_1}, \ldots, \boldsymbol{D}_{c_l})$ is generated (where l is the number of cities) as represented in Algorithm 1.

An environmental information-based immigrant ant, i.e., $A_{eii} = (c_1, \ldots, c_l)$, is generated as follows. First, randomly select the start city c_1; then, the probabilistic distribution of $\boldsymbol{D}_{c_{i-1}} = (d_1, \ldots, d_l)$ is used to select the next city c_i probabilistically as follows:

$$p_i = \frac{d_i}{\sum_{j \in \boldsymbol{D}_{c_{i-1}}} d_j}, \text{if } i \in \boldsymbol{D}_{c_{i-1}}, \tag{3}$$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Algorithm 2 Generate Environmental Information-Based Immigrant

```
1: input l                    // number of cities
2: input k                    // immigrant ant identifier
3: input frequency_of_cities  // see Algorithm 1
4: step = 1                   // counter for construction step
5: ant[k].tour[step] = random[1, l]
6: while step < l do
7:    step+=1
8:    current_city = ant[k].tour[step − 1]
9:    sum_probabilities = 0.0
10:   for j = 1 to l do
11:      if ant[k].visited[j] then
12:         probability[j] = 0.0
13:      else
14:         probability[j] = frequency[current_city][j]
15:         sum_probabilities += probability[j]
16:      end if
17:   end for
18:   if sum_probabilities = 0.0 then
19:      selected = random[1, l]
20:      while ant[k].visited[selected] do
21:         selected = random[1, l]
22:      end while
23:      ant[k].tour[step] = selected
24:   else
25:      r = random[0, sum_probabilities]
26:      selected = 1
27:      p = probability[selected]
28:      while p < r do
29:         selected+=1
30:         p += probability[selected]
31:      end while
32:      ant[k].tour[step] = selected
33:   end if
34: end while
35: ant[k].tour[l + 1] = ant[k].tour[1]
36: return ant[k]             // generated immigrant ant
```

where d_i is the frequency number where city c_i appears before or after city c_{i-1}. Note that all cities currently selected and stored in A_{eii} have a probability of 0.0 to be selected since they are already visited. In the case where the sum of $p_i = 0.0$, which means that all cities in $\boldsymbol{D}_{c_{i-1}}$ are visited, a random city j that has not been visited yet is selected. This process is repeated until all cities are used in order to generate a valid immigrant ant based on the environmental information, as represented in Algorithm 2. During lines 7–17, the probabilistic distribution is generated, during lines 18–24, the next city is selected randomly from the unvisited cities, and during lines 25–33, the next city is selected probabilistically

from the frequency of cities generated from Algorithm 1. Note that in line 35 the first city stored in the ant is added to the end since the TSP tour is cyclic.

5 Simulation Experiments

5.1 Experimental Setup

In the experiments, we compare the proposed EIIACO algorithm with the S-ACO and P-ACO algorithms, which are described in Section 3. Our implementation follows the guidelines of the ACOTSP[3] framework. All the algorithms have been applied to the `eil76`, `kroA100`, and `kroA200` problem instances, obtained from TSPLIB[4]. Most of the parameters have been optimized and obtained from our preliminary experiments while others have been inspired from the literature [5, 8, 9]. For all algorithms, $\mu = 25$ ants are used, $\alpha = 1$, $\beta = 5$ and $q_0 = 0.0$ (except P-ACO where $q_0 = 0.9$). Moreover, for S-ACO, $\rho = 0.2$. For P-ACO $\tau_{max} = 1.0$, and the size of k_{long} is 3. For EIIACO, the size of k_{short} is 10 and four immigrant ants are generated. For each algorithm on a DTSP instance, $N = 30$ independent runs were executed on the same dynamic changes. The algorithms were executed for $G = 1000$ iterations and the overall offline performance of an algorithm is calculated as follows:

$$\boldsymbol{P}_{offline} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N}P_{ij}^{*}\right), \tag{4}$$

where P_{ij}^{*} defines the best ant since the last dynamic change of iteration i of run j [7].

The value of f was set to 20 and 100, which indicates fast and slowly changing environments, respectively. The probability of m was set to 0.1, 0.25, 0.5, and 0.75, which indicates the degree of environmental changes from small, to medium, to large, respectively. The intervals of the traffic factor were set to $F_L = 0$ and $F_U = 5$. As a result, 8 dynamic environments, i.e., 2 values of $f\times$ 4 values of m, were generated from each stationary TSP instance, to systematically analyze the adaptation and searching capability of each algorithm in the DTSP.

5.2 Experimental Results and Analysis

The experimental results regarding the overall offline performance of the algorithms for DTSPs are presented in Table 1. The corresponding statistical results of two-tailed t-test with 58 degree of freedom at a 0.05 level of significance are presented in Table 2, where "$s+$" or "$s-$" means that the first or the second algorithm is significantly better, respectively, and "$+$" or "$-$" means that the first or the second algorithm is insignificantly better, respectively. Moreover, to

[3] http://www.aco-metaheuristic.org/aco-code/
[4] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Table 1. Experimental results of algorithms regarding the offline performance

Alg. & Inst.	eil76							
	$f = 20$				$f = 100$			
$m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
S-ACO	403.1	441.1	541.7	752.4	378.5	426.5	505.6	711.5
P-ACO	396.9	437.2	538.5	750.9	381.5	432.3	511.8	723.8
EIIACO	392.5	432.0	532.8	739.3	379.3	428.5	506.4	710.6
Alg. & Inst.	kroA100							
	$f = 20$				$f = 100$			
$m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
S-ACO	18806.3	21106.1	26000.6	35812.8	17726.2	19351.3	23672.0	34232.5
P-ACO	18043.5	20767.0	25777.5	35544.5	17891.0	19656.8	24089.5	34633.1
EIIACO	18041.3	20710.7	25447.4	34963.1	17789.5	19520.1	23800.9	34106.4
Alg. & Inst.	kroA200							
	$f = 20$				$f = 100$			
$m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
S-ACO	25979.4	29058.4	36383.0	50344.3	23519.0	23519.0	23519.0	23519.0
P-ACO	24621.4	28525.8	35924.7	49620.5	23529.9	26284.5	33761.4	45314.5
EIIACO	24686.9	28132.4	35164.3	48190.4	23423.4	26136.9	33302.5	44388.0

Table 2. Statistical tests of comparing algorithms regarding the offline performance

Alg. & Inst.	eil76				kroA100				kroA200			
$f = 20, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
S-ACO $\Leftrightarrow$ P-ACO	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIIACO $\Leftrightarrow$ P-ACO	$s+$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$
EIIACO $\Leftrightarrow$ S-ACO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
$f = 100, m \Rightarrow$	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75	0.1	0.25	0.5	0.75
S-ACO $\Leftrightarrow$ P-ACO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$-$
EIIACO $\Leftrightarrow$ P-ACO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
EIIACO $\Leftrightarrow$ S-ACO	$s-$	$s-$	$-$	$+$	$s-$	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$

better understand the dynamic behaviour of the algorithms, the offline performance of the first 500 iterations is plotted in Fig. 1 for fast and slowly changing environments with $m = 0.10$ and $m = 0.75$, respectively. From the experimental results several observations can be made and are analyzed as follows.

First, S-ACO significantly outperforms P-ACO in almost all slowly changing environments whereas it is beaten in fast changing environments; see the results of S-ACO $\Leftrightarrow$ P-ACO in Table 2. This result validates that the S-ACO algorithm can adapt to dynamic changes, but it needs sufficient time to recover and locate the new optimum. The S-ACO algorithm uses pheromone evaporation in order to eliminate pheromone trails that are not useful for the new environment and helps the population of ants to forget the previous solution where they have converged to. On the other hand, P-ACO uses more aggressive method to elim-

inate previous pheromone trails, which guides the population of ants to keep up with the changing environments, even if they change fast. From Fig. 1, it can be observed that P-ACO converges faster than S-ACO, which helps in fast changing environments but not in slowly changing environments. As we have discussed previously, P-ACO has a high risk to maintain identical ants in the population-list and may get trapped in a local optimum solution.

Second, EIIACO outperforms P-ACO in almost all dynamic test environments; see the results of EIIACO $\Leftrightarrow$ P-ACO in Table 2. However, P-ACO is competitive with EIIACO when the environment is slightly changing, e.g., in the `kroA200` with $f = 20$ and $m = 0.10$. This is because the solutions stored in the population-list of P-ACO may be fit only when the previous environment has many similarities with the new one. In cases where the environmental changes are medium to significant, the environmental information-based immigrants transfer more knowledge to the pheromone trails of the next iteration and increase the diversity. In slowly changing environments, EIIACO outperforms P-ACO in all dynamic test cases, either with small or large magnitude of changes. This is because EIIACO has enough time to gain knowledge from the previous environment.

Third, EIIACO outperforms S-ACO in all fast changing environments whereas it is competitive in some slowly changing environments; see the results of EIIACO $\Leftrightarrow$ S-ACO in Table 2. On the smallest problem instance, i.e., `eil76`, S-ACO is significantly better than EIIACO because the diversity provided from the immigrants scheme may not be helpful. Furthermore, it is easier for the population in S-ACO to forget previous solutions and become more adaptive. This validates our expectation for S-ACO, where the time needed to adapt depends on the size of the problem and the magnitude of change. As the problem size and magnitude of change increases, EIIACO is significantly better than S-ACO because the population in S-ACO needs more time to adapt in more complex problem instances; see Fig. 1. Moreover, it can be observed that when the magnitude of change is small, S-ACO converges slowly to a better solution, whereas when the magnitude of change is large, EIIACO converges quickly to a better solution.

Finally, in order to investigate the effect of the environmental information-based immigrants scheme in the population diversity of ACO, we calculate the mean population diversity of all iterations as follows:

$$Div = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N}\left(\frac{1}{\mu(\mu-1)}\sum_{p=1}^{\mu}\sum_{q\neq p}^{\mu}M_{pq}\right)\right), \tag{5}$$

where G is the number of iterations, N is the number of runs, μ is the size of the population, $M_{pq} = 1 - \frac{CE(p,q)}{l}$ is the metric that defines the difference between ant p and ant q, where $CE(p, q)$ is the common edges between the ants and l is the number of cities. A value of M_{pq} closer to 0 means that the two ants are similar. The total diversity results for all dynamic test cases are presented in Fig. 2. It can be observed that S-ACO maintains the highest diversity. Especially, in fast changing environments, S-ACO has an extremely high level of diversity, and this

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

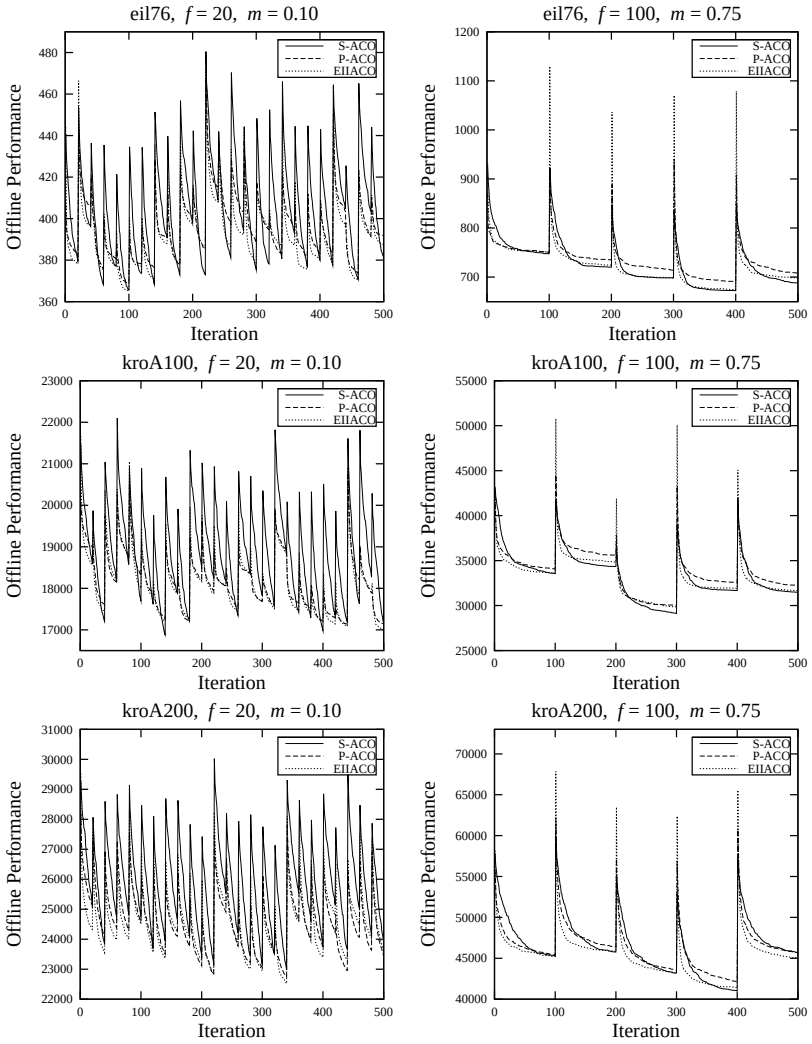


Fig. 1. Dynamic behaviour of investigated algorithms on different DTSPs

shows that it does not have sufficient time to converge. The P-ACO algorithm has the lowest diversity level, which shows the negative effect when identical ants are stored in the population-list. EIIACO maintains higher diversity than P-ACO and much lower diversity than S-ACO. This shows that the diversity generated from the proposed scheme is guided. Moreover, it shows that ACO algorithms that maintain higher diversity levels than others do not always achieve better performance for the DTSP; see Table 1 and Fig. 2.

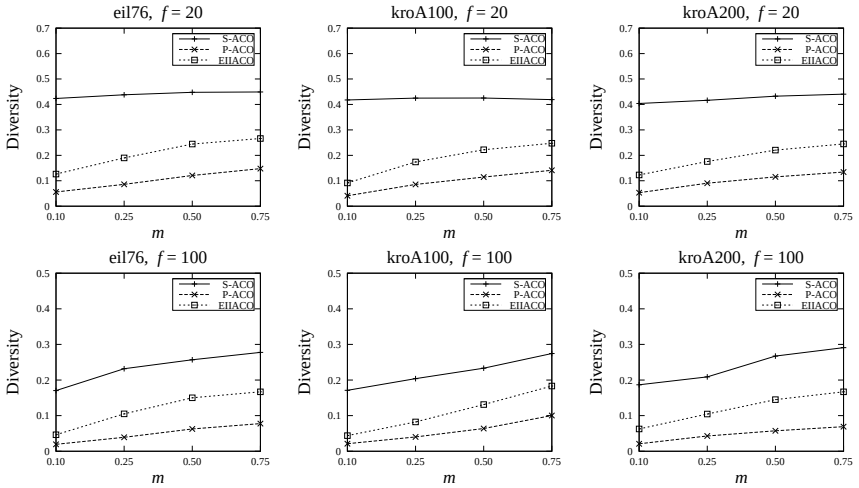


Fig. 2. Overall population diversity of algorithms for different dynamic test problems

6 Conclusions and Future Work

Several immigrants schemes based on individual information, e.g., elitist-based immigrants, have been integrated with ACO algorithms to address different DTSPs in the literature [8, 9]. In this paper, an immigrants scheme based on environmental information, i.e., the frequency of cities appearing next to each other, is proposed for ACO to address the DTSP with traffic factors. A number of generated immigrants replace the worst ants in the current population in order to maintain diversity and transfer knowledge to the pheromone trails for the ants that will construct solutions on the next iteration.

From the experimental results of comparing the proposed EIIACO algorithm with S-ACO and P-ACO algorithms on different cases of DTSPs, the following concluding remarks can be drawn. First, ACO algorithms can be benefited by transferring knowledge from previous environments to the pheromone trails using immigrants schemes for DTSPs. Second, S-ACO has good performance in slowly and slightly changing environments and it is comparable with EIIACO, especially on small problem instances. Third, EIIACO outperforms other ACO algorithms in fast changing environments, while it is comparable with P-ACO in some slightly changing environments. Fourth, EIIACO is significantly better than P-ACO in almost all slowly changing environments. Finally, guided diversity is usually better than random diversity.

For future work, it will be interesting to compare or hybridize EIIACO with other immigrants schemes, which are based on individual information [8, 9], and investigate the interaction between the two types of schemes. As another interesting future work, EIIACO can be applied in more challenging optimization problems, e.g., vehicle routing problems [11].

7 Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/2.

References

1. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm intelligence: from natural to artificial systems. Oxford University Press, New York (1999)
2. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst., Man and Cybern., Part B: Cybern. 26(1), 29–41 (1996)
3. Eyckelhof, C.J., Snoek, M.: Ant system for a dynamic TSP. In: ANTS 2002: Proc. 3rd Int. Workshop on Ant Algorithms, pp. 88–99 (2002)
4. Grefenestette, J.J.: Genetic algorithms for changing environments. In: Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, pp. 137–144 (1992)
5. Guntsch, M., Middendorf, M.: Applying population based ACO to dynamic optimization problems. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) Ant Algorithms 2002. LNCS, vol. 2463, pp. 111–122. Springer, Heidelberg (2002)
6. Guntsch, M., Middendorf, M.: Pheromone modification strategies for ant algorithms applied to dynamic TSP. In: Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tijink, H. (eds.) EvoIASP 2001, EvoWorkshops 2001, EvoFlight 2001, EvoSTIM 2001, EvoCOP 2001, and EvoLearn 2001. LNCS, vol. 2037, pp. 213–222. Springer, Heidelberg (2001)
7. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments - a survey. IEEE Trans. Evol. Comput. 9(3), 303–317 (2005)
8. Mavrovouniotis, M., Yang, S.: Ant colony optimization with immigrants schemes for dynamic environments. In: Schaefer, R., Cotta, C., Kolodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 371–380. Springer, Heidelberg (2010)
9. Mavrovouniotis, M., Yang, S.: Memory-based immigrants for ant colony optimization in changing environments. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcázar, A., Merelo, J., Neri, F., Preuss, M., Richter, H., Togelius, J., Yannakakis, G. (eds.) EvoApplications 2011. LNCS, vol. 6624, pp. 324–333. Springer, Heidelberg (2011)
10. Neumann, F., Witt, C.: Runtime analysis of a simple ant colony optimization algorithm. Algorithmica 54(2), 243–255 (2009)
11. Rizzoli, A. E., Montemanni, R., Lucibello, E., Gambardella, L. M.: Ant colony optimization for real-world vehicle routing problems – from theory to applications. Swarm Intelli. 1(2), 135–151 (2007)
12. Stützle, T., Hoos, H.: The MAX-MIN ant system and local search for the traveling salesman problem. In: Proc. 1997 IEEE Int. Conf. on Evol. Comput., pp. 309–314 (1997)
13. Yang, S.: Genetic algorithms with memory and elitism based immigrants in dynamic environments. Evol. Comput. 16(3), 385–416 (2008)
14. Yu, X., Tang, K., Yao, X.: An immigrants scheme based on environmental information for genetic algorithms in changing environments. In: Proc. 2008 IEEE Cong. of Evol. Comput., pp. 1141–1147 (2008)
15. Yu, X., Tang, K., Chen, T., Yao, X.: Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization. Memetic Comput. 1(1), 3–24 (2009)

A Pheromone Trails Model for MAX-MIN Ant System

Nikola Ivkovic[1], Marin Golub[2], Mirko Malekovic[1]

[1] Faculty of Organization and Informatics, University of Zagreb
{nikola.ivkovic, mirko.malekovic}@foi.hr
[2] Faculty of Electrical Engineering and Computing, University of Zagreb
marin.golub@fer.hr

Abstract. Pheromone trails are the main media for gathering collective knowledge about a problem, and have a central role in all ant colony optimization algorithms. Setting appropriate trail limits for the MAX-MIN ant system (MMAS) is important for good performance of the algorithm. We used rigorous analysis to develop expressions that model the influence of trail limits on MMAS behavior. Besides the general model, specific formulas for ATSP, TSP and QAP are presented. Assumptions on which our model is founded are experimentally validated. The paper gave general guidance for estimating the trail limits ratio established on exact analytical models. Experiments on tested problems showed a high level of agreement with predictions made by the presented model.

Keywords: pheromone trail, trail limit, MAX-MIN ant system, Ant colony optimization, Swarm intelligence

1 Introduction

Many practical computational problems have a too high a complexity to be solved with exact algorithms. One common approach to overcome high complexity is to use heuristic algorithms that cannot guarantee finding an optimal or sufficiently good solution, but in practice they often do.

Ant colony optimization (ACO) [1], [2] is a metaheuristic inspired by the foraging behavior of a colony of biological ants. Together with the ant colony system [3], the MAX-MIN ant system [4] is one of the most popular and successful [5] ACO algorithms. The MAX-MIN ant system has desirable characteristics like robustness, as it works with a population of solutions, natural parallelism and applicability to a wide range of problems [6], [7]. The disadvantage of MMAS, as with many nature inspired algorithms, is the usage of parameters that require appropriate tuning otherwise the algorithm may fail to find admissible solutions [8].

The main characteristic of the MMAS is the existence of trail limits, where all trails are maintained inside an interval bounded by some predefined lower and upper trail limits. The motivation for limiting pheromone trails is to avoid algorithm stagnation [4]. There are two possible approaches for setting the trails limits. The first

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

is to set minimum and maximum limits after experimental measuring of the algorithm's performance. The second is to use an analytical expression to choose appropriate limits. Of course, analytically estimated limits can be fine-tuned with experimental measurements.

The paper is structured as follows. Section 2 gives a brief description of the MMAS and Section 3 gives a brief description of studied problems. In Section 4, a trail separation effect is described, modeled and experimentally evaluated. In Section 5, formulas for trail limits and probabilities are constructed, analyzed and compared. In Section 6, predictions of the presented model are experimentally evaluated. In Section 7 we summarize our findings and stress the importance of using exact formulas in trail limit estimation.

2 MAX-MIN Ant System Description

The MAX-MIN ant system is an extension of the ant system with improved performance for many optimization problems. It uses a colony of ants that construct a population of solutions, based on pheromone trails and heuristic values, both associated with building components. Heuristic values are used only with some optimization problems and for others it is always 1. The algorithm constructs a solution by adding solution components in the list of components that specify partial solutions, until an entire solution is constructed. The probability of selecting solution component $c(i)$ is given in (1). Index i denotes a solution construction step, $\tau_{c(i)}$ is the trail value associated with component $c(i)$, and $\eta_{c(i)}$ is the heuristic value associated with $c(i)$. In step i, a component is selected from L_i, a set of components. Parameters α and β are used to maintain balance between trails and heuristic values.

$$p_{c(i)} = \frac{\tau_{c(i)}^{\alpha} \cdot \eta_{c(i)}^{\beta}}{\sum_{k \epsilon L_i} \tau_k^{\alpha} \cdot \eta_k^{\beta}} \quad (1)$$

After the population of solutions is constructed the best solution is found and trails are updated. The update process includes trail evaporation (2) for all trails, and trail reinforcement (3) for all components included in the iteration or global best solution.

$$\tau_{c(j)} = (1 - \rho) \cdot \tau_{c(j)} \quad (2)$$

$$\tau_{c(k)} = \tau_{c(k)} + \frac{1}{f(S^{best})} \quad (3)$$

Parameter ρ is the trail evaporation rate, and $f(S^{best})$ is the goodness of the iteration best or the global best solution. If a trail gets smaller than the minimum trail limit, the trail is set to the minimal value and if a trail gets bigger than the maximum trails limit, the trail is set to the maximal value. After the trails update, ants construct a new population of solutions and the process repeats. If algorithm stagnation is detected, the trails are reset to their initial values. If a fitness function $f(S^{opt})$ of the optimal solution can be estimated, then initial trails and maximum trail limit are set as:

$$\tau_0 = \tau_{max} = \frac{1}{\rho \cdot f(S^{opt})}. \quad (4)$$

3 Optimization Problems

The Quadratic Assignment Problem (QAP) and the Travelling Salesman Problem (TSP) are well known optimization problems that arise in many practical applications. These are known to be NP-hard problems. Also, TSP and QAP cannot be approximated by polynomial approximation algorithms, unlike some other NP-hard problems, and are NPO-complete [9], [10]. Only brief problem descriptions are presented here. For a more comprehensive introduction, it is advisable to consult literature that is substantial for the selected problems.

For the TSP, there is a set of cities and all distances between cities are known. The problem is to find a tour with minimum total length. All cites must be visited exactly once and a traveler must end the tour by coming back to the starting city. Alternatively, cities are called nodes or vertices and direct links between two cities are called edges or arcs. If edges have directions, the problem is called asymmetrical. Otherwise, if all edges have the same distances in both directions, the problem is called symmetrical. Although TSP is a general term and includes asymmetrical and symmetrical variants, more often only symmetrical variants are studied and denoted as a TSP.

For the QAP, there is a set of facilities and an equally sized set of locations. Flow weights between facilities and distances between locations are known in advance. The problem is to allocate facilities to locations in a way such that the sum of the products of flow weights and distances is minimized.

All ATSP and TSP test problems used in empirical studies presented in this article are taken from *TSPLIB* library publicly accessible at *http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/* and *VLSI Data Set* accessible at *http://www.tsp.gatech.edu/vlsi/*. All QAP test problems used in the article are from *QAPLIB* library publicly accessible at *http://www.seas.upenn.edu/qaplib/*.

4 Trails Separation

Trails serve as a media for transferring collectively gathered knowledge about the problem into solution construction step. As the algorithm progresses, trails are changed to hopefully raise the probability for constructing an optimal or near optimal solution. The components that were reinforced in the previous steps are more likely to be chosen in the next solution construction. Because of this autocatalytic process, it is probable that the algorithm will converge toward one solution; hopefully an optimal or near optimal one. The trails of components that form this solution will probably be around maximal and all the others will be around minimal trail value.

The trail separation can be defined as a state in which the most trails are separated into two non-overlapping intervals I_{MIN} and I_{MAX}, separated by the interval I_{MID}. The

interval I_{MIN} includes the lower and I_{MAX} includes the upper trail limit. If the set S_X denotes the set of components inside interval I_X, than the separation effect can be formally defined as a state that satisfies the inequalities (6) and (7) for some arbitrary small r, $0 \leq r < 1$.

$$S_{MID} = (S_{MIN} \cup S_{MAX})^C \quad (5)$$

$$|S_{MID}| \leq r \cdot |S_{MIN}| \quad (6)$$

$$|S_{MID}| \leq r \cdot |S_{MAX}| \quad (7)$$

4.1 Experimental Evaluations

To evaluate trail separation effect on the test problems, 100 runs of the algorithm were executed and distributions of trails were gathered. Table 1 lists selected problems that consist of 90 QAP, 18 ATSP and 59 TSP problems, giving 16700 runs in total. A problem name contains information about the problem size; but for ATSPs, this rule is sometimes violated, so the sizes for ATSPs are explicitly listed. The parameters were set to α=1, ρ=0.1 and the number of ants was set equal to a problem size n. The iteration best reinforcement strategy was used, except for bigger problems (more than 300 cities for ATSPs and more than 400 for TSPs) where the global best strategy was used. For ATSPs and TSPs β=4 and for QAPs β=0.

Table 1. List of selected problems used in experiments

ATSP									
Problem	Size	Problem	Size	Problem	Size	Problem	Size	Problem	Size
br17	17	p43	43	ft53	53	kro124p	100	rbg403	403
ftv33	34	ftv44	45	ftv55	56	ftv170	171	rbg443	443
ftv35	36	ftv47	48	ftv64	65	rbg323	323		
ftv38	39	ry48p	48	ft70	70	rbg358	358		
TSP list									
gr17, gr21, gr24, fri26, gr48, hk48, eil51, berlin52, st70, eil76, pr76, rat99, kroA100, kroB100, kroC100, kroD100, kroE100, rd100, eil101, lin105, pr107, gr120, pr124, xqf131, pr136, pr144, ch150, kroA150, kroB150, pr152, u159, rat195, d198, kroA200, kroB200, tsp225, pr226, xqg237, gil262, pr264, pr299, lin318, linhp318, pma343, pka379, bcl380, pbk411, fl417, pbn423, pbm436, pr439, pcb442, d493, u574, pr654, xql662, u724, dkg813, pr1002									
QAP list									
tai10a, tai10b, had12, nug12, rou12, tai12a, had14, chr15a, esc16a, esc16b, esc16c, esc16e, esc16g, esc16h, esc16i, esc16j, had16, nug16a, nug16b, nug17, tai17a, had18, nug18, had20, lipa20a, lipa20b, nug20, rou20, tai20a, nug21, nug22, bur26a, nug27, nug28, kra30a, kra30b, lipa30a, lipa30b, nug30, tai30a, tai30b, tho30, esc32b, esc32c, esc32d, esc32h, kra32, tai35a, lipa40a, lipa40b, tai40a, tai40b, sko49, lipa50a, lipa50b, tai50a, tai50b, wil50, sko56, lipa60a, lipa60b, tai60a, tai60b, esc64a, sko64, tai64c, lipa70a, lipa70b, sko72, lipa80a, lipa80b, tai80a, tai80b, sko81, lipa90a, lipa90b, sko90, sko100a, sko100b, sko100c, sko100d, sko100e, sko100f, tai100a, tai100b, wil100, esc128, tai150b, tho150, tai256c									

Possible trail values were divided into k intervals: $I_1 = I_{MAX} = [10^{-1/2}\tau_{max}, \tau_{max}]$, ... , $I_i = [10^{-i/2}\tau_{max}, 10^{-(i-1)/2}\tau_{max}\rangle$, ... , $I_{k-1} = [10^{-(k-1)/2}\tau_{max}, 10^{-(k-2)/2}\tau_{max}\rangle$, $I_k = I_{MIN} = [\tau_{min}, 10^{-(k-1)/2}\tau_{max}\rangle$. For ATSPs, there are $n*(n-1)$ components from which n components are selected for one solution. This gives a predicted number of components $Count(I_{MAX}) = n$ for I_{MAX} interval and $Count(I_{MIN}) = n*(n-2)$ for I_{MIN}. For TSPs, the predicted number of components is $Count(I_{MAX}) = 2n$ and $Count(I_{MIN}) = n*(n-3)$, and for QAPs $Count(I_{MAX})=n$ and $Count(I_{MIN}) = n*(n-1)$.

Figure 1 shows a scatter plot of the experimentally measured $|S_{MAX}|$ marked with circles and $|S_{MIN}|$ marked with squares for ATSPs (a), TSPs (b) and QAPs (c) after 500 iterations. Respective predicted values are drawn with lines. In all cases it is noticeable that experimentally obtained data follows predicted values rather well. There are a few reasons why the measured values differ from the predicted values.

Firstly, the algorithm has not yet reached the trail separation phase. An example of this is the trail distribution for the TSP problem kroB100 shown on Fig. 1d. After 100 iterations, there is still a noticeable number of components with trails between I_{MIN} and I_{MAX} intervals. After 200 iterations, trails outside of I_{MIN} and I_{MAX} intervals are barely noticeable on the graph and values are very close to the predicted one. After 300 iterations, the situation is further improved by a barely noticeable increase of I_{MIN}. Greater sized problems, as expected, often need more time to converge to a solution and to achieve trail separation.

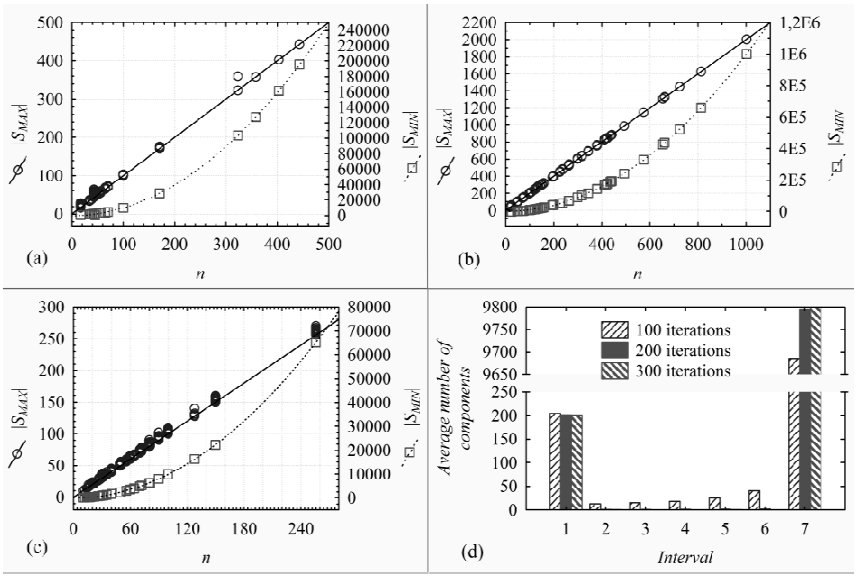


Fig. 1. First three graphs represent scatter plots for ATSP (a), TSP (b) and QAP (c) with related prediction curves. Bar graph (d) shows trails distribution for kroB100 instance of TSP problem.

Secondly, when the trail separation occurs at a certain iteration and the algorithm has constructed a near optimal solution, it is possible that in the next iteration, the algorithm would find a better solution that differs from the previous in a few components. These new components will gradually rise from I_{MIN} interval to I_{MAX} interval, and the components that are being replaced will go from I_{MAX} to I_{MIN} interval.

The third case is when two or more different solutions with some common components are constructed as iteration best solution in an alternating manner. Then, when the separation effect occurs, more components than predicted can be in the I_{MAX} interval. As the algorithm proceeds, one solution can manage to be reinforced more than others and take others out of the I_{MAX} interval. This can be observed on Fig. 1d where the average $|S_{MAX}|$ for kroB100 between 100 and 200 iterations falls closely to the predicted values.

Coefficients of the determination R^2, a measure of how well experimentally obtained values agree with predicted values, are calculated and listed in Table 2. The coefficient of determination for linear functions $Count(I_{MAX})$ is equal to the square of the correlation coefficient. Data in Table 2 shows that the predicted values agree with the experimentally obtained values very well, since for all cases R^2 is very close to 1 (exactly 1 would mean perfect matching).

Table 2. Coefficient of determination R^2

ATSP		TSP		QAP	
$\|S_{MAX}\|$	$\|S_{MIN}\|$	$\|S_{MAX}\|$	$\|S_{MIN}\|$	$\|I_{MAX}\|$	$\|I_{MIN}\|$
0.9993548	0.9999519	0.9990805	0.9996935	0.9993020	0.9999996

5 Trail Limits and Solution Construction Probabilities

For a fixed iteration of the MMAS algorithm it is possible to calculate the exact probability for one ant to construct predefined solution identified by an ordered list of components. Order in the list is not important for all optimization problems, but it is inherent for the MMAS solution construction process. The probability for constructing a solution as an ordered list is equal to the product of the probabilities of selecting individual components. To construct such expression it is necessary to know all the trails and heuristic values, and also the α and β parameters that influence the components selection.

5.1 Stützle – Hoos Expression for Trails Limits

Stützle and Hoos proposed in [4] an analytical expression (8) for calculating appropriate maximal and minimal trail values.

$$\tau_{min} = \tau_{max} \cdot \frac{1-\sqrt[n]{p_{best}}}{(avg-1)\cdot\sqrt[n]{p_{best}}}. \quad (8)$$

The average number of components that can be selected in construction steps is *avg*, the probability of constructing the best solution is p_{best}, and the number of components in the constructed solution is n. Heuristic values, as well as the α parameter, are neglected in expression (8). Mathews extended the Stützle-Hoos expression in [11] by setting the α parameter as an exponent on minimal and maximal trails. In [4] and [11] there is no differentiation between a solution described with a particular order of components, its probability is further on denoted as p_π, and a class of solutions, which are equivalent to a solution described with a particular order of components.

5.2 Exact Expressions for Trail Limits

In the presumption of complete trail separation and by neglecting heuristic values, if any are used, exact expressions that directly follow the process of constructing a solution can be obtained. All possible solution components that can be used for solution construction, in the solution construction step i, constitute the set L_i. Using the random-proportional rule (1), the probability of selecting the component with maximal trail value is:

$$p_{c(i)} = \frac{\tau_{max}^{\alpha}}{\tau_{max}^{\alpha} + \tau_{min}^{\alpha} \cdot (|L_i| - 1)} \tag{9}$$

$$\vartheta = \frac{\tau_{min}}{\tau_{max}} \tag{10}$$

$$p_{c(i)} = \frac{1}{1 + \vartheta^{\alpha} \cdot (|L_i| - 1)} \tag{11}$$

The probability for constructing the solution described with an ordered list of components constituting only of components associated with maximal trails is given in (12). The number of components contained in the solution is denoted as n.

$$p_\pi = \frac{1}{\prod_{i=1}^{n} \left(1 + \vartheta^{\alpha} \cdot (|L_i| - 1)\right)} \tag{12}$$

5.2.1 Expressions for ATSP

Solution construction starts by randomly selecting the first node. The probability of selecting the node that is first in the predefined permutation of components is $1/n$. Without losing generality, nodes can be labeled sequentially starting from 1 to n, so that edge $(i, i+1)$ is the component of the predefined solution as shown in the Fig. 2. The case when the first node is selected is shown in Fig. 2a.

The next step is to select the second node, and by this, implicitly the first edge. For the first edge that can be selected, one component is associated with maximal trail value and $|L_1|-1=n-2$ are associated with minimum trail value. In general, if an ant stays on a predefined path and has already selected i nodes, than it can select from $|L_i|-1=n-i-1$ nodes as shown in Fig. 2b. Again, only one edge has the maximal value and all the others have minimal trail value.

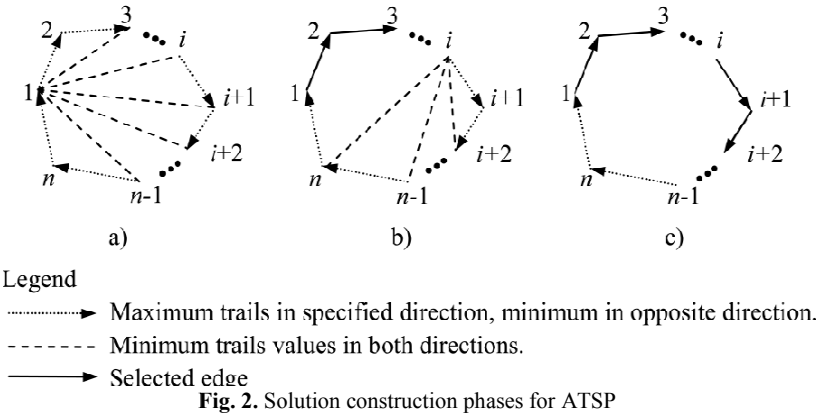


Fig. 2. Solution construction phases for ATSP

At the end of solution construction, the ant is in the node labeled with n-1. As shown in Fig. 2c, there is no real choice for selection since $|L_{n-1}|$-1= 0 and only the component with maximum trail can be selected. This permutation of nodes identifies a solution, and although the edge (n, 1) is not explicitly selected, it is selected implicitly and the solution construction is over. Multiplying single probabilities for selecting nodes in a predefined order gives (13), the probability for constructing an ATSP solution with all the edges associated with the maximum trail value. One particular permutation represents n equivalent solutions that are trivially generated by rotating permutations. All cyclic permutations of the nodes have the same tour length. The probability for constructing any of n cyclic permutations that represent one equivalent solution is given by (14).

$$p_{\pi}^{ATSP}(n, \vartheta^{\alpha}) = \frac{1}{n} \cdot [\prod_{i=1}^{n-2}(1 + \vartheta^{\alpha} \cdot i)]^{-1} \quad (13)$$

$$p^{ATSP}(n, \vartheta^{\alpha}) = n \cdot p_{\pi}^{ATSP}(n, \vartheta^{\alpha}) = [\prod_{i=1}^{n-2}(1 + \vartheta^{\alpha} \cdot i)]^{-1} \quad (14)$$

5.2.2 Expressions for TSP

The solution construction for a TSP is very similar to that for an ATSP, but the probability of selecting the first component is different. Fig. 2 can be reused as visualization aid if arrows are neglected and all edges are bidirectional. When the first node is randomly selected, the same as with ATSP, there is a choice of selecting two edges that have maximal trails (edges: (1, 2) and (1, n)) and n-3 edges with minimal trails. After that, solution construction has defined constructing direction and probabilities of selecting following components are the same as with ATSP. The probability for constructing a particular solution coded directly by one permutation of nodes is given in (15). One permutation identifies $2n$ equivalent TSP solutions. Along with choosing one of n nodes as starting node, we can choose 2 distinct directions. The expression for the class of equivalent solutions for TSP is given in (16).

$$p_{\pi}^{TSP}(n,\vartheta^{\alpha}) = \frac{1}{n}\cdot\frac{1}{2+\vartheta^{\alpha}\cdot(n-3)}\left(\prod_{i=1}^{n-3}(1+\vartheta^{\alpha}\cdot i)\right)^{-1} \quad (15)$$

$$p^{TSP}(n,\vartheta^{\alpha}) = 2n\cdot p_{\pi}^{TSP}(n,\vartheta^{\alpha}) = \frac{2\cdot\left[\prod_{i=1}^{n-3}(1+\vartheta^{\alpha}\cdot i)\right]^{-1}}{2+\vartheta^{\alpha}\cdot(n-3)} \quad (16)$$

5.2.3 Expressions for QAP

To construct a QAP solution is to select couplings of elements from a facilities set and a locations set. In every construction step, an element is selected randomly with uniform distribution from the available elements from one set (set of available facilities or locations), and then its coupling element from the other set is selected using the random-proportional rule (1). The probability of selecting one particular element from the first set is 1/*n* in the first construction step, 1/(*n*-1) in the second construction step, and generally 1/(*n*-*i*+1) in the *i*-*th* construction step. In the first step $|L_1|=n$, and generally $|L_i|=n-i+1$ for the step *i*. The probability of constructing one particular component in a predefined permutation is given by (17). The solution is the set of couplings, not their permutations, so there are *n*! equivalent solutions coded with one permutation. The expression for constructing a predefined solution that has all the trails with maximal value is given in (18).

$$p_{\pi}^{QAP}(n,\vartheta^{\alpha}) = \prod_{i=1}^{n}\left[\frac{1}{(n-i+1)}\cdot\frac{1}{(1+\vartheta^{\alpha}\cdot(n-i))}\right] \quad (17)$$

$$p^{QAP}(n,\vartheta^{\alpha}) = n!\cdot p_{\pi}^{QAP}(n,\vartheta^{\alpha}) = \prod_{i=1}^{n-1}\frac{1}{(1+\vartheta^{\alpha}\cdot i)} \quad (18)$$

5.3 Expressions Comparisons and Analysis

Expressions for ATSP, TSP and QAP are similar, so it is convenient to define a function (19) and rewrite expressions for ATSP (20), TSP (21) and QAP (22).

$$p^{LIN}(\mathrm{n},\vartheta^{\alpha}) = \frac{1}{\prod_{\mathrm{i}=1}^{\mathrm{n}-1}(1+\vartheta^{\alpha}\cdot\mathrm{i})} = \frac{\vartheta^{-\alpha\mathrm{n}}}{(\vartheta^{-\alpha})_{\mathrm{n}}} = \frac{\vartheta^{-\alpha\mathrm{n}}\cdot\Gamma(\vartheta^{-\alpha})}{\Gamma(\vartheta^{-\alpha}+\mathrm{n})} \quad (19)$$

$$p^{ATSP}(n,\vartheta^{\alpha}) = p^{LIN}(n-1,\vartheta^{\alpha}) \quad (20)$$

$$p^{STSP}(n,\vartheta^{\alpha}) = \frac{2}{(2+\vartheta^{\alpha}\cdot(n-3))}\cdot p^{LIN}(n-2,\vartheta^{\alpha}) \quad (21)$$

$$p^{QAP}(n,\vartheta^{\alpha}) = p^{LIN}(n,\vartheta^{\alpha}) \quad (22)$$

Figure 3a shows that the p^{LIN} function changes considerably with ϑ^{α} and *n* arguments, and only for a rather limiting region gives a probability that is not very close to either 0 or 1. If, instead of using $|L_i|$ in construction steps, average value is used as in Stützle-Hoos and the α parameter is taken into account, then the probability of constructing a solution is (23). An equivalent expression, in a different formulation, was previously developed by Matthews [11].

$$p^{AVG}(n,\vartheta^{\alpha}) = \left(1+\vartheta^{\alpha}\cdot(avg-1)\right)^{-n}. \quad (23)$$

5.3.1 Stützle – Hoos and Average Approximation Error

The error caused by using an average approximation instead of an exact formula is shown on Fig. 3b. Both axes use a logarithmic scale. The vertical axis represents the ratio of p^{AVG}/p^{LIN}. Although this graph was originally drawn for the QAP (note that $p^{QAP}=p^{LIN}$), the same graph gives the error ratios for the ATSP and the TSP because on a scale like this graph lines are the same.

The graph shows that except for very small ϑ^{α} values, introduced error is huge and this becomes more serious as n grows. In an extreme case, when $p^{AVG} \cdot avg^{n}<1$, ϑ^{α} becomes bigger than one, that is $\tau_{min} > \tau_{max}$. The Stützle-Hoos expression causes an error ratio with even higher magnitudes if $\alpha \neq 1$, otherwise it is equal to those for p^{AVG}. The problems with the Stützle-Hoos expression are empirically detected by others. Wong and See noted in [12] that minimal trail limits are often set too low and that this affects the algorithms performance. These findings correspond with the error ratio shown in Fig. 3b, where it is noticeable that the Stützle-Hoos probability, as a special case of p^{AVG}, underestimates ϑ^{α} often by many orders of magnitude.

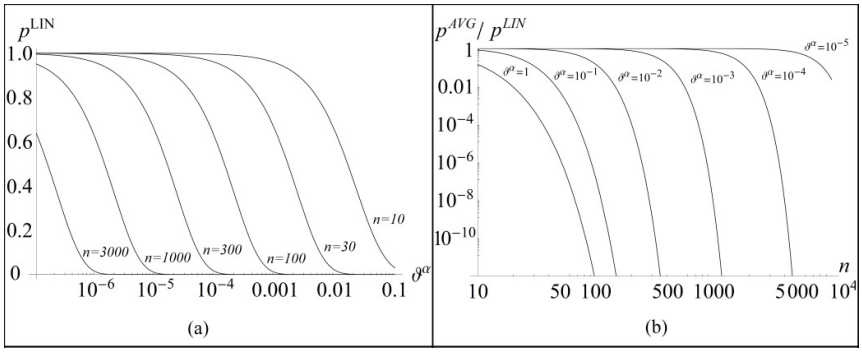


Fig. 3. Dependence of p^{LIN} function on n and ϑ^{α} arguments (a) and error ratio of average approximation (or Stützle – Hoos) and exact expressions (b).

6 Experimental model testing

To test the proposed model, we compare the performance of the algorithm for different ϑ ratios and different α parameters. The aim of these experiments is to see how well the proposed model explains differences in the performance of the algorithm. The results of the experiments are listed in the Table 3 as normalized mean solutions over 100 runs, for each parameter settings. For a problem instance and the α parameter (one row in the table), the minimal mean solution was selected and then all mean solutions were divided with this value. The parameters were set to α=1, β=4 (for the ATSP and the TSP), ρ=0.1, the number of ants was set equal to a problem size n, the iteration best strategy was used and the maximum number of iterations was set to 1000.

For minimal mean solutions, with normalized value 1, respective ϑ^{α} is provided in the brackets. The results show that optimal ϑ depends on problem size n and the α parameter as predicted by the model. In general, data shows that when α changes, so does the optimal ϑ, but the optimal ϑ^{α} stays approximately the same. The results in Table 3 also show that it is better to set ϑ^{α} too low than to set it too high (relatively close to 1), since generally the leftmost column has better mean solutions then the rightmost column. (This is of course not true for small problems that have optimal ϑ^{α} in rightmost columns. For small problems, too high ϑ^{α} values are not measured by these experiments.) These behaviors occur because too high ϑ^{α} prevents the algorithm from properly using trails to guide the algorithm towards the (near) optimal solution. When ϑ^{α} is set too low, at first the algorithm progresses normally, but after passing a certain minimum/maximum trail ratio, a search space is no longer explored. Instead, the algorithm constructs the same solutions all over again.

Table 3. Normalized mean solutions for MMAS

Problem	α	ϑ=1e-6	ϑ=1e-5	ϑ=1e-4	ϑ=1e-3	ϑ=1e-2	ϑ=1e-1
ftv35 (ATSP)	2/3	1.002462	1.001899	1.001543	**1(1.0e-2)**	1.000449	1.040856
	1	1.005004	1.004676	1.00383	1.002428	**1(1.0e-2)**	1.007097
	3/2	1.00962	1.009681	1.008015	1.007027	1.00303	**1(3.2e-2)**
ft70 (ATSP)	2/3	1.000522	1.000408	**1(2.2e-3)**	1.029509	1.108673	1.235636
	1	1.00392	1.003653	1.002417	**1(1.0e-3)**	1.009122	1.205311
	3/2	1.008973	1.008685	1.009779	1.007142	**1(1.0e-3)**	1.128595
kro124p (ATSP)	2/3	1.008777	1.004721	**1(2.2e-3)**	1.003099	1.06977	1.231649
	1	1.023612	1.022723	1.013809	1.004179	**1(1.0e-2)**	1.175048
	3/2	1.027704	1.028072	1.027028	1.021173	**1(1.0e-3)**	1.031121
eil51 (TSP)	2/3	1.002898	1.001613	1.000701	**1(1.0e-2)**	1.010144	1.065729
	1	1.005541	1.005308	1.004606	1.002128	**1(1.0e-2)**	1.014099
	3/2	1.008455	1.009225	1.006586	1.006446	1.001004	**1(3.2e-2)**
pr124 (TSP)	2/3	1.001093	1.000788	**1(2.2e-3)**	1.000431	1.012313	1.08072
	1	1.004805	1.004036	1.001035	**1(1.0e-3)**	1.000607	1.036979
	3/2	1.007934	1.008671	1.006843	1.003956	**1(1.0e-3)**	1.008198
ch150 (TSP)	2/3	1.005388	1.001836	**1(2.2e-3)**	1.013028	1.10239	1.205364
	1	1.014749	1.00848	1.00451	**1(1.0e-3)**	1.000149	1.140743
	3/2	1.025587	1.022809	1.022631	1.008084	**1(1.0e-3)**	1.034762
chr20 (QAP)	2/3	1.232387	1.114547	1.061805	**1(1.0e-2)**	1.012952	1.669481
	1	1.331188	1.310806	1.272797	1.102553	**1(1.0e-2)**	1.113854
	3/2	1.524881	1.447104	1.479266	1.411661	1.174907	**1(3.2e-2)**
lipa40a (QAP)	2/3	1.001597	1.000158	**1(2.2e-3)**	1.005087	1.011528	1.012395
	1	1.005012	1.003934	1.001738	**1(1.0e-3)**	1.005246	1.012295
	3/2	1.006576	1.006538	1.005973	1.003545	**1(1.0e-3)**	1.010981
sko100e (QAP)	2/3	**1(1.0e-4)**	1.003395	1.052245	1.085047	1.093219	1.094417
	1	1.022404	1.011157	**1(1.0e-4)**	1.018971	1.085858	1.094746
	3/2	1.028558	1.027869	1.021032	**1(3.2e-5)**	1.010537	1.085239

7 Conclusion

This paper explains the trail separation effect and shows with extensive experimental testing that in practice this effect occurs very closely as predicted with theoretical analysis. The paper stresses the importance of using exact models, instead of approximations, to avoid errors that can make calculated values completely unusable. Experimental results published by others detected that minimal trail limits are often set too low. This correlates with our findings that outside a limited domain, previous approximate expressions give significantly lower minimum trail limits. It is experimentally confirmed that choosing appropriate trail limits depends on the problem size and the α parameter as predicted by our model. If the trail limits ratio is set high (relatively close to 1) it will prevent the algorithm from learning and by this from successfully solving a problem, but setting it too low will cause algorithm stagnation and prevent it from further improving the solution. The presented model shows that appropriate trail limits, more precisely their ratio, vary considerably with a problem size and the α parameter.

References

1. Dorigo, M., Di Caro, G.: Ant colony optimization: A new meta-heuristic. In: Angeline, P.J. et al. (eds.), Proceedings of the Congress on Evolutionary Computation, vol. 2, pp. 1470–1477. IEEE Press (1999)
2. Dorigo, M., Di Caro, G., Gambardella, L. M.: Ant Algorithms for Discrete Optimization. Artificial Life, 5(2):137--172 (1999)
3. Dorigo, M., Gambardella, L.M.: Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. IEEE Transactions on Evolutionary Computation, 1(1):53--66, 1997
4. Stützle, T., Hoos, H. H.: MAX–MIN Ant System. Future Generation Computer Systems, 16(9): 889 -- 914, 2000
5. Dorigo, M.: Ant colony optimization. Scholarpedia, 2(3):1461, revision #82084 (2007)
6. Dorigo, M., Stützle, T.: Ant Colony Optimization, p. 39, MIT Press (2004)
7. Dorigo, M., Birattari, M., Stützle, T.: Ant Colony Optimization-- Artificial Ants as a Computational Intelligence Technique. IEEE Computational Intelligence Magazine (2006)
8. Pellegrini, P., Birattari, M.: Implementation effort and performance: a comparison of custom and out-of-the-box metaheuristics on the vehicle routing problem with stochastic demand. In: Proceedings of the 2007 international conference on Engineering stochastic local search algorithms: designing, implementing and analyzing effective heuristics (2007)
9. Orponen, P., Mannila, H.: On approximation preserving reductions: Complete problems and robust measures. Technical Report C-1987-28, Department of Computer Science, University of Helsinki (1987)
10. Arkin, E. M., and Hassin, R.: Approximating the maximum quadratic assignment problem. In: SODA '00, Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms (2000)
11. Matthews, D. C.: Improved Lower Limits for Pheromone Trails in Ant Colony Optimization. In: Rudolph, G., Jansen, Th., Lucas, S.M., Poloni, C., Beume, N. (eds.) PPSN X. LNCS, vol. 5199, pp. 508--517. Springer, Heidelberg (2008)
12. Wong, K.Y. and See, P.C.: A new minimum pheromone threshold strategy (MPTS) for max-min ant system. Applied Soft Computing, 9(3): 882--888 (2009)

A Comparison of Meta-modeling Techniques for Multiobjective Optimization Problems

Gerardo Montemayor-García, Gregorio Toscano-Pulido, and Eduardo Rodriguez-Tello

Information Technology Laboratory, CINVESTAV - Tamaulipas, Parque Científico y Tecnológico TECNOTAM. Km. 5.5, carretera Cd. Victoria-Soto La Marina. Cd. Victoria, Tamaulipas, 87130, MÉXICO
{gmontemayor,gtoscano,ertello}@tamps.cinvestav.mx

Abstract. Evolutionary Algorithms (EAs) are bioinspired meta-heuristics that have been successfully applied to solve multiobjective optimization problems (MOPs). When these problems are computationally expensive, they can remain intractable even by these meta-heuristics. Therefore, it is necessary to employ an additional strategy in order to reduce the response time of EAs when optimizing these expensive problems. Replacing the original problem with a surrogate model has been an usual strategy for time reduction. However, despite its success, few comparison among surrogate models for MOPs have been reported in the specialized literature. In this paper, we compare empirically four meta-modeling techniques: Radial Basis Functions, Support Vector Regression, Polynomial Regression and Kriging-DACE in different aspects such as accuracy, robustness, efficiency, and scalability with the aim to identify advantages and drawbacks of each meta-modeling technique in order to choose the most suitable one to be combined with multiobjective evolutionary algorithms.

Keywords: Multiobjective evolutionary algorithms, surrogated models

1 Introduction

In recent years, Evolutionary Algorithms (EAs) have been successfully applied for a variety of optimization problems. One of these problems is the simultaneous satisfaction of several conflicting objectives [1]. However, some computationally expensive problems require that EAs evaluate a considerable number of the objective function in order to find sub-optimal solutions, therefore such problems become intractable even for these algorithms. The EA community has used *surrogate models*[2] to approximate as much as possible the behavior of an expensive optimization function, but reducing the computational effort needed to perform the evaluation. Since surrogate models are not new, a wide variety of optimization techniques that use them have been previously proposed [1,2,3,4]. However, few works have performed a comparison among meta-models in order to find the most suitable one for the application at hand. Most of these works have focused just on single-objective problems [5,6,7], while only few works on MOPs [8,9].

[1] Also known as Multiobjective Optimization Problems or MOPs for short.
[2] Also known as meta-models, emulators, or response surface models.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

With respect to the latter subject, Fang *et al.* [9] compared meta-models using a small number of techniques and test problems and Santana *et al.* [8] selected the best surrogate model based in just one criteria. On the other hand, most of the works ignored the effects of the dimensionality on the problems and also the possibility to combine different techniques, in order to approximate independently each of the objective functions of a MOP.

In this paper, we evaluate the appropriate meta-model to be used with an evolutionary algorithm. Such evaluation will be through the appraisal of the following indicators: accuracy, robustness, efficiency, scalability and the suitability to be combined with an evolutionary approach.

The remain of this document is organized as follows: Section 2 provides a brief description of the four surrogate models to be used in this paper. Section 3 describes the comparative experiments among different meta-models. Section 4 discusses the results obtained on the performed study. Finally, Section 5 presents our main conclusions and future work.

2 Background

A meta-model is an approximation of a real model but simpler and preferably with a lower computational cost. In order to construct an approximation function, it is required a limited set of sample points. The objective of a surrogate model is to reduce the total number of evaluations performed on the real objective function, while maintaining a reasonably good quality of the obtained results. Thus, such meta-model will be used to predict new solutions but using a lower evaluation cost than the needed by the original problem. If the simulation is represented as $f(x)$, then the surrogate model will be represented as $f'(x)$, such that $f'(x) = f(x) + e(x)$, where $e(x)$ is the approximation error. The internal behavior of $f(x)$ is not necessary to be known (or understood). Only the behavior of the input/output matters (what is known as *modeling behavior* or *black box modeling*).

Currently, there are a variety of different techniques to construct surrogate functions. The most popular are the *Kriging models*, *Polynomial Regression (PR)*, *Radial Basis Functions Neural Networks (RBF)* and *Support Vector Machines (SVM)*, among others.

Kriging models (KRG) refers to a spatial prediction method based on the minimization of the mean square error (MSE). The Design of Analysis of Computer Experiments (DACE) is a parametric regression model developed by Sacks *et al.* [10,11]. The DACE approach is an extension of Kriging [12] to handle three or more dimensions. One advantage of this model is that the $2k + 2$ (where k is the number of dimensions) parameters needed to make the approximation can be roughly calculated using the *maximum likelihood function*. The Kriging method assumes that the correlation between errors is related to the distance between corresponding points. For more details about the implementation used here see [11].

Polynomial Regression (PR) The regression analysis is a methodology that studies the quantitative association between a function of interest f and N_{BF} basis functions z_j,

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

where there are N_s sample values of the function of interest f_i for a set of basis functions $z_j^{(i)}$. For each observation i, a linear equation is formulated as follows:

$$f_i(\mathbf{z}) = \sum_{j=1}^{N_{FB}} \beta_j z_j^{(i)} + \varepsilon_i, E(\varepsilon_i) = 0, V(\varepsilon_i) = \sigma^2, \quad (1)$$

where the errors ε_i are considered independent with each other with expected value equal to zero, variance σ^2 and $N_{BF} = (k+1)(k+2)/2$. In this case, we will use second order polynomials.

Radial Basis Functions Neural Network (RBFNN) The topology currently known as RBFNN was initially proposed by Poggio and Girosi [13]. The RBFNN uses a single hidden layer without feedback. The number of nodes in the input layer is equal to k independent variables of the problem. The hidden layer consists of m neurons with ϕ_i radial basis as activation functions whose connections with the input nodes encode the centers of the RBFs. Thus, the definition of the weights of the hidden layer can be seen intuitively as the position of the nodes in the space of input data. The r neurons in the output layer refer to a weighted sum of the activations of the hidden layer with weights defined by vector w. For more details about the implementation used here see [14].

Support Vector Regression (SVR) Support vector machines (SVM) are inspired from statistical learning theories. The major advantages of the SVR are 1) there is single global minimum during learning and, 2) the generalization error does not depend on the dimensions of the space. SVMs can be applied to regression problems by the introduction of an alternative loss function [15]. A widely used loss function is $\varepsilon - SVR$, whose goal is to find a function $f(x)$ that has at most ε deviation from the corresponding targets y_i for the training data.

3 Description of the experiments

An important drawback of some of the previously proposed approaches when comparing meta-models in MOPs is the lack of use of different meta-models for separate objectives [4]. In our opinion, since in multiobjective problems each function has a different landscape, intuitively each one can be approximated with a different meta-model. For simplicity, in this paper we use only bi-objective test problems, therefore, we will approximate the first objective with a meta-model $\mathcal{A}$ and the second objective with a meta-model $\mathcal{B}$, where $\mathcal{A}$ and $\mathcal{B}$ can be any of RBF, SVR, PR or KRG.

Although some approaches make a comparison with a small number of test problems or they select the best meta-model based on a single criterion, this study used several criteria to measure the effectiveness of the meta-models. Therefore, in order to allow a quantitative assessment of the performance of the meta-model techniques, we decided to consider five issues: accuracy, robustness, efficiency (both, training and prediction), scalability and the suitability to be combined with an evolutionary approach.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

To evaluate the performance of the different techniques on MOPs, we have selected eight scalable unconstrained continuous multiobjective test problems from the specialized literature (ZDT{1-4,6}, DTLZ{1,7} and Kursawe test problems) [16]. These test functions were selected taking into account the number of local minima and the shape of the Pareto front and because they contain characteristics that are representative of what can be considered as "difficult" in multiobjective optimization problems.

3.1 Parameter setting

An important advantage of the KRG method is that it is possible to perform an online calculation of its own parameters ($\theta's$ and $p's$) using the *maximum likelihood* (for more details see [11]), avoiding with this, the fine-tuning phase. Like KRG, PR does not need any parameter tuning. However, that is not the case for RBF and SVR.

In order to avoid affecting both techniques (RBF or SVR) by poor parameter settings, we decided to perform a full factorial design for each technique. To have a statistical validation, we executed 31 times each combination of parameters using 5 single-objective test functions (Rosenbrock, Rastriging, Griewangk, Sphere, and Ackley). Also, we used 100 solutions for training and other 200 solutions for prediction. The parameters used to perform the full factorial design were:

- For RBF: the number of centers in the hidden layer was in the range of $\{3 - 100\}$.
- For SVR: the parameters C,γ and ϵ, and their ranges of values were $\{2^{-5} - 2^{15}\}$, $\{0.1 - 2\}$ and $\{2^{-10} - 2^{5}\}$, respectively. Since they are continuous parameters, we performed a discretization of each parameter with 20 possible values uniformly spaced from each other. Then, we perform a full factorial design executing each combination of parameters ($20 \times 20 \times 20$) with every test function adopted.

After executing each combination, we decided to identify the best parameters with respect to the $G - metric$ for each test function. We called Best Local Setting (BLS) to these combinations. We also decided to identify the combination whose average was the best $G - metric$ for all problems. This combination was called Best Overall Setting (BOS). Then, we decided to compare both settings using the same experiment configuration previously mentioned. In Figure 1, the results produced by BLS and BOS using the $G - Metric$ are shown. From these results, it seems obvious that both sets behaved similar. However, for simplicity, we decided to select the BOS in the rest of our experiments.

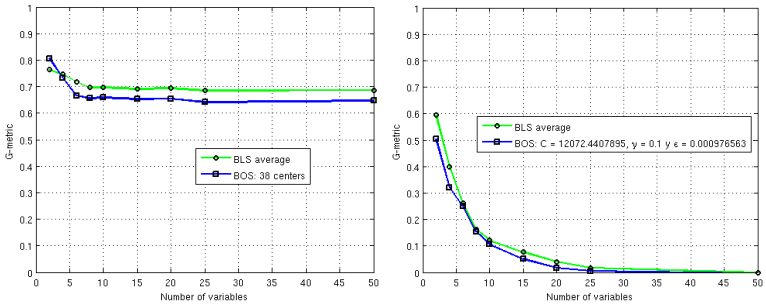
The BOS values were:

- The RBF's BOS value for the number of centers of the hidden layer was: 38 centers.
- The SVR's BOS combination was $C = 12072.4407895$, $\gamma = 0.1$, and $\epsilon = 0.000976$.

The $G - metric$ is aim to measure the accuracy of the meta-models. This metric was applied to the values obtained for each objective of a MOP:

$$G_i = 1 - \frac{\sum_{j=1}^{N} (y_{ij} - \hat{y}_{ij})^2}{\sum_{j=1}^{N} (y_{ij} - \bar{y}_i)^2} = 1 - \frac{\textbf{MSE}}{\textbf{Variance}} \tag{2}$$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



(a) BOS and BLS for RBF (b) BOS and BLS for SVR

Fig. 1. Average accuracy with the BOS and BLS for five single-objective problems.

where N is the size of the validation data set, $\hat{y}_{ij}$ is the predicted value for the objective i on the input j, and y_{ij} is the real value; $\bar{y}_i$ is the mean of the real values on the objective j. The Mean Square Error (MSE) measures the difference between the estimator and the real value. The variance describes how far the values lie from the mean. In this metric, we will prefer larger values of G since this corresponds to a more accurate meta-model. Thus, for mono-objective problems the G_1 is in the range $\{0-1\}$. However, for bi-objective problems the accuracy is the sum $G_1 + G_2$, therefore is in the range $\{0-2\}$.

3.2 Description of the different adopted criteria

- **Accuracy:** is the ability of a technique for making predictions close to the values of the real system. In this paper we use the *G-Metric*.
- **Robustness:** refers to the ability of a technique to achieve a good accuracy on different test problems. Then, the robustness of a meta-model will be given by its average behavior for all adopted test cases.
- **Scalability:** is the ability of a technique to achieve a good accuracy with the increase of the number of variables (i.e., a meta-models is more scalable if it has a good accuracy for every problem size). To measure this criteria, we used the following problem sizes: $v = \{2, 4, 6, 8, 10, 15, 20, 25, 50\}$.
- **Efficiency:**
  - *to train:* this criteria refers to the computational effort required for the technique to build the meta-model. We decided to measure the time it takes to train a model with 100 solutions.
  - *to predict:* refers the computational effort required for the technique to predict responses to new entries. We measured the time it takes to predict 500 solutions.
- **Suitability:** this metric is defined as the degree of difficulty needed to optimize two meta-models created by two different techniques. For measuring this criteria, we decided to use three approaches. The first approach refers to the way in which evolutionary multiobjective approaches (MOEAs) compare solutions. They usually

compare two solutions and select the one with the best dominance rank. Therefore, it would be interesting to build two meta-models, then, produce N distributed points with the meta-models at hand, and compare each pair of the generated points using both, the surrogate functions and the real functions. Then, we will say that a combination of meta-models is better than another if it preserves a better comparative relation with respect to the dominance rank. In order to measure this, we will use the degree of ranking preservation as the ability of the meta-models to maintain the same rank of the solutions with respect to the original functions. A combination of meta-models $\boldsymbol{f}'$ has a perfect **multiobjective ranking preservation (RP)** under the original functions $\boldsymbol{f}$ if:

$$\begin{aligned} & (\boldsymbol{f}(\boldsymbol{x}) \prec \boldsymbol{f}(\boldsymbol{y}) \wedge \boldsymbol{f}'(\boldsymbol{x}) \prec \boldsymbol{f}'(\boldsymbol{y})) \\ \forall x, y \in \mathcal{F} : \ & \vee (\boldsymbol{f}(\boldsymbol{y}) \prec \boldsymbol{f}(\boldsymbol{x}) \wedge \boldsymbol{f}'(\boldsymbol{y}) \prec \boldsymbol{f}'(\boldsymbol{x})) \\ & \vee ((\boldsymbol{f}(\boldsymbol{x}) \nprec \boldsymbol{f}(\boldsymbol{y}) \wedge \boldsymbol{f}(\boldsymbol{y}) \nprec \boldsymbol{f}(\boldsymbol{x})) \wedge (\boldsymbol{f}'(\boldsymbol{x}) \nprec \boldsymbol{f}'(\boldsymbol{y}) \wedge \boldsymbol{f}'(\boldsymbol{y}) \nprec \boldsymbol{f}'(\boldsymbol{x}))) \end{aligned} \tag{3}$$

where $\mathcal{F}$ is the feasible region of the problem. Therefore, the performance measure can be defined as follows:

$$RP = \left(\sum_{i=1}^{N} \sum_{j=i+1}^{N} h(i,j) \right) / \binom{N}{2} \tag{4}$$

where N refer to the number of solutions used to validate the model. In this article, we used $N = 500$ solutions for each size of all test problems. And $h(i, j)$ is:

$$h(i,j) = \begin{cases} & \textbf{if}((\boldsymbol{f}(i) \prec \boldsymbol{f}(j) \wedge \boldsymbol{f}'(i) \prec \boldsymbol{f}'(j)) \\ 1 & \vee (\boldsymbol{f}(j) \prec \boldsymbol{f}(i) \wedge \boldsymbol{f}'(j) \prec \boldsymbol{f}'(i)) \\ & \vee ((\boldsymbol{f}(i) \nprec \boldsymbol{f}(j) \wedge \boldsymbol{f}(j) \nprec \boldsymbol{f}(i)) \wedge (\boldsymbol{f}'(i) \nprec \boldsymbol{f}'(j) \wedge \boldsymbol{f}'(j) \nprec \boldsymbol{f}'(i)))) \\ 0 & \textbf{otherwise} \end{cases} \tag{5}$$

The second approach refers to the fact that when some MOEAs perform the optimization process on the meta-models, they commonly obtain the non-dominated solutions, evaluate them in the real functions, and add them to the elitist population. Therefore, we are interested in knowing the percentage of non-dominated solutions on the real functions that also are non-dominated on the meta-models. Thus, a combination of meta-models $\boldsymbol{f}'$ has a perfect **dominance preservation** under the original functions $\boldsymbol{f}$ if:

$$\forall x \in \mathcal{F} : \ \boldsymbol{f}(\boldsymbol{x}) \text{ is non-dominated} \wedge \boldsymbol{f}'(\boldsymbol{x}) \text{ is non-dominated} \tag{6}$$

where $\mathcal{F}$ is the feasible region of the problem. Therefore, the performance measure can be defined as follows:

$$DP = \left(\sum_{i=1}^{NDS} h(i) \right) / NDS \tag{7}$$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

where NDS is the number of non-dominated solutions in the real function and $h(i)$ is:

$$h(i) = \begin{cases} 1 \textbf{ if}((\boldsymbol{x}_i \text{ is non-dominated in } \boldsymbol{f}) \wedge (\boldsymbol{x}_i \text{ is non-dominated in } \boldsymbol{f}')) \\ 0 \textbf{ otherwise} \end{cases} \quad (8)$$

Finally, the third approach measures the ability of an evolutionary algorithm to optimize the fitness landscape produced by a specific pair of meta-models working on separate objectives of the MOP. Therefore, we decided to optimize the models produced by two different surrogate algorithms within a simple surrogated-assisted MOEA (saMOEA) based on an state of the art algorithm called *NSGA-II* [17]. The saMOEA algorithm alternates five iterations with the *NSGA-II* replacing $\boldsymbol{f}(\boldsymbol{x})$ with $\boldsymbol{f}'(\boldsymbol{x})$, and five iterations with the *NSGA-II* using the expensive $\boldsymbol{f}(\boldsymbol{x})$. A detailed description of this algorithm is presented in Algorithm 1. Since the *Pareto front* of

Algorithm 1 saMOEA($\{\mathcal{A}, \mathcal{B}\}$)

Require: The pair $\{\mathcal{A}, \mathcal{B}\}$ of meta-models to use
Ensure: An evolved population $\mathcal{P}$
1: Create an initial population $\mathcal{P}$ of size 100 with the latin hypercubes routine [18].
2: Evaluate $\mathcal{P}$ in $\boldsymbol{f}(\boldsymbol{x})$
3: **repeat**
4: $\boldsymbol{f}'(\boldsymbol{x}) \leftarrow$ train the pair of meta-models $\{\mathcal{A}, \mathcal{B}\}$ using $\mathcal{P}$
5: $\mathcal{Q} \leftarrow \mathcal{P}$ / $*\mathcal{P}$ is cloned for the optimization in meta-models $*$/
6: $\mathcal{Q} \leftarrow$ 5 iterations of the *NSGA-II* replacing $\boldsymbol{f}(\boldsymbol{x})$ with $\boldsymbol{f}'(\boldsymbol{x})$. The $\mathcal{Q}$ solutions are used in this procedure and 100 new solutions are returned.
7: Evaluate $\mathcal{Q}$ in $\boldsymbol{f}(\boldsymbol{x})$
8: $\mathcal{P} \leftarrow$ the best 100 solutions from $(\mathcal{P} \cup \mathcal{Q})$ are retained using the non-dominated sorting used in the *NSGA-II*
9: $\mathcal{P} \leftarrow$ 5 iterations of the *NSGA-II* using the expensive $\boldsymbol{f}(\boldsymbol{x})$ (returning 100 solutions).
10: **until** to reach 3000 evaluations

the tested problems are known[3], then, we will say that a pair of meta-models $\{\mathcal{A}, \mathcal{B}\}$ is easier to optimize than another $\{\mathcal{C}, \mathcal{D}\}$, if the result from the best solutions found in a fixed number of evaluations (3000 in our case) by the pair $\{\mathcal{A}, \mathcal{B}\}$ is closer to the *Pareto front* than the one from the pair $\{\mathcal{C}, \mathcal{D}\}$. The proximity to the *Pareto front* was measured using the *hypervolume ratio (HVR)* indicator (see [19] for more details).
We will prefer larger values for these performance indicators. Since the values are normalized between $0 - 1$, we will prefer those solutions closer to 1.

3.3 Approach to measure the effectiveness of the meta-models

Below we describe the approach to measure the effectiveness of meta-models used in this paper:

[3] Since we have no information about the Pareto front in the Kursawe test function for problem sizes different to 3, we decided to omit this test function in the third suitability approach.

1. Create a training data set with a latin hypercube[4] [18] of size 100 (since EAs typically handle this population size).
2. Train the two objectives with each pair $\{\mathcal{A}, \mathcal{B}\}$ of a technique[5] with the training data set.
3. Create the validation data set with a latin hypercube of size 500.
4. Predict the validation data set with meta-models $\{\mathcal{A}, \mathcal{B}\}$.
5. Compute the performance measures for the different adopted criteria.
6. Repeat 31 times the steps 1 to 5 and compute the average.

4 Results from the study

In our experiments we used 16 pairs $\{\mathcal{A}, \mathcal{B}\}$ ({KRG, KRG}, {KRG, RBF}, {KRG, SVR}, {KRG, PR}, {RBF, KRG}, ..., {PR, PR}) of techniques to train each problem. However, for simplicity we only draw 5 pairs ({KRG, KRG}, {RBF, RBF}, {SVR, SVR}, {PR, PR} and {*the pair with the best average over all others*}).

4.1 Accuracy, Robustness and Scalability

Figure 2 shows both, the results produced by the application of the G-metric on the test problems (see Figures 2a to 2h) and the mean for all test problems is shown in (see Figure 2(i)). In general, it is clear from this figure, that the worst performance for problem sizes > 10 is for $\{PR, PR\}$. However, in the problems ZDT1, ZDT2, ZDT3 and DTLZ7, $\{PR, PR\}$ had a good accuracy when the problem had a lower number of variables ($v < 10$). Even in specific instances as in ZDT4 where $v = \{2, 4, 6\}$, ZDT3 where $v = \{4, 6\}$ and DTLZ7 where $v = \{6, 8, 10\}$, $\{PR, PR\}$ had the better accuracy. Similarly to $\{PR, PR\}$, $\{KRG, KRG\}$ has an erratic behavior in all problems for $v > 10$, but is considerably better than $\{PR, PR\}$ when $v > 10$.

Although the behavior of $\{SVR, SVR\}$ is not good in ZDT4 and Kursawe test problems, it has the best accuracy on problems ZDT1, ZDT2, DTLZ1 and DTLZ7. Figure 2(d) shows that the $\{RBF, RBF\}$ presented an erratic behavior. However, in the rest of the problems, it shown a good performance. In addition, it is important to note that the best average accuracy for all problems was $\{RBF, RBF\}$ and not for $\{SVR, SVR\}$, contrary to what we expected (see Figure 2(i)).

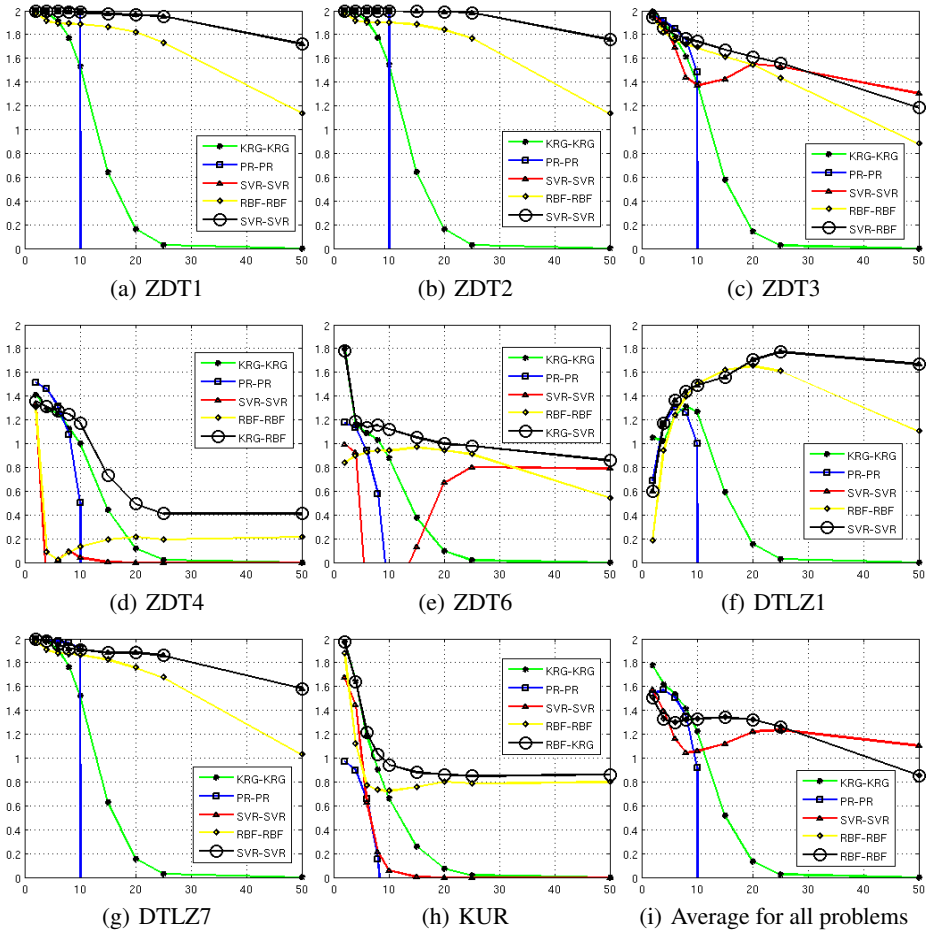
It can be say that $\{RBF, RBF\}$ was not the best in all test problems. However, in most of the problems it showed a stable behavior. For this reason we consider that $\{RBF, RBF\}$ had the best robustness. On the other hand, the worst performance for this measure was $\{PR, PR\}$.

In this case, we associated the size of the problem with respect to its dimensionality. We considered that problems with $v \leq 10$ are of low dimensionality and the rest are considered to be high dimensional. Figure 2(i) shows that $\{KRG, KRG\}$ was the best for low dimensional problems. Moreover, $\{RBF, RBF\}$ was the best for high dimensional problems, since our observations, it was slightly better than $\{SVR, SVR\}$.

[4] Statistical method of stratified sampling that can be applied to multiple variables
[5] Where $\mathcal{A}$ and $\mathcal{B}$ can be any of RBF, SVR, PR or KRG

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Also, it is important to note that for most problems, the best pair of methods was compounded by two different techniques ($\{SVR, RBF\}$ in ZDT3, $\{KRG, RBF\}$ in ZDT4, $\{KRG, SVR\}$ in ZDT6 and $\{RBF, KRG\}$ in Kursawe). Therefore, we think that MOEAs should use a different meta-model for each objective.



(a) ZDT1 (b) ZDT2 (c) ZDT3

(d) ZDT4 (e) ZDT6 (f) DTLZ1

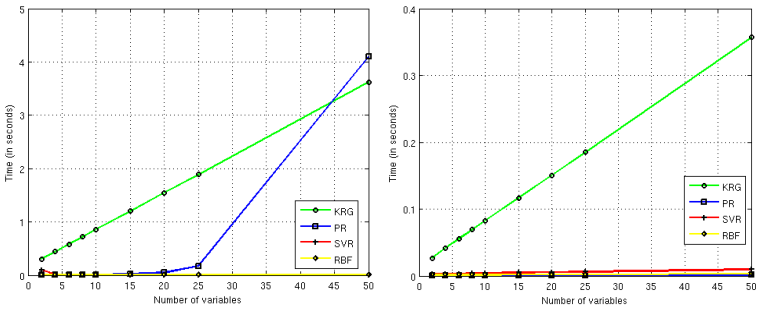
(g) DTLZ7 (h) KUR (i) Average for all problems

Fig. 2. Average accuracy (for all plots: $y - axis$ is the G-metric) by number of variables.

4.2 Efficiency

As mentioned in Section 3, we divided this performance measure in 2 parts. *a*) the time to train a meta-model and *b*) the time to predict the solutions. The results for this performance measure are shown in Figure 3. It can be seen that RBF and SVR required short periods of time in both, training and prediction. On the other side, PR needed a large training but it presented a fast prediction. Contrary, we can say that KRG presented the worst performance for both parts of this performance measure.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



(a) Time to train (b) Time to predict

Fig. 3. Average time of execution for training (a) and prediction (b) in all the problems.

4.3 Suitability

An important feature of RP is that the higher its value is, the lower the probability of adding a false optimum to the problem. This feature is a very important characteristic in evolutionary algorithms given that most of them attempt to avoid these false optima. However, in most cases, it is required to repeat the training of the meta-models, which leads into a reduction in efficiency. It can be seen in Figure 4(a) that $\{SVR, SVR\}$ could outperform the other techniques.

The second performance indicator to measure the suitability is the Dominance Preservation (DP) (see Figure 4(b)). The results for this performance measure are similar to the RP. The best pair of techniques in all problems was $\{SVR, SVR\}$ and again, the best performance in many problems was a combination of two different meta-models. This time, $\{RBF, RBF\}$ produced bad results for problem sizes < 15. In this direction, $\{PR, PR\}$ produced the worst combination for problem sizes > 10.

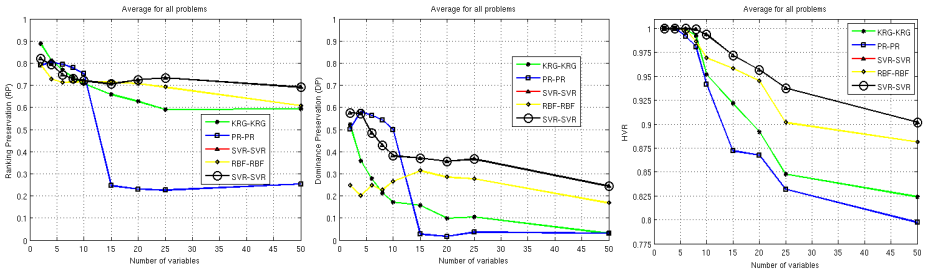
Finally, the experiment for the optimization assisted by meta-models is shown in Figure 4(c). The behavior of the meta-models in this experiment is similar to the RP and DP. With few variables, the most of pairs are close to 1.0, because the most functions are easier to optimize. In this direction, the best results for problem sizes < 10 was $\{KRG, KRG\}$. While the worst meta-models were $\{SVR, SVR\}$ and $\{RBF, RBF\}$. However, the differences are very small. Instead, for problem sizes > 10 the worst meta-model was $\{PR, PR\}$, and the best combination was again $\{SVR, SVR\}$.

5 Conclusions

The purpose of the current study was to compare meta-modeling techniques and evaluate them under multiple aspects in order identify their suitability to be used with MOEAs. The study presented in this paper has provided results about the performance of PR, KRG, RBF and SVR techniques in several test problems with different features.

From the results derived on this study, we can say that the best approach to be used in problem sizes ≤ 6, would be KRG or SVR. If the problem size is greater than 6 but

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



(a) Ranking Preservation (b) Dominance Preservation (c) Meta-modeling optimization

Fig. 4. Average of Ranking Preservation, Dominance Preservation and the optimization assisted by meta-models, by number of variables over all problems

lower than 15, then *KRG*, or *RBF* can be used. Moreover, if the problem size is equal or greater than 15, then, the most obvious technique to be used would be *SVR*.

With respect to efficiency, *RBF* and *SVR* were the winners to train the meta-models. However, *PR* also had a good efficiency to predict new solutions. The worst technique was *KRG*, which had the worst time on both, training and predicting.

An important issue is that if we have a $DP = 1$, all non-dominated solutions in the artificial function will be also non-dominated in the real function. In addition, it should be noted that not necessarily the best accurate meta-model will have the best results in the optimization process. Therefore, in our opinion, SVR presented the best results among the four approaches tested. Finally, as future works, it would be interesting to assess other experiments like the optimization with a MOEA using auto-selection of the best meta-model technique for each objective, in order to fit landscapes for each one function.

Acknowledgments

The first author acknowledges support from CONACyT through a scholarship to pursue graduate studies at the Information Technology Laboratory, CINVESTAV-Tamaulipas. The second and third authors gratefully acknowledge support from CONACyT through projects 105060 and 99276, respectively.

References

1. Ivan Voutchkov and A.J. Keane. Multiobjective Optimization using Surrogates. In I.C. Parmee, editor, *Adaptive Computing in Design and Manufacture 2006. Proceedings of the Seventh International Conference*, pages 167–175, Bristol, UK, April 2006. The Institute for People-centred Computation.
2. Michael T.M. Emmerich, Kyriakos C. Giannakoglou, and Boris Naujoks. Single- and Multi-objective Evolutionary Optimization Assisted by Gaussian Random Field Metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421–439, August 2006.

3. Joshua Knowles. ParEGO: A Hybrid Algorithm With On-Line Landscape Approximation for Expensive Multiobjective Optimization Problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, February 2006.
4. Deepti Chafekar, L. Shi, Khaleed Rasheed, and Jiang Xuan. Multiobjective GA optimization using reduced models. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 35(2):261–265, 2005.
5. L. V. Santana-Quintero, A.A. Montaño, and C. A. Coello. A Review of Techniques for Handling Expensive Functions in Evolutionary Multi-Objective Optimization. *Computational Intelligence in Expensive Optimization Problems*, 2:29–59, 2010.
6. Nestor V. Queipo, Raphael T. Haftka, Wei Shyy, Tushar Goel, Rajkumar Vaidyanathan, and Kevin. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41(1):1–28, January 2005.
7. Ruichen Jin, Wei Chen, and Timothy W. Simpson. Comparative studies of metamodeling techniques under multiple modeling criteria. *Structural and Multidisciplinary Optimization*, 23:1–13, 2000.
8. Luis V. Santana-quintero, Carlos A. Coello Coello, and Alfredo G. Hybridizing surrogate techniques, rough sets and evolutionary algorithms to efficiently solve multi-objective optimization problems. In *Genetic and Evolutionary Computation Conference (GECCO'2008)*, pages 763–764, 2008, ISBN 978-1-60558-131-6.
9. H. Fang, M. Rais-Rohani, Z. Liu, and M.F. Horstemeyer. A comparative study of metamodeling methods for multiobjective crashworthiness optimization. *Computers & Structures*, 83(25-26):2121 – 2136, 2005.
10. Jerome Sacks, William J. Welch, Toby J. Mitchell, and Henry P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–423, 1989.
11. D. Jones. A taxonomy of global optimization methods based on response surfaces. *J. of Global Optimization*, 21(4):345–383, December 2001.
12. Georges Matheron. Principles of geostatistics. *Economic Geology*, 58(8):1246–1266, 1963.
13. T. Poggio and F Girosi. A theory of networks for approximation and learning. Technical report, Massachusetts Institute of Technology, Artificial Intelligence Laboratory and Center for Biological Information Processing, Whitaker College., Zurich, Switzerland, 1989.
14. Tshilidzi Marwala and Monica Lagazio. *Conflict Modeling Using Computational Intelligence Advanced Information and Knowledge Processing Series*. Springer, 2011. ISBN 0857297899, 9780857297891.
15. Steve R. Gunn. Support vector machines for classification and regression. Technical report, Faculty of Engineering, Science and Mathematics School of Electronics and Computer Science, Zurich, Switzerland, 1998.
16. S. Huband, P. Hingston, L. Barone, and L. While. A review of multiobjective test problems and a scalable test problem toolkit. *Evolutionary Computation, IEEE Transactions on*, 10(5):477 –506, Oct. 2006.
17. Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.
18. M D McKay, R J Beckman, and W J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.
19. Xiaodong Li, J. Branke, and M. Kirley. On performance metrics and particle swarm methods for dynamic multiobjective optimization problems. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 576 –583, sept. 2007.

A Surrogate-based Intelligent Variation Operator for Multiobjective Optimization

Alan Díaz-Manríquez, Gregorio Toscano-Pulido, and Ricardo Landa-Becerra

Information Technology Laboratory, CINVESTAV-Tamaulipas
Parque Científico y Tecnológico TECNOTAM
Km. 5.5 carretera Cd. Victoria-Soto La Marina
Cd. Victoria, Tamaulipas 87130, MÉXICO
{adiazm,gtoscano,rlanda}@tamps.cinvestav.mx

Abstract. Evolutionary algorithms are meta-heuristics that have shown flexibility, adaptability and good performance when solving Multiobjective Optimization Problems (MOPs). However, in order to achieve acceptable results, Multiobjective Evolutionary Algorithms (MOEAs) usually require several evaluations of the optimization function. Moreover, when each of these evaluations represents a high computational cost, these expensive problems remain intractable even by these meta-heuristics. To reduce the computational cost in expensive optimization problems, some researchers have replaced the real optimization function with a computationally inexpensive surrogate model. In this paper, we propose a new intelligent variation operator which is based on surrogate models. The operator is incorporated into a stand-alone search mechanism in order to perform its validation. Results indicate that the proposed algorithm can be used to optimize MOPs. However, it presents premature convergence when optimizing multifrontal MOPs. Therefore, in order to solve this drawback, the proposed operator was successfully hybridized with a MOEA. Results show that this latter approach outperformed both, the former proposed algorithm and the evolutionary algorithm but without the operator.

Keywords: Evolutionary Algorithms, Intelligent Genetic Variation Operator, Multiobjective Optimization

1 Introduction

Evolutionary Algorithms (EAs) have been successfully applied to an important variety of difficult multiobjective optimization problems (MOPs). EAs are population-based techniques that make a multidimensional search, finding more than one solution within an execution. Their main advantage lies on their ability to locate solutions close to the global optimum even in highly accident search spaces. However, for many real-world optimization problems, the number of calls of the objective function to locate a suboptimal solution may be high. In many science and engineering problems, researchers have been using computer simulation in order to replace expensive physical experiments with the aim of improving the quality and performance of engineered products and devices, but using

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

a fraction of the needed effort. For example, Computational Fluid Dynamics solvers, Computational Electro Magnetics and Computational Structural Mechanics have been shown to be very accurate. However, such simulations are often computationally expensive, such that, some simulations can take several days or even weeks in order to be completed. Hence, in many real-world optimization problems, the computational cost is dominated by the quantity of objective function evaluations needed to obtain a good solution. This suggests that it is crucial to use any information gained during the optimization process in order to increase the efficiency of optimization algorithms and consequently, to reduce the number of objective function evaluations.

A natural approach used by the EA community to reduce the number of evaluations has been the use of a surrogate model instead of the exact problem [9, 5, 2]. Building a model of the fitness function to assist in the selection of candidate solutions for evaluation has been a natural approach used by the EA community since it reduces the number of evaluations [9, 5, 2].

An alternative approach to reduce the number of evaluations needed to obtain suboptimal solutions by an EA is to use information from the local search space in order to generate several promising solutions for the next generation, these techniques are known as **Intelligent Genetic Operators** [3, 1, 11, 10, 6]

Hence, in this work, we propose a new intelligent genetic variation operator that uses a surrogate model to guide the search with the aim of accelerating its convergence i.e., achieve similar results but using a lower number of function evaluations. The remainder of this work is organized as follows: In Section 2, we give a brief background needed for the understanding of the paper. Section 3 presents the proposed intelligent genetic variation operator denominated Surrogate-based Intelligent Variation Operator (SIVO) and we validate it using a simple approach called Surrogate Assisted Optimizer (SAO). Section 4 presents the incorporation of SIVO into the NSGA-II. Experimental results obtained on synthetic test problems are presented in Section 5. Finally, Section 6 summarizes our main conclusions as well as some possible directions for future work.

2 Background

Definition 1 (Multiobjective Optimization Problem - MOP): A multiobjective optimization problem can be defined as: Find the vector $\boldsymbol{x}^* = [x_1^*, x_2^*, ..., x_n^*]^T \in \mathcal{F}$ which will satisfy the m inequality constrains, $g_i(\boldsymbol{x}) \leq 0$ $i = 1, 2, ..., m$, the p equality constrains $h_j(\boldsymbol{x}) = 0$ $j = 1, 2, ..., p$ and optimize the vector function: $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), ..., f_k(\boldsymbol{x})]^T$, where $\boldsymbol{x} = [x_1, x_2, ..., x_n]^T$ is the vector of decision variables, and the set of inequality and equality constrains will define the feasible region $\mathcal{F}$ and any point in $\mathcal{F}$ will constitute a feasible solution.
Definition 2 (Pareto Optimality): A point $\boldsymbol{x}^* \in \mathcal{F}$ is **Pareto-optimal** if every $\boldsymbol{x} \in \mathcal{F}$ and $I = \{1, 2, \ldots, k\}$ either, $\forall_{i \in I}(f_i(\boldsymbol{x}) = f_i(\boldsymbol{x}^*))$ or, there is at least one $i \in I$ such that: $f_i(\boldsymbol{x}) > f_i(\boldsymbol{x}^*)$.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Definition 3 (Pareto Dominance): A vector $\boldsymbol{x} = [x_1, x_2, ..., x_n]^T$ is said to dominate $\boldsymbol{y} = [y_1, y_2, ..., y_n]^T$ (denoted by $\boldsymbol{x} \preceq \boldsymbol{y}$ if and only if $\boldsymbol{x}$ is partially less than $\boldsymbol{y}$, i.e., $\forall_{i \in \{1,...,n\}} : x_i \leq y_i \wedge \exists_{i \in \{1,...,n\}} : x_i < y_i$.

Definition 4 (Pareto Optimal Set): For a given MOP $\boldsymbol{f}(\boldsymbol{x})$, the Pareto optimal set ($\mathcal{P}^*$) is defined as: $\mathcal{P}^* = \{\boldsymbol{x} \in \mathcal{F} \mid \nexists \boldsymbol{y} \in \mathcal{F} \mid \boldsymbol{f}(\boldsymbol{y}) \preceq \boldsymbol{f}(\boldsymbol{x})\}$

Definition 5 (Pareto Front): For a given MOP $\boldsymbol{f}(\boldsymbol{x})$, and Pareto optimal set $\mathcal{P}^*$, the Pareto front ($\mathcal{PF}^*$) is defined as: $\mathcal{PF}^* = \{\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), ..., f_k(\boldsymbol{x})]^T | \boldsymbol{x} \in \mathcal{P}^*\}$

Definition 6 (Surrogate Model): A surrogate model is a mathematical model that mimics the behavior of a computationally expensive simulation code over the complete parameter space as accurately as possible, using the least amount of data points.

3 Surrogate-based Intelligent Variation Operator (SIVO)

The Surrogate-based Intelligent Variation Operator **SIVO** can be an alternative or a complement to the common genetic variation operators. A key feature which we want for **SIVO** is to improve the solutions previously found by the generation of new non-dominated solutions. For this purpose, we first select a non-dominated solution in order to improve it. Then, we propose three new (desired) solutions in the objective space which outperform the selected one. After that, we try to locate their positions in the variable space. Therefore, we construct the local surrogate model and launch a new optimization algorithm, which tries to find the surrogated solutions which are closest to the three previously proposed points. At the end, the final surrogate solutions are evaluated in the real optimization problem with the aim that the new solutions found be better than the original one.

3.1 Selection in the objective space of the new solutions to be created

The fundamental idea of **SIVO** is that given a solution $\boldsymbol{x}$ and its evaluation $f(\boldsymbol{x})$, we want to find new solutions (S) whose evaluation ($f(S)$) are situated at a distance Δf. For a bi-objective problem, the evaluation of the solutions $f(S)$ that we wish to find are shown graphically in Figure 3.1, such solutions can be calculated as follows.:

$$f''(S_1) = [f_1(\boldsymbol{x}) - \Delta f_1 \quad f_2(\boldsymbol{x})] \tag{1}$$
$$f''(S_2) = [f_1(\boldsymbol{x}) \quad f_2(\boldsymbol{x}) - \Delta f_2] \tag{2}$$
$$f''(S_3) = [f_1(\boldsymbol{x}) + \Delta f_1 \quad f_2(\boldsymbol{x}) - 2\Delta f_2] \tag{3}$$

It is obvious that solutions $f''(S_1)$ and $f''(S_2)$ will help us to improve the algorithm's convergence with respect to the original point. Moreover, $f''(S_3)$ will help to improve the diversity with respect to $f(\boldsymbol{x})$. Finally, we can distinguish that $f''(S_1)$, $f''(S_2)$ and $f''(S_3)$ are not dominated among them.

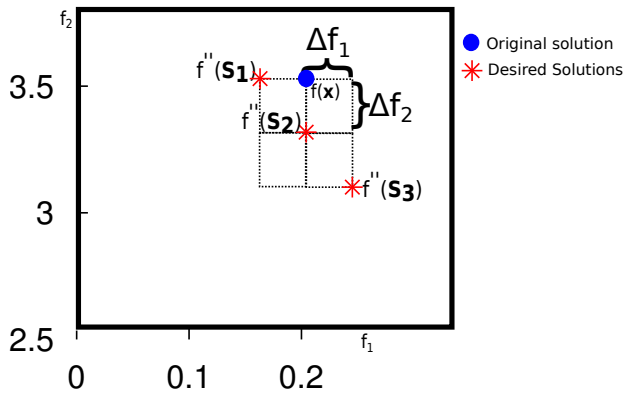


Fig. 1. Solutions that we want to create with the operator, $\Delta f = [\Delta f_1 \quad \Delta f_2]$

3.2 Creation and optimization of the local surrogate model

Once we have our targets (the three non-dominated points in the objective space that we would like to find), then we proceed to create the surrogate model with the solutions previously stored.

Once we know the position of the desired solution we would like to find $S = [S_1 \quad S_2 \quad S_3]$ where:

$$f(S_1) \approx f''(S_1) \wedge f(S_2) \approx f''(S_2) \wedge f(S_3) \approx f''(S_3) \tag{4}$$

Then we proceed with the creation of a local surrogate model with any surrounding solutions previously stored. It should be obvious that we need to store every evaluated solution in a repository in order to use such information for building the surrogate models. Therefore, in order to select the surrounding solutions, we use a simple clustering algorithm, such that, we can find the solutions which belong to the same cluster of $\boldsymbol{x}$ (the clustering algorithm is performed in the variable space). After we have the surrounding solutions, then we create a surrogate model $f'(x)$ with such solutions. After that, we minimize the distance of the surrogate model to each solution in S. The formulation of the optimization problem for each solution i in S is as follows:

Find x' such that,

$$\text{Minimize} \sum_{j=1}^{M} (f'_j(\boldsymbol{x}'_i) - f''_j(S_i))^2 \tag{5}$$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

where M refers to the number of objectives of the MOP. In this work, we use the Nelder-Mead algorithm to optimize the above optimization problem. The solutions x'_i found will be the new solutions created by the operator $NS = \{S_1, S_2, S_3\}$. We recommend to optimize the previous optimization problem with a local search algorithm and use the solution $\boldsymbol{x}$ as initial solution, because in this way if the optimizer can not find an exact solution, at least we can get a solution close to $\boldsymbol{x}$. Although the error (Equation 6) is a important factor, there is not a strong effect in **SIVO**, because if we create new solutions with **SIVO** and they have a big error with respect to the original point, then, in the next generation these solutions created will serve as new solutions for the creation of the surrogate model, and therefore the error would be lower. The pseudocode for **SIVO** is shown in Algorithm 1.

$$error = \sum_{1=1}^{|S|} \text{abs}(f(S_i) - f''(S_i)) \tag{6}$$

Algorithm 1 SIVO$(\boldsymbol{x}, \Delta f, P)$

Input: Solution $\boldsymbol{x}$, Δf, Stored solutions P
Output: New solutions NS
  $NS = \emptyset$
  $f''(S_1) \leftarrow [f_1(\boldsymbol{x}) - \Delta f_1 \quad f_2(\boldsymbol{x})]$
  $f''(S_2) \leftarrow [f_1(\boldsymbol{x}) \quad f_2(\boldsymbol{x}) - \Delta f_2]$
  $f''(S_3) \leftarrow [f_1(\boldsymbol{x}) + \Delta f_1 \quad f_2(\boldsymbol{x}) - 2\Delta f_2]$
  $C \leftarrow Clustering(P)$
  $S_C \leftarrow \{\forall_{i \in P}\, C_i = C_x\}$ {C_x is the Cluster which belong $\boldsymbol{x}$}
  f'=Metamodel(Sc) {The surrogate model is created with S_c}
  for $i \leftarrow 1$ to 3 **do**
    $x'_i \leftarrow$Solve problem in Equation 5 with $f''(S_i)$
    $NS \leftarrow NS \cup x'_i$
  end for

3.3 Surrogate Assisted Optimizer (SAO)

In order to evaluate the effectiveness of the proposed operator, we decided to build a simple stand-alone algorithm based on SIVO. The resulting algorithm was called the Surrogate Assisted Optimizer or SAO for short. SAO works as follows: first an initial population of solutions P_0 of size $|P|$ is created using a latin hypercube [1] [4]. Then, it starts the main procedure of SAO which refers to the selection of the solutions that SIVO will try to improve. Therefore, we select β solutions in P according to their dominance rank and their crowding distance. Then the SIVO algorithm is applied to those solutions. The resulting solutions from this procedure will be stored in P. Since P is fixed-size archive, then dominance rank and crowding distance will be applied again in order to maintain the file size. The SAO pseudocode is show in Algorithm 2.

[1] A latin hypercube is a statistical method for stratified sampling that can be applied to multiple variables

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Algorithm 2 SAO Algorithm

Input: $|P|$ (size of population), $tmax$ (number of iterations), β (percentage of solutions to apply **SIVO**, $\beta = [0, |P_0|])$, Δf
Output: P_{tmax} (Final population)
1: $P_0 \leftarrow$GenerateRandomPopulation()
2: **for** $t \leftarrow 0$ to $tmax$ **do**
3: $S \leftarrow$Choose β solutions of P_t (with dominance rank and crowding distance)
4: **for** $j \leftarrow$ to $|S|$ **do**
5: $NewSolutions \leftarrow$**SIVO**$(S_j, \Delta f, P_t)$
6: $NS \leftarrow NS \cup NewSolutions$
7: **end for**
8: $P_t \leftarrow P_t \cup NS$
9: $F \leftarrow$fast-non-dominated-sorting(P_t)
10: $P_{t+1} = \emptyset$
11: **repeat**
12: Assign crowding distance on F_i
13: $P_{t+1} \leftarrow P_{t+1} \cup F_i$
14: **until** $|P_{t+1}| + |F_i| \leq |P_t|$
15: Crowding distance on F_i
16: $P_{t+1} \leftarrow P_{t+1} \cup$(Choose the first $(|P| - |P_{t+1}|)$ individuals of F_i)
17: **end for**

3.4 SAO Results

In order to allow a quantitative assessment of the performance of the proposed algorithms, $ZDT\{1-4, 6\}$ test problems [12] and two metrics were adopted.

1. **Inverted Generational Distance (IGD)**: The concept of generational distance was introduced by Van Veldhuizen & Lamont [7, 8] as a way of estimating how far are the elements in the Pareto front produced by our algorithm from those in the true Pareto front of the problem. This measure is defined as:

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \tag{7}$$

   where n is the number of non-dominated vectors found by the algorithm being analyzed and d_i is the Euclidean distance (measured in objective space) between each of these and the nearest member of the true Pareto front. It should be clear that a value of $GD = 0$ indicates that all the elements generated are in the true Pareto front of the problem. Therefore, any other value will indicate how "far" we are from the global Pareto front of our problem. In our case, we implemented the "inverted" generational distance measure (IGD) in which we use as a reference the true Pareto front, and we compare each of its elements with respect to the front produced by an algorithm. In this way, we are calculating how far are the elements of the true Pareto front, from those in the Pareto front produced by our algorithm. In this metric, we will prefer smaller values of IGD, since this represents a closer result to the real front.
2. **Hypervolume (HV)** The HV calculates the area covered by all the solutions in Q using a reference point W, which is a vector identified as having the worst objective function values.

$$HV = volume(\bigcup_{i=1}^{|Q|} v_i) \tag{8}$$

where for each solution $i \in Q$, a hypercube v_i, is constructed with reference to W. The HV is the union of all hypercubes. The HV depends strongly of the choice of W. A set of non-dominated solutions can be different values for two different choices of W. In this metric, we will prefer larger values of HV, since this represents a closer result to the real front.

The parameters used for **SAO** are a initial population of $|P| = 100$, a $\Delta f =$ [0.1 0.5], and $\beta = 5$. With a $\beta = 5$, the algorithm by generation will do 15 evaluations, because the **SIVO** creates 3 new solutions by point, therefore, $\beta * 3 = 15$. We measure the hypervolume at every 100 evaluations during 2000 function evaluations in our approach. This procedures was executed during 31 times and the mean of the results was plotted.

Figure 2 shows the performance of **SAO** in the five test functions adopted. In all the problems, we can see that the hypervolume had a rapid growth such that, **SAO** required only (approx.) 400 evaluations to obtain good results. However, since the HV does not tell us whether the algorithm is close to the real front, we decided to use the IGD performance measure. IGD confirms, in most of the problems (ZDT1, ZDT2, ZDT3, and ZDT5), the results obtained by HV, due to SAO obtained an IGD very close to 0 with only 400 evaluations. But, when optimizing the ZDT4 test function, the IGD reached its best value within 700 evaluations, and from this point the IGD improvement was practically null. Moreover, this value is relatively far from 0 (where 0 refers that the solutions found are on the true Pareto front). This result shows that the algorithm tends to stuck in local optima in this type of problems (multifrontal problems). This allows us to speculate that an hybridization of **SIVO** with an Evolutionary Algorithm (EA), can improve the overall results.

4 The NSGA-II + SIVO

SAO could work properly when trying to optimize most of the adopted problems. However, it was also obvious that SAO did not behave as it was expected when it tried to optimize multifrontal problems, since it presented premature convergence. Therefore, it was clear that an additional effort in order to develop a more "robust" algorithm was necessary. In order to solve such a problem, we decided hybridize the **SIVO** with an Evolutionary Algorithm. In this section, we describe the methodology used for the integration of **SIVO** with the **NSGA-II**.

The **NSGA-II+SIVO** works as follows: First, it creates a random initial population of solutions P_0 of size $|P|$ (in this work, the initial population was created with latin hypercubes). Then, in the iteration t a child population Q_t is created with the traditional genetic operators (mutation and crossover), a mixed population R_t is created with $P_t \cup Q_t$, the solutions in R_t are used as

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

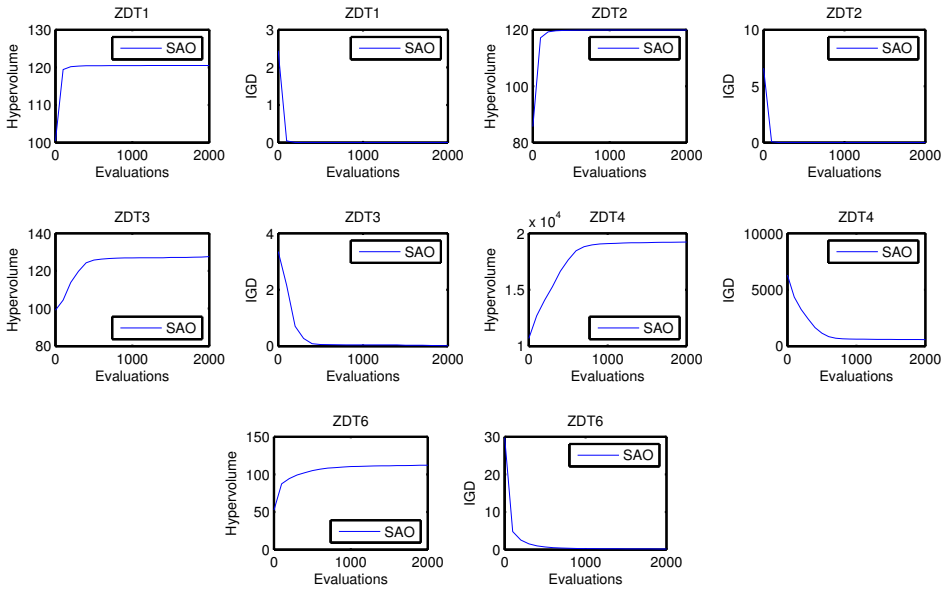


Fig. 2. Hypervolume and Inverted Generational Distance for SAO.

stored solutions for **SIVO**. It is easy to see that the main procedure of the integration of **NSGA-II** with **SIVO** is the choice of the solutions to apply the operator, we choose β solutions in R_t to apply **SIVO**, the solutions are chosen according to their dominance rank and their crowding distance (as in NSGA-II), the new solutions created NS are mixed with R_t and its size is limited to $|P|$ (with dominance rank and crowding distance), the $|P|$ solutions are stored in P_{t+1}, and they are the new population for the next generation. The pseudocode for **NSGA-II+SIVO** is show in Algorithm 3.

5 Comparison of results

We selected the following parameters for NSGA-II and NSGA-II+SIVO: $|P| = 100$, and for NSGA-II+SIVO were used a $|P| = 84$, a $\Delta f = [0.1\ 0.5]$, and $\beta = 5$. Both algorithms used a mutation rate of $1/Number\_of\_variables$, crossover rate of 0.8, an index crossover of 15, and a mutation index of 20. We also decided to compare with respect to our previously proposed approach (SAO), therefore, we selected the same parameters as indicated before. In order to know how the behavior of the proposed approach is, we applied the hypervolume and IGD performance measures each 400 evaluations during 2000 evaluations. This procedure was executed 31 times and the results were plotted as boxplot graphics.

Figure 3 shows the results of hypervolume in NSGA-II, NSGA-II+SIVO and SAO. As pointed out above, SAO had troubles to escape from local optima in multifrontal problems (ZDT4). While NSGA-II+SIVO clearly shows how the

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Algorithm 3 NSGA-II+SIVO Algorithm

Input: $|P|$ (size of population), $tmax$ (number of iterations), β (number of solutions to apply **SIVO**, β), Δf
Output: P_{tmax} (Final population)
1: $P_0 \leftarrow$GenerateRandomPopulation()
2: **for** $t \leftarrow 0$ to $tmax$ **do**
3: $Q_t \leftarrow$make-new-pop(P_t){use selection, crossover and mutation to create a new population}
4: $R_t \leftarrow P_t \cup Q_t$
5: $S \leftarrow$Choose β solutions of R_t (with dominance rank and crowding distance)
6: **for** $j \leftarrow$ to $|S|$ **do**
7: $NewSolutions \leftarrow$**SIVO**$(S_j, \Delta f, R_t)$
8: $NS \leftarrow NS \cup NewSolutions$
9: **end for**
10: $R_t \leftarrow R_t \cup NS$
11: $F \leftarrow$fast-non-dominated-sorting(R_t)
12: $P_{t+1} = \emptyset$
13: **repeat**
14: Assign crowding distance on F_i
15: $P_{t+1} \leftarrow P_{t+1} \cup F_i$
16: **until** $|P_{t+1}| + |F_i| \leq |P_t|$
17: Crowding distance on F_i
18: $P_{t+1} \leftarrow P_{t+1} \cup$(Choose the first $(|P| - |P_{t+1}|)$ individuals of F_i)
19: **end for**

algorithm achieves a significant improvement with respect to the increase of the number of evaluations. Moreover, comparing both, the original NSGA-II and the NSGA-II+SIVO, it is clear that the SIVO operator helped to improve the convergence rate of NSGA-II+SIVO.

The IGD results are shown in the Figure 4, in most of the problems the SAO algorithm obtained good results. However, it can be seen in this metric that only the hybrid approach (NSGA-II+SIVO) could deal with the ZDT4 test problem.

6 Conclusions

The purpose of this paper was to propose a new operator based on surrogate models. The operator can be used as a simple stand-alone optimizer, or it can be added to an evolutionary algorithm, such that, we propose both schemes, a stand-alone optimizer: The Surrogate Assisted Optimizer (SAO), and an evolutionary algorithm: The NSGA-II+SIVO. The first algorithm proposed has the property to have a rapid convergence, but it also can be trapped in a false front when optimizing multifrontal problems. For this reason, we developed the second algorithm, this algorithm maintain the speed of convergence of SAO, but also incorporates the exploratory capability of NSGA-II. After comparing this algorithm with respect to the NSGA-II, we found that the proposed algorithm could outperform both, the original NSGA-II and SIVO stand alone algorithm. Further investigation and experimentation on the hybridization of the SIVO with other evolutionary algorithms, and an extension of SIVO for three objective problems are needed. Also, we are planning to work in the automated calculation of δ.

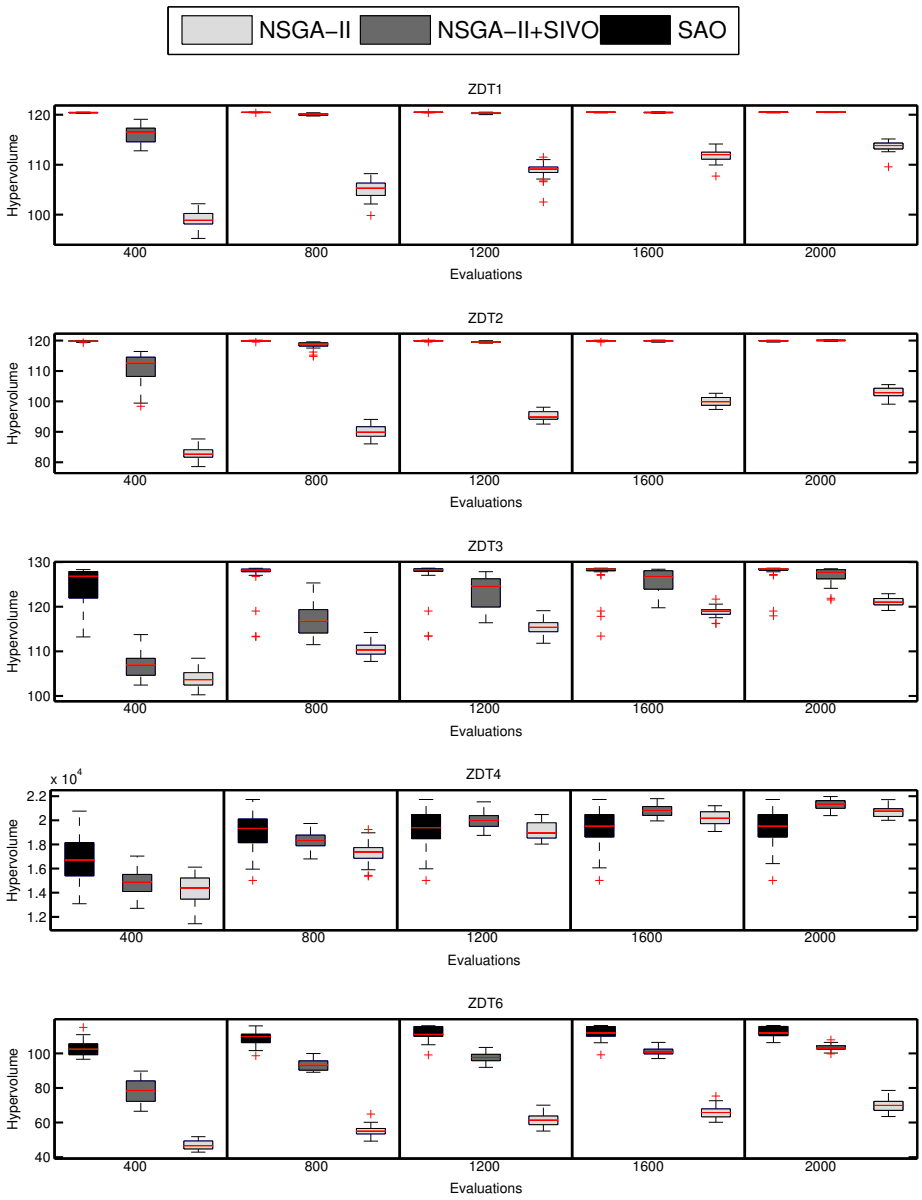


Fig. 3. Hypervolume of SAO, NSGA-II and NSGA-II+SIVO in the five test problems.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

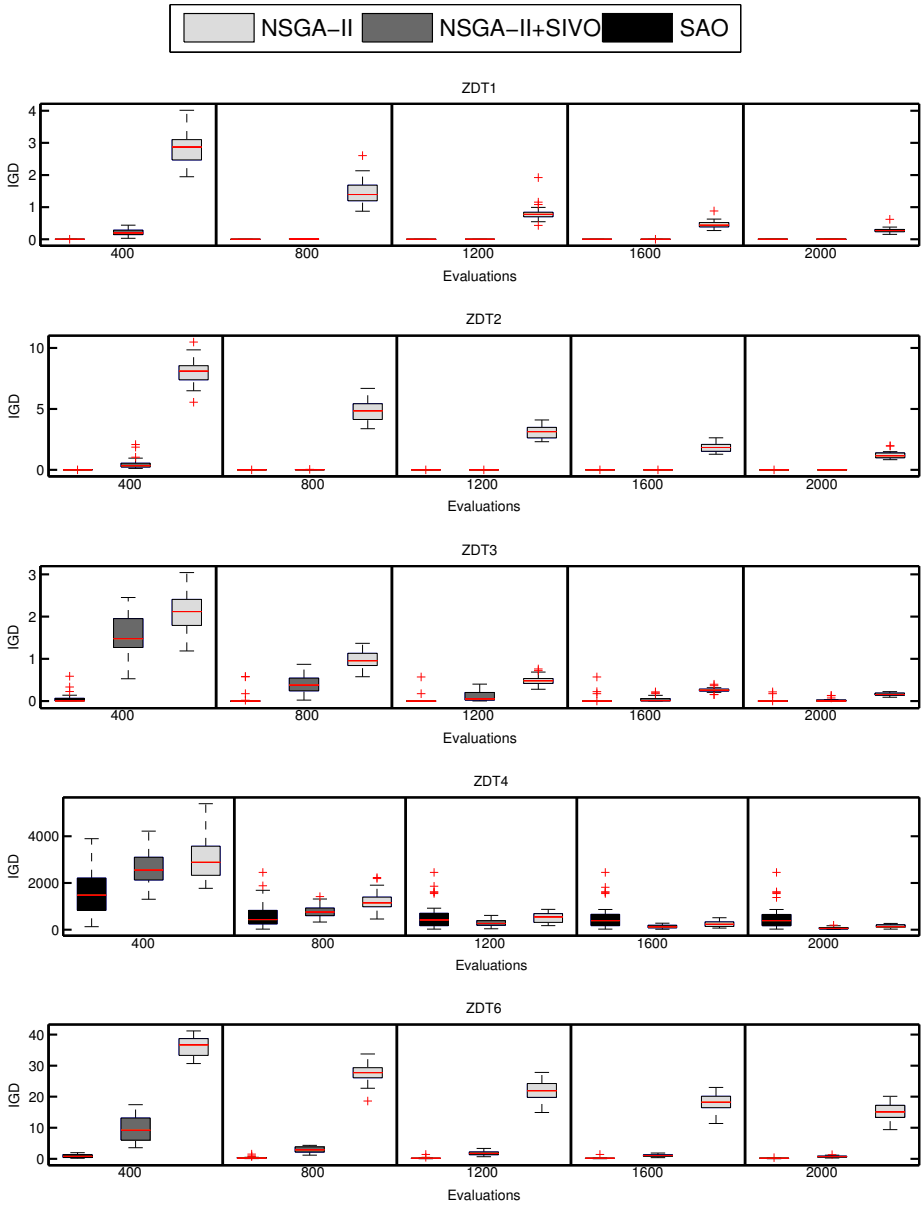


Fig. 4. Inverted Generational Distance of SAO, NSGA-II and NSGA-II+SIVO in the five test problems.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Acknowledgments

The first author acknowledges support from CONACyT through a scholarship to pursue graduate studies at the Information Technology Laboratory, CINVESTAV-Tamaulipas. The second author gratefully acknowledges support from CONACyT through project 105060.

References

1. K.S. Anderson and Y. Hsu. Genetic Crossover Strategy Using an Approximation Concept. In *Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99.*, volume 1, page 533, Washington D.C., USA, 1999. P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala,Eds.
2. M. Emmerich, K. Giannakoglou, and B. Naujoks. Single and Multiobjective Evolutionary Optimization Assisted by Gaussian Random Field Metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421 –439, aug. 2006.
3. S. Jun and K. Shigenobu. Extrapolation-Directed Crossover for Real-coded GA: Overcoming Deceptive Phenomena by Extrapolative Search. In *Proceedings of the 2001 Congress on Evolutionary Computation (CEC 2001)*, volume 1, pages 655 –662 vol. 1, Seoul, Korea, 2001. IEEE Press.
4. M. D. McKay, R. J. Beckman, and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239–245, 1979.
5. L. V. Santana-Quintero, V. A. Serrano-Hernandez, C. A. Coello, A. G. Hernández-Díaz, and J. Molina. Use of Radial Basis Functions and Rough Sets for Evolutionary Multi-Objective Optimization. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Multicriteria Decision Making (MCDM'2007)*, pages 107–114, Honolulu, Hawaii, USA, April 2007. IEEE Press.
6. A.K.M.K.A. Talukder, M. Kirley, and R. Buyya. The Pareto-Following Variation Operator as an Alternative Approximation Model. In *IEEE Congress on Evolutionary Computation, 2009. CEC '09.* , pages 8 –15, may 2009.
7. D. A. V. Veldhuizen and G. B. Lamont. Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1998.
8. D. A. V. Veldhuizen and G. B. Lamont. On Measuring Multiobjective Evolutionary Algorithm Performance. In *2000 Congress on Evolutionary Computation*, volume 1, pages 204–211, Piscataway, New Jersey, July 2000. IEEE Service Center.
9. Q. Li X. Liao, X. Yang, W. Zhang, and W. Li. Multiobjective optimization for crash safety design of vehicles using stepwise regression model. *Structural and Multidisciplinary Optimization*, 35:561–569, 2008.
10. Q. Zhang, J. Sun, and E. Tsang. An Evolutionary Algorithm with Guided Mutation for the Maximum Clique Problem. *IEEE Transactions on Evolutionary Computation*, 9(2):192 – 200, april 2005.
11. Q. Zhou and Y. Li. Directed Variation in Evolution Strategies. *IEEE Transactions on Evolutionary Computation*, 7(4):356 – 366, aug. 2003.
12. E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.

The Relationship Between the Covered Fraction, Completeness and Hypervolume Indicators

Viviane Grunert da Fonseca[1,3] and Carlos M. Fonseca[2,3]

[1] INUAF – Instituto Superior D. Afonso III, Loulé, Portugal
viviane.grunert@sapo.pt
[2] CISUC, Department of Informatics Engineering
University of Coimbra, Coimbra, Portugal
cmfonsec@dei.uc.pt
[3] CEG-IST – Center for Management Studies
Instituto Superior Técnico, Lisbon, Portugal

Abstract. This paper investigates the relationship between the covered fraction, completeness, and (weighted) hypervolume indicators for assessing the quality of the Pareto-front approximations produced by multiobjective optimizers. It is shown that these unary quality indicators are all, by definition, weighted Hausdorff measures of the intersection of the region attained by such an optimizer outcome in objective space with some reference set. Moreover, when the optimizer is stochastic, the indicators considered lead to real-valued random variables following particular probability distributions. Expressions for the expected value of these distributions are derived, and shown to be directly related to the first-order attainment function.

Keywords: stochastic multiobjective optimizer, performance assessment, covered fraction indicator, completeness indicator, (weighted) hypervolume indicator, attainment function, expected value, Hausdorff measure.

1 Introduction

The performance assessment of stochastic multiobjective optimizers (MOs) has become an emerging area of research, enabling the comparison of existing optimizers and supporting the development of new ones. In general, stochastic MO performance can be associated with the distributional behavior of the random outcomes produced in one optimization run, seen either as random nondominated point (RNP) sets in objective space or, alternatively, as the corresponding random unbounded *attained* sets [7].

Typically, realizations of such random closed sets in $\mathbb{R}^d$ can be observed arbitrarily often through *multiple* optimization runs. This allows stochastic MO performance assessment and comparison to be carried out with frequency-based statistical inference methodology using (simple) random samples of independent and identically distributed MO outcome sets.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

However, employing the frequency argument for MO outcome sets is not an easy task. Often, the overall outcome-set distribution is very complex – too complex to be considered as a whole in most practical situations. Therefore, in order to formulate suitable estimators and/or hypothesis tests for MO performance assessment and comparison, it needs to be agreed upon what *partial* aspect of this set distribution one is interested in.

Currently, two main stochastic MO performance assessment approaches are in use, but without really exploiting the relationship between them: the *attainment function approach* [8, 5, 7] and the *quality indicator approach* [17, 15].

The first approach describes the optimizer outcome distribution directly via a hierarchy of nested, increasingly informative attainment functions, where all functions beyond a certain order lead to a full performance description. It is known that the first-order attainment function relates to the *location* and *spread* of an outcome-set distribution, while second and higher-order versions address the corresponding *inter-point dependence structures* [7]. However, to the authors' knowledge, third- and higher-order attainment functions have not yet been considered in practice, due to computational difficulties [5, 6].

The quality indicator approach addresses the complexity of the outcome-set distribution in a different way, through the definition of so-called (unary) quality indicators which somehow transform each realized outcome set into a scalar. The corresponding, much simpler, univariate distributions of indicator values are usually studied via an estimate of their expected value (average indicator value), although other statistics could be considered as well. Even though this approach clearly implies a loss of information about the overall MO performance, there is hardly any discussion in the literature about what aspect of the outcome-set distribution is, or is not, addressed by each indicator.

On the whole, it is generally difficult to combine or compare results of different MO performance studies, unless exactly the same assessment methodology is used. Little is known about which indicators can complement each other and which, when used together, supply redundant information. Hence, there is a need to classify quality indicators with respect to the information they provide. Acknowledging the fact that unary quality indicators are transformations of the original optimizer outcome set, it seems natural to explore this link and attempt to relate the resulting indicator-value distributions with the attainment function hierarchy. As a first step in this direction, the present paper discusses the covered fraction, the completeness and the (weighted) hypervolume indicators with respect to their definition and to the mean (or expected value) of the corresponding distributions.

In Section 2, the attainment function and the quality indicator approaches are briefly outlined. Subsequently, the covered fraction indicator, the completeness indicator and the (weighted) hypervolume indicator are considered in Sections 3 to 5, respectively. The paper ends with a discussion of the results in Section 6 and some concluding remarks in Section 7.

2 Approaches to MO Performance Assessment

In the context of performance assessment, the outcome set of a multiobjective optimizer is considered to be the image in objective space of the non-dominated solutions produced in one optimization run (obeying some stopping criterion). When the optimizer is stochastic, such a set of objective vectors is random, and its probability distribution reflects the performance of the optimizer on a given optimization problem instance.

Mathematically, the outcome set of a d-objective stochastic optimizer can be interpreted as a random closed set [11]. More specifically, it is a random non-dominated point set (RNP set) [7]

$$\mathcal{X} = \{X_1, \ldots, X_M \in \mathbb{R}^d : \; P(X_i \leq X_j) = 0, \; i \neq j\} \; , \tag{1}$$

where both the cardinality M and the elements X_i, $i = 1, \ldots, M$, are random, and $P(0 \leq M < \infty) = 1$. In other words, $\mathcal{X}$ has a finite, but random, number of elements which do not weakly dominate one another in the Pareto sense [17]. Therefore, stochastic MO performance is related to

1. the (identical) multivariate distribution of the random vectors $X_1, \ldots, X_M$,
2. the way in which these random vectors depend on each other (in pairs, triples, quadruples, etc.) and, finally,
3. the univariate distribution of the discrete random variable M.

For simplicity, it has been common practice to condition on M with realizations up to a certain value m^* and/or to study only *partial* aspects of this set distribution. The two main approaches in this context are outlined below.

2.1 Attainment Function Approach

Let m^* be the maximum number of non-dominated objective vectors in $\mathbb{R}^d$ that may be generated by a d-objective optimizer. Then, increasing amounts of distributional information about the corresponding outcome set $\mathcal{X}$ with growing $k = 1, \ldots, m^*$ are comprised in the attainment functions $\alpha_{\mathcal{X}}^{(k)} : \mathbb{R}^{d \times k} \longrightarrow [0, 1]$, where

$$\alpha_{\mathcal{X}}^{(k)}(z_1, \ldots, z_k) = P\left(\mathcal{X} \trianglelefteq z_1 \; \wedge \; \ldots \; \wedge \; \mathcal{X} \trianglelefteq z_k\right) \; , \tag{2}$$

and the event

$$\left[\mathcal{X} \trianglelefteq z\right] \quad \Longleftrightarrow \quad \left[X_1 \leq z \; \vee \; X_2 \leq z \; \vee \ldots \vee \; X_M \leq z\right] \tag{3}$$

denotes the attainment of a goal $z \in \mathbb{R}^d$ by $\mathcal{X}$, assuming minimization without loss of generality. Thus, (2) gives the probability that the outcome set $\mathcal{X}$ weakly dominates the set of goals $\{z_1, \ldots, z_k\}$. The random unbounded but closed set $\mathcal{Y} = \{z \in \mathbb{R}^d : \; \mathcal{X} \trianglelefteq z\}$, which contains all goals $z \in \mathbb{R}^d$ dominated by at least one element of $\mathcal{X}$, is known as the *attained set* [7].

A *complete* distributional performance description (given that $M \leq m^*$) is provided by the attainment function of order $k = m^*$, while the first-order attainment function $\alpha_{\mathcal{X}}^{(1)}(\cdot) = \alpha_{\mathcal{X}}(\cdot)$ is sufficient to characterize where in objective

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

space goals tend to be attained (*location*) and with what degree of variability this happens across multiple runs (*spread*).

The above (theoretical) attainment functions can be estimated from a random sample of independent and identically distributed RNP sets $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$ via the cumulative frequencies of the corresponding (combinations of) events of the type $\left[\mathcal{X} \trianglelefteq z\right]$. For $k = 1$, for example, the non-parametric estimator is the *empirical first-order attainment function*

$$\alpha_n(z) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{I}\{z \in \mathcal{Y}_i\} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{I}\{\mathcal{X}_i \trianglelefteq z\} \,, \tag{4}$$

where $\mathbf{I}_A(\cdot) = \mathbf{I}\{\cdot \in A\}$ denotes the indicator function of the set A defined over $\mathbb{R}^d$. Estimators for higher-order attainment functions can be constructed in a similar way [7].

2.2 Quality Indicator Approach

Here, the outcome RNP set $\mathcal{X}$ is transformed into a single real-valued random variable $I(\mathcal{X})$, usually with respect to some non-empty, deterministic, closed reference set $Z_{ref} \subset \mathbb{R}^d$. A variety of such set-transformations are currently in use, each of which defining a particular unary quality indicator that reflects stochastic MO performance, in a restricted sense, via the associated *univariate* probability distribution of indicator values.

Like the distribution of $\mathcal{X}$, these indicator (value) distributions are unknown for a given MO application. Arising from the transformation applied to $\mathcal{X}$, they depend on the distribution of $\mathcal{X}$ itself, which can be described through the attainment function hierarchy. Hence, determining the form of this dependence should reveal what aspect(s) of the optimizer outcome-set distribution each indicator actually addresses.

Indicator distributions can, in principle, be estimated as a whole using the information of n independent optimization runs, in a non-parametric way. For simplicity, however, it is common to begin by estimating their expected values $\mathrm{E}[I(\mathcal{X})]$ through the sample average $\frac{1}{n}\sum_{i=1}^{n} I(\mathcal{X}_i)$. Similarly, this work focuses on the expected value of the distributions produced by the quality indicators considered.

3 Covered Fraction Indicator

The original version of the (unary) covered fraction indicator, or coverage indicator, considers the fraction of the Pareto-optimal front $\mathcal{X}^*$ (assumed to be a finite point set in $\mathbb{R}^d$) that is attained by the outcome set $\mathcal{X}$ [13, 15]. In this work, a more general definition will be used which, instead of referring to $\mathcal{X}^*$, may refer to any deterministic, non-empty, compact (i.e., closed and bounded) reference set Z_{ref} in objective space $\mathbb{R}^d$.

3.1 Definition

Given a general subset $A \subset \mathbb{R}^d$, denote its (normalized) *Hausdorff measure* of dimension j by $\mathcal{H}^j(A)$, where $j \geq 0$. Provided that A is non-empty, there is a single value of j, known as the *Hausdorff dimension* of A, for which $\mathcal{H}^j(A)$ is finite and positive [1, 12, p. 105]. Then, $\mathcal{H}^j(A)$ may be understood as the "size" of A. For example:

- $\mathcal{H}^0(\cdot)$, also known as the *counting measure*, measures the cardinality of a point set in $\mathbb{R}^d$ (where any such set has Hausdorff dimension zero),
- $\mathcal{H}^1(\cdot)$ measures the length of a Hausdorff one-dimensional smooth curve in $\mathbb{R}^d$ (for example a straight line, a circle or an ellipse),
- $\mathcal{H}^2(\cdot)$ measures the area of a Hausdorff two-dimensional smooth surface[4] in $\mathbb{R}^d$ (for example a plane or the surface of a sphere), and finally
- $\mathcal{H}^d(\cdot)$ measures the hypervolume of a Hausdorff d-dimensional set in $\mathbb{R}^d$, and, as such, corresponds to the usual Lebesgue measure on $\mathbb{R}^d$.

Thus, taking a Hausdorff j-dimensional reference set $Z_{ref} \subset \mathbb{R}^d$, $0 \leq j \leq d$, the covered fraction indicator of $\mathcal{X}$ can be generally defined as[5]

$$I_{CF}(\mathcal{X}, Z_{ref}) = \frac{\mathcal{H}^j(\{z \in Z_{ref} : \ \mathcal{X} \trianglelefteq z\})}{\mathcal{H}^j(Z_{ref})} \tag{5}$$

$$= \frac{1}{\displaystyle\int_{\mathbb{R}^d} \mathbf{I}\{z \in Z_{ref}\} \ \mathcal{H}^j(dz)} \cdot \int_{Z_{ref}} \mathbf{I}\{\mathcal{X} \trianglelefteq z\} \ \mathcal{H}^j(dz) \ . \tag{6}$$

For Hausdorff d- and zero-dimensional reference sets in objective space $\mathbb{R}^d$, the above definition (with integrals with respect to the measure $\mathcal{H}^j$) can be written in more familiar ways, as follows:

1. For a reference set $Z_{ref} \subset \mathbb{R}^d$ of Hausdorff dimension d, the covered fraction indicator of $\mathcal{X}$ can be defined as a quotient of Lebesgue integrals:

$$I_{CF}(\mathcal{X}, Z_{ref}) = \frac{1}{\displaystyle\int_{\mathbb{R}^d} \mathbf{I}\{z \in Z_{ref}\} \ dz} \cdot \int_{Z_{ref}} \mathbf{I}\{\mathcal{X} \trianglelefteq z\} \ dz \ . \tag{7}$$

[4] A j-dimensional smooth surface is the image of a continuously differentiable mapping (i.e. of class C^1) from $\mathbb{R}^j$ onto $\mathbb{R}^d$, where $j < d$ [9, p. 355]. Note that a countable union of Hausdorff j-dimensional sets preserves the Hausdorff dimension [4, p. 112], i.e. any such union of smooth sets is also of integer Hausdorff dimension. A non-mathematical discussion of the concept of "smoothness" can be found in [3, p. 335].

[5] The Hausdorff dimension of any realization of the outcome set $\mathcal{X}$ is equal to zero, and the Hausdorff dimension of a non-empty *attained set* realization is equal to d. Non-smooth reference sets, such as fractal sets, may have a non-integer Hausdorff dimension.

2. For a discrete reference set $Z_{ref}^k = \{z_1, \ldots, z_k\}$ of k not necessarily non-dominated points in $\mathbb{R}^d$, the definition simplifies to

$$I_{CF}(\mathcal{X}, Z_{ref}^k) = \frac{1}{k} \cdot \sum_{j=1}^{k} \mathbf{I}\{\mathcal{X} \trianglelefteq z_j\} \; . \tag{8}$$

In any case, the covered fraction indicator of $\mathcal{X}$ takes realizations in $[0, 1]$, where a larger observed indicator value is considered to represent a "better" optimization result (in the particular sense of the indicator).

3.2 Expected Value

Due to the "linearity property of expectation", and since the binary random variable $\mathbf{I}\{\mathcal{X} \trianglelefteq z\}$ follows a Bernoulli distribution with expected value $P(\mathcal{X} \trianglelefteq z) = \alpha_{\mathcal{X}}(z)$, for all $z \in \mathbb{R}^d$, it can be seen that the expected value of the covered fraction indicator distribution is related to the first-order attainment function. For a Hausdorff j-dimensional reference set Z_{ref}, $0 \leq j \leq d$, it holds that

$$\mathrm{E}[I_{CF}(\mathcal{X}, Z_{ref})] = \frac{\mathrm{E}[\mathcal{H}^j(\{z \in Z_{ref} : \; \mathcal{X} \trianglelefteq z\})]}{\mathcal{H}^j(Z_{ref})} \tag{9}$$

$$= \frac{1}{\displaystyle\int_{\mathbb{R}^d} \mathbf{I}\{z \in Z_{ref}\} \; \mathcal{H}^j(dz)} \cdot \int_{Z_{ref}} \alpha_{\mathcal{X}}(z) \; \mathcal{H}^j(dz) \; , \tag{10}$$

where (10) can be obtained by applying Robbins's theorem [11, p. 59] to the random compact set $\{z \in Z_{ref} : \; \mathcal{X} \trianglelefteq z\}$, since $\mathcal{H}^j$ is *locally finite* on a Hausdorff j-dimensional Z_{ref}.

For Hausdorff d- and zero-dimensional reference sets, the above formula can again be simplified:

1. For a reference set $Z_{ref} \subset \mathbb{R}^d$ of Hausdorff dimension d,

$$\mathrm{E}[I_{CF}(\mathcal{X}, Z_{ref})] = \frac{1}{\displaystyle\int_{\mathbb{R}^d} \mathbf{I}\{z \in Z_{ref}\} \; dz} \cdot \int_{Z_{ref}} \alpha_{\mathcal{X}}(z) \; dz \; . \tag{11}$$

2. For a discrete reference set $Z_{ref}^k = \{z_1, \ldots, z_k\}$ of Hausdorff dimension zero,

$$\mathrm{E}[I_{CF}(\mathcal{X}, Z_{ref}^k)] = \frac{1}{k} \cdot \sum_{j=1}^{k} \alpha_{\mathcal{X}}(z_j) \; . \tag{12}$$

4 Completeness Indicator

In its original form, the completeness indicator is defined for a given (observed) solution set in decision space (after one optimization run), as the probability of selecting a point uniformly at random from the feasible set which is weakly dominated by that solution set [10, 15].

4.1 Definition

For a random outcome set $\mathcal{X}$ in objective space, the original completeness indicator can be defined as the conditional probability of $\mathcal{X}$ attaining a non-uniformly distributed random reference point V from the image of the feasible set, given $\mathcal{X}$. However, the indicator may also be defined more generally by considering some deterministic, non-empty, closed reference set $Z_{ref} \subset \mathbb{R}^d$ (which may or may not be the image of the feasible set), together with a random vector V taking realizations in Z_{ref} and following an explicitly defined distribution. Hence,

$$I_{CO}(\mathcal{X}, V, Z_{ref}) = P\left(\mathcal{X} \trianglelefteq V \mid \mathcal{X}\right) \ . \tag{13}$$

Like the covered fraction indicator of $\mathcal{X}$, the completeness indicator of $\mathcal{X}$ is a random variable which takes realizations in $[0, 1]$, where larger values correspond to "better" optimization results.

In fact, it can be seen that the definitions of the two indicators are very similar, when interpreting $I_{CO}(\mathcal{X}, V, Z_{ref})$ as the conditional expectation of $\mathbf{I}\{\mathcal{X} \trianglelefteq V\}$ given $\mathcal{X}$: with a Hausdorff j-dimensional reference set $Z_{ref} \subset \mathbb{R}^d$, $0 \leq j \leq d$, and a random vector V supported on Z_{ref}, stochastically *independent* from the outcome set $\mathcal{X}$, and for which the probability density function $f_V(\cdot)$ exists with respect to Hausdorff measure $\mathcal{H}^j$, it holds that

$$\begin{aligned} I_{CO}(\mathcal{X}, V, Z_{ref}) &= P\left(\mathcal{X} \trianglelefteq V \mid \mathcal{X}\right) && (14)\\ &= \mathrm{E}\left[\mathbf{I}\{\mathcal{X} \trianglelefteq V\} \mid \mathcal{X}\right] && (15)\\ &= \int_{Z_{ref}} \mathbf{I}\{\mathcal{X} \trianglelefteq z\} \cdot f_{V|\mathcal{X}}(z)\ \mathcal{H}^j(dz) && (16)\\ &= \int_{Z_{ref}} \mathbf{I}\{\mathcal{X} \trianglelefteq z\} \cdot f_V(z)\ \mathcal{H}^j(dz)\ . && (17) \end{aligned}$$

Clearly, the particular choice of a *uniform* distribution for $V = V_u$ over some compact reference set Z_{ref}, again stochastically independent from $\mathcal{X}$, leads to the identity of the covered fraction indicator and the completeness indicator. In this case,

$$f_{V_u}(z) = f_u(z) = 1/\mathcal{H}^j(Z_{ref}) \tag{18}$$

for all $z \in Z_{ref}$, and $f_u(z) = 0$ otherwise.

Finally, note that for both Hausdorff d-dimensional reference sets and zero-dimensional reference sets, the formulation in (17) can be simplified, respectively in the sense of (7) and (8), by substituting the Hausdorff measure $\mathcal{H}^j$ either by the usual Lebesgue measure on $\mathbb{R}^d$ or by the counting measure for point sets.

4.2 Expected Value

From (17) and the argumentation given in subsection 3.2, it immediately follows that

$$\mathrm{E}[I_{CO}(\mathcal{X}, V, Z_{ref})] = \int_{Z_{ref}} \alpha_{\mathcal{X}}(z) \cdot f_V(z)\ \mathcal{H}^j(dz)\ , \tag{19}$$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

while more familiar formulations can be achieved for Hausdorff d- and zero-dimensional reference sets:

1. For a reference set $Z_{ref} \subset \mathbb{R}^d$ of Hausdorff dimension d,

$$\mathrm{E}[I_{CO}(\mathcal{X}, V, Z_{ref})] = \int_{Z_{ref}} \alpha_{\mathcal{X}}(z) \cdot f_V(z)\ dz\ . \tag{20}$$

2. For a discrete reference set $Z^k_{ref} = \{z_1, \ldots, z_k\}$ of Hausdorff dimension zero,

$$\mathrm{E}[I_{CO}(\mathcal{X}, V, Z^k_{ref})] = \sum_{j=1}^{k} \alpha_{\mathcal{X}}(z_j) \cdot P(V = z_j)\ . \tag{21}$$

5 Hypervolume Indicator

The hypervolume indicator (also known as dominated space), introduced in [16], considers the size of the portion of a deterministic, non-empty, closed reference set in objective space $\mathbb{R}^d$ that is attained by the outcome set $\mathcal{X}$. Later, this idea was refined by including a weight function that assigns varying levels of importance to different regions of the reference set [14].

In contrast to the covered fraction and completeness indicators, the reference set for the (weighted) hypervolume indicator is usually specified through a single reference point $z^*_{ref} \in \mathbb{R}^d$, and can be written as

$$Z^*_{ref} = \{z \in \mathbb{R}^d :\ z \leq z^*_{ref}\}\ , \tag{22}$$

though in some cases the Pareto-optimal front $\mathcal{X}^*$ or some other set of non-dominated points below z^*_{ref} is used to bound Z^*_{ref} from below, in such a way that it is still Hausdorff d-dimensional.

5.1 Definition

For a reference set Z^*_{ref} as given in (22), the (weighted) hypervolume indicator is defined as

$$I_H(\mathcal{X}, w, Z^*_{ref}) = \int_{Z^*_{ref}} \mathbf{I}\{\mathcal{X} \trianglelefteq z\} \cdot w(z)\ dz\ , \tag{23}$$

where $w(\cdot)$ is a non-negative valued weight function, integrable over Z^*_{ref}, i.e.

$$\int_{Z^*_{ref}} w(z)\ dz < \infty\ . \tag{24}$$

In this general form, the (weighted) hypervolume indicator of $\mathcal{X}$takes realizations in the interval

$$\left[0,\ \int_{Z^*_{ref}} w(z)\ dz\right]\ , \tag{25}$$

where larger observed indicator values are associated with "better" optimization results.

Some choices for the weight function $w(\cdot)$ deserve special attention:

- The indicator weight function, $w(z) = \mathbf{I}_{Z^*_{ref}}(z), z \in \mathbb{R}^d$, defines the classical, non-weighted hypervolume indicator with possible values ranging from 0 to the size of Z^*_{ref}, i.e., $\mathcal{H}^d(Z^*_{ref})$, if the reference set is compact.
- When $w(\cdot)$ is the *probability density* function [2] of a random variable V,

$$\int_{Z^*_{ref}} w(z)\ dz = 1\ , \tag{26}$$

  and all possible indicator values are contained in the interval $[0, 1]$. Further, if V is independent from $\mathcal{X}$, the (weighted) hypervolume indicator corresponds to the completeness indicator with respect to the reference set Z^*_{ref}.
- For the probability density function $w(\cdot) = f_u(\cdot)$ of a random variable V_u, *uniformly* distributed over some compact reference set Z^*_{ref} and independent from $\mathcal{X}$, all three indicators considered in this paper are identical, i.e.

$$I_{CF}(\mathcal{X}, Z^*_{ref}) = I_{CO}(\mathcal{X}, V_u, Z^*_{ref}) = I_H(\mathcal{X}, f_u, Z^*_{ref})\ . \tag{27}$$

5.2 Expected value

Again, from the argumentation given in subsection 3.2, it follows that

$$\mathrm{E}[I_H(\mathcal{X}, w, Z^*_{ref})] = \int_{Z^*_{ref}} \alpha_{\mathcal{X}}(z) \cdot w(z)\ dz\ . \tag{28}$$

6 Discussion

The covered fraction, completeness and weighted hypervolume indicators are all, by definition, weighted Hausdorff measures of the intersection of the attained set $\mathcal{Y} = \{z \in \mathbb{R}^d :\ \mathcal{X} \trianglelefteq z\}$ with some reference set Z_{ref} in objective space $\mathbb{R}^d$. As such, they can be seen as special cases of a *generalized hypervolume indicator* of $\mathcal{X}$, defined as

$$I_{GH}(\mathcal{X}, w, Z_{ref}) = \int_{Z_{ref}} \mathbf{I}\{\mathcal{X} \trianglelefteq z\} \cdot w(z)\ \mathcal{H}^j(dz)\ , \tag{29}$$

where

- Z_{ref} is a Hausdorff j-dimensional, deterministic, closed reference set in $\mathbb{R}^d$, $0 \leq j \leq d$, representing the region in objective space which is of interest for performance assessment, and
- $w(\cdot)$ is a non-negative valued weight function, integrable over Z_{ref}, which assigns different levels of importance to the points in Z_{ref}.

Note that, technically, the weight function $w(\cdot)$ alone would be sufficient to parametrize this indicator, as its support would define the corresponding reference set Z_{ref}. In other words, the generalized hypervolume indicator measures the *mass* of the region attained by an optimizer outcome set, as determined by

Table 1. Indicators as special cases of the *generalized hypervolume indicator*.

Indicator	Reference set	Weight function
weighted hypervolume	Z^*_{ref}, closed	integrable over Z^*_{ref}
non-weighted hypervolume	Z^*_{ref}, compact	indicator function $\mathbf{I}_{Z^*_{ref}}(\cdot)$
completeness	Z_{ref}, closed	density function f_V with support on Z_{ref}, V independent of $\mathcal{X}$
covered fraction	Z_{ref}, compact	uniform density function with support on Z_{ref}

some *mass density* function $w(\cdot)$, which may or may not be also a probability density function. The generalized hypervolume indicator can be reduced to each of the indicators considered in this paper, as summarized in Table 1.

As a function of the outcome set $\mathcal{X}$, the indicator $I_{GH}(\mathcal{X}, w, Z_{ref})$ leads to a random variable with realizations in the interval

$$\big[0,\ \int_{Z_{ref}} w(z)\ \mathcal{H}^j(dz)\big]\ , \tag{30}$$

where larger indicator values indicate a "better" optimization result. At this point it should be noted, however, that for the purpose of "unit-independent" comparisons of optimizer performance, it may be preferable to always use a normalizing density weight function, so that indicator values can be limited to the interval $[0, 1]$.

For a general weight function $w(\cdot)$, the expected value of the generalized hypervolume indicator distribution can be expressed as

$$\mathrm{E}[I_{GH}(\mathcal{X}, w, Z_{ref})] = \int_{Z_{ref}} \alpha_{\mathcal{X}}(z) \cdot w(z)\ \mathcal{H}^j(dz)\ . \tag{31}$$

For the *probability density* function of a random vector V, supported on Z_{ref} and distributed *independently* from the outcome set $\mathcal{X}$, this additionally leads to the interesting relationship:[6]

$$\mathrm{E}[I_{GH}(\mathcal{X}, f_V, Z_{ref})] = \mathrm{E}[\alpha_{\mathcal{X}}(V)] = P\left(\mathcal{X} \trianglelefteq V\right)\ . \tag{32}$$

Hence, the expected value of the generalized hypervolume indicator, and consequently of all three indicators considered in this paper, can be obtained from the first-order attainment function.

This result is important because it sheds light on the relationship between the quality indicator approach and the attainment function approach. It can be seen that the three indicators considered, unified in the generalized hypervolume indicator, convey information related to the *location* of the optimizer

[6] For independent $\mathcal{X}$ and V: $\mathrm{E}\big[P\left(\mathcal{X} \trianglelefteq V \mid V\right)\big] = \mathrm{E}\big[P\left(\mathcal{X} \trianglelefteq V \mid \mathcal{X}\right)\big] = P\left(\mathcal{X} \trianglelefteq V\right)$.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

outcome-set distribution via their mean indicator values. Moments of the generalized hypervolume indicator distribution other than the mean may still contain information beyond that captured by the first-order attainment function.

The generalized hypervolume indicator also allows new indicators to be constructed using less usual reference sets and/or weight functions. For example, assume that the outcome set $\mathcal{X}$ of a hypothetical two-objective optimizer, when applied to a given problem instance, is simply the set of minima [6] of a set of two stochastically independent random vectors in $\mathbb{R}^2$. Assume also that both vectors are distributed according to a bivariate exponential distribution with parameter $\lambda > 0$, with density function

$$f(t_1, t_2) = \lambda^2 \cdot e^{-\lambda \cdot t_1 - \lambda \cdot t_2} \cdot \mathbf{I}_{[0,\infty)^2}((t_1, t_2)') \tag{33}$$

and cumulative distribution function

$$F(t_1, t_2) = \left(1 - e^{-\lambda \cdot t_1} - e^{-\lambda \cdot t_2} + e^{-\lambda \cdot t_1 - \lambda \cdot t_2}\right) \cdot \mathbf{I}_{[0,\infty)^2}((t_1, t_2)') \; . \tag{34}$$

Then, the first-order attainment function of $\mathcal{X}$ at a goal $z = (t_1, t_2)' \in \mathbb{R}^2$ is

$$\alpha_{\mathcal{X}}(z) = 1 - \left(1 - F(t_1, t_2)\right)^2 \; . \tag{35}$$

Given $(r_1, r_2)' \in \mathbb{R}^2_+$, the reference set $Z_{seg} = \{(r_1 \cdot t, r_2 - r_2 \cdot t)' : \; t \in [0, 1]\}$ is a line segment in $\mathbb{R}^2$, and has Hausdorff dimension 1. In this case, the generalized hypervolume indicator with weight function $\mathbf{I}_{Z_{seg}}(\cdot)$ measures the *length* of the intersection of this segment with the attained set. From (31), the expected indicator value may then be calculated via the line integral

$$\mathrm{E}[I_{GH}(\mathcal{X}, \mathbf{I}_{Z_{seg}}, Z_{seg})] = \sqrt{r_1^2 + r_2^2} \cdot \int_0^1 \alpha_{\mathcal{X}}((r_1 \cdot t, r_2 - r_2 \cdot t)') \; dt \; . \tag{36}$$

Clearly, the expected value can also be directly evaluated using the standard integral based on the joint density of the two exponential random vectors, but the latter leads to much more complicated expressions.

7 Concluding Remarks

In this paper the relationship between three popular quality indicators for multiobjective optimizer performance assessment has been studied with respect to their definitions. By considering Hausdorff measures, a *single* notation was introduced for all indicators, leading to their unification. Furthermore, it was shown that, when the optimizer is stochastic, the corresponding expected indicator values depend on the first-order attainment function of the optimizer outcomes.

Investigating how the variance and other aspects of these univariate indicator-value distributions relate to the attainment function hierarchy will be the subject of future work, as well as considering other unary quality indicators. In particular, it would be interesting to see how indicators designed to assess the dispersion or uniformity of points in observed approximation sets relate in distribution to (higher-order) attainment functions.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

References

1. Alberti, G.: Geometric measure theory. In: Françoise, J.P., et al. (eds.) Encyclopedia of Mathematical Physics, vol. 2, pp. 520–527. Elsevier, Oxford (2006)
2. Bader, J.M.: Hypervolume-Based Search for Multiobjective Optimization: Theory and Methods. Ph.D. thesis, Swiss Federal Institute of Technology, Zurich (2009)
3. Barenblatt, G.I.: Scaling, Self-similarity, and Intermediate Asymptotics. Cambridge University Press, Cambridge (1996)
4. DiBenedetto, E.: Real Analysis. Birkhäuser, Boston (2002)
5. Fonseca, C.M., Grunert da Fonseca, V., Paquete, L.F.: Exploring the performance of stochastic multiobjective optimisers with the second-order attainment function. In: Coello Coello, C.A., et al. (eds.) EMO 2005, LNCS, vol. 3410, pp. 250–264. Springer, Heidelberg (2005)
6. Fonseca, C.M., Guerreiro, A.P., López-Ibáñez, M., Paquete, L.: On the computation of the empirical attainment function. In: Takahashi, R.H.C., et al. (eds.) EMO 2011, LNCS, vol. 6576, pp. 106–120. Springer, Heidelberg (2011)
7. Grunert da Fonseca, V., Fonseca, C.M.: The attainment-function approach to stochastic multiobjective optimizer assessment and comparison. In: Bartz-Beielstein, T., et al. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, chap. 5, pp. 103–130. Springer, Berlin (2010)
8. Grunert da Fonseca, V., Fonseca, C.M., Hall, A.O.: Inferential performance assessment of stochastic optimisers and the attainment function. In: Zitzler, E., et al. (eds.) EMO 2001, LNCS, vol. 1993, pp. 213–225. Springer, Heidelberg (2001)
9. Ito, K. (ed.): Encyclopedic Dictionary of Mathematics 2. The Mathematical Society of Japan. The MIT Press (1987)
10. Lotov, A.V., Bushenkov, V.A., Kamenev, G.K.: Interactive Decision Maps: Approximation and Visualization of Pareto Frontier. Kluwer Academic Publishers, Dordrecht (2004)
11. Molchanov, I.: Theory of Random Sets. Springer, London (2005)
12. Ott, E.: Chaos in Dynamical Systems. Cambridge University Press, Cambridge (2002)
13. Ulungu, E.L., Teghem, J., Fortemps, P.H., Tuyttens, D.: MOSA method: A tool for solving multiobjective combinatorial optimization problems. Journal of Multi-Criteria Decision Analysis 8(4), 221–236 (1999)
14. Zitzler, E., Brockhoff, D., Thiele, L.: The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration. In: Obayhashi, S., et al. (eds.) EMO 2007, LNCS, vol. 4403, pp. 862–876. Springer, Heidelberg (2007)
15. Zitzler, E., Knowles, J., Thiele, L.: Quality assessment of Pareto set approximations. In: Branke, J., et al. (eds.) Multiobjective Optimization. Interactive and Evolutionary Approaches, LNCS, vol. 5252, pp. 373–404. Springer, Heidelberg (2008)
16. Zitzler, E., Thiele, L.: Multiobjective optimization using evolutionary algorithms – A comparative case study. In: Eiben, A.E., et al. (eds.) PPSN V 1998. LNCS, vol. 1498, pp. 292–301. Springer, Heidelberg (1998)
17. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Grunert da Fonseca, V.: Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on Evolutionary Computation 7(2), 117–132 (2003)

A Rigorous Runtime Analysis for Quasi-Random Restarts and Decreasing Stepsize

Marc Schoenauer, Fabien Teytaud and Olivier Teytaud

TAO (Inria), LRI, UMR 8623(CNRS - Univ. Paris-Sud), bat 490 Univ. Paris-Sud
91405 Orsay, France

Abstract. Multi-Modal Optimization (MMO) is ubiquitous in engineering, machine learning and artificial intelligence applications. Many algorithms have been proposed for multimodal optimization, and many of them are based on restart strategies. However, only few works address the issue of initialization in restarts. Furthermore, very few comparisons have been done, between different MMO algorithms, and against simple baseline methods. This paper proposes an analysis of restart strategies, and provides a restart strategy for any local search algorithm for which theoretical guarantees are derived. This restart strategy is to decrease some 'step-size', rather than to increase the population size, and it uses quasi-random initialization, that leads to a rigorous proof of improvement with respect to random restarts or restarts with constant initial step-size. Furthermore, when this strategy encapsulates a (1+1)-ES with 1/5th adaptation rule, the resulting algorithm outperforms state of the art MMO algorithms while being computationally faster.

1 Introduction

The context of this work is continuous black-box Multi-Modal Optimization (MMO). Given an objective function F, the goal of MMO is to discover **all** global optima of F (and not just a few of them). Because of the "black-box" hypothesis, this paper focuses on gradient-free methods, more particularly Evolution Strategies (ES) [5], addressing both theoretical and experimental issues.

Note that there are to-date very few rigorous analyses of ES for MMO. But it has been experimentally shown that in the MMO setting, choosing a large population size λ reduces the risk being trapped in local minima [27, 9, 1]. Unfortunately, it has also been shown that some classical ES are not very efficient for large λ [4], and that self-adaptive ES are quite fast and reliable in that case, with a good speed-up as a function of λ [4].

Several MMO-specific methods have been proposed in the evolutionary framework, and their precise description cannot be included here for space considerations. Please refer to the survey proposed in [23] (or to the original papers of course). A popular family of MMO algorithms is based on niching techniques: sharing [8]; clearing [22], including the modified clearing approach proposed in [23]; crowding [6], including deterministic [18] and probabilistic [19] versions; clustering [30]; species conserving genetic algorithms [16]; and finally, different

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

restart strategies, to the recent state-of-the-art restart with increasing population size [1]. Several other works have been devoted to MMO outside the evolutionary community. For instance, EGO [12], IAGO [29] provide very efficient algorithms in terms of precision/number of fitness evaluations, but are computationally far too expensive unless the objective function is itself very costly (requiring hours or days of computation per point). Moreover, they have no theoretical guarantee in spite of their elegant derivation. UNLEO [3] proposed an approximation of optimal optimization algorithm under robustness constraints, with nice theoretical guarantees; however, it is, too, far too expensive, and could hence be tested with a few tenths of function evaluations.

This paper proposes a very simple restart strategy that can encapsulate any (local) optimization algorithm, and for which convergence rates can be theoretically derived. This strategy is based on quasi-random restarts, and a decreasing schedule for some 'step-size' parameter. Comparative experiments with the extensive results published in [23] are used to demonstrate the applicability and efficiency of the proposed strategy, in the case where the embedded local algorithm is a simple (1+1)-ES with 1/5th adaptation rule. However, the proposed restart strategy heavily relies on a specific parameter d (see Alg. 1), somehow analogous to a parameter used in the modified clearing which outperformed by far all other algorithms [23]. Hence, because this parameter d will be tuned for the problems used in the experiments, no fair comparison can be done with other published results for which no such parameter tuning was performed. Our conclusions will therefore be limited to the design of a generic restart algorithm based on quasi-random sequences with theoretical guarantees and tight complexity bounds (see Corollary 1), that outperforms uniform restarts, and reaches state of the art performance provided a good setting of parameter d, which essentially quantifies some prior knowledge about the minimal distance between two optima.

The remaining of the paper is organized in the following way: Section 2 introduces the notations used throughout the paper, and surveys the state-of-the-art in quasi-random (QR) sequences. Section 3 introduces the proposed restart strategy, and rigorously quantifies the improvement provided by QR points in the initialization of MMO and the requirement that the initial step-size after a restart goes to zero as the number of restarts increases. Section 4 provides comparative experimental results with state-of-the-art MMO methods as well as the simple random restart method with constant step-size.

2 Mathematical Background

Notations: For each E, subset of a topological space, $\overline{E}$ will denote the *topological closure* of E, i.e., the intersection of all closed sets containing E, and $\#\mathbb{E}$ denote the number of elements of E. For each sequence $S = s_1, s_2, s_3, \dots$ of points in a topological space, the *accumulation* of S is defined as $Acc\ S = \cap_{n\geq 1}\overline{\{s_{n+1}, s_{n+2}, \dots\}}$. For each sequence $X = x_1, x_2, \dots$ of points in $[0,1]^D$, the *dispersion* of X is defined as $Disp(X,n) = \sup_{x\in[0,1]^D} \inf_{i\in[[1,n]]} ||x - x_i||$.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Quasi-random (QR) points A quasi-random sequence is a (possibly randomized) sequence with some uniformity properties that make it, intuitively, "more uniform" than a pseudo-random sequence. After astonishing results in numerical integration (convergence in $1/n$ instead of $1/\sqrt{n}$ for numerical integration, within logarithmic factors) and successful experiments in random searchs [20], QR points have been used in several works dealing with evolution strategies, during initialization [13, 7] or mutation [2, 26, 25]. Furthermore, "modern" QR sequences using scrambling [14] have been demonstrated to outperform older QR sequences [26, 25]. Hence, following [28, 25], this paper will only consider Halton sequences with random scrambling, and this Section will only briefly introduce them – please refer to [20, 21] for more details and references.

Let us first define Van Der Corput's sequence: Given a prime number p, the n^{th} element $vdc_{n,p}$ of the Van Der Corput sequence in basis p is defined by:

- write n in basis p: $n = d_k d_{k-1} \dots d_1$, i.e. $n = \sum_{i=0}^{k} d_i p^i$ with $d_i \in [[0, p-1]]$;
- $vdc_{n,p} = 0.d_1 d_2 \dots d_k$ in basis p, i.e. $vdc_{n,p} = \sum_{i=1}^{k} d_i p^{-i} \in [0, 1]$.

A classical improvement in terms of discrepancy for moderate values of n and large values of d, termed *scrambling*, defines $vdc_{n,p}$ as $vdc_{n,p} = 0.\pi(d_1)\pi(d_2)\dots\pi(d_k)$ where π is some permutation of $[[0, p-1]]$ such that $\pi(0) = 0$ in order to ensure $\forall n, vdc_{n,p} \neq 0$.

Halton sequences generalize Van Der Corput sequences to dimension D by using one different prime number per dimension. Let $p_i, i \in [[1, D]]$ be D prime numbers. The n^{th} element h_n of a Halton sequence in dimension D is defined by $h_n = (vdc_{n,p_1}, vdc_{n,p_2}, \dots, vdc_{n,p_D}) \in [0, 1]^D$. Scrambled-Halton sequences, like the ones used in this paper, are scrambled using a randomly drawn permutation for each $i \in [[1, D]]$.

The N^{th} Hammersley point set is $\{\text{HAMM}_{N,1}, \text{HAMM}_{N,2}, \dots, \text{HAMM}_{N,N}\}$, where $\text{HAMM}_{N,n} = ((n-1)/N, vdc_{n,p_1}, vdc_{n,p_2}, \dots, vdc_{n,p_{D-1}})$.

3 Theoretical Analysis of a Simple Restart Strategy

Let $\mathcal{D}$ be the optimization domain, embedded in a normed vector space (with norm $||.||$). Let $\mathcal{F}$ be a family of fitness functions, i.e., of mappings from $\mathcal{D}$ to $\mathbb{R}$. For any fitness function $f \in \mathcal{F}$, let $X^*(f)$ be the set of interest[1]. Let ES be an optimization algorithm that takes as input $x \in \mathcal{D}$ and $\sigma > 0$ (initial point and step-size respectively), depends on f, and outputs some $x' = ES(x, \sigma, f) \in \mathcal{D}$ [2]. Think of σ as a radius at which local optima are searched; however, it is not necessary to assume that ES always finds an optimum x' at distance $< \sigma$ of x, but only that for σ sufficiently small and x sufficiently close to x^*, $x' = ES(x, \sigma, f) = x^*$. Finally, for any given sequence $S = (x_i, \sigma_i)_{i \in \mathbb{N}} \in (\mathcal{D} \times]0, \infty[)^{\mathbb{N}}$, denote $RS(S)$ the restart algorithm that successively starts from (x_1, σ_1), (x_2, σ_2), ...

[1] in most cases, $X^*(f)$ will be the set of local optima of f, though the results below have some generality w.r.t. $X^*(f)$.

[2] x' is the output of a whole run of ES: in general, it will be the best point of the run; however, here again the results are more general.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Definitions:
(i) ES has the **convergence property** if

$$(\forall f \in \mathcal{F})\ (\forall (x,\sigma) \in \mathcal{D}\times]0,\infty[)\ (ES(x,\sigma,f) \in X^*(f)) \tag{1}$$

(ii) ES has the **locality property** w.r.t. $\mathcal{F}$ if

$$\begin{array}{l}(\forall x^* \in X^*(f))\ (\exists \epsilon > 0)\ (\exists \sigma_0 > 0) \text{ s.t.} \\ (||x - x^*|| < \epsilon) \text{ and } (0 < \sigma < \sigma_0) \Rightarrow (ES(x,\sigma,f) = x^*).\end{array} \tag{2}$$

Note that this does not imply that $ES(x,\sigma,f^*)$ is necessarily within distance σ of x, which would be an unrealistic assumption.

(iii) ES has the **strong locality property** w.r.t. $\mathcal{F}$ if

$$\begin{array}{l}(\exists \epsilon > 0)\ (\exists \sigma_0 > 0) \text{ s.t. } (\forall f \in \mathcal{F})\ (\forall x^* \in X^*(f)) \\ (||x - x^*|| < \epsilon) \text{ and } (0 < \sigma < \sigma_0) \Rightarrow ES(x,\sigma,f) = x^*.\end{array} \tag{3}$$

(iv) For any sequence $S \in (\mathcal{D}\times]0,\infty[)^{\mathbb{N}}$, the restart algorithm $RS(S)$ is said to be **consistent w.r.t.** $\mathcal{F}$ if

$$(\forall f \in \mathcal{F})\ \{ES(x_i,\sigma_i,f); i \in \mathbb{N}\} = X^*(f).$$

We can now state the following consistency theorem.

Theorem 1 (Consistency of the restart algorithm). *Assume that ES has the convergence property (Eq. 1) and the locality property (Eq. 2). Then:*

1. *If $(X^*(f) \times \{0\} \subset Acc\ S)$ for all $f \in \mathcal{F}$, then $RS(S)$ is consistent w.r.t. $\mathcal{F}$;*
2. *If $\mathcal{D} \times \{0\} \subset Acc\ S$, then $RS(S)$ is consistent w.r.t. all $\mathcal{F} \subset \mathbb{R}^{\mathcal{D}}$.*

Proof: (2) is an immediate consequence of (1) so let us prove (1).
Let $f \in \mathcal{F}$. Eq. 1 immediately implies that $\{ES(x_i,\sigma_i,f); i \geq 1\} \subset X^*(f)$.
Let $x^* \in X^*(f)$; then $(x^*,0)$ is in $Acc\ S$ by assumption. Using Eq. 2, it follows that $x^* = ES(x_i,\sigma_i,f)$ for some $i \geq 1$; this proves that $X^*(f) \subset \{ES(x_i,\sigma_i,f); i \geq 1\}$; hence the expected result. □

Remark 1 The assumption $X^*(f) \times \{0\} \subset Acc\ S$ is necessary. It holds in particular with random restarts points and random initial step-sizes with non-zero density close to 0; or random restart points and step-sizes decreasing to 0; in both cases, quasi-random restarts can be used instead of random restarts.

Next result now considers the number of restarts required for finding all points in $X^*(f)$. Sequences of starting points are now considered stochastic, hence the expectation operator:

Proposition 1 (QR restarts are faster). *Assume that ES has the convergence property (Eq. 1) and the strong locality property (Eq. 3) for some ϵ, σ_0. Assume that $\sigma_i = \sigma_0$ for all $i \geq 1$. Define*
$\#RS(\mathcal{F}) = \sup_{f\in\mathcal{F}} \mathbb{E}[\inf\{n \in \mathbb{N}; X^*(f) = \{ES(x_1,\sigma_1,f),\ldots,ES(x_n,\sigma_n,f)\}\}]$.
Then, $\#RS(\mathcal{F}) \leq \mathbb{E}[\inf\{n \in \mathbb{N}; Disp((x_i)_{i\in\mathbb{N}}, n) \leq \epsilon\}]$.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Proof: By Eq. 1, $X^*(f) \subset \{ES(x_1, \sigma_1, f), \dots, ES(x_n, \sigma_n, f)\}$ for all $n \in \mathbb{N}$. On the other hand, if $Disp((x_i)_{i\in\mathbb{N}}, n) < \epsilon$, then Eq. 3 implies

$$\#RS(\mathcal{F}) \leq \sup_{f\in\mathcal{F}} \mathbb{E}[\inf\{n \in \mathbb{N}; X^*(f) \subset B(x_1, \epsilon) \cup B(x_2, \epsilon) \cup \dots \cup B(x_n, \epsilon)\}]$$

and this is at most n; hence the expected result. □

The bound is tight for some simple cases, e.g., optima distributed on a grid with sphere-like functions on Voronoi-cells built on the optima, and ES converging locally:

$$X^*(f) = \{(k_1\epsilon, k_2\epsilon, \dots, k_D\epsilon); (k_1, \dots, k_D) \in [[0, \lfloor 1\epsilon \rfloor]]^D\}. \quad (4)$$

The dispersion $Disp$ can therefore be used for quantifying the quality of a sequence of restart points. The optimal dispersion is reached by some grid-based sampling (e.g. Sukharev grids [17]) reaching $Disp(x, n) = O(n^{1/D})$. The **dispersion complexity**, i.e., the number n of points required for reaching dispersion ϵ is then[3]:

$O\left((1/\epsilon)^D \log(1/\epsilon)\right)$ for all low-discrepancy sequences; th. 6.6 p152, (C1)
$O\left((1/\epsilon)^D\right)$ for Halton or Hammersley (Section 2); th. 6.12+6.13 p157, (C2)
$\Omega\left((1/\epsilon)^D\right)$ for all sequences or point sets; th. 6.8 p154, (C3)
$\Omega\left((1/\epsilon)^D \log(1/\epsilon)\right)$ for random sequences (expected value; see Theorem 2) (C4)

The results above are based on the notion of discrepancy. In particular, low-discrepancy (also termed quasi-random) sequences have *extreme discrepancy* $O(\log(n)^D/n)$. However, only complexity result (C1) will be necessary here.

To the best of our knowledge, complexity (C4) for random sequences has not yet been published. It can nonetheless easily be derived by reduction to the classical Coupon collector theorem (proof omitted for space reasons):

Theorem 2 (Dispersion of random points). *Consider $x_1, \dots, x_n, \dots$ randomly independently uniformly distributed in $[0, 1]^D$ and $\epsilon > 0$. Then $\mathbb{E}[inf\{n \in \mathbb{N}; Disp(x, n) < \epsilon] = \theta\left(1/\epsilon^D \log(1/\epsilon)\right)$.*

From Proposition 1, and the complexity of dispersions above, it comes

Corollary 1 (Complexity in terms of number of restarts). *Let $(\mathcal{F}^{(k)})_{k\in\mathbb{N}}$ be a family of sets of fitness functions defined on $[0, 1]^D$ for some D. Suppose that for each $k \in \mathbb{N}$, $\mathcal{F}^{(k)}$ has strong local property (Eq. 3) with values $\epsilon^{(k)}, \sigma_0^{(k)}$. Then, for some $C > 0$, a quasi-random restart with Halton sequence[4] and $\sigma_i < \sigma_0$ ensures that*

$$(\forall k \in \mathbb{N}), (\forall f \in \mathcal{F}^{(k)}), (X^*(f) \subset \{ES(x_i, \sigma_i, f); i \in [[1, C/\epsilon^D]]\}). \quad (5)$$

This is not true for random restart, and C/ϵ^D cannot be replaced by $o(1/\epsilon^D)$.

[3] all references are to be found in [20]. See references therein for more details.
[4] or any sequence with dispersion complexity (C1)

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Algorithm 1 This generic algorithm includes Random restart with Decreasing Step-size (RDS) as well as Quasi-Random restart with Decreasing Step-size (QRDS) algorithm. Constant, linearly or quadratically decreasing versions can be implemented on line 5. The case with murder is the case $d > 0$ (line 14).

Require: $(x_1, x_2, \ldots)$ sequence of starting points, (d, σ^*) precision thresholds
1: $n = 0, optimaFound = \emptyset$
2: **while** Maximum number of evaluations not reached, and all optima not found **do**
3: $n \leftarrow n + 1$
4: $y = x_n$ // nth point the sequence of starting points
5: $\sigma = nextSigma(n, \sigma_0)$ // constant or decreasing σ
6: $State \leftarrow alive$
7: **while** $State = alive$ **do**
8: $y' = y + \sigma\mathcal{N}(0, I_d)$ // $\mathcal{N}(0, I_d)$ is an isotropic Gaussian random variable
9: **if** y' better than y **then**
10: $y = y'$; $\sigma = 2\sigma$
11: **else**
12: $y = y'$; $\sigma = 2^{-1/4}\sigma$
13: **for all** $(opt \in optimaFound)$ **do**
14: **if** $(||y - opt|| < d)$ or $(\sigma < \sigma^*)$ **then**
15: State$\leftarrow$dead
16: $optimaFound = optimaFound \cup \{y\}$

Proof: Eq. 5 is a consequence of (C1) and Proposition 1.

The fact that this is not true for random restart is the application of complexity (C4) to the particular case shown above (Eq. 4).

Finally, $\Omega(1/\epsilon^D)$ restarts are needed in order to find the $\Omega(1/\epsilon^D)$ optima in Eq. 4: it is hence not possible in the general case to replace C/ϵ^D by $o(1/\epsilon^D)$. □

4 Experimental results

This section presents comparative experimental results for some particular instances of restart algorithms to which the theoretical results of above Section 3 can be applied. The main ingredient of a restart algorithm is its embedded 'local' optimizer. The choice made here is that of an Evolution Strategy, for the robustness of this class of algorithm. However, only local convergence properties are required, far from any sophisticated variant designed for multimodal fitness functions for instance [27, 9, 1]. Furthermore, the testbed we want to compare to [23] does not require large values of λ, nor covariance matrix adaptation. Hence the simple (1+1)-ES with $\frac{1}{5}^{th}$-rule was chosen, as it gives very good results in very short time according to some preliminary runs. Finally, a (1+1)-ES satisfies the hypotheses of Theorem 1, and hence all results of Section 3 apply depending only on the properties of the sequence of starting points and initial step-sizes.

The precise instances of restart algorithm under scrutiny here are defined in Algorithm 1: this includes random and quasi-random sequences of starting points in the domain given as input (line 4), as well as different strategies for the step-size change from one run to the other, depending on line 5: function *nextSigma* can return a constant value σ_0, a linearly decreasing value $(\sigma_0/(n+1))$ or a quadratically decreasing value $(\sigma_0/(n+1)^2)$.

One run of the local algorithm can be killed when either the step-size goes beyond a given thresholds σ^* (defaulted to 10^{-6} unless otherwise stated), or

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

when the current solution gets too close to a previously discovered local optimum, up to a tolerance d (line 14). In the latter case, the algorithm is said *with murder*. But the choice of the threshold distance d is not trivial: a too small d wastes time, and too large values give imprecise results. Parameter d is a clear and strong drawback of the proposed algorithms, as the results are very sensitive to the value of d. Note however that the same remark holds for the modified clearing approach which outperformed by far all other methods in [23]. Nevertheless, because of this parameter, the generality of the results shown below is limited. In particular, we will limit our claims to the comparison with the results in [23], obtaining results that are comparable with the very good results of the modified clearing, with a similar parameter d, but with a simple restart algorithm.

Let us define K as the number of optima for the murder operator (the murder operator is not required for those complexity bounds). Let us first compare the complexity of the proposed restart algorithm with those provided in [23] (Table 1 (left)). The complexity for RDS/QRDS is immediate. In the case of murder operator, the complexity bound holds provided that K and d are such that the proportion of the domain which is forbidden by the murder operator is never larger than a fixed proportion of the whole domain. λ is the population size, 1 in our case (but the complexity results hold for any value of λ). RTS [10, 11] and SCGA [15] stand for Restricted Tournament Selection and Species Conserving Genetic Algorithm respectively. w is window size of RTS. N_c is the number of clusters.

Algorithm	Complexity
RDS/QRDS, Clearing	$\Theta(\lambda)$
Deterministic or probabilistic crowding	$\Theta(\lambda)$
RTS	λw
QRDS/RDS with murder	$O(\lambda K)$
SCGA	$\Omega(\lambda)$ and $O(\lambda^2)$
Sharing, modified clearing	$\Theta(\lambda^2)$
Clustering	$\Theta(\lambda N_c)$

	Number of evaluations	Number of restarts	Computational overhead for RDS vs QRDS
Constant initial step-size $\sigma_0 = 0.1$			
RDS	1652	13.7	1%
QRDS	1628	13.41	
Constant initial step-size $\sigma_0 = 0.01$			
RDS	740 ± 15.81	6.07 ± 0.23	64%
QRDS	452 ± 8.95	2.3 ± 0.11	
Constant initial step-size $\sigma_0 = 0.001$			
RDS	808 ± 18.37	6.44 ± 0.25	79%
QRDS	451 ± 8.54	2.21 ± 0.10	
Quadratically decreasing step-size, $\sigma_0 = 1.$			
RDS	726 ± 18.34	6.73 ± 0.28	46%
QRDS	498 ± 10.15	3.72 ± 0.15	
Quadratically decreasing step-size, $\sigma_0 = 0.1$			
RDS	755 ± 17.40	6.65 ± 0.25	64%
QRDS	461 ± 8.74	2.79 ± 0.11	

Table 1. Left: Complexity of various algorithms. Most results are from [23] (see text). Right: Results on the sine function in dimension 1, for RDS and WRDS, and different strategies for σ. The last column is the percentage of additional evaluations when using random instead of quasi-random.

Let us then test the different approaches on the test functions provided in [23]. The first function is a n-dimensional sine, with 5^n peaks, defined on $[0, 1]$ as$f(x) = 1 - \frac{1}{n}\sum_{i=1}^{n}(1 - \sin^6(5\pi x_i))$. In all experiments with this function, a

point x^* will be considered an optimum if $f(x^*) > 0.997$.
Another important test function is the hump function defined in [23] as: $f(x) = h \max(1 - (\inf_k ||x - x_k||/r)^{\alpha_k}, 0)$where $\alpha_k = 1$, $r = 1.45$, $h = 1$; sequence $x_1, \ldots, x_k$ is randomly drawn in domain $[0, 1]^D$; the problem is used in dimension 25 with $k = 50$, the most difficult case according to [23].

4.1 Constant initial step-size is dangerous

First introduced in [24], the idea of decreasing σ at each restart is somewhat natural, when considering the convergence proof (see Remark 1). From the results on the sine function in dimension 1 (Table 1-right), it is clear that a too large fixed σ leads to poor results, while a too small σ also hinders the performances. Indeed, when looking for all optima, a large initial step-size might be helpful in order to avoid poor local optima. However it is a bad idea for finding optima close to the frontier of the domain when there are big basins of attractions. Furthermore, the proved version, with quasi-random restarts and decreasing σ, performs well without any tuning, though less efficiently than the version with *a posteriori* chosen fixed σ. Yet, its good performances independently of any parameter tuning is a strong argument for the proved method.

4.2 Validating quasi-random sequences

Let us now focus on the comparison between the use of quasi-random vs random restarts, i.e., RDS vs QRDS. From results on the multi-dimensional sine function (Tables 1-right and 2), it is clear that quasi-random becomes more and more efficient as the number of optima increases.

Dimension D	RDS	QRDS	Additional cost for RDS over QRDS
3	143986 (803)	109128 (609)	32%
2	11673 (98)	8512 (71)	37%
1	777 (13)	447 (8)	74%

Table 2. RDS vs QRDS on the multidimensional sine function: number of evaluations (number of restarts), averaged over 30 runs, for finding the 5^D optima in dimension D. Here the step-size is quadratically decreasing with initial step-size $\sigma_0 = 0.1$.

Let us now revisit the sine function in dimension 1, and increase the number of optima by increasing its parameter K from 5 to 50, 500, and 1000. Other parameters of the algorithms are here $\sigma_{init} = 10^{-1}/K$, $\sigma^* = 5.10^{-4}/K$, and $d = 0.5/K$ for the murder threshold. The performances reported in Table 3-left witness the computational effort until all optima are found, i.e. there is a point of fitness > 0.997 at distance lower than half the distance between optima (please remember than 0.997 is the chosen threshold in this benchmark). Those results show that QRDS becomes more and more efficient when compared to RDS as the number of optima increases. Furthermore, the murder operator is clearly highly efficient.

	Nb of evaluations	Nb of restarts	Ratio
5 optima			
RDS	1484 ± 33.59	5.79 ± 0.24	25 %/
QRDS	1187 ± 30.27	4.17 ± 0.20	152%
QRDS+M	588 ± 3.24	1.31 ± 0.04	
50 optima			
RDS	30588 ± 156.28	171.45 ± 1.12	46% /
QRDS	20939 ± 100.04	102.24 ± 0.71	364%
QRDS+M	6583 ± 2.39	17.39 ± 0.05	
500 optima			
RDS	470080 ± 548.60	2900.75± 3.96	67% /
QRDS	281877± 279.15	1540.5± 2.01	603%
QRDS+M	66789 ± 3.07	161.92 ± 0.07	
1000 optima			
RDS	1009144 ± 747.05	6298.16 ± 5.40	61 % /
QRDS	627696 ± 409.41	3545.61 ± 2.96	655%
QRDS+M	133587 ± 3.11	320.8 ± 0.07	

Algorithm	Nb of optima found
Sharing	$\simeq 0$
D./P. Crowding	$\simeq 0$ / 0
RTS	$\simeq 0$
SCGA	$\simeq 0$
Clustering	$\simeq 0$
Clearing	$\simeq 43$
Modified Clearing	$\simeq 50$
RDS	49.92 (over 100 runs)
QRDS	49.95 (over 100 runs)

Table 3. Left: Performance of RDS, QRDS, and QRDS with Murder when the number of optima on the 1-D sine function increases. The last column shows the percentage of additional evaluations for RDS over the given algorithm. Right: Comparative results on the 25-dimensional hump problem. All results but RDS/QRDS are taken from [23], where the modified clearing is reported to have found all 50 optima in all of the 30 runs. RDS and QRDS being much faster (see Table 1), 100 experiments have been run easily, and almost all 50 optima have been consistently found. See text for details.

4.3 Comparison with Modified Clearing

In this section, we compare the restart as previously defined (quasi-random restarts, decreasing σ and murder) to the best techniques in the survey [23]. Interestingly, some algorithms tested in [23] (Probabilistic Crowding and SCGA) could not even find all optima of the simple sine function in dimension 1. Note that QRDS finds all optima within a few hundred evaluations, whereas according to [23] nearly 100 generations of population size 50 are necessary for finding the 5 optima for all methods. However, because [23] does not provide quantitative results, precise comparisons are not possible here. Therefore, the following comparative results use the hump function, for which [23] provides extensive experiments with detailed results. This function is particularly challenging, because [23] points out that no algorithm except their modified clearing can solve the problem. The modified clearing, however, finds all optima in each of the 30 runs, whereas all methods (as reported in Table 1), except clearing, do not even find a single optimum. Table 3-right reports the results of RDS and QRDS for the hardest instance, in dimension 25 where the number of optima is 50. Averaged over 100 runs, both methods find almost all 50 optima (on average more than 49.9).

5 Conclusion

This paper has introduced some generic framework for restart algorithms embedding a (local) optimization algorithm. With limited hypotheses about the

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

local properties of the embedded algorithm, in particular with respect to some initial parameter σ describing the size of its basins of attraction, we have proved some convergence properties with speed depending on the dispersion of the sequence of starting points, independently of any other parameter of the embedded algorithm.

Actual instances of the generic algorithm have then been proposed, embedding a (1+1)-Evolution Strategy with $\frac{1}{5}^{th}$ rule, for which the initial step-size plays the role of parameter σ above. Random and quasi-random sequences of starting points can be used. The proposed algorithms have mathematically proved convergence properties. Thanks to the decreasing of the initial step-size to 0, the convergence is proved independently of initialization parameters. Furthermore, experimental validation of the proposed algorithm have been conducted, comparing random and quasi-random sequences of starting points, and judging performance with respect to other previously published algorithms [23].

A first conclusion is the not surprising superiority of quasi-random restarts over random restarts. The improvement due to QR points is moderate (a logarithmic factor), but regular and increasing when considering more complicated cases (more optima to be found). We have no experimental evidence in this paper or in the literature for the superiority of any algorithm (whatever complicated and computationally expensive) over the simple restart algorithm with decreasing σ and quasi-random initializations that has been proposed here. QRDS performed equally or better than modified clearing [23], whilst keeping a linear computational cost. All methods with non quadratic cost benchmarked in [23] are much less efficient than QRDS, RDS or modified clearing, while RDS and QRDS are much cheaper than modified clearing.

The murder operator, that kills runs that get close to previously found optima, was found highly beneficial (up to more than 700 % speed-up for 500 optima). However, QRDS with murder operator has the same weakness than modified clearing: it introduces a crucial parameter taking into account the distance between different optima, somewhat similar to σ_{clear} parameter of modified clearing (that has additionally two arbitrary constants 1.5 and 3 for solving the difficult hump function).

Let us now discuss the limitations of our analysis, and some general elements. An important element in this work is the choice of benchmark functions. The hump function is in fact a distribution of fitness functions, and not only a fitness function, or random translations of a fitness function. We agree with the authors of [23] that solving the hump function is by no means an easy task. However, this benchmark has its limitations: as the function is locally very simple, and as a very loose precision is required, the best choice for the embedded ES is the 1+1-ES with a very loose halting condition: this is perhaps not the less realistic scenario, but this certainly does not covers all cases. A strong advantage of the hump function is that it cannot easily be overfitted. However, all successful methods on the hump function have a parameter which is intuitively related to the distance between optima: This suggests that, in spite of the random part in the hump function, some overfitting nevertheless happens when tuning an

algorithm on the hump function. Also, we feel these experiments are definitely convincing regarding the importance of mathematical analysis in MMO: it is otherwise very easy to conclude positively about an algorithm, without seeing its precise limitations. Thanks to theoretical analysis, we could clearly see that decreasing the step-size provides a real advantage (a decreasing step-size provides consistency) and that quasi-randomizing the restarts provides an improvement - and to the best of our knowledge, as QRDS has the best experimental results and the best proved results, this paper provides a clear and simple baseline for future research in MMO. A case in which clearing approaches might perform better than the proposed approach is the parallel case. The murder operator might be less efficient if several populations evolve simultaneously. The proposed approach has been designed for the case of MMO in which the goal is to locate all optima, opposed to the case where the goal is to locate the global optimum, but there exist many local optima. However, these two forms of MMO are probably not so different - if the local optima have fitness value close to the one of the global optimum x^*, we have to check all local optima.

References

1. A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005*, pages 1769–1776, 2005.
2. A. Auger, M. Jebalia, and O. Teytaud. XSE: quasi-random mutations for evolution strategies. In *Proceedings of EA'05, 12 pages*, 2005.
3. A. Auger and O. Teytaud. Continuous lunches are free plus the design of optimal optimization algorithms. *Algorithmica*, 57(1):121–146, 2010.
4. H. Beyer and B. Sendhoff. Covariance Matrix Adaptation Revisited–The CMSA Evolution Strategy–. *Parallel Problem Solving from Nature–PPSN X*, pages 123–132, 2008.
5. H.-G. Beyer. *The Theory of Evolution Strategies*. Springer, Heidelberg, 2001.
6. K. DeJong. *The Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Harbor, 1975. *Dissertation Abstract International*, 36(10), 5140B. (University Microfilms No 76-9381).
7. A. Georgieva and I. Jordanov. A hybrid meta-heuristic for global optimisation using low-discrepancy sequences of points. *Computers and Operations Research, - special issue on hybrid metaheuristics*, In press.
8. D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodalfunction optimization. In J. J. Grefenstette, editor, *ICGA*, pages 41–49. Lawrence Erlbaum Associates, 1987.
9. N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 282–291. Springer, 2004.
10. G. Harik. Finding multiple solutions in problems of bounded difficulty. Technical Report 94002, University of Illinois at Urbana Champaign, 1994.
11. G. Harik. Finding multimodal solutions using restricted tournament selection. In L. J. Eshelman, editor, *Proc. Sixth International Conference on Genetic Algorithms*, pages 24–31. Morgan Kaufmann, 1995.

12. D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13(4):455–492, 1998.
13. S. Kimura and K. Matsumura. Genetic algorithms using low-discrepancy sequences. In *GECCO*, pages 1341–1346, 2005.
14. P. L'Ecuyer and C. Lemieux. *Recent Advances in Randomized Quasi-Monte Carlo Methods*, pages 419 – 474. Kluwer Academic, 2002.
15. J. Li, M. Balazs, G. Parks, and P. Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evolutionary computation*, 10(3):207–234, 2002.
16. J.-P. Li, M. E. Balazs, G. T. Parks, and P. J. Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evol. Comput.*, 10(3):207–234, 2002.
17. S. R. Lindemann and S. M. LaValle. Incremental low-discrepancy lattice methods for motion planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2920–2927, 2003.
18. S. W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1995.
19. O. J. Mengshoel and D. E. Goldberg. Probabilistic crowding: Deterministic crowding with probabilisitic replacement. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 409–416, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.
20. H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia: SIAM, 1992.
21. A. B. Owen. Quasi-Monte Carlo sampling. In H. W. Jensen, editor, *Monte Carlo Ray Tracing: Siggraph 2003 Course 44*, pages 69–88. SIGGRAPH, 2003.
22. A. Pétrowski. A clearing procedure as a niching method for genetic algorithms. In *International Conference on Evolutionary Computation*, pages 798–803, 1996.
23. G. Singh and D. Kalyanmoy Deb. Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1305–1312, New York, NY, USA, 2006. ACM.
24. F. Teytaud and O. Teytaud. Log(lambda) Modifications for Optimal Parallelism. In *Parallel Problem Solving From Nature–PPSN XI*, Krakow Pologne, 09 2010.
25. O. Teytaud. When does quasi-random work? *Parallel Problem Solving from Nature–PPSN X*, pages 325–336, 2008.
26. O. Teytaud and S. Gelly. DCMA: yet another derandomization in covariance-matrix-adaptation. In D. Thierens et al., editor, *ACM-GECCO'07*, pages 955–963, New York, NY, USA, 2007. ACM.
27. F. van den Bergh and A. Engelbrecht. Cooperative learning in neural networks using particle swarm optimizers, 2000.
28. B. Vandewoestyne and R. Cools. Good permutations for deterministic scrambled halton sequences in terms of l2-discrepancy. *Computational and Applied Mathematics*, 189(1,2):341:361, 2006.
29. J. Villemonteix, E. Vazquez, and E. Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509–534, 2009.
30. X. Yin and N. Germay. A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization. In R. F. Albrecht, N. C. Steele, and C. R. Reeves, editors, *Artificial Neural Nets and Genetic Algorithms*, pages 450–457, Wien, 1993. Springer Verlag.

Local Optima Networks with Escape Edges

Sebastien Verel[2], Fabio Daolio[3], Gabriela Ochoa[1], Marco Tomassini[3]

[1] School of Computer Science, University of Nottingham, Nottingham, UK.
[2] INRIA Lille - Nord Europe and University of Nice Sophia-Antipolis, France.
[3] Faculty of Business and Economics, University of Lausanne, Lausanne, Switzerland.

Abstract. This paper proposes an alternative definition of edges (*escape edges*) for the recently introduced network-based model of combinatorial landscapes: *Local Optima Networks (LON)*. The model compresses the information given by the whole search space into a smaller mathematical object that is the graph having as vertices the local optima and as edges the possible weighted transitions between them. The original definition of edges accounted for the notion of transitions between the basins of attraction of local optima. This definition, although informative, produced densely connected networks and required the exhaustive sampling of the basins of attraction. The alternative escape edges proposed here do not require a full computation of the basins. Instead, they account for the chances of escaping a local optima after a controlled mutation (e.g. 1 or 2 bit-flips) followed by hill-climbing. A statistical analysis comparing the two LON models for a set of NK landscapes, is presented and discussed. Moreover, a preliminary study is presented, which aims at validating the LON models as a tool for analyzing the dynamics of stochastic local search in combinatorial optimization.

1 Introduction

The performance of heuristic search algorithms crucially depends on the structural aspects of the spaces being searched. An improved understanding of this dependency, can facilitate the design and further successful application of these methods to solve hard computational search problems. Local optima networks (LON) have been recently introduced as a novel model of combinatorial landscapes [10, 11]. This model allows the use of complex network analysis techniques [7] in connection with the study of fitness landscapes and problem difficulty in combinatorial optimization. The model is based on the idea of compressing the information given by the whole problem configuration space into a smaller mathematical object, which is the graph having as vertices the optima configurations of the problem and as edges the possible transitions between these optima. This characterization of landscapes as networks has brought new insights into the global structure of the landscapes studied, particularly into the distribution of their local optima. Moreover, some network features have been found to correlate and suggest explanations for search difficulty on the studied domains.

The definition of the edges in the LON model critically impacts upon its descriptive power with regards to heuristic search. The initial definition of edges in [10, 11], *basin-transition* edges, accounted for the notion of transitions between the local optima basins' frontiers. This definition, although informative, produces highly connected networks and requires the exhaustive sampling of the basins of attraction. We explore in

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

this article an alternative definition of edges, which we term *escape* edges, that does not require a full computation of the basins. Instead, the edges account for the chances (of a prospective heuristic search algorithm) of escaping a local optima after a controlled mutation (e.g. 1 or 2 bit-flips) followed by hill-climbing. This new definition produces less dense and easier to build LONs, which are more amenable to sampling and get us closer to a fitness landscape model that can be used to understand (and eventually exploit) the dynamics of local search on combinatorial problems.

The first goal of the present study is to compare and explore the relationships between the two LON models, based on (i) basin-transition edges and (ii) escape edges, respectively. Thereafter, we present a preliminary study that aims at validating the LON models in their descriptive power of the dynamics of stochastic local search algorithms. We conduct this validation by considering the behavior of two well-known stochastic local search heuristics, namely, Tabu Search [4] and Iterated Local Search [6]. The well known family of NK landscapes [5] is used in our study.

The article is structured as follows. Section 2, includes the relevant definitions and algorithms for extracting the LONs. Section 3, describes the experimental design, and reports a comparative analysis of the extracted networks of the two models. Section 4, presents our model validation study. Finally, section 5 discusses our main findings and suggest directions for future work.

2 Definitions and algorithms

A Fitness landscape [9] is a triplet (S, V, f) where S is a set of potential solutions i.e. a search space, $V : S \longrightarrow 2^S$, a neighborhood structure, is a function that assigns to every $s \in S$ a set of neighbors $V(s)$, and $f : S \longrightarrow R$ is a fitness function that can be pictured as the *height* of the corresponding solutions. In our study, the search space is composed of binary strings of length N, therefore its size is 2^N. The neighborhood is defined by the minimum possible move on a binary search space, that is the single bit-flip operation. Thus, for a bit string s of length N, the neighborhood size is $|V(s)| = N$.

The $HillClimbing$ algorithm to determine the local optima and therefore define the basins of attraction, is given in Algorithm 1. It defines a mapping from the search space S to the set of locally optimal solutions S^*. Hill climbing algorithms differ in their so-called *pivot-rule*. In best-improvement local search, the entire neighborhood is explored and the best solution is returned, whereas in first-improvement, a neighbor is selected uniformly at random and is accepted if it improves on the current fitness value. We consider here a best-improvement local searcher (see Algorithm 1). For a comparison between first and best-improvement LON models, the reader is referred to [8]

2.1 Nodes

As discussed above, a best-improvement local search algorithm based on the 1-move operation is used to determine the local optima. A local optimum (LO), which is taken to be a maximum here, is a solution s^* such that $\forall s \in V(s)$, $f(s) \leq f(s^*)$.

Let us denote by $h(s)$, the stochastic operator that associates to each solution s, the solution obtained after applying the best-improvement hill-climbing algorithm (see

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Algorithm 1 Best-improvement local search (hill-climbing).

```
Choose initial solution s ∈ S
repeat
  choose s' ∈ V(s), such that f(s') = max_{x∈V(s)} f(x)
  if f(s) < f(s') then
    s ← s'
  end if
until s is a Local optimum
```

Algorithm 1) until convergence to a LO. The size of the landscape is finite, so we can denote by LO_1, LO_2, $LO_3 \ldots, LO_p$, the local optima. These LOs are the vertices of the *local optima network*.

2.2 Basin-transition edges

The basin of attraction of a local optimum $LO_i \in S$ is the set $b_i = \{s \in S \mid h(s) = LO_i\}$. The size of the basin of attraction of a local optimum i is the cardinality of b_i, denoted $\sharp b_i$. Notice that for non-neutral[4] fitness landscapes, as are standard NK landscapes, the basins of attraction as defined above, produce a partition of the configuration space S. Therefore, $S = \cup_{i \in S^*} b_i$ and $\forall i \in S \; \forall j \neq i$, $b_i \cap b_j = \emptyset$.

We can now define the weight of an edge that connects two feasible solutions in the fitness landscape. For each pair of solutions s and $s^{'}$, $p(s \rightarrow s^{'})$ is the probability to pass from s to $s^{'}$ with the given neighborhood structure. In the case of binary strings of size N, and the neighborhood defined by the single bit-flip operation, there are N neighbors for each solution, therefore, considering a uniform selection of random moves:
if $s^{'} \in V(s)$, $p(s \rightarrow s^{'}) = \frac{1}{N}$ and
if $s^{'} \notin V(s)$, $p(s \rightarrow s^{'}) = 0$.
The probability to go from solution $s \in S$ to a solution belonging to the basin b_j, is[5]:

$$p(s \rightarrow b_j) = \sum_{s^{'} \in b_j} p(s \rightarrow s^{'}) \quad .$$

Thus, the total probability of going from basin b_i to basin b_j, i.e. the weight w_{ij} of edge e_{ij}, is the average over all $s \in b_i$ of the transition probabilities to solutions $s^{'} \in b_j$:

$$p(b_i \rightarrow b_j) = \frac{1}{\sharp b_i} \sum_{s \in b_i} p(s \rightarrow b_j) \quad .$$

2.3 Escape edges

The escape edges are defined according to a distance function d (minimal number of moves between two solutions), and a positive integer $D > 0$.

[4] For a definition of basins that deals with neutrality, the reader is referred to [11].

[5] Notice that $p(s \rightarrow b_j) \leq 1$ and notice also that this definition, disregarding the fitness values, is purely topological and is not related to any particular search heuristic.

There exists an edge e_{ij} between LO_i and LO_j if it exists a solution s such that $d(s, LO_i) \leq D$ and $h(s) = LO_j$. The weight w_{ij} of this edge is then: $w_{ij} = \sharp\{s \in S \mid d(s, LO_i) \leq D \text{ and } h(s) = LO_j\}$, which can be normalized by the number of solutions within reach w.r.t. such a distance $\sharp\{s \in S \mid d(s, LO_i) \leq D\}$.

2.4 Local optima network

The weighted local optima network $G_w = (N, E)$ is the graph where the nodes $n_i \in N$ are the local optima, and there is an edge $e_{ij} \in E$, with weight $w_{ij} = p(b_i \to b_j)$, between two nodes n_i and n_j if $p(b_i \to b_j) > 0$.

According to both definitions of edge weights, $w_{ij} = p(b_i \to b_j)$ may be different than $w_{ji} = p(b_j \to b_i)$. Thus, two weights are needed in general, and we have an oriented transition graph.

Figure 1 depicts a representative example of the alternative LON models. The figures corresponds to a real NK landscape with N = 18, K = 2, which is the lowest ruggedness value explored in our study. The left plot illustrates the basin-transition edges, while the center and right plots the escape edges with $D = 1$ and $D = 2$, respectively. Notice that the basin-transition edges (left) produce a densely connected network, while the escape edges produce more sparse networks.

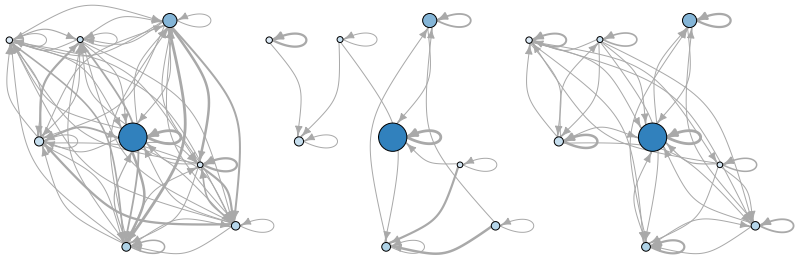


Fig. 1. Local optima network of an NK-landscape instance with N = 18, K = 2. Left: basin-transition edges. Center and Right: escape edges with $D = 1$ and $D = 2$, respectively. The size of the circles is proportional to the logarithm of the size of the corresponding basins of attraction; the darker the color, the better the local optimum fitness. The edges' width scales with the transition probability (weight) between local optima, according to the respective definitions. Notice that the basin-transition edges model (Left) is much more densely connected.

3 Comparative analysis of the LON models

In this section, we compare the LONs resulting from the different edges definitions discussed above. We chose to perform this analysis on the NK-model artificial landscapes, primarily to be able to compare directly with previous work [10, 11], but also

because this problem provides a framework that is of general interest in studying the structure of complex combinatorial problems [5].

The NK family of correlated landscapes is in fact a problem-independent model for constructing multimodal landscapes that can gradually be tuned from smooth to rugged. In the model, N refers to the number of (binary) genes in the genotype, i.e. the string length, and K to the epistatic interaction, i.e. the number of genes that influence a particular gene. By increasing the value of K from 0 to $N-1$, the landscapes can be tuned from smooth to rugged. The K variables that form the context of the fitness contribution of a gene, can be chosen according to different models, the two most widely studied being the *random neighborhood* model and the *adjacent neighborhood* model. As no significant differences between the two were found, neither in terms of the landscape global properties [5] nor in terms of their local optima networks (preliminary studies), we conduct our full study on the more general random model.

In order to minimize the influence of the random creation of landscapes, we considered 30 different and independent problem instances for each combination of N and K parameter values. In all cases, the measures reported are the average of these 30 landscapes. In the present study, $N = 18$ and $K \in \{2, 4, 6, 8, 10, 12, 14, 16, 17\}$, which are the largest possible parameter combinations that allow the exhaustive extraction of local optima networks. LONs for the two definitions of edges: (i) basin-transition and (ii) escape edges with $D \in \{1, 2\}$, were extracted and analyzed[6].

Table 1. Local optima network features. Values are averages over 30 random instances, standard deviations are shown as subscripts. K = epistasis value of the corresponding NK-landscape ($N = 18$); N_v = number of vertices; D_{edge} = density of edges ($N_e/(N_v)^2 \times 100\%$); L_{opt} = average shortest path to reach the global optimum ($d_{ij} = 1/w_{ij}$).

K	N_v	D_{edge} (%)			L_{opt}		
	all	Basin-trans.	Esc.$D1$	Esc.$D2$	Basin-trans.	Esc.$D1$	Esc.$D2$
2	$43.0_{27.7}$	$74.182_{13.128}$	$8.298_{4.716}$	$22.750_{9.301}$	$21.2_{8.0}$	$16.8_{4.7}$	$33.5_{14.1}$
4	$220.6_{39.1}$	$54.061_{4.413}$	$1.463_{0.231}$	$7.066_{0.810}$	$41.7_{10.5}$	$19.2_{5.1}$	$53.7_{12.4}$
6	$748.4_{70.2}$	$26.343_{1.963}$	$0.469_{0.047}$	$3.466_{0.279}$	$80.0_{19.1}$	$22.2_{3.9}$	$66.7_{12.9}$
8	$1668.8_{73.5}$	$12.709_{0.512}$	$0.228_{0.009}$	$2.201_{0.066}$	$110.1_{13.8}$	$24.0_{4.9}$	$76.6_{9.1}$
10	$3147.6_{109.9}$	$6.269_{0.244}$	$0.132_{0.004}$	$1.531_{0.036}$	$152.8_{19.3}$	$27.3_{5.0}$	$90.7_{8.4}$
12	$5270.3_{103.9}$	$3.240_{0.079}$	$0.088_{0.001}$	$1.115_{0.015}$	$185.1_{23.8}$	$30.3_{6.7}$	$108.3_{12.3}$
14	$8099.6_{121.1}$	$1.774_{0.035}$	$0.064_{0.001}$	$0.838_{0.009}$	$200.2_{16.0}$	$38.9_{9.6}$	$124.7_{8.6}$
16	$11688.1_{101.3}$	$1.030_{0.013}$	$0.051_{0.000}$	$0.647_{0.004}$	$211.8_{15.0}$	$47.9_{11.4}$	$146.2_{11.2}$
17	$13801.0_{74.1}$	$0.801_{0.007}$	$0.047_{0.000}$	$0.574_{0.002}$	$214.3_{17.5}$	$55.7_{12.5}$	$155.9_{12.2}$

3.1 Network features and connectivity

Number of nodes and edges. The 2^{nd} column of Table 1, reports the number of nodes (local optima), N_v, which is the same for all the studied landscapes and models. The

[6] Some of the the tools for fitness landscape analysis and the local search heuristics, were used from the "ParadisEO" library [2]; data treatment and network analysis are done in "R" with the "igraph" package [3].

number of nodes increase exponentially with increasing values of K. The networks, however, have a different number of edges, as can be appreciated in the 3^{rd}, 4^{th}, and 5^{th} columns of Table 1, which report the number of edges normalized by the square of the number of nodes (density of edges). Clearly, the density is higher for the basin-transition edges, followed by the escape edges with $D = 2$, and the smaller density corresponds to $D = 1$. The trend is, however, that density decreases steadily with increasing values of K, which supports the correlation between the two models.

With the basin-transition edges, LONs are densely connected, especially when K is low: 74% and 54% of all possible edges are present, on average, for $K \in \{2, 4\}$. The escape edges produce sparsely connected graphs. Indeed, the $D = 1$-escape edge model, produces networks that are not completely connected, with the number of connected components ranging between 1.67 and 8.37 in average. The global optimum, though, always happens to belong to the largest connected component, which, in our analysis, comprises an average proportion of solutions raising, with increasing values of K, from 0.9392764 to 0.9999879.

The networks with escape edges and $D = 2$, are always connected. The density decreases with the epistasis degree. For high Ks, the density values are close to those of the basin-transition networks. Figure 2 (Left) illustrates what is happening in terms of the average degree of the outgoing links. First, notice that the difference with the basin-transition networks is maximal when K is between 4 and 12. Whereas for $D = 1$ the outgoing degree only increases from 1.7 to 5.5 across the range of K values, for $D = 2$ the growth is faster and reaches 78.2, not far from the 109.5 score of the LON with basin-transition edges. The size of the basins could provide an explanation for this: at high values of K basins are so small that a 2-bit mutation from the local optimum is almost enough to recover the complete topology.

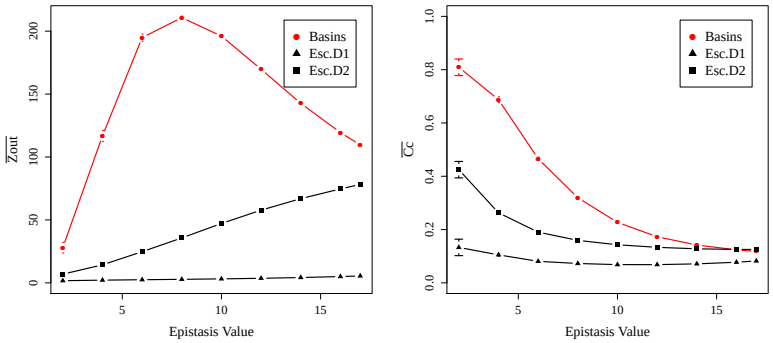


Fig. 2. Average out-degree (Left) and average clustering coefficient (Right) vs epistasis value.

Clustering coefficient. LONs with basins-transitions edges present a somewhat symmetric structure: when two nodes i and j are connected, both edges e_{ij} and e_{ji} are present (even though their weights are in general different, $w_{ij} \neq w_{ji}$). Moreover,

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

those connections often form triangular closures whose frequency is given by the global clustering coefficient. As Figure 2 (Right) shows, this measure of transitivity is lower with the escape-transition edges, but the difference could be due to the different number of edges of those LONs. Values for $D = 1$ are remarkably low, even if the calculation disregarded the direction of edges. Overall though, the decreasing trend w.r.t. the landscape ruggedness, remains common. In other words, even with escape-transition edges, the clustering coefficient can be retained as a measure related to problem complexity: it decreases with the non-linearity of the (NK-) problem.

Shortest paths. Due to the differences in topology, in the escape edges networks not all the paths are possible: few nodes might be disconnected, or they might not be reachable due to direction constraints (these are more "asymmetric" networks, as can be seen in Fig. 1). Thus, while evaluating shortest paths, only paths connecting reachable couples of nodes are averaged. Moreover, there are different ranges of weights, so the values displayed in Figure 3 have been normalized. An unexpected behavior can be observed for $D = 2$: the average path length peaks at $K = 4$ and stays always high. Maybe the increasing connectivity of nodes (see Fig. 2) counteracts their increase in numbers. However, some paths are more important than others, for example those who lead to the global optimum (see Fig. 3 (right)). With respect to these paths, all the LON models show the same trend: the paths increase in length as ruggedness increases.

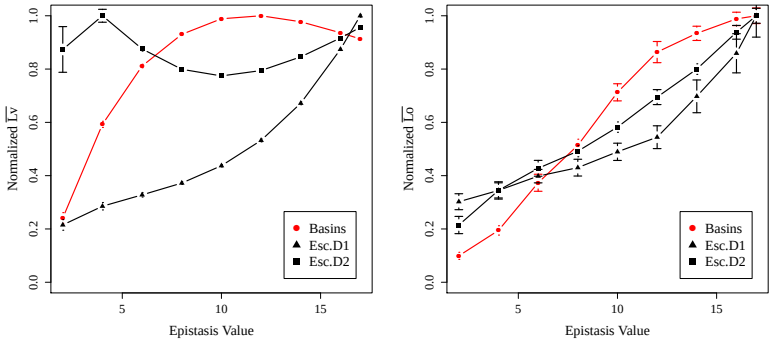


Fig. 3. Shortest paths over the LON vs epistasis value. Left: average geodesic distance between optima. Right: average shortest path to the global optimum. Each curve has been divided by its respective maximum value.

3.2 Characterization of weights

Disparity. Figure 2 (Left) gave the average connectivity of vertices, counting outgoing links. One might then ask whether or not there are preferential directions when leaving a particular basin, i.e. if for a given node i, the weights w_{ij} (with $j \neq i$) are equivalent. For this purpose, we measure the disparity Y_2 [1], which gauges the heterogeneity of the contributions of the edges of node i to the total weight. For a large enough degree

z_i, when there is not a dominant weight, then $Y_2 \approx 1/z_i$. The connectivity of LONs with escape edges with $D = 1$ is weak, and it is difficult to draw conclusions based on disparity only, as Figure 4 (left) illustrates. In the example illustrated, where $K = 4$, (i.e. relatively low epistasis, and so not a random structure), Y_2 approaches $1/z$ for escape-$D = 1$, whereas it has distinctively higher values for both the escape-$D = 2$, and the basin-transition edges.

However, the common trend is that disparity decreases with increasing epistasis: as the landscapes become more rugged, the transition probabilities to leave a particular basins appear to become more uniform, which could relate to the search difficulty. This is clear from Fig 4 (right), where Y_2 approaches $1/z$ on average as K grows.

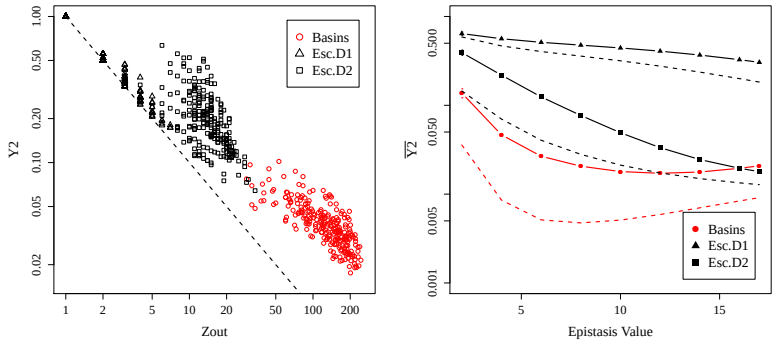


Fig. 4. Disparity of edges' weights. Left: scatter plot of disparity against outgoing degree (lin-log scale) for an instance with epistasis $K = 4$. Right: average vertex disparity for outgoing edges vs epistasis value. Dotted lines represent the inverse of the outgoing degree.

Strength. In a general weighted network, the degree of a vertex naturally extends into its strength, which measures its weighted connectivity. In LON model, basins size, as well as connectivity, generally correlate with fitness value [8]. Thus we ask if the incoming strength of a given node, i.e. the sum of the transition probabilities for all the incoming connections, correlates with the fitness of its LO. Figure 5 gives a clear affirmative answer for all the definition of edges.

Correlation of weights among edges' definitions. The LON models resulting from the alternative definition of edges show different structures but common trends w.r.t. the features that are related to problem difficulty. We conclude this subsection by directly comparing the transition probabilities that result from the alternative edge definitions. For this purpose, Figure 6 shows the Spearman's rank correlation between the corresponding rows of the weighted adjacency matrix, for different LON models of the same instance. The statistic is always positive, but, given the sparser nature of the escape-$D = 1$ networks, the result is weaker in this case. With $D = 2$, though, the correlation with the basin-transition definition is consistently good, which agrees with the previous

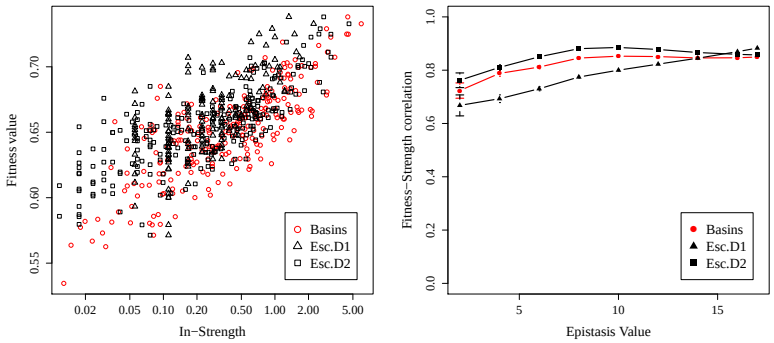


Fig. 5. Correlation between the strength of a node and its fitness value. Left: scatter plot of fitness against weighted connectivity (lin-log scale) for an instance with epistasis $K = 4$. Right: average correlation Spearman's coefficient between in-strength and fitness value vs epistasis value.

findings on this topology. Indeed, as the landscape ruggedness increases, the correlation between $D = 1$ and $D = 2$ becomes smaller.

4 Model validation: local search dynamics

In this section, we analyze the connection between the LON model and dynamics of local search (LS) in an attempt to validate its descriptive power. Does a LS follow the edges of the LON? Which edge definition is the most accurate to predict the dynamics of a local search heuristic? This is a preliminary study on one particular NK-landscape instance with $N = 18$ and $K = 4$, a larger analysis will be the subject of future work.

4.1 Experiment setup

We choose two simple but efficient stochastic LS heuristics, namely Iterated Local Search (ILS) [6], and Tabu Search (TS) [4]. Both are given $2.6 * 10^4$ function evaluations ($\approx 10\%$ of the search space), or stop when reaching the global optimum.

The search trajectory is traced and filtered according to the basins of attraction: each solution belongs to one basin, so it is labelled by the corresponding local optimum. We then considered that a LS stays in the same basin, if the solutions s_t and s_{t+1} both belong to the same basin and the fitness increases: $f(s_t) \leq f(s_{t+1})$. Otherwise, the LS jumps from one basin to another one. In that manner, when accumulated from a number of independent runs of the LS, 10^4 in our experiments, it is possible to compare the empirical transition frequencies between basins with the corresponding edge weights of a given LON. Such a high number of independent runs is necessary because often times single trajectories go through few LOs and only once.

When a LS performs a transition between two basins that are not connected in the LON model, we add to the latter a virtual edge with weight equal to 0.0. Of course, we

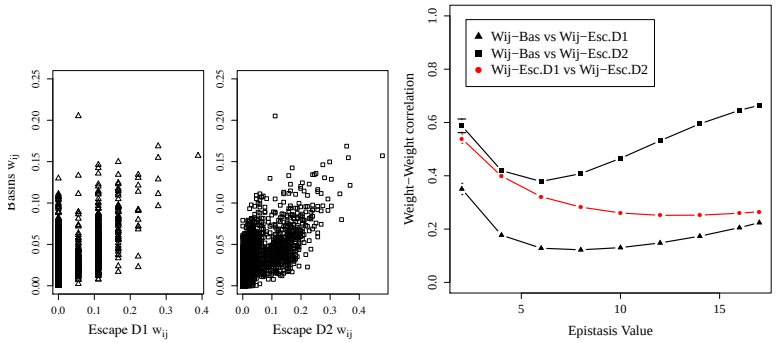


Fig. 6. Correlation between the weights resulting from the the new and the old definition of edges. Left: scatter plot for an instance with epistasis $K = 4$. Right: average Spearman's rank correlation coefficient vs epistasis value. No self-loops.

do not consider the LON edges that are not sampled, as it is not possible to compute the transition frequency between nodes that have not been visited. Finally, the LS under study are based on hill-climbing, thus the LON self-loops are also discarded.

4.2 Results

Figure 7 shows the scatter plots for the correlations between the edges weights of the different LON models and the empirical transition frequencies between basins.

Iterated Local Search. Our implementation is based on a steepest-ascent (best-improvement) hill-climbing, which, at each iteration, moves to the best neighboring solution, and stops on a local optimum. The ILS perturbation is the k bit-flip mutation, which flips k bits at random, $k \in \{1, 2\}$ in the present study. A mutation is accepted as a new solution if its fitness value is strictly better than the current one. Intuitively, the ILS follows the same edges of the escape definition: from the LO, k bits are flipped; the the main difference lies in the acceptance criterion used by the ILS. Indeed (see Fig. 7), correlations are all significant. As expected, for the 1-bit-flip perturbation, Spearman coefficient is higher for escape edges with $D = 1$, and lower for $D = 2$-escape edges (0.49 for $D = 1$, 0.45 for $D = 2$, and 0.48 for basin edges). For the 2-bit-flips perturbation, the highest correlation is for the escape edges with $D = 2$, and the smallest for escape edges with $D = 1$ (0.41 for $D = 1$, 0.75 for $D = 2$, and 0.72 for basin edges). Notice how figures for basin edges and $D = 2$-escape edges are very similar.

Tabu Search. Our implementation uses the 1-bit-flip mutation, and, in order to obtain a better diversification, a set of moves are forbidden during the search; those tabu-moves are stored in a memory that lasts $\lambda = N = 18$ iterations. A tabu move is nonetheless accepted when it finds a new best-so-far solution. Referring to Figure 7, the correlation values are 0.27 for the $D = 1$-escape edges, 0.53 for $D = 2$, and 0.79 for the basin edges. All values are significant and positive, particularly so for the more interesting LON definitions: $D = 2$-escape edges and basin edges.

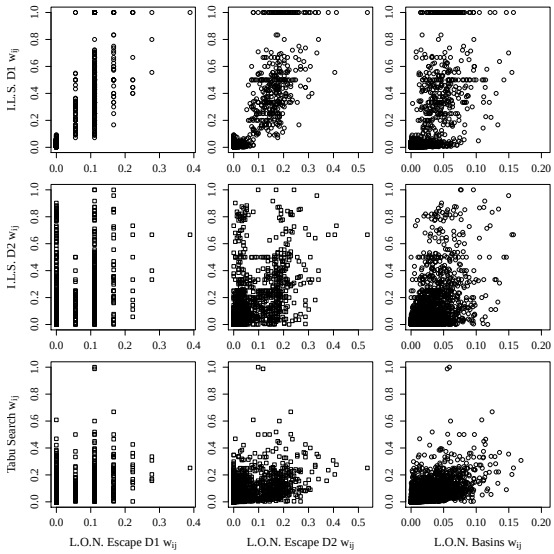


Fig. 7. Correlation (Spearman coefficient) between the edge weight and the empirical transition frequency of a local search. 10^4 independent runs on an NK-instance with $N = 18$, $K = 4$. From top to bottom, Iterated Local Search with 1- and 2-bit-flips, and Tabu Search have been tested. From left to right, the empirical frequencies are plotted against the corresponding edge weights according to the Escape $D \in \{1, 2\}$ and Basins definition, respectively. Only transitions between different basins are considered (no self-loops).

The result of this preliminary study is encouraging because it shows that the LON could capture the coarse-grained dynamics of a LS. In particular, the escape edge definition with $D = 2$ seems to be informative enough to correlate with the trajectories of a simple ILS, or a simple TS. Of course, further studies have to confirm this result for a larger class of LS, a broader number of instances, and on other problems, as to delineate the validity domain of the LON model.

5 Concluding remarks

The local optima networks (LON) model is a mesoscopic representation of a problem search space, which deals with the local-optima basins of attractions as the meso-state level of description. In this contribution, an alternative definition of edges (*escape edges*), has been proposed. Our statistical analysis, on a set of NK-landscapes, shows that the escape edges are as informative as the original (basin- transition) edges. We reach this conclusion because, for both LON models, the analyzed network features (such as the clustering coefficient, disparity, correlation between in-strength and fitness of local optima, and path length to global optimum) are always consistent with the non-linearity of the problem (the K parameter), which tunes the landscape ruggedness. Indeed, the edges' weights are positively correlated between the different definitions.

We also present a preliminary analysis that aims at validating the model. We show that the dynamics of simple stochastic local search heuristics such as Iterated Local Search, or Tabu Search, tend to follow the edges of LONs according to the rate defined by the weights. The model validation in the present study is a first step. Starting from the work by Reidys and Stadler [9] on combinatorial fitness landscapes, we could conduct a spectral analysis of the LON model. From the adjacency matrix of the LON graphs, it should be possible to build a Markov Chain having the previously discussed transition probabilities. That would allow us to compare the stationary distribution of different LON models of the same search space. In this case, an empirical assessment could be performed in a more informed way, because we could estimate the number of local search runs that are necessary to have a good sampling, as in a Monte Carlo method. The model validation could, then, go together with a prediction of the LS dynamics.

Overall, the present study opens up relevant perspectives. The escape edges definition will allow us to design a sampling methodology of the LONs. The enumeration of the basins of attraction is impossible on realistic search spaces. Therefore, the original definition of edges is restricted to study small search spaces. With the new definition, and through sampling of large networks such as breadth-first search, forest-fire or snowball sampling, we will be able to study the properties of LONs for real-world combinatorial search spaces. Our hope is that, by combining the LON model of the search dynamics and the ability to build LONs for large search spaces, we will be able to perform off-line parameter tuning of evolutionary algorithms and local search heuristics. Moreover, if the LON features are collected along the search process, on-line control of the parameters could also be achieved.

References

1. Barthélemy, M., Barrat, A., Pastor-Satorras, R., Vespignani, A.: Characterization and modeling of weighted networks. Physica A 346, 34–43 (2005)
2. Cahon, S., Melab, N., Talbi, E.G.: Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. Journal of Heuristics 10, 357–380 (2004)
3. Csardi, G., Nepusz, T.: The igraph software package for complex network research. InterJournal Complex Systems, 1695 (2006)
4. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Norwell, MA, USA (1997)
5. Kauffman, S.A.: The Origins of Order. Oxford University Press, New York (1993)
6. Lourenço, H.R., Martin, O., Stützle, T.: Iterated local search. In: Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 57, pp. 321–353. Kluwer Academic Publishers (2002)
7. Newman, M.E.J.: The structure and function of complex networks. SIAM Review 45, 167–256 (2003)
8. Ochoa, G., Verel, S., Tomassini, M.: First-improvement vs. best-improvement local optima networks of nk landscapes. In: Parallel Problem Solving from Nature - PPSN XI. Lecture Notes in Computer Science, vol. 6238, pp. 104–113. Springer (2010)
9. Reidys, C., Stadler, P.: Combinatorial landscapes. SIAM review 44(1), 3–54 (2002)
10. Tomassini, M., Verel, S., Ochoa, G.: Complex-network analysis of combinatorial spaces: The NK landscape case. Phys. Rev. E 78(6), 066114 (2008)
11. Verel, S., Ochoa, G., Tomassini, M.: Local optima networks of NK landscapes with neutrality. IEEE Transactions on Evolutionary Computation (to appear)

Visual Analysis of population scatterplots

Evelyne Lutton[1], Julie Foucquier[2], Nathalie Perrot[2],
Jean Louchet[3], and Jean-Daniel Fekete[1]

[1] AVIZ Team, INRIA Saclay - Ile-de-France,
Bat 490, Université Paris-Sud, 91405 ORSAY Cedex, France.
Evelyne.Lutton@inria.fr, Jean-Daniel.Fekete@inria.fr
[2] UMR782 Génie et Microbiologie des Procédés Alimentaires.
AgroParisTech, INRA, 78850 Thiverval-Grignon, France.
Julie.Foucquier@grignon.inra.fr, nathalie.perrot@grignon.inra.fr
[3] Artenia, 24 rue Gay-Lussac, 92320 Châtillon, France
Jean.Louchet@gmail.com

Abstract. We investigate how visual analytic tools can deal with the huge amount of data produced during the run of an evolutionary algorithm. We show, on toy examples and on two real life problems, how a multidimensional data visualisation tool like ScatterDice/GraphDice can be easily used for analysing raw output data produced along the run of an evolutionary algorithm. Visual interpretation of population data is not used very often by the EA community for experimental analysis. We show here that this approach may yield additional high level information that is hardly accessible through conventional computation.

Keywords : Optimisation, Artificial Evolution, Genetic algorithms, Visual Analytics, Experimental analysis of algorithms, parameter tuning, fitness landscape visualisation.

1 Introduction

Experimental analysis is an important aspect of research in Evolutionary Computation, both in the development of new algorithms and in the understanding of the internal mechanisms of artificial evolution. It is particularly difficult to deal with the huge amount of information that can be collected during the run of an evolutionary algorithm, as the information is usually highly multidimensional, and combines continuous, discrete and symbolic data.

Visualisation utilities associated with available EA software usually display fitness curves, convergence diagrams, and various statistics, but they more rarely allow to visualise raw genomic data. The importance of efficient visualisation tools for EAs is not a new issue [18], and various solutions have been proposed. The methods are commonly split into two types of visualisation tools: on-line tools, that display a set of monitoring curves during an EA run (e.g. fitness of the best individual, statistics, diversity), and off-line tools, that do a "post-mortem" analysis with more sophisticated results. However, important, difficult issues remain unsolved:

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

- How to visualise an individual. This task is particularly complex when the search space is large and multidimensional, or when the genome combines symbolic and numerical (discrete and/or continuous) values. The particular case of genetic programming has also been considered[7]. Existing solutions are usually problem dependent, and may call for the display of phenotypes (signals, images, sounds, graphs, networks, etc ...).
- How to visualise a population, i.e. a possibly large set of multidimensional points. The ability to efficiently visualise fitness landscapes is still a challenge: approaches like fitness-distance correlation only give a partial answer.
- How to visualise the history and evolutionary mechanisms. This issue is important, as visualising various statistics about an evolving population may not be enough to understand some of the complex or hidden mechanisms of EAs, like the action of operators. Being able to follow the transmission of genetic material inside a population has been partially addressed by schemata analysis, however there is still a strong need when dealing with continuous landscapes.
- How to visualise the result in non-standard EAs. A good example of this is the case of multi-objective EAs, as the growing size of the problem EA are able to solve lead to outputs made of large, high dimensional Pareto datasets. There is a strong need of efficient visualisation tools, that may help to monitor (on line or even interactively) multidimensional Pareto fronts [13].

In this paper, we investigate recent tools developed by the visual analytics community, that may provide efficient and generic answers to some of the previous challenges. The paper is organised as follows. A short review of existing visualisation systems for EAs is given in Section 2. Section 3 presents the ScatterDice / GraphDice tool. An analysis of population clouds using ScatterDice/GraphDice is developed in Section 4 for some classical test functions. Tests are based on the EASEA language[4][4]. Two real life examples are then presented in Section 5, and as a conclusion Section 6 sketches future developments for a GraphDice version adapted to EA visualisation.

2 Visualisation of EA data

2.1 On-line visualisation

Almost any evolutionary software proposes nowadays its own on-line display facilities. It is often reduced to visualise how the best fitness value evolves along generations. We give below some examples that provide additional features. and Bullock[1, 3] show the importance of tracking evolutionary activity via plots that visualise the history at different levels. For instance, genotype's activity corresponds to the frequence of a given genotype in a population, which appears, increases or decreases along generations, forming what they call "waves".

Pohlheim[15] proposed in 1999 a visualisation system adapted to the Genetic and Evolutionary Algorithm Toolbox for Matlab - GEATbx[14]. His system allows various

[4] The software is available at `http://sourceforge.net/projects/easea/`

visualisation modes, and gives for instance the current state of a population (one generation), visualises a run (all the generations), or different runs for comparisons. Additionally he copes with the problem of visualising high-dimensional data using multidimensional scaling (reduction to a 2D visualisation that preserves distance relationships), and uses a 2D representation to show the paths followed by the best individual of every generation.

Kerren and Egger in 2005 [6] developed a Java-based on-line visualisation tool, EAVis, in which it is possible to embed an application-dependent Phenotype View.

Collins in 2003 [5] provided a survey chapter on this topic and identified some directions for future research. His main conclusion concerns the strong need for flexible visualisation environments, as he considered that current solutions were still too problem dependent.

2.2 Off-line visualisation

Off-line visualisation systems allow displaying more data, including multidimensional data, which is one of the important issues in current visualisation systems. Spears [19] provided in 1999 a brief overview of multidimensional visualisation techniques for visualising EAs, such as the use of colour, glyphs, parallel coordinates or coordinates projections. For discrete data, Routen in 1994 [16] suggested to adopt a modified Hinton diagram[5] in which a rectangle represents a chromosome and its size is determined by the fitness of the chromosome. Let us give below a list of some off-line systems:

- William Shine and Christoph Eick[18] describe the features of a GA-visualization environment that uses quadcodes to generate search space coverage maps, employs 2D-distance maps to visualize convergence, and uses contour maps to visualize fitness.
- The VIS system [21] proposed in 1999 allows a navigation at various levels of detail, and manages transitions between related data levels. The visualisation of the most detailed level is based on ad-hoc representations (bar codes, colors, alleles frequencies) but does not allow visualising multidimensional continuous genomes.
- Emma Hart [10] proposed GAVEL in 2001, an off-line visualisation system adapted to generational GAs, that provides a means to understand how crossover and mutation operations assemble the optimal solution, and a way to trace the history of user-selected sets of alleles. It allows a display of the complete history across all generations of chromosomes, individual genes, and schemata.
- Marian Mach [12] presented in 2002 a simple and interactive “post-mortem” GA visualising tool focused on the visualisation of multidimensional data via 1D projections.

Annie Wu [21], who developed the VIS system, gave the following list of desirable tasks for visualisation systems: *(a)* to examine individuals and their encodings in detail, *(b)* to trace the source and survival of building blocks or partial solutions, *(c)* to trace

[5] A Hinton diagram provides a qualitative display of the values in a matrix. Each value is represented by a square whose size is related to the magnitude, and color indicates sign.

family trees to examine the effects of genetic operators, *(d)* to examine populations for convergence, speciation, etc, *(e)* to trace gross population statistics and trends to move freely in time and through populations.

The genome representation issue, item *(a)*, is perhaps the most complex one, and as we have seen above, various solutions have been proposed, depending if we are dealing with continuous or discrete genomes. For Genetic Programming, the question is even more complex: a solution proposed by Jason Daida [7] in 2005 consists in visualizing big tree structures as large sized graphs.

The issue addressed by the GAVEL system, that appears as very challenging in off-line systems and that is mentioned by Annie Wu as items *(c)* and *(d)*, is to be able to trace the history of individuals. Spears [19] mentioned also that being able to track the origin of the fittest individual per generation is a very important issue for parameter tuning.

The question of family trees visualisation has more recently been considered by Zbigniew Walczak in 2005 in a short chapter [20] where he proposed to visualise evolutionary processes using graph drawing software.

3 ScatterDice / GraphDice

Visual analytics is a multidisciplinary field that integrates various sophisticated computational tools with innovative interactive techniques and visual representations to facilitate human interpretation of raw data. We present in this section a tool developed by researchers in this field, that seems to answer in a generic way to some needs identified in the previous sections.

ScatterDice[8] is a multidimensional visual exploration tool, that enables the user to navigate in a multidimensional set via simple 2D projections, organised as scatterplot matrices. The visual coherence between various projections is based on animated 3D transitions. A scatterplot matrix presents an overview of the possible configurations, thumbnails of the scatterplots, and support for interactive navigation in the multidimensional space. Various queries can be built using bounding volumes in the dataset, sculpting the query from different viewpoints to become more and more refined. Furthermore, the dimensions in the navigation space can be reordered, manually or automatically, to highlight salient correlations and differences among them[6].

A recent evolution of ScatterDice using the same principles but with many additional features is GraphDice[2]. It allows reading the same type of data (.csv files), and other more sophisticated formats, as it also embeds graph visualisation utilities[7].

4 Analysing successive populations

ScatterDice or GraphDice can be used to visualise data collected during the run of an EA[11]. At each generation, the content of the current population can be written

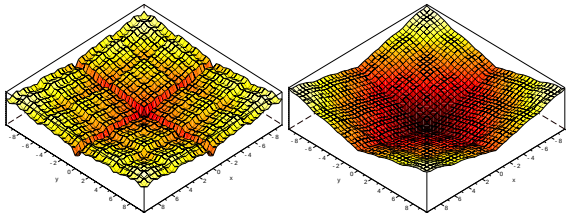
[6] A demo of ScatterDice can be launched from `http://www.aviz.fr/~fekete/scatterdice/`, it accepts standard .csv files (although it may be necessary to add a second line after the header giving the data type for each column - INT, STR, REAL, etc).

[7] A demo of GraphDice is also accessible at `http://www.aviz.fr/graphdice/`

into a ".csv" file as on figure 1, creating what can be called a "cloud" of successive populations. This cloud of multidimensional points is visualised using ScatterDice or GraphDice, to produce various, sometimes unusual, viewpoints.

```
Generation;Fitness;x[0];sigma[0];x[1];sigma[1]
INT;DOUBLE;DOUBLE;DOUBLE;DOUBLE;DOUBLE
0;3.75447;-0.12508;0.195626;0.524069;0.255402
0;1.17484;-0.573358;0.142053;0.924887;0.392851
0;2.28066;-0.533583;0.183641;0.546523;0.461643
0;1.92319;-0.70052;0.338246;0.582533;0.406443
0;2.75539;0.784538;0.182157;-0.940648;0.383136
0;3.08894;-0.770051;0.190012;-0.840338;0.359992
0;2.30766;0.380979;0.124581;0.0379446;0.469388
0;3.30957;-0.704403;0.453222;0.208484;0.182612
...
```

Fig. 1. A simple .csv file collected during a run (minimisation of the 2D Weierstrass function $H = 0.2$)



$H = 0.2$ (very irregular) $H = 0.9$ (more regular).

Fig. 2. 2D Weierstrass functions with Hölder Exponent H.

The test functions used in the experiments below are the following:

– Weierstrass functions (see figure 2) defined in a space of dimension 2, of Hölder exponents $H = 0.2$ (very irregular) and $H = 0.9$ (more regular).

$$f(x,y) = \sum_{n=-\infty}^{+\infty} 2^{-nH}(1-\cos 2^n x) + \sum_{n=-\infty}^{+\infty} 2^{-nH}(1-\cos 2^n y)$$

– Rosenbrock function[8] in a space of dimension 10.

$$f(x_1,...,x_{10}) = \sum_{n=1}^{9} 100(x_i^2 - x_{i+1})^2 + (1-x_i)^2$$

The genetic engine is a simple generational algorithm on R^n using tournament selection, geometric (barycentric) crossover and log-normal self-adaptive mutation. Additional dimensions are thus considered in the search space, the σ_i values, that represent the mutation radius for each coordinate x_i. The population size is 100 and the algorithm runs for 100 generations. This genetic engine is available in the sample programs (weierstrass.ez) distributed with the EASEA software.

[8] See for instance http://en.wikipedia.org/wiki/Rosenbrock_function

VI

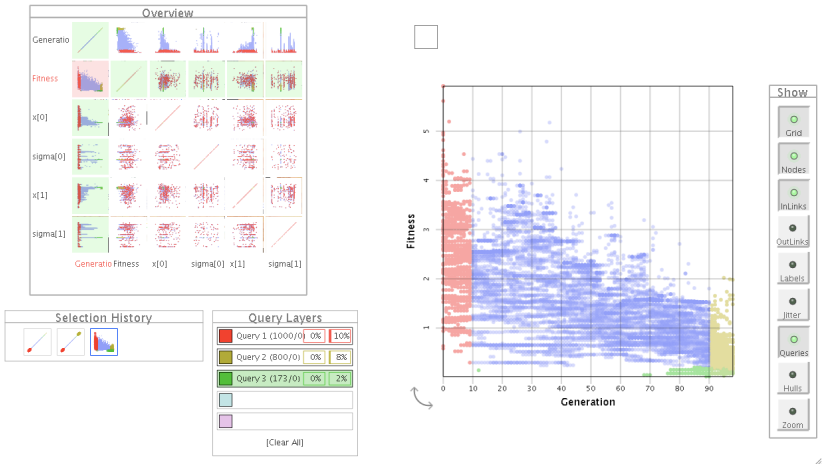


Fig. 3. 2D Weierstrass function of Hölder exponent 0.2. Scatterplot and Fitness versus generation view. Red points correspond to the first 10 generations, yellow points, to the 10 last ones, and green points, to the best fitness areas.

A visualisation of the population cloud for the 2D Weierstrass function with dimension 0.2 is given on figure 3. The scatterplot matrix on the left of the figure, gives an overview of the possible visualisations. The columns and lines of this matrix can be dragged and dropped as wished by the user. A default order is proposed, based on an algorithm that reduces "clutter"[2]. On the right, a detailed view is given, corresponding to the selected cell of the matrix (hightlighted in red), on which some queries have been visualised: in red, the points corresponding to the first 10 generations of the run, in yellow, the last 10 generations, and in green, the best fitness points. The queries are organised as layers, the "Query layers" window gives the details of the three queries, with some additional measurements (percentage of selected points, and percentage of selected edges if a graph is visualised : GraphDice actually considers a set of points as a degenerate graph, made of a collection of nodes with no edges). Bottom left, a "Selection History" window shows how the queries have been sculpted: queries 1 and 2 have been activated on the Generation versus Generation view, i.e. the top left plot of the scatterplot matrix, while query 3 has been made on the fitness versus generation plot. On the extreme right of the window, a toolbar proposes various visualisation options (the "Show" window), for instance "Grid" activates a grid on the dataset, "Labels" allow to display attributes attached to a point. It is thus possible for instance to identify different runs of an EA using a label, and use this option to separate the data when needed. "Hull" displays a convex hull for each query, and "Zoom" activates an automatic zoom focused on the selected data.

The Fitness versus Generation plot, that displays the whole set of individuals generated along the evolution, provides additional information about the distribution of successive populations. When observed from a different viewpoint, for instance according to x_0 and x_1 or to σ_0 and σ_1 like in figure 4, it can be noticed that the population diver-

sity decreases slowly, and converge toward a point of the 2D plane, while stabilising on some mutation parameters. The first generations (red points of the query 1) are spread in a rather uniform way on the whole search space, while the last ones (yellow points) are concentrated in the areas of best fitness (green points). A green point appears rather early (see generation 12 on the Fitness versus Generation plot), but green points start to multiplicate rather late (from generation 68) in the evolution.

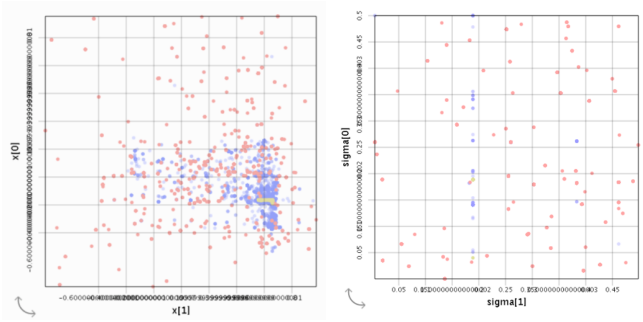


Fig. 4. 2D Weierstrass function $H = 0.2$. Left, projection on the 2D plane (x_0, x_1). Right, parameters σ_0 and σ_1. Points in red belong to generations 0 to 10, points in yellow to 90 to 100.

Figure 5 gives the same view as figure 3 but for a 2D Weierstrass function of dimension 0.9. This function is much more regular than the previous one. Visually, it seems obvious that the population is able to converge more rapidly.

Figure 6 gives an overview of the visualisation window for a dimension 10 space. Once again, as the function is more regular, the population seems to converge rapidly, even if it uses the same parameters as for the Weierstrass functions, in a search space with higher dimension.

5 Analysing an evolved population

GraphDice can also be used a more conventional way, in order to visualise the final population of an evolutionary algorithm. In the sequel, we present the analysis that can be made on two real life problems. The first one is a classical steady-state genetic algorithm, for which visual inspection of the evolved population allows identifying a linear dependency between the variables. The second one is an example of visualisation for a cooperative-coevolution algorithm, the fly algorithm, for which a major part of the evolved population represents the searched solution.

VIII

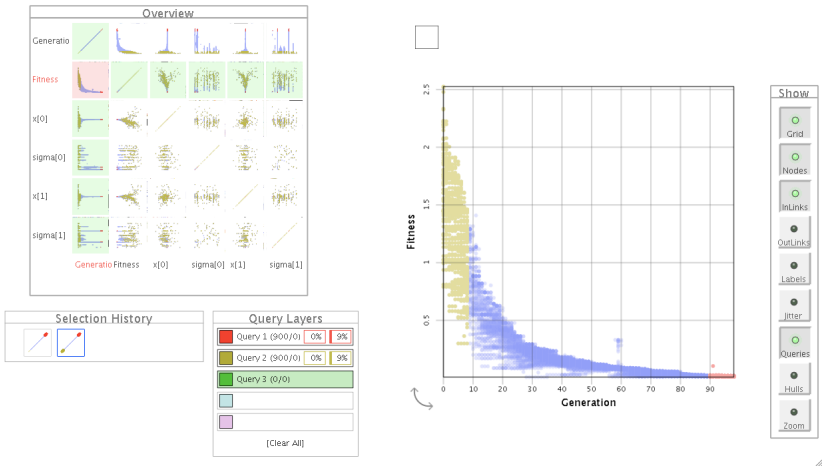


Fig. 5. 2D Weierstrass function of Hölder exponent 0.9. Yellow points correspond to the first generations, red points to the last generations.

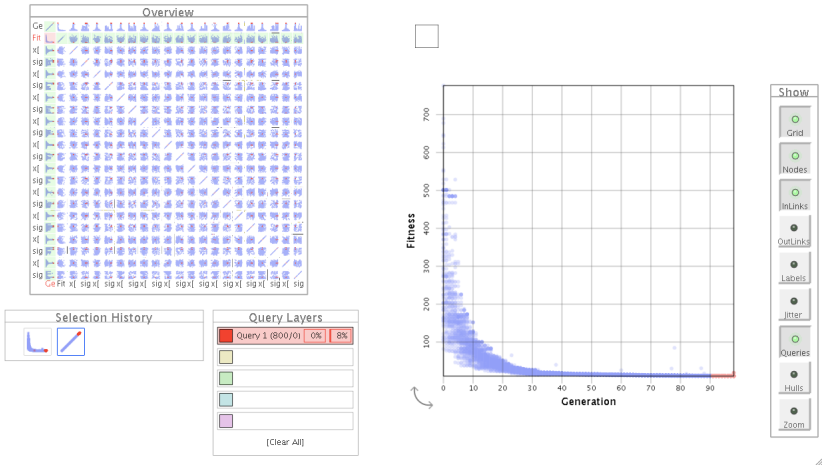


Fig. 6. Rosenbrock function in dimension 10. Red points correspond to the 10 last generations.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

5.1 Classical optimisation in R^2

The optimisation problem considered here is related to modelling the behaviour of an emulsion of milk, proteins and fat. A model of whey protein[9] and casein micelles adsorbtion on fat droplets has been built, based on geometrical reasoning[9]. The resulting differential equations depend on two unknown parameters k_{cas} and k_{wp}. An evolutionary algorithm has been used to learn these parameters using experimental data. The final population has been visualised using GraphDice.

A linear dependency between k_{cas} and k_{wp} has been made obvious on Figure 7: best fitted points of the final population are centred on the line (red points); when the fitness decreases, the dependency becomes less strict (yellow and green points) and the corresponding points are distributed around the diagonal line. It seems thus more convenient to deal with a one dimensional problem, i.e. to find the best ratio k_{cas}/k_{wp}. This experimental evidence sheds a new light on the *a priori* geometrical model, which needs now to be refined.

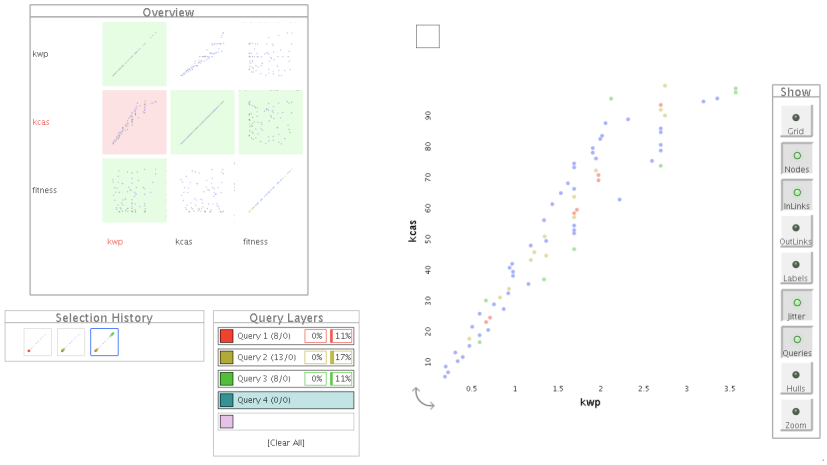


Fig. 7. Red points correspond to the 11% best fitness individuals, yellow points, to the 17% "next" best points, and green points, to the 11% worst fitness points. Queries have been interactively sculpted on the fitness versus fitness view. On the $kcas$ versus kwp view, a linear dependence between the two parameters has been made obvious.

5.2 Cooperative Coevolution of a set of 3D points

In the computer vision domain, Cooperative Coevolution algorithms (CCEAs) have been applied to stereovision, to produce the fly algorithm[17]. This algorithm evolves a

[9] Whey is left over when milk coagulates and contains everything that is soluble from milk.

population of 3-D points, the flies, so that the population matches the shapes of the objects on the scene. It is a cooperative coevolution in the sense that the searched solution is represented by the whole population rather than by the single best individual.

An individual of the population, a “fly”, is defined as a 3-D point with coordinates (x,y,z). If the fly is on the surface of an opaque object, then the corresponding pixels in the two images will normally have highly similar neighbourhoods. Conversely, if the fly is not on the surface of an object, their close neighbourhoods will usually be poorly correlated. The fitness function exploits this property and evaluates the degree of similarity of the pixel neighbourhoods of the projections of the fly, giving higher fitness values to those probably lying on objects surfaces.

GraphDice allows visualising a population of flies, and rapidly provides various viewpoints on the evolved data set (figures 9 and 10), in addition to *ad hoc* visualisations (figure 8)

Fig. 8. Visualisation of results by reprojection on the stereo pair: flies are in pink.

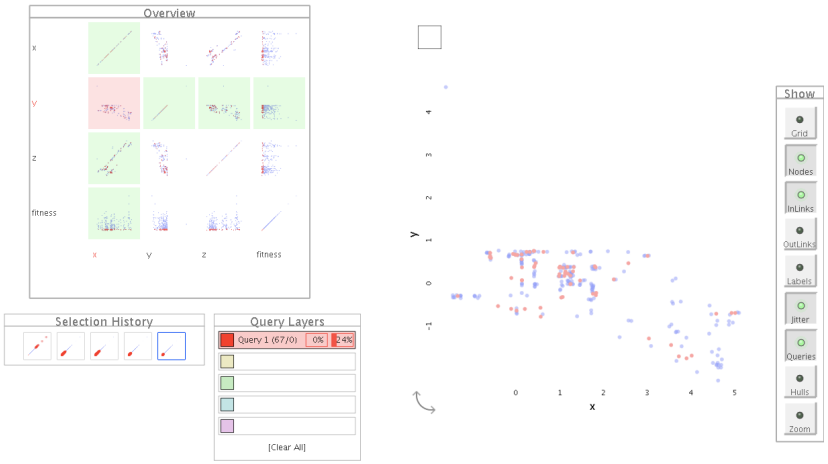


Fig. 9. Front view of the flies cloud: furniture structures are visible. Red points are the best fitted flies.

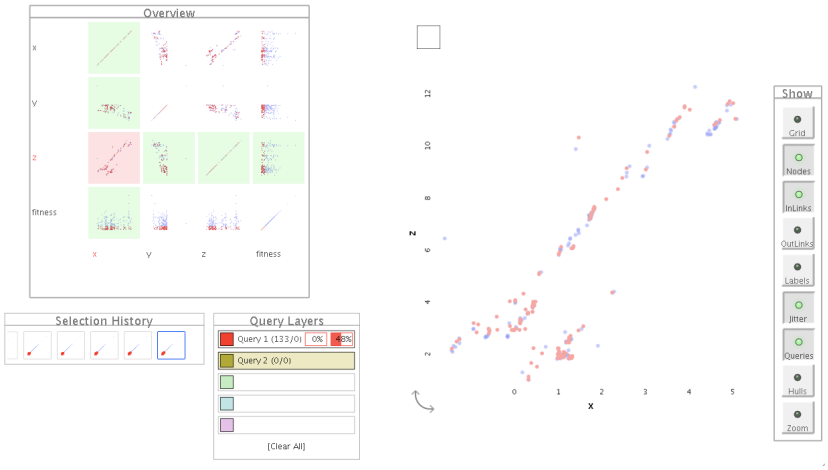


Fig. 10. View from above: the direction of the wall is visible.

6 Conclusion and future work

We have shown above how GraphDice allows fast visual inspection of raw EA data. For instance it is easy to visualise the algorithm's exploration capability and population diversity, and to localise possible visually obvious dependencies between variables.

Some features of GraphDice are particularly useful to make it a good candidate for a generic answer to the issues identified in section 1: The visualisation scheme is simple and easy to use (2D scatterplots organised in a matrix). Interactions and graphic queries allow a fast navigation in the datased. Data exchange is performed via simple csv files.

Our intention is now to adapt this general purpose visualisation tool to the specific needs of EA analysis. The following issues will guide future developments of a GraphDice version adapted to EAs:

- Tests have been performed on relatively small data sets (up to 100x100 individuals in 10 dimensional space). The scalability issue will be tested more extensively.
- Various usual statistics (per generation, per fitness level) and query-based statistics will be implemented, including comparison of distributions (p-values).
- On-line visualisation issues will also be considered.

References

1. Mark A. Bedau, Shareen Joshi, and Benjamin Lillie. Visualizing waves of evolutionary activity of alleles. In *Proceedings of the 1999 GECCO Workshop on Evolutionary Computation Visualization*, pages 96–98, 1999.
2. Anastasia Bezerianos, Fanny Chevalier, Pierre Dragicevic, Niklas Elmqvist, and Jean-Daniel Fekete. Graphdice: A system for exploring multivariate social networks. *Computer Graphics Forum (Proc. EuroVis 2010)*, 29(3):863–872, 2010.

3. Seth Bullock and Mark A. Bedau. Exploring the dynamics of adaptation with evolutionary activity plots. *Artif. Life*, 12:193–197, March 2006.
4. P. Collet, E. Lutton, M. Schoenauer, and J. Louchet. Take it EASEA. In M. Schoenauer, K. Deb, G. Rudolf, X. Yao, E. Lutton, Merelo. J.J., and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Paris, France, September 16-20 2000. Springer Verlag. LNCS 1917.
5. Trevor D. Collins. *Visualizing evolutionary computation*, pages 95–116. Springer-Verlag New York, Inc., New York, NY, USA, 2003.
6. Andreas Kerren Computer and Andreas Kerren. Eavis: A visualization tool for evolutionary algorithms. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 05*, pages 299–301, 2005.
7. Jason Daida, Adam Hilss, David Ward, and Stephen Long. Visualizing tree structures in genetic programming. *Genetic Programming and Evolvable Machines*, 6:79–110, 2005.
8. Niklas Elmqvist, Pierre Dragicevic, and Jean-Daniel Fekete. Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis 2008)*, 14(6):1141–1148, 2008.
9. J Foucquier, S Gaucel, C Surel, M Anton, C Garnier, A Riaublanc, C Baudrit, and N Perrot. Modelling the formation of the fat droplets interface during homogenisation in order to describe texture. In *ICEF, 11th International Congress on Engineering and Food, may 22-26, Athens, Greece*, 2011. http://www.icef11.org/.
10. Emma Hart and Peter Ross. Gavel - a new tool for genetic algorithm visualization. *IEEE Trans. Evolutionary Computation*, 5(4):335–348, 2001.
11. Evelyne Lutton and Jean-Daniel Fekete. Visual analytics of ea data. In *Genetic and Evolutionary Computation Conference, GECCO 2011*, 2011. July 12-16, 2011, Dublin, Ireland.
12. Z. Mach, M. Zetakova. *Intelligent Technologies - Theory and Applications.*, chapter Visualising genetic algorithms: A way through the Labyrinth of search space., pages 279–285. IOS Press, Amsterdam, 2002. P. Sincak - J. Vascak - V. Kvasnicka - J. Pospichal (eds.).
13. I.C. Parmee and J.A.R. Abraham. Supporting implicit learning via the visualisation of coga multi-objective data. In *CEC2004, Congress on Evolutionary Computation, 19-23 June*, volume 1, pages 395 – 402, 2004.
14. H Pohlheim. Geatbx - genetic and evolutionary algorithm toolbox for matlab. http://www.geatbx.com/.
15. Hartmut Pohlheim. Visualization of evolutionary algorithms - set of standard techniques and multidimensional visualization. In *GECCO'99 - Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, CA*, pages 533–540, 1999.
16. T.W Routen. Techniques for the visualisation of genetic algorithms. In *The First IEEE Conference on Evolutionary Computation*, volume II, pages 846–851, 1994.
17. Emmanuel Sapin, Jean Louchet, and Evelyne Lutton. The fly algorithm revisited: Adaptation to cmos image sensor. In *ICEC 2009, International Conference on Evolutionary Computation*, Madeira, Portugal, October, 5-7 2009.
18. W Shine and C Eick. Visualizing the evolution of genetic algorithm search processes. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation*, pages 367–372. IEEE Press, 1997.
19. W. M. Spears. An overview of multidimensional visualization techniques. In *Evolutionary Computation Visualization Workshop*, 1999. T. D. Collins, editor, Orlando, Florida, USA.
20. Zbigniew Walczak. Graph-based analysis of evolutionary algorithm. In M Klopotek, S Wierzchon, and K Trojanowski, editors, *Intelligent Information Processing and Web Mining*, volume 31 of *Advances in Soft Computing*, pages 329–338. Springer, 2005.
21. Annie S. Wu, Kenneth A. De Jong, Donald S. Burke, John J. Grefenstette, and Connie Loggia Ramsey. Visual analysis of evolutionary algorithms. In *In Proceedings of the 1999 Conference on Evolutionary Computation (CEC'99)*, pages 1419–1425. IEEE Press, 1999.

An On-line On-board Distributed Algorithm for Evolutionary Robotics

Robert-Jan Huijsman, Evert Haasdijk, and A.E. Eiben

Dept. of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
r.huijsman@student.vu.nl,e.haasdijk@vu.nl,a.e.eiben@vu.nl
http://www.cs.vu.nl/ci/

Abstract. Imagine autonomous, self-sufficient robot collectives that can adapt their controllers autonomously and self-sufficiently to learn to cope with situations unforeseen by their designers. As one step towards the realisation of this vision, we investigate on-board evolutionary algorithms that allow robot controllers to adapt without any outside supervision and while the robots perform their proper tasks. We propose an EVAG-based on-board evolutionary algorithm, where controllers are exchanged among robots that evolve simultaneously. We compare it with the $(\mu + 1)$ ON-LINE algorithm, which implements evolutionary adaptation inside a single robot. We perform simulation experiments to investigate algorithm performance and use parameter tuning to evaluate the algorithms at their *best possible* parameter settings. We find that distributed on-line on-board evolutionary algorithms that share genomes among robots such as our EVAG implementation effectively harness the pooled learning capabilities, with an increasing benefit over encapsulated approaches as the number of participating robots grows.

Keywords: evolutionary robotics, on-line evolution, distributed evolution

1 Introduction

The work presented in this paper is inspired by a vision of autonomous, self-sufficient robots and robot collectives that can cope with situations unforeseen by their designers. An essential capability of such robots is the ability to adapt –evolve, in our case– their controllers in the face of challenges they encounter in a hands-free manner, "the ability to learn control without human supervision," as Nelson *et al.* put it [13]. In a scenario where the designers cannot predict the operational circumstances of the robots (e.g, an unknown environment or one with complex dynamics), the robots need to be deployed with roughly optimised controllers and the ability to evolve their controllers autonomously, on-line and on-board.

This contrasts with the majority of evolutionary robotics research, which focusses on *off-line* evolution, where robot controllers are developed -evolved- in a separate training stage before they are deployed to tackle their tasks in earnest.

When dealing with multiple autonomous robots, one can distinguish three options to implement on-line evolution [7]:

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Encapsulated Each robot carries an isolated and self-sufficient evolutionary algorithm, maintaining a population of genotypes inside itself;

Distributed Each robot carries a single genome and the evolutionary process takes place by exchanging genetic material between robots;

Hybrid Which combines the above two approaches: each robot carries multiple genomes and shares these with its peers.

In this paper, we compare instances of each of these three schemes: the encapsulated $(\mu + 1)$ ON-LINE algorithm, the distributed Evolutionary Agents algorithm (EVAG) [9] and a hybrid extension of EVAG.

One of EVAG's distinguishing features is that it employs the newscast algorithm [8] to exchange genomes between peers (in our case: robots) and to maintain an overlay network for peer-to-peer (robot-to-robot) communication. Newscast-based EVAG has proved very effective for networks of hundreds or even thousands of peers, but in the case of swarm robotics, network sizes are likely to be much smaller. Therefore, it makes sense to compare the efficacy of newscast-based EVAG with a panmictic variant where the overlay network is fully connected.

It is well known that the performance of evolutionary algorithms to a large extent depends on their parameter values [11]. To evaluate the algorithms at their *best possible* parameter setting, we tune the algorithm parameters with REVAC, an evolutionary tuning algorithm specifically designed for use with evolutionary algorithms [10].

Summarising, the main question we address in this paper is: how do the three algorithms compare in terms of performance and can we identify circumstances in which to prefer one of the three schemes over the others? Secondly, we investigate how EVAG's newscast population structure influences its performance compared to a panmictic population structure. Thirdly, we briefly consider the sensitivity of the algorithms to parameter settings.

2 Related work

The concept of on-board, on-line algorithms for evolutionary robotics was discussed as early as 1995 in [15], with later research focussed on the 'life-long learning' properties of such a system [20].

A distributed approach to on-line, on-board algorithms was first investigated as 'embodied evolution' in [21], where robots exchange single genes at a rate proportional to their fitness with other robots that evolve in parallel. Other work on on-line evolution of robot controllers is presented in [4] that describes the evolution of controllers for activating hard-coded behaviours for feeding and mating. In [1], Bianco and Nolfi experiment with open-ended evolution for robot swarms with self-assembling capabilities and report results indicating successful evolution of survival methods and the emergence of multi-robot individuals with co-ordinated movement and co-adapted body shapes.

The $(\mu + 1)$ ON-LINE algorithm – this paper's exemplar for the encapsulated approach – has been extensively described in [7] and [2], where it was shown to be capable of evolving controllers for a number of tasks such as obstacle avoidance, phototaxis and patrolling. [5] uses encapsulated evolution to evolve spiking circuits for a fast forward

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

task. Encapsulated on-line evolution as a means for continuous adaptation by using genetic programming is suggested in [16].

The distributed approach to on-line evolutionary robotics has a clear analogy with the field of parallel evolutionary algorithms, in particular to the fine-grained approach, where each individual in the population has a processor of its own. The primary distinguishing factor among fine-grained parallel algorithms is their population structure, with small-world graphs proving competitive with panmictic layouts [6].

The hybrid scheme can be implemented as what in parallel evolutionary algorithms is known as the island model: the population is split into several separately evolving sub-populations (the islands), that occasionally exchange genomes. This approach is used in [19] and [4]. A variant where the robots share a common hall of fame is implemented in [12]. As will become apparent, we take a slightly different approach where genome exchange between sub-populations is the norm.

3 Algorithms

Autonomous on-line adaptation poses a number of requirements that regular evolutionary algorithms don't necessarily have to contend with. We take a closer look at two especially relevant considerations.

To begin with, fitness must be evaluated *in vivo*, i.e., the quality of any given controller must be determined by actually using that controller in a robot as it goes about its tasks. Such real-life, real-time fitness evaluations are inevitably very noisy because the initial conditions for the genomes under evaluation vary considerably. Whatever the details of the evolutionary mechanism, different controllers will be evaluated under different circumstances; for instance, the nth controller will start at the final location of the $(n-1)$th one. This leads to very dissimilar evaluation conditions and ultimately to very noisy fitness evaluations. To address this issue, the algorithms we investigate here implement re-evaluation: whenever a new evaluation period commences, the robot can choose (with a probability ρ) not to generate a new individual but instead re-evaluate an existing individual to refine the fitness assessment and so combat noise.

The second issue specific to on-line evolution is that, in contrast to typical applications of evolutionary algorithms, the best performing individual is not the most important factor when applying on-line adaptation. Remember that controllers evolve as the robots go about their tasks; if a robot continually evaluates poor controllers, that robot's *actual* performance will be inadequate, no matter how good the best known individuals as archived in the population. Therefore, the evolutionary algorithm must converge rapidly to a good solution (even if it is not the best) and search prudently: it must display a more or less stable but improving level of performance throughout the continuing search.

3.1 $(\mu + 1)$ ON-LINE

The $(\mu + 1)$ ON-LINE algorithm is based on the classical $(\mu + 1)$ evolutionary strategy [17] with modifications to handle noisy fitness evaluations and promote rapid convergence. It maintains a population of μ individuals within each robot and these are

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

evaluated in a time-sharing scheme, using an individual's phenotype as the robot's controller for a specified number of time units. A much more detailed description of $(\mu+1)$ ON-LINE is given in [7].

3.2 EVAG

EVAG was originally presented in [9] as a peer-to-peer evolutionary algorithm for parallel tackling of computationally expensive problems, with the ability to harness a large number of processors effectively. The analogies between parallel evolutionary algorithms and a swarm of robots adapting to their environment and tasks in parallel make EVAG a suitable candidate for an on-board, on-line distributed evolutionary algorithm for evolutionary robotics.

The basic structure of EVAG is straightforward and similar to a $1 + 1$ evolution strategy: each peer (robot) maintains a record of the best solution evaluated by that peer up until that point – the champion. For every new evaluation a new candidate is generated, using crossover and mutation; if the candidate outperforms the current champion it replaces the champion.

The basic definition of EVAG leaves many decisions open to the implementer, such as the choice of recombination and mutation operators and the details of parent selection (other than that it should select from peers' champions). Because we are interested in the effects of the distributed nature of the algorithm rather than those due to, say, different recombination schemes, we have chosen our evolutionary operators to match the $(\mu + 1)$ ON-LINE algorithm. As a result, the only difference between EVAG and $(\mu + 1)$ ON-LINE (with $\mu = 1$) lies in the exchange of genomes between robots and using a cache of received genomes rather than only a locally maintained population when selecting parents.

In this light, the extension of regular EVAG to a hybrid form is a straightforward one: rather than maintaining only a single champion on-board, the robots now maintain a population of μ individuals locally. With $\mu = 1$, this implements the distributed scheme, with $\mu > 1$, it becomes a hybrid implementation. With the cache of received genomes disabled, it boils down to $(\mu + 1)$ ON-LINE, our encapsulated algorithm. The pseudo code in algorithm 1 illustrates the overlap between these three implementations.

EVAG normally uses newscast [8] to exchange solutions efficiently while maintaining a low number of links between peers: each robot locally maintains a cache of recently received genomes. Periodically, each robot randomly takes one of the genomes in its cache and contacts the robot at which that genome originated. These two robots then exchange the contents of their cache. When needed, parents are selected from the union of this cache and the local champion using binary tournament selection. Because [8] showed that with this update scheme, picking a genome randomly from the cache of received genomes is all but equivalent to picking one randomly from the entire population, this assures that the binary tournament takes place as if the contestants were randomly drawn from the combined population across all robots.

Earlier research showed very promising results for EVAG [9], but these results were obtained using thousands of nodes, while evolutionary robotics generally takes place with group sizes of no more than a few dozen robots. To investigate if EVAG's newscast overlay network remains efficient in these smaller populations we evaluate not only the

```
for i ← 1 to μ do                                          // Initialisation
    population[i] ← CreateRandomGenome();
    population[i].σ ← σ_initial; // Mutation step size, updated cf. [2]
    population[i].Fitness ← RunAndEvaluate(population[i]);
end
for ever do                                          // Continuous adaptation
    if random() < ρ then // Don't create offspring, but re-evaluate
    selected individual
        Evaluatee ← BinaryTournament(population);
        Evaluatee.Fitness ← (Evaluatee.Fitness + RunAndEvaluate(Evaluatee)) / 2;
        // Combine re-evaluation results through exponential
        moving average
    else    // Create offspring and evaluate that as challenger
        ParentA ← BinaryTournament(pool of possible parents);
        ParentB ← BinaryTournament(pool of possible parents - parentA);
        if random() < crossoverRate then
            Challenger ← AveragingCrossover(ParentA, ParentB);
        else
            Challenger ← ParentA;
        end
        if random() < mutationRate then
            Mutate(Challenger);        // Gaussian mutation from N(0, σ)
        end
        Challenger.Fitness ← RunAndEvaluate(Challenger);
        if Challenger.Fitness > population[μ].Fitness then      // Replace last
        (i.e. worst) individual in population w. elitism
            population[μ] ← Challenger;
            population[μ].Fitness ← Challenger.Fitness;
        end
    end
    Sort(population);
end
```

Algorithm 1: The on-line evolutionary algorithm. For $(\mu + 1)$ ON-LINE, the pool of possible parents is the on-board population of size μ; for both regular and hybrid EVAG, it is the union of individuals received from the robot's peers through newscast and the on-board population (with $\mu = 1$ for standard EVAG).

standard newscast-based EVAG, but also an EVAG variant that uses a panmictic population structure where a robot's choice of genomes for the binary tournament parent selection is truly uniform random from the entire population across all robots. This variant of EVAG requires full connectivity among peers (robots) and is therefore not suitable for use in truly large-scale applications. However, evaluation of the panmictic structure compared to that of newscast is interesting, since it allows us to determine the performance penalty of using of a peer-to-peer approach.

4 Experiments

To investigate the performance of the algorithms and their variants we conduct experiments with simulated e-pucks in the ROBOROBO[1] environment. The experiments have the robots running their own autonomous instance of $(\mu + 1)$ ON-LINE, EVAG or its hybrid extension EVAG, governing the weights of a straightforward perceptron neural net controller with hyperbolic tangent activation function. The neural net has 9 input nodes (8 sensors and a bias), no hidden nodes and 2 output nodes (the left and right motor values for the differential drive), giving a total of 18 weights. To evolve these 18 weights, the evolutionary algorithm uses the obvious representation of real-valued vectors of length 18 for the genomes.

The robots' task is movement with obstacle avoidance: they have to learn to move around in a constrained arena with numerous obstacles as fast as possible while avoiding the obstacles. The robots are positioned in an arena with a small loop and varied non-looping corridors (see Fig. 1). The fitness function we use has been adapted from [14]; it favours robots that are fast and go straight ahead. Fitness is calculated as follows:

$$f = \sum_{t=0}^{\tau} (v_t \cdot (1 - v_r)) \quad (1)$$

where v_t and v_r are the translational and the rotational speed, respectively. v_t is normalised between -1 (full speed reverse) and 1 (full speed forward), v_r between 0 (movement in a straight line) and 1 (maximum rotation). In our simulations, whenever a robot touches an obstacle, $v_t = 0$, so the fitness increment for time-steps where the robot is in collision is 0. A good controller will turn only when necessary to avoid collisions and try to find paths that allow it to run in a straight line for as long as possible.

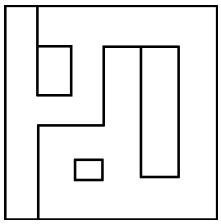


Fig. 1. The arena

We run our simulations with each robot in an arena of its own so that they can't get in each others' way, although the robots obviously can communicate across arena instances. The reasons for this are twofold: firstly, eliminating physical interaction between the robots ensures that a robot's performance is due to its own actions rather than that of others around it; this allows us a clearer view of the effects of genome exchange. Secondly, it allows us to scale our simulations from a very small number of robots to a very large number of robots while using the exact same arenas; this guarantees that in those cases any change in performance of the robots is due to their increased group size rather than a change in environment.

We evaluate the EVAG variants with group sizes of 4, 16, 36 and 400 robots: we hypothesise that differences in group size influence the performance of distributed and hybrid algorithms due to their facilities for genome exchange. Since a larger group of robots is able to evaluate a larger number of candidate solutions simultaneously, the odds of finding a successful genome are higher. EVAG is intended to distribute these successful genomes across all robots, thus improving the performance of the entire group.

[1] http://www.lri.fr/~bredeche/roborobo/

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Because the experiments were designed so that genome exchange is the only possible interaction between robots in a group, the robots can have no physical interaction. Without physical interaction and no genome exchange the *average* performance of a group of 4 robots running $(\mu + 1)$ ON-LINE would be identical to that of a group of 400 robots doing the same. Since average performance is our only metric for success and since for $(\mu + 1)$ ON-LINE this metric is not influenced by the number of robots in the experiment, we can perform experiments for $(\mu + 1)$ ON-LINE for a group of 4 robots and use these results as a fair comparison to an experiment with EVAG using 400 robots. We can therefore safely omit the costly $(\mu+1)$ ON-LINE simulations for the group sizes 16, 36 and 400 as their performance would be the same as that of the group of 4.[2]

Rather than comparing the algorithms at identical (so far as possible) parameter settings, we compare the performance at their relative *best possible* parameter settings after tuning as described in Sec 4.1. Note that, where applicable, values of μ may vary from one experiment to the next: this does not imply that the number of evaluations is influenced as the robots have a fixed amount of (simulated) wall-clock time in which to learn the task.

4.1 Evaluation with parameter tuning

It is a well-known fact that the performance of an evolutionary algorithm is greatly determined by its parameter values. Despite this, many publications in the field of evolutionary computing evaluate their algorithms using fairly arbitrary parameter settings, based on ad hoc choices, conventions, or a limited comparison of different parameter values. This approach can easily lead to misleading comparisons, where method A is tested with very good settings and method B based on poor ones. An recommendable alternative is the use of *automated parameter tuning*, where a tuning algorithm is used to optimize the parameter values and one compares the best variants of the evolutionary algorithms in question [3]. This approach helps prevent misleading comparions.

In our experiments with 4, 16 and 36 robots we evaluate the performance of the algorithms by performing parameter tuning for a fixed length of time and comparing the results. We use the MOBAT toolkit[3] to automate our tuning process. MOBAT is based on REVAC [10], which has been shown to be an efficient algorithm for parameter tuning [18]. For every combination of robot group size and algorithm MOBAT evaluates 400 parameter settings; each parameter setting is tested 25 times to allow statistically significant comparisons. Unless otherwise specified, performance comparisons were made with the best-performing parameter setting that was found for each of the algorithms-group size combinations.

Due to the computational requirements of tuning the parameters of very large simulations it was infeasible to tune parameters for our experiments with a group size of 400 robots. Instead, these experiments were performed at the parameter settings found for the 36 robots.

Each algorithm variant has its own parameters that need tuning. These parameters and the range within which they were tuned are listed in Table 1.

[2] Source code and scripts to repeat our experiments can be found at `http://www.few.vu.nl/~ehaasdi/papers/EA-2011-EvAg`.

[3] `http://sourceforge.net/projects/mobat/`

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Parameter	*Tunable range*
τ (Evaluation time steps per candidate)	300 – 600
μ (Population size – for encapsulated and hybrid schemes)	3 – 15
$\sigma_{initial}$ (Initial mutation step size)	0.1 – 10.0
ρ (Re-evaluation rate)	0.0 – 1.0
Crossover rate	0.0 – 1.0
Mutation rate	0.0 – 1.0
Newscast item TTL (for newscast-based EVAG variants)	3 – 20
Newscast cache size (for newscast-based EVAG variants)	2 – group size

Table 1. The parameters as tuned by REVAC.

5 Results and Discussion

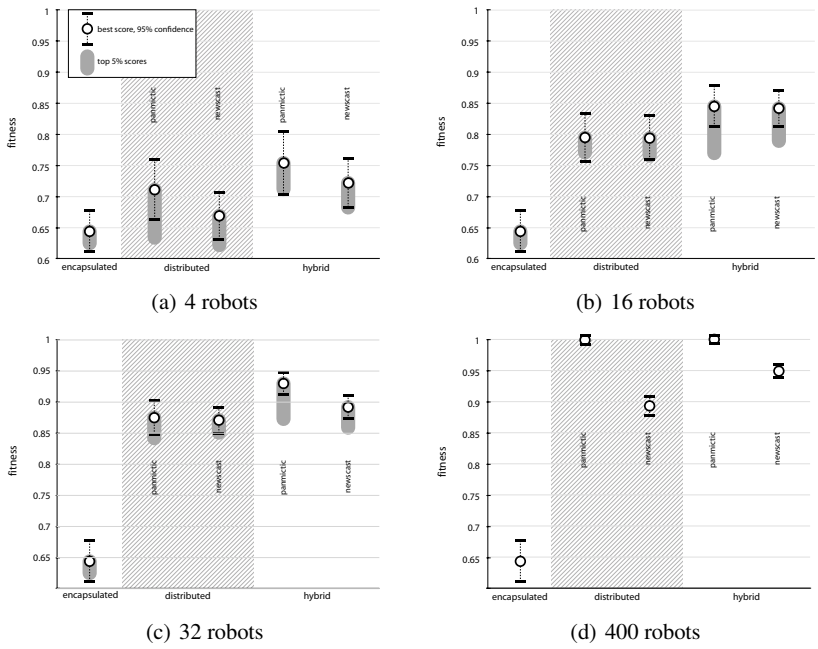
Fig. 2 shows the results for the experiments for group sizes 4, 16, 36 and 400. Each graph shows the results for $(\mu + 1)$ ON-LINE (*encapsulated*), for EVAG (*distributed*) and for EVAG's *hybrid* extension. The latter two have results for the panmictic as well as the newscast variant. The white circles indicate the average performance (over 25 repeats) of the best parameter vector for that particular algorithm variant and the whiskers extend to the 95% confidence interval using a t-test. To investigate how sensitive the algorithm variants are to the choice of parameter settings, we also show the performance variation in the top 5% of parameter vectors, indicated by the grey ovals. The results are normalised so that the highest performance attained overall is 1. Note that because –as discussed above– we only ran $(\mu + 1)$ ON-LINE experiments with group size 4, the data for $(\mu + 1)$ ON-LINE are the same in all four graphs.

The performances shown are always the average performance of the entire group of robots in an experiment, not just the performance of the best robot: we are interested in developing an algorithm that performs well for *all* robots, rather than an algorithm that has a very high peak performance in one robot but does not succeed in attaining good performance for the entire group. Performances are compared using a t-test with $\alpha = 0.05$.

In the scenario with four robots (Fig. 2(a)) the hybrid algorithm significantly outperforms the encapsulated scheme, but it is not significantly better than the distributed scheme. The differences between the distributed and the encapsulated scheme are not significant. The same goes for the differences between the panmictic and the newscast variants. All four EVAG variants show a relatively large variation in the top-5% performances, indicating that they are more sensitive to the quality of parameter settings than is $(\mu + 1)$ ON-LINE.

It is surprising that the distributed scheme matches (even improves, although not significantly) performance with the encapsulated scheme when we consider that four robots running EVAG have access to only four (shared) genotypes, compared to the 12 (isolated) genotypes that are stored by each of the robots running $(\mu+1)$ ON-LINE (with μ tuned to 12).

With group size 16 (Fig. 2(b)), EVAG starts to come into its own: both the distributed and the hybrid schemes outperform $(\mu + 1)$ ON-LINE significantly, although the differences among the EVAG variants are not significant at 95%. Moreover, the distributed variants now perform almost as consistently as their encapsulated counterpart:



(a) 4 robots (b) 16 robots

(c) 32 robots (d) 400 robots

Fig. 2. Performance plots for various group sizes: performance is averaged over 25 repeats. Note that there was no tuning for group size 400 and hence no top 5% parameter vectors.

both variation in scores for a single parameter setting and the variation of scores in the top-5% are at the level of $(\mu + 1)$ ON-LINE, indicating that the distributed algorithm becomes less sensitive to sub-optimal parameter settings as the number of robots participating in the evolution grows.

With the increase in group size from 16 to 36 (Fig. 2(c)) EVAG again shows a significant performance increase and the confidence intervals and range of top-5% values are further reduced. For the first time we see a significant difference between the EVAG variants, with the panmictic hybrid approach performing significantly better than the alternatives.

Finally, Fig. 2(d) shows the results for 400 robots. The computational requirements of so large a simulation make parameter tuning infeasible; instead we investigate the performance of the algorithms at the best parameter settings found for the 36-robot scenario. The plot shows a large jump in performance for the panmictic variants, a respectable increase for the hybrid newscast variant, but little difference for the distributed newscast implementation. For all EVAG variants the confidence interval of the scores has shrunk considerably, indicating that EVAG continues to become more reliable as the robot group size increases.

The results show that, as the group size increases, there is an increasing benefit to using an algorithm such as (the hybrid extension of) EVAG rather than a purely encapsulated algorithm such as $(\mu + 1)$ ON-LINE. In particular, the hybrid scheme consistently

reaches the highest performance (although the difference with the distributed scheme is rarely shown to be significant at 95%). EVAG's panmictic variants perform consistently better than its newscast-based version, but the difference is only significant at the largest group sizes we consider. In truly large-scale environments it would be infeasible to have a panmictic population structure; unfortunately it is especially in the large-scale 400-robot scenario that the newscast-based EVAG lags behind its panmictic counterpart.

One possible explanation for the lower performance of the newscast-based EVAG is that it has two extra parameters to tune (newscast cache size and the news items' TTL), making it more difficult for REVAC to find optimal parameter settings within the 400 attempts that we allowed it. However, this does not explain the large performance gap in the 400-robot scenario, where the same parameter settings as in the 36-robot scenario were used. One suspect is the 'newscast cache size' parameter, of which an interesting trend can be seen when looking at the progression of the value across the different scenarios: as the number of robots grows, so does the value of the cache size parameter. Earlier research on newscast suggests that a cache size of 10 should be sufficient for very large-scale applications [8], but nevertheless REVAC favours higher numbers, approximately in the range of $\frac{3}{4}$th of the number of robots. To see if the setting of a cache size of 27 is sub-optimal for the 400-robot scenario we have investigated if increasing the cache size to 300 leads to an improved performance; this turned out not to be the case, with both cache sizes performing at the same level. This indicates that a lack of cache space is not to blame for the gap in performance between newscast-based EVAG and its panmictic counterpart and its exact cause remains unknown.

6 Conclusion

In this paper we have compared the $(\mu + 1)$ ON-LINE on-board encapsulated algorithm to a distributed and a hybrid implementation of EVAG for on-line, on-board evolution of robot controllers. We have performed simulation experiments to investigate the performance of these algorithms, using automated parameter tuning to evaluate each algorithm at its *best possible* parameter setting.

Comparing the algorithms in terms of performance, EVAG performs consistently better than $(\mu + 1)$ ON-LINE, with the effect being especially prominent when the number of robots participating in the scenario is large. Even at the smallest group size we considered, the distributed scheme is competitive, despite having a population of only four individuals.

To estimate the sensitivity to sub-optimal parameter settings we have observed the variation in performance in the top-5% of parameter vectors. We have seen that $(\mu + 1)$ ON-LINE is quite stable, with the top-5% close to the best performance. In both the distributed and the hybrid variant, EVAG's performance is quite unstable when the group of robots is small, but becomes increasingly reliable as the group size increases.

For large numbers of robots, the newscast population structure has a small negative effect on the performance of the EVAG variants when compared to a panmictic population structure. However, in a large-scale scenario it may be infeasible to maintain a panmictic population structure. In those scenarios the small performance loss when

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

using a newscast-based population structure may well be outweighed by the practical advantages of being able to implement the algorithm at all.

Although we have only performed experiments with a single and quite straightforward task, we conclude that both the distributed and hybrid approaches to on-board online evolutionary algorithms in evolutionary robotics are feasible and provide a promising direction for research in this field.

The hybrid scheme can be preferred over the encapsulated and the distributed case because it efficiently harnesses the opportunities of parallelising the adaptation process over multiple robots while performing well even for small numbers of robots. Although the panmictic variant does outperform the newscast-based implementation for very large numbers of robots, we do not know if or how tuning specifically for 400 robots would have influenced the apparent performance difference between newscast and panmixia. We would still prefer the latter because of its inherent scalability and robustness.

Future research should confirm our findings in different scenarios. Additionally, there is research to be done regarding the study of which parameters are influential and why certain parameter settings are more effective than others.

Acknowledgements This work was made possible by the European Union FET Proactive Intiative: Pervasive Adaptation funding the SYMBRION project under grant agreement 216342. The authors would like to thank Selmar Smit and our partners in the SYMBRION consortium for many inspirational discussions on the topics presented here.

References

1. Raffaele Bianco and Stefano Nolfi. Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce. *Connection Science*, 4:227–248, 2004.
2. A. E. Eiben, Giorgos Karafotias, and Evert Haasdijk. Self-adaptive mutation in on-line, on-board evolutionary robotics. In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW 2010)*, pages 147–152. IEEE Press, Piscataway, NJ, September 2010.
3. A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011. to appear.
4. S. Elfwing, E. Uchibe, K. Doya, and H.I. Christensen. Biologically inspired embodied evolution of survival. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation*, volume 3, pages 2210–2216, Edinburgh, UK, 2-5 September 2005. IEEE Press.
5. Dario Floreano, Nicolas Schoeni, Gilles Caprari, and Jesper Blynel. Evolutionary bits'n'spikes. In Russell K. Standish, Mark A. Bedau, and Hussein A. Abbass, editors, *Artificial Life VIII : Proceedings of the eighth International Conference on Artificial Life*, pages 335–344, Cambridge, MA, USA, 2002. MIT Press.
6. Mario Giacobini, Mike Preuss, and Marco Tomassini. Effects of scale-free and small-world topologies on binary coded self-adaptive CEA. In Jens Gottlieb and Günther R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP˜2006*, volume 3906 of *LNCS*, pages 85–96, Budapest, April 2006. Springer Verlag.

7. Evert Haasdijk, A. E. Eiben, and Giorgos Karafotias. On-line evolution of robot controllers by an encapsulated evolution strategy. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, Barcelona, Spain, 2010. IEEE Computational Intelligence Society, IEEE Press.
8. Márk Jelasity and Maarten van Steen. Large-scale newscast computing on the internet. Technical report, Vrije Universiteit Amsterdam, 2002.
9. J. L. Laredo, A. E. Eiben, M. van Steen, and J. J. Merelo. Evag: a scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11:227–246, June 2010.
10. V Nannen and A E Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. *Proc of IJCAI 2007*, pages 975–980, 2007.
11. Volker Nannen, S. K. Smit, and A. E. Eiben. Costs and benefits of tuning parameters of evolutionary algorithms. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X*, pages 528–538, Berlin, Heidelberg, 2008. Springer-Verlag.
12. Ulrich Nehmzow. Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. In C.G. Prince, Y. Demiris, Y. Marom, H. Kozima, and C. Balkenius, editors, *Proceedings of The Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, number 94 in Lund University Cognitive Studies, Edinburgh, UK, August 2002. LUCS.
13. Andrew L. Nelson, Gregory J. Barlow, and Lefteris Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345 – 370, 2009.
14. Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA, 2000.
15. Peter Nordin and Wolfgang Banzhaf. Genetic programming controlling a miniature robot. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 61–67. AAAI, 1995.
16. Peter Nordin and Wolfgang Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5:107–140, 1997.
17. Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.
18. S. K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 399–406, Piscataway, NJ, USA, 2009. IEEE Press.
19. Yukiya Usui and Takaya Arita. Situated and embodied evolution in collective evolutionary robotics. In *Proceedings of the 8th International Symposium on Artificial Life and Robotics*, pages 212–215, 2003.
20. Joanne H. Walker, Simon M. Garrett, and Myra S. Wilson. The balance between initial training and lifelong adaptation in evolving robot controllers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(2):423–432, 2006.
21. Richard A. Watson, Sevan G. Ficici, and Jordan B. Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1 – 18, 2002.

Improving Performance via Population Growth and Local Search: The Case of the Artificial Bee Colony Algorithm

Doğan Aydın[1], Tianjun Liao[2], Marco A. Montes de Oca[3], and Thomas Stützle[2]

[1] Dept. of Computer Engineering, Dumlupınar University, 43030 Kütahya, Turkey
dogan.aydin@dpu.edu.tr
[2] IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
{tliao,stuetzle}@ulb.ac.be
[3] Dept. of Mathematical Sciences, University of Delaware, Newark, DE, USA
mmontes@math.udel.edu

Abstract. We modify an artificial bee colony algorithm as follows: we make the population size grow over time and apply local search on strategically selected solutions. The modified algorithm obtains very good results on a set of large-scale continuous optimization benchmark problems. This is not the first time we see that the two aforementioned modifications make an initially non-competitive algorithm obtain state-of-the-art results. In previous work, we have shown that the same modifications substantially improve the performance of particle swarm optimization and ant colony optimization algorithms. Altogether, these results suggest that population growth coupled with local search help obtain high-quality results.

1 Introduction

Thinking of optimization algorithms as being composed of components has changed the way high-performance optimization algorithms are designed. Research based on algorithm components and not on specific implementations of optimization algorithms promises to lead to breakthroughs because a more systematic approach can be taken in order to explore the space of algorithm designs. In previous work, we have taken a component-based approach to the design of optimization algorithms for continuous optimization and have obtained promising results. In particular, we integrated a growing population size and a local search components into a particle swarm optimization (PSO) algorithm and an ant colony optimization algorithm for continuous domains. The resulting algorithms, called IPSOLS [3,4] in the case of PSO, and IACO$_{\mathbb{R}}$-LS [5] in the case of ACO$_{\mathbb{R}}$, exhibited a significantly better performance than the original algorithms. In fact, their performance was competitive with state-of-the-art algorithms in the special issue of the Soft Computing journal on large-scale continuous optimization [6]. (Throughout the rest of the paper, we will refer to this special issue as SOCO.) In this paper, we present a third case study based on the artificial

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

bee colony (ABC) algorithm [7, 8]. The goal of this case study is to test the "pure chance" hypothesis, which explains the results obtained with PSO and $ACO_{\mathbb{R}}$ as consequence of mere good luck. The results obtained in this third case study, shown in Section 3.4, suggest that the pure-chance hypothesis is false. We find that population growth together with a local search procedure are algorithmic components that make swarm intelligence algorithms for continuous optimization obtain high-quality results.

2 Related Work

The idea of increasing the size of the population in swarm intelligence algorithms for continuous optimization derives from the incremental social learning (ISL) framework [3]. The ISL framework's aim is to reduce the time needed by a swarm intelligence system to reach a desired state. In the case of swarm intelligence systems for optimization, the desired state may be associated with a solution better than or equal to a desired quality. The ISL framework achieves this goal by starting the system with a small population and increasing its size according to some addition criterion. By starting a swarm-based optimization algorithm with a smaller population than usual, the framework encourages a rapid convergence toward promising regions of the search space. Adding new solutions increases the diversity of the swarm. The new solutions are generated using information from the "experienced" swarm; thus, new solutions are not completely random.

Modifying the size of the population while an algorithm is operating is not a new idea. For example, a number of researchers in the field of evolutionary computation (EC) have proposed several schemes that increase or reduce the size of the population in order to adjust the diversification-intensification properties of the underlying optimization algorithms (see e.g. [9–11]). The ISL framework is different from most previous dynamic population sizing approaches. In most of these cases, diversification is favored during the first phases of the optimization process through a large population, and intensification toward the end by reducing its size. In contrast, the ISL framework modifies the population size in one direction only: increasing its size. A notable exception of the strategy used in most EC algorithms is the one used in the G-CMA-ES [12] algorithm. In this algorithm, the population size also increases over time. It is important to note that G-CMA-ES is considered to be a state-of-the-art algorithm for continuous optimization.

The utilization of an auxiliary local search method is a common approach to enhance the intensification properties of EC algorithms. In ISL, local search can be integrated as a means to simulate individual learning, that is, learning without any social influence.

In [3], we introduced IPSOLS, a PSO-local search hybrid algorithm with increasing population size as an instantiation of the ISL framework. In subsequent work [4], we used iterated F-Race [14], an automatic parameter tuning software, throughout the redesign process of IPSOLS. The redesigned IPSOLS algorithm [4] was benchmarked on SOCO's large-scale optimization benchmark

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

problems and compared favorably to other state-of-the-art algorithms that include differential evolution algorithms, memetic algorithms, particle swarm optimization algorithms and other types of optimization algorithms [6]. In SOCO, a differential evolution algorithm (DE) [15], G-CMA-ES, and the real coded CHC algorithm (CHC) [16] were used as reference algorithms. A second instantiation of ISL was presented in [5] in the context of $\text{ACO}_{\mathbb{R}}$. The introduced algorithm, called $\text{IACO}_{\mathbb{R}}$-LS, features the same two components: a growing solution population size and a local search procedure. $\text{IACO}_{\mathbb{R}}$-LS was also benchmarked on SOCO's benchmark problems and on the IEEE CEC 2005 functions [17]. We can say that $\text{IACO}_{\mathbb{R}}$-LS is a state-of-the-art algorithm because it obtained better results than IPSOLS and it is competitive with G-CMA-ES.

3 An Artificial Bee Colony Algorithm with Population Growth and Local Search

In this section, we present our third case study of the utilization of a growing population size and a local search procedure in a swarm intelligence algorithm for continuous optimization.

3.1 The Artificial Bee Colony Algorithm

The ABC algorithm [7, 8] is inspired by the foraging behavior of a honeybee swarm. At the initialization step of the algorithm, ABC generates a number of randomly located food sources, and it creates a number of employed and onlooker bees. The number of food sources, which is denoted by SN, is equal to the number of employed and onlooker bees. Each cycle of the algorithm consists of three successive steps. In the first step, each employed bee selects uniformly at random a food source and then modifies the location of the chosen food source according to

$$v_{i,j} = x_{i,j} + \phi_{i,j}(x_{i,j} - x_{k,j}),\ i \neq k\,, \tag{1}$$

where $k \in \{1, 2, \ldots, SN\}$, $j \in \{1, 2, \ldots, D\}$ (D is the problem's dimension), $\phi_{i,j}$ is a uniform random number between $[-1, 1]$, $x_{i,j}$ and $x_{k,j}$ is the position of the reference food source i and a randomly selected food source k in dimension j. The candidate food source created by an employed bee can be better than the reference food source. In this case, the reference food source is replaced with the candidate food source. In the second step, onlooker bees try to find new food sources near existing food sources as the employed bees do. Different from employed bees, onlooker bees randomly select a food source i with a food source selection probability p_i, which is determined as

$$p_i = \frac{\mathit{fitness}_i}{\sum_{n=1}^{SN} \mathit{fitness}_n}\,, \tag{2}$$

where $\mathit{fitness}_i$ is the *fitness* value of the food source i, which is inversely proportional to the objective value of the food source i for function minimization.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Algorithm 1 The Incremental ABC Algorithm with Local Search

```
initialization {Initialize SN number of food sources}
while termination condition is not met do
  if FailedAttempts = Failures_Max then
    Invoke local search on randomly selected food source
  else
    Invoke local search on the best food source
  end if
  Employed Bees Stage {Use x_gbest,j as the reference food source}
  Onlookers Stage
  Scout Bees Stage {Use eq. 5}
  if Food source addition criterion is met then
    Add a new food source to the environment {Use eq. 4}
    SN ← SN + 1
  end if
end while
```

Onlooker bees explore in the vicinity of good food sources. This behavior is responsible for the intensification behavior of the algorithm since information about good solutions is exploited. In the last step, a few food sources, which have not been improved during a predetermined number of iterations (controlled by a parameter *limit*), are detected and abandoned. Then, scout bees search for a new food source randomly according to

$$x_{i,j} = x_j^{min} + \varphi_{i,j}(x_j^{max} - x_j^{min}) \tag{3}$$

where $\varphi_{i,j}$ is a uniform random number between $[0, 1]$ for dimension j and food source i, and x_j^{min} and x_j^{max} are the minimum and maximum limits of the search range on dimension j, respectively. Clearly, in the ABC algorithm employed and onlooker bees intensify the algorithm's search and the scout bees diversify it [18].

3.2 Integrating Population Growth and Local Search

In this section, we describe the integration of a growing population size and a local search procedure into the ABC algorithm. The outline of the proposed algorithm, called IABC-LS, is shown in Algorithm 1. In IABC-LS, the number of food sources and, indirectly, the population size of the bee colony (that is, the number of onlooker and employed bees), is increased according to a control parameter g. Every g iterations, a new food source is added to the environment until a maximum number of food sources is reached.

IABC-LS begins with few food sources. New food sources are placed biasing their location toward the location of the best-so-far solution. This is implemented as

$$x'_{new,j} = x_{new,j} + \varphi_{new,j}(x_{gbest,j} - x_{new,j}), \tag{4}$$

where $x_{new,j}$ is the randomly generated new food source location, $x'_{new,j}$ is the updated location of the new food source, $x_{gbest,j}$ refers to best-so-far food source

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

location, and $\varphi_{new,j}$ is a number chosen uniformly at random in $[0, 1]$. A similar replacement mechanism is applied by the scout bees step in IABC-LS. The difference is a replacement factor parameter, R_{factor}, that controls how much the new food source locations will be closer to the best-so-far food source. This modified rule is:

$$x'_{new,j} = x_{gbest,j} + R_{factor}(x_{gbest,j} - x_{new,j})\,. \tag{5}$$

Another difference between the original ABC algorithm and IABC-LS is that employed bees search in the vicinity of $x_{gbest,j}$ instead of a randomly selected food source. This modification enhances the intensification behavior of the algorithm and helps it to converge quickly toward good solutions. Although intensification in the vicinity of the best-so-far solution may lead to premature convergence to local optima, the algorithm detects stagnation by using the *limit* parameter, and the scouts can discover another random food source to escape a local optimum.

IABC-LS is a hybrid algorithm that calls a local search procedure at each iteration. The best-so-far food source location is usually used as the initial solution from which the local search is called. The result of the local search replaces the best-so-far solution if there is an improvement on the initial solution. In this paper, the IABC-LS algorithm is implemented with Powell's conjugate direction set [19] (IABC-Powell) and Lin-Yu Tseng's Mtsls1 [20] (IABC-Mtsls1) as local search procedures. Both local search procedures are terminated after a maximum number of iterations, itr_{max}, or when the tolerance $FTol$ is reached. $FTol$ is the threshold value of the relative difference between two successive iterations' solutions. An adaptive step size for each local search procedure is used. The step size is set to the maximum norm ($\|\cdot\|_\infty$) of the vector that separates a randomly selected food source from the best-so-far food source.

For fighting stagnation, the local search procedures are applied to a randomly selected location if local search calls cannot improve the results after a maximum number of repeated calls, which is controlled by a parameter $Failures_{max}$. The original versions of the local search algorithms do not enforce bound constraints. To enforce bound constraints, the following penalty function is used in both local search procedures [4, 5]:

$$P(x) = fes \times \sum_{j=1}^{D} Bound(x_j), \tag{6}$$

where $Bound(x_j)$ is defined as

$$Bound(x_j) = \begin{cases} 0, & if\ x_j^{max} > x_j > x_j^{min} \\ (x_j^{min} - x_j)^2, & if\ x_j < x_j^{min} \\ (x_j^{max} - x_j)^2, & if\ x_j > x_j^{max} \end{cases} \tag{7}$$

where *fes* is the number of function evaluations that have been used so far.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Table 1: The best parameter configurations obtained through iterated F-race for ABC algorithms in 10-dimensional instances

Algorithm	SN	$limit$	SN_{max}	$growth$	R_{factor}	itr_{max}	$Failures_{max}$	$FTol$
ABC	5	98	—	—	—	—	—	—
IABC	8	96	13	8	10^{-6}	—	—	—
IABC-Mtsls1	4	13	80	6	$10^{-2.94}$	199	17	—
IABC-Powell	7	99	44	9	$10^{-0.94}$	82	9	$10^{-5.6}$

3.3 Experimental Setup

In our study, we performed three sets of experiments. First, we ran the original ABC algorithm, IABC (IABC-LS without local search) and the two instantiations of IABC-LS on the 19 SOCO benchmark functions proposed by Herrera et al. [21] for SOCO. A detailed description of the benchmark function set can be found in [21].

All experiments were conducted under the same conditions, and grouped by the dimensionality of the optimization problems ($D \in \{50, 100, 200, 500, 1000\}$). All investigated algorithms were run 25 times for each SOCO function. Each run stops when the maximum number of evaluations is achieved or the solution value is lower than 10^{-14}, which is approximated to 0. The maximum number of evaluations is $5000 \times D$.

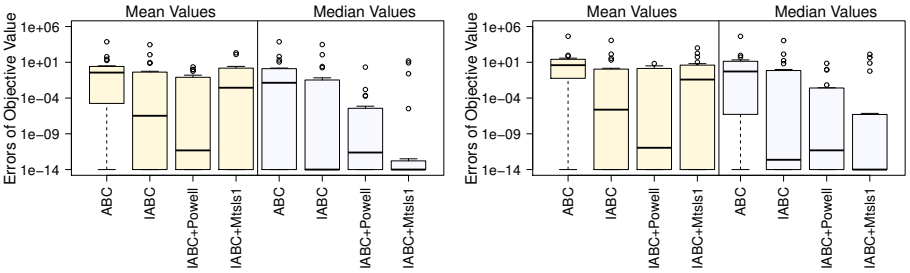
The parameters of all ABC algorithms used in this paper were tuned using iterated F-race [14]. The best parameters found for each algorithm are given in Table 1. Iterated F-race was run using the 10-dimensional versions of the 19 benchmark functions as tuning instances and the maximum tuning budget was set to 50,000 algorithm runs. The number of function evaluations used in each run is 50,000.

We compared the computational results of IABC-Powell and IABC-Mtsls1 with $\text{IACO}_{\mathbb{R}}$-LS and IPSOLS. Finally, all three IABC variants were compared with the 16 algorithms featured in SOCO special issue.

3.4 Results

The results of the first set of experiments, where we compare the ABC algorithm variants, are shown in Figure 1. The box-plots indicate the distribution of average results (left side) and the median results (right side) obtained by each algorithm on the 19 SOCO benchmark functions. (Each point for the box-plot measures the mean and median performance for 25 independent trials on each function.) In all cases, except in the comparison between IABC-Mtsls1 and ABC on the 50-dimensional functions, the mean and median results obtained by the proposed algorithms are statistically significantly better than those of the original ABC algorithm. Statistical significance was determined with Wilcoxon's test at $\alpha = 0.05$ using Holm's method for multiple test corrections. There are interesting differences in performance depending on whether we base our conclusions

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

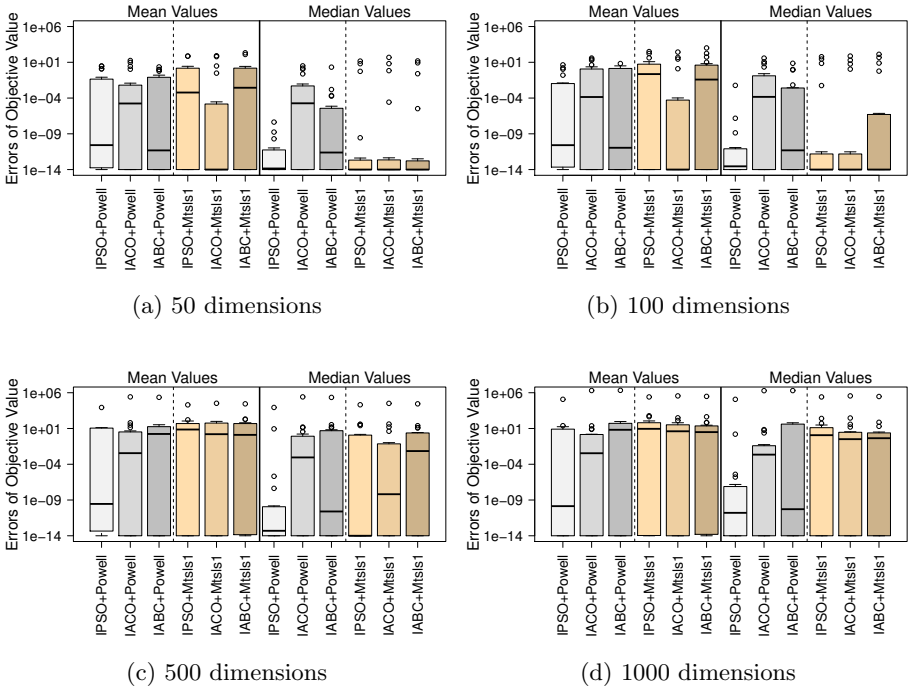


(a) 50 dimensions (b) 100 dimensions

Fig. 1: Distribution of mean and median errors of ABC, IABC, IABC-Powell and IABC-Mtsls1 on the 19 SOCO benchmark functions.

on the median or the mean performance. Based on mean errors, IABC-Powell performs better than IABC and IABC-Mtsls1. However, based on median errors (that is, for each function, we measure the median result across the 25 independent trials), IABC-Mtsls1 outperforms all others. This difference is the result of the fact that IABC-Mtsls1 shows early stagnation effects in few trials in which it obtains rather poor results. This difference also indicates that a further refinement of IABC-Mtsls1 could be interesting for future work. Notice that the results obtained with IABC are very promising and competitive with IABC-Mtsls1 and IABC-Powell for several functions. This indicates that the improved performance over ABC is not only due to the usage of a local search procedure, but that the incremental social learning mechanism and the stronger exploitation of the best found solutions are decisive to improve over ABC.

It is interesting to compare the performance of IABC and its variants with that of other algorithms that have been obtained by adopting the ISL framework to improve them. Therefore, we compared IABC-LS with IPSOLS (IPSO-Powell and IPSO-Mtsls1) and IACO$_{\mathbb{R}}$-LS (IACO$_{\mathbb{R}}$-Powell and IACO$_{\mathbb{R}}$-Mtsls1). All experiments were conducted using tuned parameter configurations based on [4, 5]. The distributions of the mean and median errors obtained on the 19 SOCO test functions for various dimensions are shown in Figure 2. Based on to the mean errors, IABC-Powell appears to be competitive with IPSOLS and IACO$_{\mathbb{R}}$-LS variants, especially when $D = 50$ and $D = 100$. It is also apparent that the performance of IABC-Powell degrades for higher dimensions. Based on median performance, IABC-Powell seems to be slightly better than IACO$_{\mathbb{R}}$-Powell. When using Mtsls1 as local search procedure, the results are different. In this case, Mtsls1 seems to work better with IACO$_{\mathbb{R}}$. Another interesting observation is that the hybrids with Mtsls1 appear to degrade in performance when compared to the hybrids with Powell's direction set method, indicating a better scaling behavior for the latter.



(a) 50 dimensions (b) 100 dimensions

(c) 500 dimensions (d) 1000 dimensions

Fig. 2: Distribution of the mean and the median errors obtained on the 19 benchmark functions

Finally, we compare all IABC variants with the 16 algorithms featured in SOCO. To test the significance of the observed differences, we again conducted pairwise Wilcoxon tests with Holm's corrections of multiple comparisons, at the 0.05 α-level. Figure 3 shows these results on the 100- and 1000-dimensional functions. A + symbol on top of a box-plot denotes a significant difference at the 0.05 level between the results obtained with the indicated algorithm and those obtained with the indicated IABC variant. If an algorithm is significantly better than an IABC variant, a − symbol is put on top of a box-plot for indicating this difference. The numbers on top of a box-plot denotes the number of optima found by the corresponding algorithm (in other words, the number of functions on which the statistic, either mean or median, is smaller then the zero threshold 10^{-14}). Figure 3 indicates that IABC, IABC-Powell and IABC-Mtsls1 significantly outperform CHC and G-CMA-ES in each case. This is noteworthy since G-CMA-ES is an acknowledged state-of-the-art algorithm for continuous optimization. When taking into account the median errors of the algorithms, IABC variants outperform CHC, G-CMA-ES, EvoPROpt, MA-SSW, RPSO-vm, and VXQR1 in almost all dimensions. The IABC variants exhibit a performance sim-

ilar to rest of the state-of-the-art algorithms with the exception of IPSO-Powell and MOS-DE.

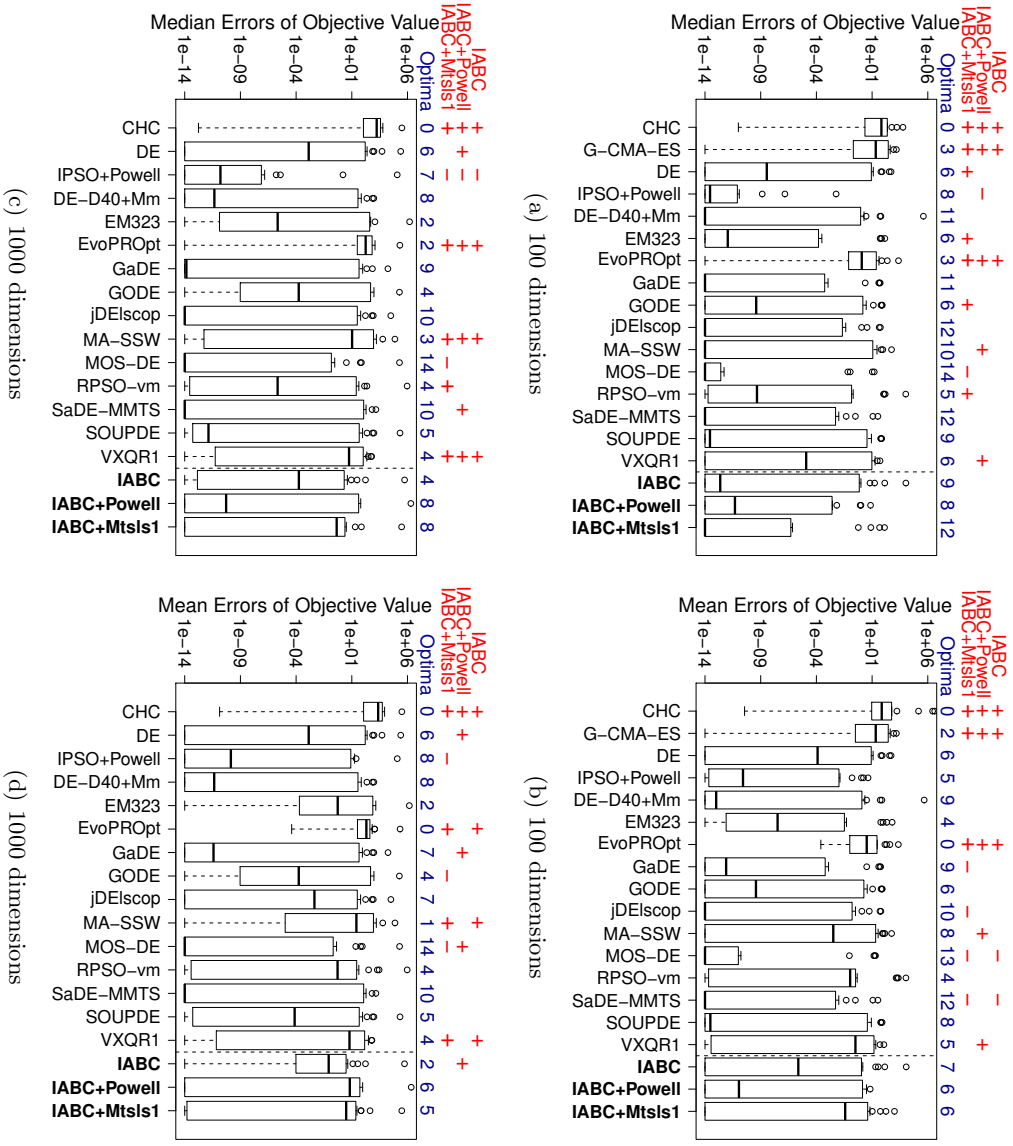
4 Discussion

In this paper, we have shown that the ISL framework with local search can enhance the performance of ABC. In earlier work we have shown the same result for two other swarm intelligence algorithms: PSO and $ACO_{\mathbb{R}}$. In swarm intelligence algorithms for optimization, individual agents can learn from each other. However, the specific knowledge-sharing mechanism used in any given algorithm can be slow. In the ISL framework, new agents can obtain knowledge directly from experienced ones. Thus, ISL allows the new agent to more directly focus the search in the vicinity of the best results. On the other hand, although local search procedures cannot always find good-enough solutions alone, local search procedures play an important role in population-based algorithms. At the same time, the behavior of a population-based algorithm affects directly the behavior of the local search procedure. For example, information from the population may be used to select restart positions and adaptively determine the step size of the local search procedure. In our case, the step size is determined by the position of the best-so-far agent and two agent positions, respectively.

Concerning the impact of the ISL framework on algorithm performance, we observed that IABC improved strongly upon the original ABC algorithm, more than what we observed when moving to the incremental $ACO_{\mathbb{R}}$ and PSO algorithms. In fact, the IABC algorithm alone was sufficient to find optimal solutions for a number of benchmark functions. Possible reasons for this strong improvements may be that (i) the bee colony metaphor has not been as explored as the metaphors behind PSO or $ACO_{\mathbb{R}}$ have; thus, improvements are easier to obtain; and (ii) ABC algorithms have a better local search type behavior for the refinement of solutions than PSO and $ACO_{\mathbb{R}}$ algorithms. A more in-depth analysis of the ABC algorithm could be an interesting direction for future research. In addition to the usage of ISL, we note that another possible reason for the observed performance improvements is the stronger focus around the best-so-far solutions, which is a feature also present in other ABC variants [22].

The performance of the hybrid method, that is, the combination of IABC with local search (but also that of $IACO_{\mathbb{R}}$ and IPSO with local search) depends significantly on the local search algorithm chosen. In fact, we observed that for "low" dimensions of 50 and 100, Mtsls1 seems to work well with IABC and $ACO_{\mathbb{R}}$, while Powell's direction set method seems to be less affected by increasing dimensionality. Since the best local search may further depend on the particular benchmark function, an adaptive choice of the local search algorithm to be used would be desirable. As example, we have tried a simple strategy in which the algorithm selects the better local search procedure after the first iteration of the algorithm, resulting in further improvements of the IABC results.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



(a) 100 dimensions (b) 100 dimensions (c) 1000 dimensions (d) 1000 dimensions

Fig. 3: Distribution of the average and the median errors obtained on the 19 benchmark functions for all featured algorithms in SOCO and IABC variants on 100- and 1000-dimensional functions. G-CMA-ES results on 1000-dimensional functions are unknown.

5 Conclusions

In this paper, we have introduced an ABC algorithm with a growing population size and we hybridized it with a local search procedure for tackling large-scale benchmark functions. The increasing population size and a stronger focus on the best-so-far solution have contributed to make the proposed IABC algorithm much better performing than the original ABC algorithm. For the hybridization with local search two different local search procedures were used, using either Powell's conjugate direction set or Mtsls1. The parameters of IABC and the hybrid IABC with local search were tuned using iterated F-Race, an automatic parameter tuning tool. For the benchmark functions of 50 and 100 dimensions we found that the hybrid algorithm typically improves over IABC. When compared with other algorithms for large scale continuous optimization from the SOCO special issue, the hybrid IABC algorithm was found to reach high-quality solutions; it was outperformed, according to the median results, only by an incremental PSO algorithm and the overall best performing from the SOCO benchmark competition.

It is maybe more noteworthy that also in the ABC case, extending this algorithm with the incremental social learning framework led to significant performance improvements. Certainly, further analysis of the hybrid IABC algorithm is necessary to determine the contribution of the specific algorithm components. Nevertheless, the that fact that apart from ABC we also could improve PSO and continuous ACO algorithms by embedding them into the incremental social learning framework, gives strong evidence that an increasing population size and the hybridization with local search algorithms are important to obtain high performing swarm intelligence algorithms for continuous optimization.

Acknowledgments This work was supported by the E-SWARM project, funded by an ERC Advanced Grant, and by the Meta-X project, funded by the Scientific Research Directorate of the French Community of Belgium. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Associate.

References

1. Kennedy, J., Eberhart, R.: Particle swarm optimization. Proc. IEEE International Conference of Neural Networks, 4 (1995) 1942–1948
2. Socha, K., Dorigo, M.: Ant colony optimization for continuous domains. European Journal of Operational Research 185 (2008) 1155–1173
3. Montes de Oca, M., Stützle, T., Van den Enden, K., Dorigo, M.: Incremental social learning in particle swarms. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 41(2) (2011) 368–384
4. Montes de Oca, M.A., Aydın, D., Stützle, T.: An incremental particle swarm for large-scale optimization problems: An example of tuning-in-the-loop (re)design of optimization algorithms. Soft Computing (2011) In press.

5. Liao, T., Montes de Oca., M.A., Aydın, D., Stützle, T., Dorigo, M.: An Incremental Ant Colony Algorithm with Local Search for Continuous Optimization. In: GECCO 2011, New York, ACM Press (2011) 125–132.
6. Lozano, M., Molina, D., Herrera, F.: Editorial: Scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. Soft Computing (2011) In Press.
7. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical Report-TR06, Erciyes Universitesi, Computer Engineering Department (2005)
8. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization 39(3), 459–471 (2007)
9. Eiben, A., Marchiori, E., Valk, V.: Evolutionary algorithms with on-the-fly population size adjustment. In Yao, X. et al., eds.: PPSN VIII. Vol. 3242 of LNCS. Springer, Heidelberg (2004) 41–50
10. Chen, D., Zhao, C.: Particle swarm optimization with adaptive population size and its application. Applied Soft Computing **9**(1) (2009) 39–48
11. Hsieh, S., Sun, T., Liu, C., Tsai, S.: Efficient population utilization strategy for particle swarm optimizer. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **39**(2) (2009) 444–456
12. Auger, A., Hansen, N.: A restart CMA evolution strategy with increasing population size. In: CEC 2005, IEEE Press (2005) 1769–1776
13. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the f-race algorithm: sampling design and iterative refinement. In: HM 2007. pp. 108–122. Springer-Verlag, Heidelberg (2007)
14. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-Race and iterated F-Race: An overview. Experimental Methods for the Analysis of Optimization Algorithms pp. 311–336. Springer-Verlag, Heidelberg (2010)
15. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4) (1997) 341–359
16. Eshelman, L., Schaffer, J.: Real-coded genetic algorithms and interval-schemata. Foundations of Genetic Algorithms **2**(1993) (1993) 187–202
17. Suganthan, P., Hansen, N., Liang, J., Deb, K., Chen, Y., Auger, A., Tiwari, S.: Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report 2005005, Nanyang Technological University (2005)
18. Karaboga, D., Akay, B.: A comparative study of Artificial Bee Colony algorithm. Applied Mathematics and Computation 214(1), 108 – 132 (2009)
19. Powell, M.: An efficient method for finding the minimum of a function of several variables without calculating derivatives. The Computer Journal **7**(2) (1964) 155
20. Tseng, L., Chen, C.: Multiple trajectory search for large scale global optimization. In: CEC 2008, IEEE Press (2008) 3052–3059
21. F. Herrara, M.L., Molina, D.: Test suite for the special issue of soft computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems (2010), http://sci2s.ugr.es/eamhco/updated-functions1-19.pdf
22. Diwold, K. and Aderhold, A., Scheidler, A. and Middendorf, M.: Performance evaluation of artificial bee colony optimization and new selection schemes. Memetic Computing (2011) In press.

Two ports of a full evolutionary algorithm onto GPGPU

Ogier Maitre[1], Nicolas Lachiche[1], and Pierre Collet[1]

LSIIT, University of Strasbourg,67412 Illkirch, France,
{ogier.maitre,nicolas.lachiche,pierre.collet}@unistra.fr

Abstract. This paper presents two parallelizations of a standard evolutionary algorithm on an NVIDIA GPGPU card, thanks to a parallel replacement operator.

These algorithms tackle new problems where previously presented approaches do not obtain satisfactory speedup. If programming is more complicated and fewer options are allowed, the whole algorithm is executed in parallel, thereby fully exploiting the intrinsic parallelism of EAs and the many available GPGPU cores.

Finally, the method is validated using two benchmarks.

1 Introduction

The use of General Purpose Graphic Processing Unit (GPGPU) many-core processors for generic computing has raised a lot of interest within the scientific community. Artificial evolution is no exception to this trend and many works propose to port evolutionary algorithms onto GPGPU cards.

Many papers have been published on porting Genetic Programming on GPGPU, because of its great need for computing power. In addition, its model is highly asymmetric: evaluation is by far the most consuming part of the algorithm. Yet genetic algorithms and evolution strategies have proven to be effective and many problems could benefit from a massive parallelization of EAs on multi-core processors, in order to widen search space exploration, or to find equivalent results faster and therefore allow EAs to tackle much larger problems.

Nevertheless, only few works try to port a standard EA on a GPU. When this is done, the algorithm is non standard and performance comparisons are therefore difficult.

In this paper, we propose two complete parallelizations of evolutionary algorithms on GPGPU, which are comparable with sequential algorithms. Thus, users familiar with evolutionary algorithms can access the GPGPU cards computing power without losing their habits. These implementations exhibit interesting speedups with various algorithm configurations.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

2 Related works

2.1 Parellization of Evolutionary Algorithms onto GPGPU

Different implementations of evolutionary algorithms on GPGPU have been attempted in the past. Genetic programming is part of the approaches discussed most recently. This trend is surely due to the evaluation time needed to compute the fitness value of GP individuals. Among these works, we can cite [3], [8] and [11].

Other studies target genetic algorithms and evolutionary strategies. Works of this type are presented in [2, 4, 13], where the authors program the card at the vertex and pixel shader level. The use of such programming paradigms makes porting complex and requires algorithmic changes that tend to make the resulting algorithms difficult to compare with sequential ones. [4, 13] use techniques such as neighbouring selection, while [2] uses a selection with replacement in order to select next parents, which could affect the diversity of the population (there may be clones in the new generation). Furthermore, the raw speedups of these approaches are disappointing considering the efforts invested, even if it is necessary to keep in mind the relative age of the equipment used.

Another approach presented in [10] uses an island model in which each processor group manages an island. The local population is stored in on-chip memory and implementation shows impressive speedups($\approx 7000\times$). However the use of this memory limits the size of the individuals and the number of individuals into an island. Moreover the use of an internal island model on the card questions about the scalability of this approach to a multi-cards architecture.

Recently, the EASEA platform implements another model of parallel evolutionary algorithm on GPGPU. The evolutionary engine runs on the CPU and only evaluation is parallelized on GPU: the population is transferred to the GPU card to be evaluated, then only the fitness values are transferred back to CPU memory in order to continue the algorithm. This model is similar to a standard sequential algorithm, but as stated in Amdhal's law [1], the gain of porting an algorithm on a parallel architecture is limited by the sequential part of the algorithm. Furthermore, it is similar to what is done for the parallelization of GP on GPGPU: only costly evaluation functions can really benefit from this model. The EASEA plateform is thoroughly presented in [7].

2.2 DISPAR: a parallel remplacement operator

One of the main problems about parallelizing a complete EA is the population reduction step. In the case of an (1000+500)ES algorithm, for instance, with 1000 individuals in the population and 500 children, it is necessary to reduce the temporary population of parents + children (1500 individuals) back to only 1000 individuals for the next generation.

Ideally, this reduction operator must not select twice the same individual, in order to preserve diversity in the new generation. The selection must be operated without replacement (an individual selected to be part of the new generation

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

must be removed from the initial pool of individuals). This is very problematic if a parallel selection were to be performed, as it would be impossible to prevent two different cores from selecting the same individual.

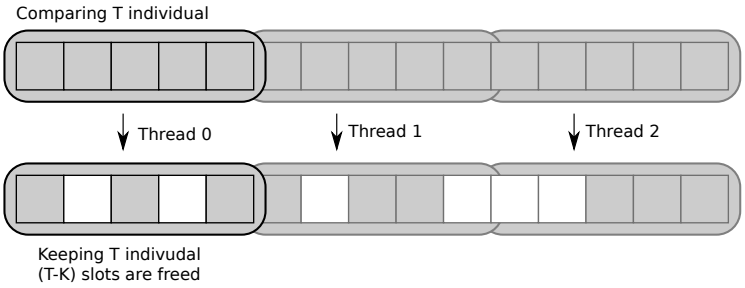


Fig. 1. Parallel individual comparison by dispar operator

DISPAR-tournament (Disjoint Sets Parallel tournament) is a parallel selection operator that mimics a tournament without replacement [9]. The population is divided into blocks of T neighbouring individuals, each block being given to a single thread. The K best individuals are kept, the $(T-K)$ other are deleted. The released slots are reused to store the children for the next generation. The tournaments can be performed by $(\mu+\lambda)/T$ threads in parallel, where λ (resp. μ) is the number of parents (resp. children).

Using this operator allows to fully parallelize any evolutionary algorithm (by simulating a $(\mu+\lambda)$-ES) without the need to execute a serial reduction of the population on the CPU.

3 CUDA Hardware

A GPGPU is a processor containing a large number of cores (typically several hundreds). These computing units are grouped into Single Instruction Multiple Data (SIMD) core bundles, which execute the same instruction at the same time, applied to different data. In CUDA, these units are called Multi-Processors (MP).

Recent Fermi processors have 32 cores that execute each instruction only once. Thus, an instruction is executed by 32 different threads (called a warp), on possibly different data.

NVIDIA hardware implements synchronization mechanisms between the cores of the same warp but there is no explicit mechanism to synchronize different warps. They are implicitly synchronized at the end of the execution of a GPGPU program.

The GPGPU card contains a large global memory that can be accessed by the CPU thanks to Direct Memory Access (DMA) transfers. In addition, each MP embeds a few kilobytes of memory (16 or 48KB, depending on the card), that are only accessible by the MP cores. This extremely fast memory is called shared

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

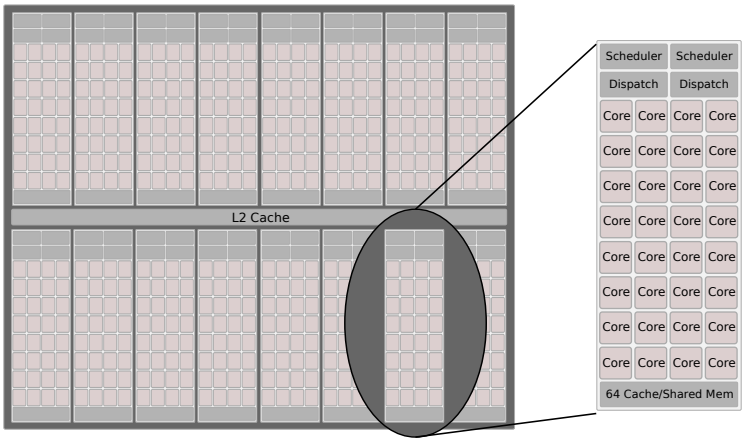


Fig. 2. Architectural details of a Fermi GPGPU Card. GPGPU chip and detail of a Multi-processor.

memory, and replaces what should be a cache to access the global memory. It is implemented using 16 banks that should not be accessed by multiple threads at the same time. The shared memory has a switch which handles complex access patterns. Indeed, if some threads of the same warp attempt to access the same bank, memory accesses are serialized. Otherwise, if threads access different banks, the switch provides simultaneous access to the shared memory.

Without cache, a memory access is very slow (in the order of several hundreds of cycles) and due to its limited size, shared memory can not always compensate for the lack of cache. A second mechanism complementing the shared memory is a hardware thread scheduling mechanism. It is comparable to the Intel Hyper-Threading mechanism, that can be found on Pentium IV or Core i7, but at a much higher level. These units switch frozen warps, for example on memory reads, with other warps, which are ready for execution. Scheduling units can choose among a set of w warps ($w = 32$), which can come from different blocks of threads per MP. The blocks are groups of threads that are alloted to one MP, e.g. the minimal pool of threads to schedule. The size of this scheduling pool can explain why GPGPUs are not efficient with only a few threads, as shown in [6].

The structure of the cores and the availability of hardware scheduling units give GPGPUs a SPMD (Single Program Multiple Data) / SIMD structure, as a warp among w runs in parallel on each multi-processor. Indeed, the threads in a warp must necessarily execute the same instruction, because they are executed by the cores of the same MP at the same time. However, each warp can execute a different instruction without causing any divergence among the cores, because only one warp is executed at a time. In addition, the MPs can execute different instructions on their respective cores at the same time.

Finally, the bus to global memory is particularly large (for a 480GTX it is 384bits). The correct use of the memory bus allows to obtain a high theoretical

bandwidth (177GB/s for a 480GTX) if, when accessing the global memory, the requested data is loaded, as well as neighbouring data, up to the width of the bus. If memory accesses of a warp are aligned (neigbouring threads reading neighbouring data), the number of memory accesses is reduced compared to totally random schemes and bandwidth is better utilised.

4 EAs on GPGPUs

Parallelization of evolutionary algorithms can tackle problems with larger search spaces than those accessible to sequential algorithms because they can deal with much larger population sizes for no extra temporal cost. The EASEA approach (evolutionary engine on CPU, parallel evaluation on remote processors) allows to benefit from GPGPUs on problems with heavy evaluation functions [6, 5].

This gives two advantages: first, a heavy evaluation function compensates for the transfer of the individuals to the computation cores (GPGPU or remote nodes). Secondly, a CPU intensive evaluation makes the parallelization of the evolutionary engine less critical. Indeed, if the evaluation function takes 99% of the total time, it is possible to obtain an acceleration of up to 100x, by parallelizing the evaluation function only, according to Amdahl's law.

But these principles restrict parallelization to algorithms with heavy evaluation functions. Indeed, if the evaluation step takes 50% of the total execution time, even if parallelization makes it virtually instantaneous, the global speedup will only be x2.

In this paper, two approaches are presented that run the whole evolutionary algorithm on the GPGPU architecture. The first uses the DISPAR population reduction operator. The second focuses on improving memory usage, but is restricted to the generational algorithm, therefore eliminating the reduction phase (parents are simply replaced by children).

4.1 DISPAR GPU Evolutionary Algorithm

Population organization As explained in 2.2, DISPAR requires that the individuals are randomly distributed in the population, *i.e.* there must not exist a clear correlation between spatially related individuals. This is done here, by keeping a global population mixing parents and children.

This population is stored in the global memory of the card. Accessing the individuals cause memory latencies but this memory is less constrained in terms of space than the shared-memory.

ind0 G0	ind0 G1	ind0 G2	ind0 G3	ind1 G0	ind1 G1	ind1 G2	ind1 G3	ind2 G0	ind2 G1	ind2 G2	ind2 G3	ind3 G0	ind3 G1	ind3 G2	ind3 G3
ind0 G0	ind1 G0	ind2 G0	ind3 G0	ind0 G1	ind1 G1	ind2 G1	ind3 G1	ind0 G2	ind1 G2	ind2 G2	ind3 G2	ind0 G3	ind1 G3	ind2 G3	ind3 G3

Fig. 3. Classical population layout *vs.* population organization by gene.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

A thread is dedicated to an individual. A population organization as presented in the top of the Figure 3 allows to benefit from cache memory on standard single-core processors. On SIMD processors, the cores access the same gene of different individuals at the same time. A population organization as the one presented on the bottom of fig. 3 should allow to benefit from the large memory bus. For instance, accessing all genes 0 on different individuals takes fewer loading operations.

Initialization The algorithm initialisation phase requires the participation of both CPU and GPGPU. The CPU allocates buffer dedicated to storing individuals on the GPU card and launches the GPGPU procedures. Buffers are needed to store the genomes, the mutation coefficients, the list of free slots and the random generators data, as explained below.

The population is initialized using the parallel random number generator library now provided by CUDA.

In order to use the DISPAR reduction operator, it is necessary to maintain a list of free slots for future children. These locations are uniformly distributed throughout the population. These free locations are referenced in a dedicated table. This table contains, for each thread, a slot where the thread will place the child it will create.

After the replacement step it is still necessary to synchronize the cores, to be sure that all children have been produced. This global synchronisation is done by returning to the CPU at every generation.

Evolving Step Each thread executes a GPGPU evolutionary procedure. This procedure performs the creation of a child, into the free slot allocated to the current thread and then executes DISPAR to determine a number of free slots for a given number of threads.

Two tournament operators are run in order to choose the two parents. These operators are classical n-ary tournament operators. As the operator randomly choses the individuals to compare, the cores execute the same instruction at the same time, but on non-contiguous data.

The crossover accesses all the genes of both parents and computes a weighted centroid (barycentric crossover) for each of them. This means that there is no divergence between threads of a single warp. But accesses to the genes are very likely to be non-contiguous in memory, because parents are coming from the tournament operator.

The mutation operator takes as input a location and applies a Schwefel self-adaptive mutation [12]. A tosscoin function tells whether a gene should be mutated or not. This causes divergences between the cores of a warp, if some threads do not mutate a gene while the other threads do. However, this divergence creates two execution paths, one of which being empty.

Reduction As explained in section **??**, the reduction step requires the population to be distributed randomly, which can be maintained by randomly choosing

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

the parents of an individual within the whole population, the resulting child being placed in this same population. The population is initialized from a size $(\mu+\lambda)$. μ slots are initialized while λ are left free, alternately. These free slots will be used to store the children, ensuring their equal distribution in the population.

The reduction algorithm compares the individuals by bunch of T, (parents or children) and selects λ slots to be freed, *i.e.* among T, $T-K$ while be freed. Again, these slots will be used during the next generation, as storage location for the new children.

This way, there is no clear correlation (sibling or parentage) between spatially related individuals, which induces a memory bandwidth waste because neighbor threads handle non-neighbor individuals.

4.2 Generational Algorithm

The second algorithm reuses all the phases of the first, except the reduction phase, that is no longer needed in the case of a generational algorithm. But thanks to predictable memory accesses, the behaviour of some of these functions changes and the computing performances can be improved.

Population organization The DISPAR-based algorithm exhibits a lack of memory alignment. Indeed, its reduction procedure requires the free slots to be distributed randomly in the population. In the case of a generational algorithm, this constraint does not need to be respected, as the new population will totally replace the previous one.

It is therefore possible to place individuals in an appropriate manner in the memory. To do this, two different populations are used. In a generation, a buffer is used as the parent population, while the other will receive the created children. At the end of the generation, when returning to the CPU, the buffers are exchanged.

In both buffers, the genes of the individuals are placed in the memory using the same technique as for the previous algorithm and described on the bottom of fig. 3.

Evolutionary engine Each generation ends before the replacement phase. The parent selection operator uses the table of fitness. It randomly picks individuals in the population and compares their fitness. Here, the alignment of individuals in memory has no interest.

The crossover operator uses the individuals chosen by the tournament operator. As they can be anywhere in memory, the alignment has no positive effect on parents reading. However, neighbouring threads (for example, those of a warp) will produce children who are neighbours in the second buffer. The alignment accelerates this step.

The mutation function works on children produced by the crossover function. Therefore, neighbour threads will work on neighbour individuals.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Finally, the evaluation function uses children. As for mutation, individuals are accessed in an aligned manner from the memory point of view.

To summarize, parents selection and crossover operators work on non-neighbour individuals and therefore use the memory in a non-aligned way. Operators working on children access memory in an aligned manner.

5 Experiments

To illustrate the behaviour of the algorithms presented above, various tests are performed on a set of artificial problems, on an Intel$^{(R)}$ Core$^{(TM)}$ i7 950 CPU clocked at 3.07GHz running under Linux 2.6.32, 10.04.2 ubuntu with NVIDIA driver 260.19.26. The GPGPU card is a NVIDIA GTX480.

Comparisons are between a parallel algorithm on the GPU card and a sequential algorithm on the CPU, therefore, only one core is used on the CPU.

5.1 Benchmarks

Two well-known toy problems are used, to show the achievable speedup. The first problem is the optimization of the Rosenbrock function. It was used because it is very short to compute and therefore takes longer to execute when the evaluation step only is parallelized on a GPU card, because of the general overhead induced by the paralellization (population transfer to the GPU card, ...). The dimension of the problem varies between 10 and 80. Equation 1 presents the Rosenbrock function used here.

$$W_{b,h}(x) = \sum_{i=1}^{dim}[(1-x_i)^2 + 100(x_{x+1} - x_i^2)^2] \quad (1)$$

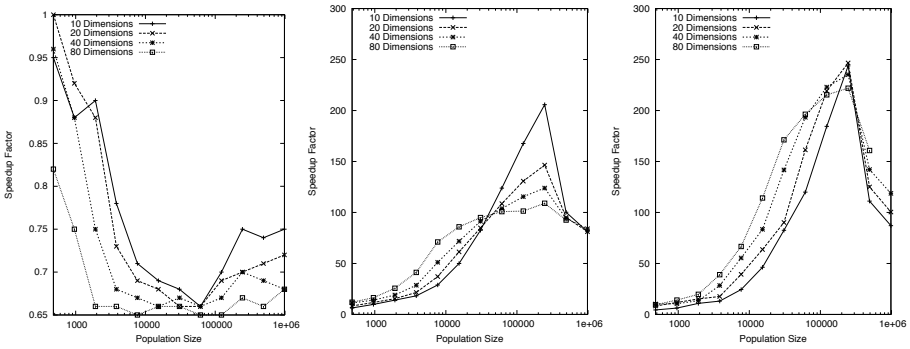
The second problem is to optimize the multi-dimensional Weierstrass function, which in theory is an infinite sum of sines. In practice, the number of sums allows to vary the computational load of the evaluation function. In the following tests, the number of sum iterations is varies between 10 and 80. Here, the number of dimensions is fixed to 40. This problem was also used in [6, 5]. Finally, Equation 2 presents the variant that has been used in the next experiments.

$$W_{b,h}(x) = \sum_{i=1}^{dim}\sum_{j=1}^{iter} b^{-jh} sin(b^j x_i) \text{ with } b > 1 \text{ and } 0 < h < 1 \quad (2)$$

Achieved speedups The following curves compare executions on CPU (ezCPU), with parallel evaluations on GPU (ezGPU), with DISPAR-GPU (DISPAR-GPU) and generational GPU (genGPU) algorithms. Both ezGPU and ezCPU algorithms are implemented using the same EASEA code.

Rosenbrock is a typical problem where parallelizing the evaluation function only on the GPU does not bring any improvement, as seen on fig. 4(a). With

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

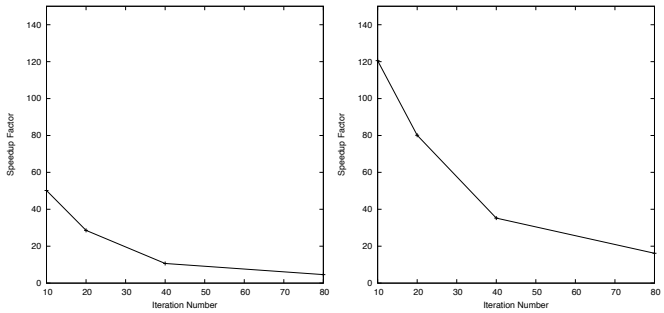


(a) ezGPU *vs.* ezCPU (b) DISPAR-GPU *vs.* ezCPU (c) genGPU *vs.* ezCPU

Fig. 4. Speedup factor for 10 to 80 dimensions Rosenbrock problem.

DISPAR-GPU, speedups starts between 6 and 16x (cf. fig. 4(b)) and reach 200x for a small-sized problem, with a relatively large population (10 dimensions and ≈ 245000 individuals). Above 10 dimensions the speedups fall, probably due to competition problems for the GPGPU memory cache.

For genGPU, the speedup is lower for a small population and small-sized individuals as shown in fig. 4(c), but rapidly increases up to $265\times$ on 10 dimensions with ≈ 245000 individuals.



(a) DISPAR-GPU *vs.* ezGPU (b) genGPU *vs.* ezGPU

Fig. 5. Speedup factor for Weierstrass *w.r.t* the number of iterations for a 40 dimensions problem, with a population 32768 individuals.

Speedup curves are different on the more computation intensive Weierstrass problem (cf. fig. 5(a) and 5(b)). The speedup is more interesting for a small number of iterations. However, when this number increases, the evolutionary engine becomes proportionally less important in the overall execution time. The individuals evaluation speedup should be substantially the same between ezGpu, DISPAR-Gpu and genGpu, only the evolutionary engine can be accelerated for

the two last. The speedup is therefore higher at the beginning and tends to decrease, as long as the evolutionary engine loses its significance in execution time.

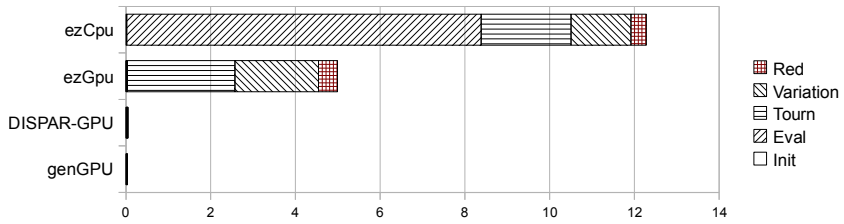


Fig. 6. Time distribution for Weierstrass function with 4800 individuals, 10 iterations and 50 generations (in seconds), over an average of 20 runs.

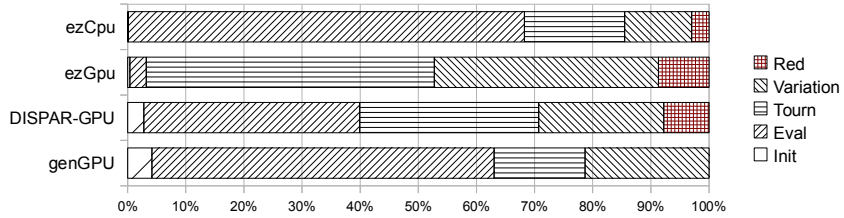


Fig. 7. Same as above, but in percentage of execution time.

Study of time distribution The comparison between ezCPU (sequential execution on one CPU core implemented in EASEA) and ezGPU (EASEA parallelization of the evaluation step on GPU card as described in [6]) shows that evaluation time (60% of the total time in the ezCPU implementation) virtually disappears thanks to parallelizatin on the GPU cores. However, the sequential part of the evolution engine remains identical (cf. fig. 6).

When DISPAR-GPU and genGPU are used, both evaluation and evolution engine execution times are collapsed by parallelization.

Figure 7 shows these same measurements, as a percentage of the total execution time. Here again, one sees that evaluation time disappears on ezGPU. Interestingly enough, when zooming on the collapsed DISPAR-GPU implementation, one sees that the proportions of the different evolutionary steps are roughly equivalent to those of the ezCPU implementation, showing that ezCPU and DISPAR-GPU are very similar.

Things are slightly different for genGPU because the population reduction step is removed (it is a generational replacement).

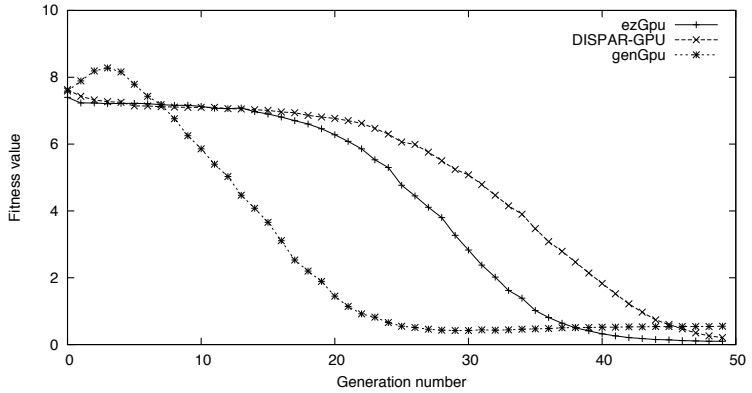


Fig. 8. Evolution of the best individual for the three GPGPU algorithms.

Quality of the algorithms (fitness evolution) ezGPU, DISPAR-GPU and genGPU are different algorithms, therefore, one can expect them to solve the problem with a different efficiency. Fig. 8 shows the evolution of the best individual among a population of 30720 individuals on the Weierstrass benchmark on 10 dimensions with 10 iterations, over 20 runs. ezGpu and DISPAR-Gpu roughly follow the same trend because their population reduction operator (tournament and DISPAR tournament) are similar. genGPU is not elitist (hence the increase in best fitness in the first generations) and the lack of convergence towards 0 as the children are not always better as their parents.

6 Conclusion and Future Works

Two fully parallel evolutionary algorithms have been presented, that were implemented on a GPGPU card. They allow to remove the sequential part of the evolutionary algorithm that was the bottleneck of the EASEA parallelization of evolutionary algorithms proposed in [6, 5]. Parallelization on GPU cards becomes interesting even on very lightweight evaluation functions such as Rosenbrock, with speedups up to about 200, with a parallel DISPAR tournament, or up to about 250 on a generational replacement, with however its disadvantages (no elitism).

The lack of a global synchronization mechanism on GPGPU, still imposes to go back to the CPU at every generation for this only purpose (the cores or the CPU are mostly left idling during the algorithm).

These speedups have been obtained with algorithms that are as standard as possible, allowing to compare their results with purely CPU implementations (as shown in fig. 8). Their behaviour is slightly different from standard ones, that can still largely benefit from GPU parallelization if the evaluation step is heavy to compute. However, more experiments have to be conducted on how our algorithm compares with standard $(\mu + \lambda)$-ES.

For lightweight evaluation function, these adaptations of standard algorithms may be interesting for real-time algorithms where low consumption embedded GPGPU cards can be used for a very high computation/consumption ratio. Their integration into the EASEA platform is in progress.

References

1. Amdahl, G.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18-20, 1967, spring joint computer conference. pp. 483–485. ACM New York (1967)
2. Fok, K.L., Wong, T.T., Wong, M.L.: Evolutionary computing on consumer graphics hardware. Intelligent Systems, IEEE 22(2), 69–78 (2007)
3. Langdon, W.B.: A many threaded CUDA interpreter for genetic programming. In: Esparcia-Alcazar, A.I., Ekart, A., Silva, S. (eds.) EuroGP 2010. Istanbul (7-9 Apr 2010)
4. LI, J.M., WANG, X.J., HE, R.S., CHI, Z.X.: An efficient fine-grained parallel genetic algorithm based on gpu-accelerated. Network and Parallel Computing Workshops, IFIP International Conference on 0, 855–862 (2007)
5. Maitre, O., Lachiche, N., Clauss, P., Baumes, L., Corma, A., Collet, P.: Efficient Parallel Implementation of Evolutionary Algorithms on GPGPU Cards. Euro-Par 2009 Parallel Processing pp. 974–985 (2009)
6. Maitre, O., Baumes, L.A., Lachiche, N., Corma, A., Collet, P.: Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea. In: GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. pp. 1403–1410. ACM, New York, NY, USA (2009)
7. Maitre, O., Krüger, F., Querry, S., Lachiche, N., Collet, P.: Easea: Specification and execution of evolutionary algorithms on gpgpu. Soft Computing - A Fusion of Foundations, Methodologies and Applications p. 1, special issue on Evolutionary Computation on General Purpose Graphics Processing Units
8. Maitre, O., Querry, S., Lachiche, N., Collet, P.: Easea parallelization of tree-based genetic programming. In: et al., F. (ed.) IEEE CEC 2010. pp. 1–8. IEEE (2010)
9. Maitre, O., Sharma, D., Lachiche, N., Collet, P.: Dispar-tournament: A parallel population reduction operator that behaves like a tournament. In: et al., C.D.C. (ed.) EvoApplications 2011. LNCS, vol. 6624, pp. 284–293. Springer, Heidelberg (2011)
10. Pospíchal, P., Jaroš, J., Schwarz, J.: Parallel genetic algorithm on the cuda architecture. In: Applications of Evolutionary Computation. pp. 442–451. LNCS 6024, Springer Verlag (2010)
11. Robilliard, D., Marion-Poty, V., Fonlupt, C.: Genetic programming on graphics processing units. Genetic Programming and Evolvable Machines 10(4), 447–471 (December 2009)
12. Schwefel, H.P.: Numerical Optimization of Computer Models. Wiley, Chichester (1981)
13. Yu, Q., Chen, C., Pan, Z.: Parallel genetic algorithms on programmable graphics hardware. In: Advances in Natural ComputationICNC 2005, Proceedings, Part III. LNCS, vol. 3612, pp. 1051–1059. Springer, Changsha (August 27-29 2005)

A Multilevel Tabu Search with Backtracking for Exploring Weak Schur Numbers

Cyril Fonlupt, Denis Robilliard, Virginie Marion-Poty, and Amine Boumaza

{fonlupt,robilliard,poty,boumaza}@lisic.univ-littoral.fr
Univ Lille Nord de France
ULCO, LISIC, BP 719
62228 CALAIS Cedex, France

Abstract. In the field of Ramsey theory, the *weak Schur number* $WS(k)$ is the largest integer n for which their exists a partition into k subsets of the integers $[1, n]$ such that there is no $x < y < z$ all in the same subset with $x + y = z$. Although studied since 1941, only the weak Schur numbers $WS(1)$ through $WS(4)$ are precisely known, for $k \geq 5$ the $WS(k)$ are only bracketed within rather loose bounds. We tackle this problem with a tabu search scheme, enhanced by a multilevel and backtracking mechanism. While heuristic approaches cannot definitely settle the value of weak Schur numbers, they can improve the lower bounds by finding suitable partitions, which in turn can provide ideas on the structure of the problem. In particular we exhibit a suitable 6-partition of $[1, 574]$ obtained by tabu search, improving on the current best lower bound for $WS(6)$.

Keywords: tabu search, weak Schur numbers, optimization

1 Introduction

1.1 Mathematical description

Ramsey theory is a branch of mathematics that studies the existence of orderly sub-structures in large chaotic structures: *"complete disorder is impossible"*[1]. Many questions in Ramsey theory remain open, and they are typically phrased as *"how many elements of some structure must there be to guarantee that a particular property will hold?"*.

In this paper we focus on the following Ramsey theory problem: finding weak Schur numbers. A set P of integers is called *sum-free* if it contains no elements $x, y, z \in P$ such that $x + y = z$. A theorem of Schur [1] states that, given $k \geq 1$, there is a largest integer n for which the integer interval set $[1, n]$ (i.e. $\{1, 2, \ldots, n\}$) admits a partition into k sum-free sets. This largest integer n is called the k-th Schur number, denoted by $S(k)$.

Adding the assumption that x, y, z must be *pairwise distinct*, i.e. $x \neq y, y \neq z, x \neq z$, we define *weakly sum-free* sets, and a similar result was shown by

[1] Quote attributed to Theodore Motzkin.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Rado [2] : given $k \geq 1$, there is a largest integer n for which the interval set $[1, n]$ admits a partition into k weakly sum-free sets: this largest integer n is the weak Schur number denoted $WS(k)$.

For example, in the case $k = 2$, a weakly sum-free partition of the first 8 integers is provided by:

$$\{1, 2, 3, 4, 5, 6, 7, 8\} = \{1, 2, 4, 8\} \cup \{3, 5, 6, 7\}$$

It is straightforward to verify that increasing the interval to $[1, 9]$ yields a set that does not admit any weakly sum-free partition into 2 sets, thus we have $WS(2) = 8$.

Very few exact values are known for $WS(k)$ (or for $S(k)$):

- $WS(1) = 2$ and $WS(2) = 8$ are easily verified
- $WS(3) = 23$, $WS(4) = 66$ were shown by exhaustive computer search in [3].

The same study [3] shows the following lower bound: $WS(5) \geq 189$, while a much older note by Walker [4] claimed, without proof, that $WS(5) = 196$. Recently a paper by Eliahou, Marín, Revuelta and Sanz [5] provided a weakly sum-free partition of the set $[1, 196]$ in 5 sets, confirming Walker's claim that $WS(5)$ is at least as large as 196, and also gave a weakly sum-free partition of $[1, 572]$ in 6 sets, thus establishing that:

$$WS(5) \geq 196$$

$$WS(6) \geq 572$$

For the general case $k \geq 5$, the best known results are a lower bound from Abbot and Hanson [6] and an upper bound by Bornsztein [7]:

$$c89^{k/4} \leq S(k) \leq WS(k) \leq \lfloor k!ke \rfloor$$

with c a small positive constant.

1.2 Tabu Search

Local search methods are among the simplest iterative search methods. In local search, for a given solution i a neighborhood $N(i)$ is defined, and the next solution is searched among the solutions in $N(i)$.

Tabu search (TS) is based on local search principles. It was introduced by Glover [8]. Unlike the well-known hill-climber search, the current solution may deteriorate from one iteration to the next to get out of a local optimum. In order to avoid possible cycling in the vicinity of a local optimum, tabu search introduces the notion of tabu list, which prohibits moves towards solutions which were recently explored. The duration for a move (or attribute of a move) to remain tabu is called the tabu-tenure. However, the tabu list can be overridden if some conditions, called aspiration criterion, are met. For instance, a typical aspiration criterion is the discovery of a new best solution (better than any

previously visited solution): the move towards such a solution is retained even if it is forbidden by the tabu list. TS was improved with many other schemes like intensification (to focus on some part of the search space) or diversification (to explore some new part of the search space). Details about tabu search and enhancements of tabu search can be found in [9, 10].

In the case of the weak Schur numbers, the search space size, including non feasible solutions, is $k^{WS(k)}$. Even with small values of k a basic tabu search approach is unable to solve the problem in reasonable time limits. The multilevel approach was already successfully used for solving some combinatorial optimization problems [11]. The basic idea of a multilevel approach is to create a sequence of successive coarser approximations of the problem to be solved in reverse order. In our case, we skip the construction of the sequence. We start with $WS(2)$ as the coarsest level and we solve the sequence of problems of increasing difficulty obtained by adding successive partitions, i.e. increasing the value of k as explained further. For convenience we will keep the multilevel term in this paper, although it skips the usual sub-problems sequence construction and thus differs from standard practice while retaining its inspiration.

This multilevel approach proved useful but not sufficient, and a backtracking mechanism was added to allow the search algorithm to rewind back to previous levels. Furthermore, we used a parallel version of the algorithm running on a GPU[2] based server to obtain the results presented in Sect. 3. Due to space constraints, we do not present the details of the machine architecture and parallel implementation.

The rest of the paper is organized as follows. In Sect. 2, we detail our approach, with a description of the multilevel framework and the backtracking mechanism. In Sect. 3, we present the new results that were discovered using our tabu scheme.

2 Detailed Framework

2.1 Problem description and multilevel paradigm

As said in the Introduction, our aim is to find weakly sum-free partitions for the currently known lower bounds for $WS(5)$ and $WS(6)$, and possibly improve (increase) these lower bounds.

In [3], the authors pointed out that there exists 29931 different possible solutions to the $WS(4)$ problem (there are 29931 partitions of $[1, WS(4)]$ into four weakly sum-free subsets). Eliahou and Chappelon [12] found one interesting point: in all these 29931 solutions, the first 23 integers are always stored in only 3 of the 4 sets. Indeed, since $WS(3) = 23$, all the partitions that solve the $WS(4)$ problem are **extensions** of partitions that solve the $WS(3)$ problem. They also noticed that only 2 of the 3 possible weakly sum-free partitions of $[1, WS(3)]$ can be extended to form a weakly sum-free partition of $[1, WS(4)]$.

[2] Graphics Processing Unit.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Eliahou *et al.* in [5] used the `march-pl` SAT solver [13], with a suitable translation of the problem in CNF form to study weak Schur numbers. As the search space increases exponentially with higher values of k for $WS(k)$, the authors could not explore in an exhaustive way all possible solutions. So they conjectured that the property that was exhibited from $WS(3)$ to $WS(4)$ could also hold true from $WS(4)$ to $WS(5)$. So when searching for $WS(5)$, Eliahou *et al.* first searched for solutions to $WS(4)$, then froze the 66 first integers into their respective sets (since $WS(4) = 66$), before searching for $WS(5)$. Conjecturing $WS(5) = 196$ leaves 130 remaining integers to assign, thus greatly reducing the size of the search space. Anyway this reduction was not yet enough to use the `march-pl` solver, and the authors of [5] had to bet on the position of part of the remaining 130 numbers in order to lower even further the computational cost. These hypotheses led them to discover a 5-partition of $[1, 196]$, proving $WS(5) \geq 196$ as part of G. W. Walker's claim. Using further placement hypotheses, they also obtained a 6-partition of $[1, 572]$, proving $WS(6) \geq 572$.

In this paper, we use the same multilevel approach for studying $WS(5)$ and $WS(6)$ by way of tabu search. When a weakly sum-free 4-partition is found for $[1, WS(4)]$, the solution is kept as an un-mutable part of the 5-partitions of $[1, x]$, with $x \geq 196$ the best known lower bound for $WS(5)$. When such a weakly sum-free 5-partition is found, it is again kept unchanged to form the basis of 6-partitions of $[1, y]$, with $y \geq 572$ the best current value for $WS(6)$.

Because some solutions at level k may be impossible to extend to give a level $(k + 1)$ solution, we enhance this technique with the following scheme: when a fixed number of trials fail to give a weakly sum-free $(k+1)$-partition, we assume that we may be trapped in a local optimum. Thus the algorithm is allowed to backtrack to the level k to find a new weakly sum-free k-partition. The details of the algorithm can be found in the next subsection.

2.2 Multilevel Tabu Search with Backtracking

The main components of our tabu search are given here, while the pseudo-code is available in Tab. 2.

Representation: a solution to the $WS(k)$ problem consists in a partition of the set $[1, n]$ into k sets (possibly non weakly sum-free).

Fitness function: the fitness function is simply the number of integers that are involved in a constraint violation. For example, if 2, 8 and 10 are in the same set, this implies the fitness is increased by 3 (as $2 + 8 = 10$). A perfect solution has a fitness $= 0$, i.e. whenever no constraint violations remain.

Formally, let c be a partition of $[1, n]$ into k sets (i.e. a solution), and let $P_i(c)$ be the i^{th} set of the k-partition of c:

$$f(c) = 3 * \sum_{i=1}^{k} |\{x \in P_i(c) \mid \exists y, z \in P_i(c), x < y < z, x + y = z\}|$$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Move operator: a move operator should transform a solution in such a way that it implies only a small change in the search space. In our case, we chose to move an integer from a partition to another. This is performed in two steps: removing the integer from its origin set, and adding it to its destination set (different from its origin set).

The move operator defines the notion of move gain, indicating how much the fitness of a solution is improved (or downgraded) according to the optimization objective when moving an integer to another set. Here a negative move gain (fitness change) characterizes a move that leads to a better solution (minimization problem).

A cheap computation of the move gain, if available, allows a fast, incremental evaluation of the fitness of solutions, hence a fast exploration of the neighborhood in search for the best non tabu neighbor.

In order to accelerate the computation of the move gain, a vector of constraint count is maintained for each set of the partition. This vector associates a number of constraints to every integer in the interval to be partitioned. More precisely, for each pair of integers x and y in the set, we associate one constraint to $x + y$, $y - x$ and $x - y$ (provided these positions are in the interval $[1, n]$ and $y \neq (x - y)$, $x \neq (y - x)$ to respect the pairwise distinct numbers assumption). This constraint count is performed even if the integer is not currently in the set, as illustrated in Fig. 1: this is akin to *forward arc checking* in constraint propagation algorithms.

set 1	indices	1	2	3	4	5	6	7	8
	integers in subset	X	X		X		X		
	constraint count	0	1	2	1	2	1	1	1
set 2	indices	1	2	3	4	5	6	7	8
	integers in subset			X		X		X	X
	constraint count	1	2	1	1	1	0	0	1

Explanation (on set 2):
- integer 1 has 1 pending constraint since if we move 1 to set 2 a conflict will occur with 7 and 8;
- integer 2 has 2 pending constraints due to the presence of respectively $\{3, 5\}$ and $\{5, 7\}$ in set 2;
- integers 3, 5 and 8 have 1 actual constraint each, since they are involved in the same constraint violation (and only this one);
- integer 4 has 1 pending constraint due to the presence of the pair $\{3, 7\}$ in set 2;
- integers 6 and 7 have no pending (nor actual) constraint (this notably implies that we can insert 6 in set 2 without degrading fitness).

Fig. 1. Illustration of the constraint count vectors for a 2-partition of $[1, 8]$.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

- The move gain is obtained by subtracting the constraint count associated to the former position of the integer from the constraint count of its new position.
- The update of the constraint vector can be easily performed:
  - When removing an integer y from a set s, for each other integer x in set s, the number of constraints is decremented by one for all the following indices : $x + y$, $y - x$ and $x - y$ (provided these positions are in the interval $[1, n]$ and $y \neq (x - y)$, $x \neq (y - x)$ to respect the pairwise distinct numbers assumption) — these changes are done in the constraint vector associated to set s;
  - When adding an integer y to a set, the reverse operation is done, i.e. incrementing the constraint count by one at the same positions;

Neighborhood: as explained before, we are exploring partitions of $[1, WS(k)]$ by extending already found weakly sum-free solutions for the $WS(k-1)$ level. So only the integers in $[WS(k-1)+1, n]$ are allowed to be moved in a $WS(k)$ level solution. The neighborhood of a solution is then the set of all possible solutions that can be reached in one application of the move operator on the allowed subset of integers.
Formally, the neighborhood $N(c)$ of a k-partition c is the set of solutions c': $N(c) = \{c' \mid \exists\, ! \, x \in [WS(k-1)+1, n], i, j \in [1, k], i \neq j, x \in P_i(c), x \in P_j(c')\}$

Tabu list and tabu tenure management: each time an integer i is moved from a partition to another partition, it is forbidden to move again the integer i in any partition for the next tt iterations. Each time we launch the tabu search on a given sub-level of the problem, we set dynamically this tabu tenure tt to a random number between 2 and half the number of integers allowed to move.

Aspiration criterion: the aspiration criterion overrides the tabu list when a solution is better than the currently best-known solution.

Perturbation strategy: a random neighbour can always be chosen with a small probability, or else the algorithm chooses the non-tabu neighbor with the best gain move (taken into account the aspiration criterion). As said above we keep the previous sub-level solution unchanged in order to focus on critical elements and narrow the search space. To complement this search strategy, a rewind back mechanism is applied when no improvement to the current solution is detected for a given number of restarts of the tabu (see Sect. 3 for details). In that case, we conjecture that the algorithm is working on a non-extensible solution and we settle back to the previous $(k-1)$ sub-level problem.

3 Experimental results

3.1 Experimental setting

Experiments were conducted with four different types of tabu: the first one is a "classic" tabu that tried to solve directly the $WS(5)$ problem. The second scheme

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

is the same tabu with a restart strategy. The third is a multilevel approach that first tried to solve $WS(2)$, then $WS(3)$ and so on, also using restarts. The fourth scheme is the multilevel with backtracking introduced in Sect. 2.2, referred to as "MLB tabu", starting from $WS(2)$.

In order to allow a fair comparison of the four approaches, all methods were allowed the same total of 10.000.000 fitness evaluations. For the classic tabu with restarts, the multilevel and the backtracking scheme, the maximum number of iterations for a tabu trial is set to 1.000.000 before a restart occurs. For the backtracking scheme we allow $Max\_Run = 5$ restarts before backtracking. We conducted 30 independent runs for the four approaches (each run with a maximum number of evaluations of 10.000.000).

Note that the comparison setting could be deemed slightly biased in favor of the two classic tabu versions, as they directly start with the $WS(5)$ problem unlike the other approaches which start with the $WS(2)$ problem. As a fitness evaluation for the $WS(5)$ problem is actually more time consuming than for the $WS(2)$ problem, the classic versions are in fact also allowed more computing time.

3.2 Results and Analysis

Experiments on $WS(5)$: Table 1 summarizes the results of our experiments on the $WS(5)$ problem: we call "successful run" a run that reaches the current best known bound, 196. For each scheme we report the results obtained within 30 independent runs.

Table 1. Experimental results for Schur number $WS(5)$

	Average fitness	# of successful runs
Classic tabu	17.2	0
Classic tabu with restarts	8.3	0
Multilevel tabu	6.15	1
MLB tabu	2.3	10

The multilevel backtracking mechanism outperforms the other methods considered here by a considerable margin. First, both classic versions were unable to find the current best known bound for the $WS(5)$ problem. Anyhow the restart version is better, certainly benefiting from a necessary diversification. The third algorithm, multilevel only, performs slightly better as it is able to find, once, the bound to the $WS(5)$ problem and gets an average error of 6.15 (2 triplet constraint violations remain on average). The multilevel coupled with backtracking is the best of the 3 approaches, rediscovering the best bound 10 times out of 30 runs, and getting the lowest average error.

As the algorithm was unable to improve the 196 lower bound for $WS(5)$, this gives some confidence that it may be the exact bound.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Experiments on $WS(6)$: We also used the multilevel and backtracking version to work on the $WS(6)$ problem, obtaining several weakly sum-free 6-partitions of the set $[1, 572]$, the best known bound at the time. This algorithm was also able to find a weakly sum-free 6-partition of $[1, 574]$, setting a new lower bound to the 6^{th} weak Schur number problem:

$$WS(6) \geq 574$$

Table 3.2 presents this partition for the $WS(6)$ problem.

4 Conclusions

This work shows that local search optimization techniques such as tabu search can be used to tackle hard Ramsey theory problems, allowing to refine bounds and to obtain instances of solutions that would be difficult, if not impossible, to obtain by other methods, such as exhaustive constraint propagation.

While heuristic approaches cannot definitely settle the value of weak Schur numbers, the solutions we obtained can suggest ideas on the structure of the problem. This study also gives some confidence that 196 is a very probable tight bound for $WS(5)$, while it also sets a new lower bound for $WS(6) \geq 574$.

In our case, the tabu search had to be enhanced by multilevel search and backtracking in order to successfully tackle this problem. Hybridizing our tabu search with some constraint propagation mechanisms could probably enhance further the power of this heuristic.

Tuning the tabu parameters, e.g. by systematic search or by using a tool based on racing algorithms like [14], is also considered, although the computing time cost would be large.

References

1. Issai Schur. Über die kongruenz $x^m + y^m \equiv z^m \pmod{p}$. *Jahresbericht der Deutschen Mathematiker Vereinigung*, 25:114–117, 1916.
2. R. Rado. Some solved and unsolved problems in the theory of numbers. *Math. Gaz.*, 25:72–77, 1941.
3. Peter F. Blanchard, Frank Harary, and Rogerio Reis. Partitions into sum-free sets. In *Integers 6*, volume A7, 2006.
4. G.W. Walker. A problem in partitioning. *Amer. Math. Monthly*, 59:253, 1952.
5. S. Eliahou, J. M. Marín, M. P. Revuelta, and M. I. Sanz. Weak Schur numbers and the search for G. W. Walker's lost partitions. Technical Report 443, Cahiers du LMPA. Univ. Littoral Côte d'Opale., Nov. 2010. To appear in Computers & Mathematics with Applications.
6. H.L. Abbott and D. Hanson. A problem of Schur and its generalizations. *Acta Arith.*, 20:175–187, 1972.
7. Pierre Bornsztein. On an extension of a theorem of Schur. *Acta Arith.*, 101:395–399, 2002.
8. Fred Glover. Tabu search - part I. *ORSA Journal of Computing*, 1(3):190–206, 1989.

9. Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.
10. J.P. Hamiez, J. K. Hao, and F. W. Glover. A study of tabu search for coloring random 3-colorable graphs around the phase transition. *Int. Journal of Applied Metaheuristic Computing*, 1(4):1–24, 2010.
11. C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. *Annals Oper. Res.*, 131:325–372, 2004.
12. Jonathan Chappelon and Shalom Eliahou, 2010. Personal communication.
13. M. Heule and H. V. Maaren. Improved version of march ks. http://www.st.ewi.tudelft.nl/sat/Sources/stable/march_pl. Last accessed on May 2010.
14. Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. The *irace* package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

Fig. 2. Pseudo-code for the tabu search algorithm.

```
Step 1: Initialization
Initialize solution s to WS(2)
Set k = 3 {initial number of partitions}

Step 2: Extension of solution
Initialize iteration_counter, tabu_list, tabu_tenure
Initialize n to either WS(k) if known, or the expected value of WS(k)
for each integer i such that WS(k − 1) < i ≤ n
    randomly assign i to a subset in [1, k]
end for
f = fitness(s)      { evaluate the fitness of s }
Repeat
    { either a random move or a neighborhood exploration }
    if (random in [0, 1] > 0.9)
      perform a random move
    else      { apply neighborhood search to s }
      Initialize δ to ∞ {worst possible move gain}
      for each integer i such that WS(k − 1) < i ≤ n
          { P_i is the current subset holding i }
          for all possible subset P' ≠ P_i
              evaluate δ the fitness improvement when moving i to P'
              if ((δ is the greater so far) &&
                  (move not in tabu_list || aspiration criterion is true))
                  record this move as best_move
              end if
          end for
      end for
      perform best_ move
    end if
    update fitness f      { quick computation using move gain }
until stop condition
{ either the problem WS(k) is solved or maximum number of iterations is reached }

Step 3: Extension or rewind back
if (s solution to WS(k))
    k = k + 1
    Go to Step 2
else
    if (maximum number of trial runs for WS(k) is not yet reached)
      runs = runs + 1
      Go to Step 2
    else
      { seemingly this solution is not extensible, rewind back }
      k = k - 1
      reset number of trial for WS(k)
      Go to Step 2
    end if
end if
```

Table 2. Sum free 6-partition of the 574 first integers.

$E_1 =$ [1, 2, 4, 8, 11, 22, 25, 31, 40, 50, 60, 63, 69, 84,
97, 135, 140, 145, 150, 155, 164, 169, 178, 183, 193, 199,
225, 258, 273, 330, 353, 356, 395, 400, 410, 415, 438, 447,
461, 504, 519, 533, 547, 556, 561, 568, 571]

$E_2 =$ [3, 5, 6, 7, 19, 21, 23, 35, 36, 51, 52, 53, 64, 65,
66, 80, 93, 109, 122, 124, 137, 138, 139, 151, 152, 153, 165,
180, 181, 182, 194, 195, 196, 210, 212, 226, 241, 251, 255,
298, 310, 314, 325, 340, 341, 354, 355, 369, 371, 384, 397,
398, 399, 411, 412, 413, 426, 440, 441, 442, 458, 472, 473,
482, 486, 498, 500, 502, 514, 515, 529, 530, 531, 542, 543,
558, 559, 560, 572, 573, 574]

$E_3 =$ [9, 10, 12, 13, 14, 15, 16, 17, 18, 20, 54, 55,
56, 57, 58, 59, 61, 62, 101, 103, 104, 107, 141, 142, 143,
144, 146, 147, 148, 149, 184, 185, 186, 187, 188, 189, 190,
191, 192, 227, 229, 230, 232, 233, 234, 235, 269, 270, 276,
317, 319, 320, 321, 322, 359, 360, 361, 363, 364, 365, 401,
402, 403, 404, 405, 406, 407, 408, 409, 443, 444, 445, 446,
448, 449, 450, 451, 476, 477, 478, 479, 483, 484, 520, 521,
522, 523, 524, 525, 526, 527, 528, 562, 563, 564, 565, 566,
567, 569, 570]

$E_4 =$ [24, 26, 27, 28, 29, 30, 32, 33, 34, 37, 38, 39,
41, 42, 43, 44, 45, 46, 47, 48, 49, 154, 156, 157, 158, 159,
160, 161, 162, 163, 166, 167, 168, 170, 171, 172, 173, 174,
175, 176, 177, 179, 284, 286, 288, 289, 292, 294, 295, 303,
304, 305, 309, 414, 416, 417, 418, 419, 420, 421, 422, 423,
424, 425, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436,
437, 439, 532, 534, 535, 536, 537, 538, 539, 540, 541, 544,
545, 546, 548, 549, 550, 551, 552, 553, 554, 555, 557]

$E_5 =$ [67, 68, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
81, 82, 83, 85, 86, 87, 88, 89, 90, 91, 92, 94, 95, 96, 98,
99, 100, 102, 105, 106, 108, 110, 111, 112, 113, 114, 115,
116, 117, 118, 119, 120, 121, 123, 125, 126, 127, 128, 129,
130, 131, 132, 133, 134, 136, 271, 274, 275, 279, 280, 282,
283, 287, 291, 296, 299, 302, 307, 308, 312, 313, 315, 452,
453, 454, 455, 456, 457, 459, 460, 462, 463, 464, 465, 466,
467, 468, 469, 470, 471, 474, 475, 480, 481, 485, 487, 488,
489, 490, 491, 492, 493, 494, 495, 496, 497, 499, 501, 503,
505, 506, 507, 508, 509, 510, 511, 512, 513, 516, 517, 518]

$E_6 =$ [197, 198, 200, 201, 202, 203, 204, 205, 206, 207,
208, 209, 211, 213, 214, 215, 216, 217, 218, 219, 220, 221,
222, 223, 224, 228, 231, 236, 237, 238, 239, 240, 242, 243,
244, 245, 246, 247, 248, 249, 250, 252, 253, 254, 256, 257,
259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 272, 277,
278, 281, 285, 290, 293, 297, 300, 301, 306, 311, 316, 318,
323, 324, 326, 327, 328, 329, 331, 332, 333, 334, 335, 336,
337, 338, 339, 342, 343, 344, 345, 346, 347, 348, 349, 350,
351, 352, 357, 358, 362, 366, 367, 368, 370, 372, 373, 374,
375, 376, 377, 378, 379, 380, 381, 382, 383, 385, 386, 387,
388, 389, 390, 391, 392, 393, 394, 396]

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

An Ant Colony Optimization Algorithm for the Two-Stage Knapsack Problem

Stefanie Kosuch

Institutionen för datavetenskap (IDA),
Linköpings Universitet, Sweden
stefanie.kosuch@liu.se
http://www.kosuch.eu/stefanie

Abstract. We propose an Ant Colony Optimization algorithm for the Two-Stage Knapsack problem with discretely distributed weights and capacity. To the best of our knowledge this is the first attempt to solve a Two-Stage Knapsack problem using a metaheuristic. Two heuristic utility measures are proposed and compared. Moreover, we introduce the novel idea of non-utility measures in order to obtain a criterion for the construction termination. We argue why for the proposed measures it is more efficient to place pheromone on arcs instead of vertices or edges of the complete search graph. Numerical tests show that our algorithm is able to produce, in much shorter computing time, solutions of similar quality than CPLEX after 2h. Moreover, with increasing number of scenarios the percentage of runs where our algorithm is able to produce better solutions than CPLEX (after 2h) increases.

1 Introduction

The knapsack problem is a widely studied combinatorial optimization problem. Special interest arises from numerous real life applications for example in logistics, network optimization and scheduling. The basic problem consists in choosing a subset out of a given set of items such that the total weight of the subset does not exceed a given limit (the capacity of the knapsack) and the total benefit of the subset is maximized (for more information on (deterministic) knapsack problems see the book by Kellerer et al. [10]). However, most real life problems are non-deterministic in the sense that some of the parameters are not (exactly) known at the moment when the decision has to be made. If randomness occurs in the capacity constraint, the main question that has to be answered is if a violation of the capacity constraint (i.e. an *overload*) could be acceptable. If an overload cannot be permitted in any case, the model maker has two possibilities: Either to force the feasible solutions of the resulting problem to satisfy the capacity constraint in any case. This generally leads to very conservative decisions and the resulting problem might even be infeasible or only have trivial feasible solutions. Or to allow for later corrective decisions at, naturally, additional costs. This latter model is called a *multi-stage* decision model in the literature (for an introduction to stochastic programming models see e.g. [21]).

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Several variants of stochastic knapsack problems have been studied so far and the interest seems still to increase. Among the publications on stochastic knapsack problems released in the last year you can find papers on the simple-recourse knapsack problem ([17]), the chance-constrained knapsack problem ([8]), two-stage knapsack problems ([7]) as well as dynamic settings of the stochastic knapsack problem ([2]).

In this paper we allow the item weights and the capacity to be random and study a *two-stage* variant of the knapsack problem, denoted $TSKP$ in the remainder. We assume the vector that contains capacity and item weights to be discretely distributed, i.e. to only admit a finite number of realizations with non-zero probability. In fact, in [12] it has been shown that a stochastic combinatorial optimization problem such as the $TSKP$ can be approximated to any desired precision by replacing the underlying distribution by a finite random sample.

It is well known that in the case of finite weight distributions the $TSKP$ can be equivalently reformulated as a deterministic linear programming problem with binary decision variables (see e.g. [7]). However, the set of constraints and binary decision variables in the reformulation grows with both the number of items as well as the number of scenarios. It is thus typically very large, or even exponential in the number of items. The aim of this paper is therefore to propose an Ant Colony Optimization (ACO) algorithm for the $TSKP$ in order to obtain near optimal or even optimal solutions in short computing time (for an introduction to ACO-algorithms and standard procedures see [15]). We think that an ACO-algorithm is a good choice to solve the $TSKP$ due to the possibility to effectively use utility measures. Moreover, ants are building (new) solutions without needing to evaluate the objective function, which, in the case of the $TSKP$, is an $\mathcal{NP}$-hard task itself.

In the last decade, several metaheuristics for Stochastic Combinatorial Optimization problems ($SCOP$s) have been presented. There are two aspects why metaheuristics are important tools to solve $SCOP$s: the problem size (especially in the case of independently discretely distributed parameters or simply a high number of possible scenarios) and the question of how to evaluate the objective function. In fact, in most cases evaluating the objective function of an $SCOP$ is $\mathcal{NP}$-hard. In other cases, no deterministic equivalent reformulation is known and only approximate values can be obtained (e.g. using Sample Average Approximation). Both difficulties can be tackled by applying appropriate metaheuristics (see e.g. [3]). However, to the best of our knowledge, no special purpose metaheuristic for the $TSKP$ has yet been proposed. Our work is, however, inspired by previous works on ACO-algorithms for the related Multiply Constrained Knapsack problem $MCKP$ (also known as multidimensional knapsack problem or multiple constraint knapsack problem, see e.g. [6],[9]).

This paper is in continuation of the preliminary work presented in the extended abstract [13]. Main differences are the modified pheromone update procedure (see subsection 3.2 for more details), the important decrease of CPU time due to a smaller number of solution evaluations and the use of a specific knapsack algorithm (see subsection 3.5) as well as the extension of the numerical

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

tests and their analysis (see section 4). The algorithmic changes also entailed important changes in the parametrization. Moreover, the ratio measure that has been considered in [13] is omitted in this paper due to its inferiority compared to the simple and difference measures (see subsection 3.3).

2 Problem Formulation, Properties and an Application

We consider a stochastic two-stage knapsack problem where both the knapsack capacity as well as the item weights are not known in the first stage but come to be known before the second-stage decision has to be made. Therefore, we handle capacity and weights as random variables and assume that the capacity-weight-vector $(\gamma, \chi) \in \mathbb{R}^{1+n}$ is discretely distributed with K possible realizations (or scenarios) $(\gamma^1, \chi^1), \ldots, (\gamma^K, \chi^K)$. The corresponding, non-zero probabilities are denoted $p^1, \ldots, p^K$. Weights and capacity are assumed to be strictly positive in all scenarios.

In the first stage, items can be placed in the knapsack. The corresponding first-stage decision vector is $x \in \{0,1\}^n$. Placing item i in the knapsack in the first stage results in a reward $r_i > 0$. At the beginning of the second stage, the weights of all items as well as the capacity are revealed. First-stage items can now be removed and additional items be added in order to make the capacity constraint be respected and/or to increase the total gain.

If item i is removed, a penalty d_i has to be paid that is naturally strictly greater than the first-stage reward r_i. The removal of item i is modeled by the decision variable y_i^- that is set to 1 if the item is removed and to 0 otherwise. Similarly, we assume that the second-stage reward $\overline{r}_i > 0$ is strictly smaller than the first-stage reward. If an item is added in the second stage we set the corresponding binary decision variable y_i^+ to 1. The resulting Two-Stage Knapsack problem with discrete weight distributions can be formulated as follows:

Two-Stage Knapsack Problem with Discretely Distributed Weights ($TSKP$)

$$\max_{x\in\{0,1\}^n} \quad \sum_{i=1}^{n} r_i x_i + \sum_{k=1}^{K} p^k \mathcal{Q}(x, \gamma^k, \chi^k) \tag{1}$$

$$\text{s.t.} \quad \mathcal{Q}(x,\gamma,\chi) = \max_{y^+,y^-\in\{0,1\}^n} \sum_{i=1}^{n} \overline{r}_i y_i^+ - \sum_{i=1}^{n} d_i y_i^- \tag{2}$$

$$\text{s.t.} \quad y_i^+ \le 1 - x_i, \quad \forall\, i = 1, \ldots, n, \tag{3}$$

$$y_i^- \le x_i, \quad \forall\, i = 1, \ldots, n, \tag{4}$$

$$\sum_{i=1}^{n} (x_i + y_i^+ - y_i^-)\chi_i \le \gamma. \tag{5}$$

The $TSKP$ is a *relatively complete recourse* problem, i.e. for every feasible first-stage decision there exists a feasible second-stage decision. Moreover, given a

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

first-stage decision and a realization of (γ, χ), solving the second-stage problem means solving a deterministic knapsack problem. Evaluating the objective function for a given first-stage solution is thus $\mathcal{NP}$-hard.

The $TSKP$ has a deterministic equivalent reformulation as a combinatorial optimization problem with linear objective and constraints. This reformulation is obtained by introducing K copies of the second-stage decision vector and treating the second-stage constraints for each scenario separately (see e.g. [7]). However, the obtained reformulation has $(2K + 1)n$ binary decision variables and $(2n + 1)K$ constraints. Note that K can be exponential in the number of items, e.g. in the case of independently discretely distributed weights.

Although its structure is on the first sight similar to that of an $MCKP$, the deterministic reformulation of the $TSKP$ contains negative rewards and weights (for removed items). To the best of our knowledge the $TSKP$ cannot be equivalently reformulated as an $MCKP$ (with strictly positive rewards and non-negative weights).

As a simplified application of the $TSKP$ consider an (online) travel agency that aims to fill the vacant beds (the random capacity) of a hotel complex. Clients are travel groups whose exact number of travelers (the "weight" of the group) is still unknown at the moment the decision which groups to accept has to be made. The randomness of the groups' sizes can for example be a result of later cancellations, and the randomness in the number of beds to be filled can be due to reservations by other agents or reparation works. If an upper bound on the sizes of the travel groups is known, the probability space for the weights is finite. In order to maximize the final occupancy of the beds, the travel agent might allow an overbooking. If, in the end, the number of beds is not sufficient, one or more of the groups need to be relocated in neighboring hotels which leads to a loss of benefit. If beds are left unoccupied, last minute offers at reduced prices might be an option to fill these vacancies. A simple recourse version of this problem with a set of hotel sites has been previously considered in [1].

3 The *ACO*-Metaheuristic

In this section we will propose an ACO-metaheuristic to solve the $TSKP$. Numerical results and comparisons of the considered variants are summarized in section 4. We use the following notations:

- $\mathcal{A}$: set of ants
- t: "time", i.e. passed number of construction steps in current iteration ($t \leq n$)
- $S_a(t)$: set of items chosen by ant a after time t
- $\tau_i(t)$: pheromone level on vertex/arc/edge i at time t
- η_i: utility ratio of item i
- ν_i: non-utility ratio of item i
- $\rho \in (0, 1)$: global evaporation parameter
- $\rho_{loc} \in (0, 1)$: local evaporation parameter
- $p^a_{uv}(t)$: transition probability = probability for ant a to go from vertex u to vertex v at time t

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

The basic structure of the *ACO*-algorithm for the $TSKP$ is given in Algorithm 3.1. Its functioning is detailed in the following subsections. The `Transition of ants` step consists of the transition of the ants following the transition probabilities and the update of $S_a(t)$.

```
IT ← 0
while IT < IT_MAX do
  IT ← IT + 1
  Initialization
  t ← 0
  while t < n and (∃ a ∈ A: (n+1) ∉ S_a(t − 1)) do
    t ← t + 1
    Computation of transition probabilities
    Transition of ants
    Local pheromone update
  end while
  Global pheromone update
end while
return  Globally best solution
```

Algorithm 3.1: ACO-algorithm for the $TSKP$

3.1 The Complete Search Graph

Our search graph is based on the search graph proposed for the $MCKP$ in [6] and [14], i.e. on a complete graph whose n vertices represent the n items. Note that the ants only construct the first-stage solution (solution vector x). In order to model the randomness of the first item chosen by an ant, we add a *starting vertex* to the complete graph. Initially, all ants are placed on this vertex.

In the case of the $MCKP$ one has a natural certificate of when an ant has come to an end of its solution construction: when either all items have been chosen or when adding any of the remaining items would lead to the violation of at least one of the constraints. As for the $TSKP$ even adding all items in the first stage would yield a feasible solution, we add a *termination vertex* $n+1$ which is connected to all other vertices, including the starting vertex.

3.2 Pheromone Trails and Update Procedure

Several choices could be made for the way pheromone is laid by the ants (see [9]). In the simplest setting, the search graph is simple and non-directed and pheromone is laid on *vertices*. In the second variant, pheromone is placed on the *edges* of the simple, non-directed search graph, or, equivalently, pairs of items. More precisely, if in a solution both items i and j are selected, pheromone is increased on the edge $(i, j) = (j, i)$ (and on all other edges of the clique defined

by the chosen items). In the third variant the graph is assumed to be a complete directed graph and pheromone is laid on *arcs*, i.e. directed edges. Contrary to the two former settings, this setting does not only take into account which items (or item pairs) had been added to former good solutions, but also in which order. In the following, when talking of an *element*, this refers to either a vertex, edge or arc of the search graph.

We use a local as well as a global update procedure. The local update procedure is performed after every construction step. The pheromone level on the elements chosen by the ants during this step is slightly reduced, in order to diversify the produced solutions. For an element i the local update rule is as follows:

$$\tau_i \leftarrow (1 - \rho_{loc}) \cdot \tau_i + \rho_{loc}\tau_{min} \tag{6}$$

Here τ_{min} is a lower bound for the pheromone level. Note that no local update is done during the first iteration, i.e. only the utility measures are taken into account for the solutions constructed in the initial iteration.

The global update procedure is done once all ants have constructed their solutions. In our setting we intensify the pheromone level on the elements of the globally best solution (i.e., the best solution found so far) as well as on the elements of the λ best solutions of the last iteration:

$$\tau_i \leftarrow (1 - \rho) \cdot \tau_i + \rho \cdot \frac{f}{f_{glob}} \tag{7}$$

Here f is the objective function value of the respective solution and f_{glob} the globally best solution value. Note that the maximum pheromone level is 1. If after the global update procedure the pheromone level on an element lies below a fixed lower bound $\tau_{init} \geq \tau_{min}$, it is set to τ_{init}. This means that, while the pheromone level on certain elements might fall beneath τ_{init} during the local update procedure (but never below τ_{min}), the level is at least τ_{init} on all elements at the beginning of each iteration.

Note that for $\lambda = 0$ only the pheromone on the elements of the globally best solution is increased. We also tested implementations where only the pheromone on the best solution(s) of the current iteration are considered. This variant, however, showed less good convergence properties due to the apparently high importance of exploitation when solving the $TSKP$ with an ACO-algorithm (see also section 4).

The here chosen update procedure (see also [5]) has turned out to be superior to the previously tested, generally applied update procedure where pheromone is evaporated on all items during the global update procedure (see [13]).

3.3 Heuristic Utility Measures

An advantage of the $TSKP$ compared to the $MCKP$ is that we have a clearly defined "relevance factor" for each knapsack constraint: the probability of the corresponding scenario (see [10] for more information on utility measures for the

$MCKP$). Our idea is thus to compute the overall utility ratio of an item as an average over the utility ratios of those scenarios where the item still fits the capacity. The problem is, however, that once adding an item would lead to a violation of the capacity in one or more scenarios, deciding whether it is more profitable to remove an item and add the new one, or to discard the current item, is $\mathcal{NP}$-hard. We overcome this problem by relying on the chosen utility measure: If the utility measure is chosen wisely, one might get good solutions by always discarding the current item (in the case of an overload).

While in the case of the $MCKP$ two factors have to be considered (reward and weight), there are 2 more factors that play a role for the utility of an item in the two-stage setting: the second-stage reward and the second-stage penalty. This renders the definition of a good utility measure much more complex.

The utility ratio for the termination vertex should depend on the penalty we would have to pay in the second stage if we add another item and/or the reward we could gain in the second stage if we do not add any of the remaining items. We thus compute an additional "non-utility" ratio for each item. The utility ratio of the termination vertex is then defined as the minimum over these ratios: If for all items the non-utility ratio is high, termination might be the best choice. Note that when it comes to removing items, items with a small penalty-weight ratio might be more attractive.

We propose two different (non-)utility ratios. Both are calculated with respect to the set $\mathcal{K}_i$ of scenarios where item i still fits in the knapsack. As usual, the utility ratio of an item that had already be chosen by the respective ant is defined to be zero.

Simple measure: Here we define the utility of an item i to be the "average" ratio of first-stage reward and weight.

$$\eta_i^S = \sum_{k \in \mathcal{K}_i} p^k \frac{r_i}{\chi_i^k} \tag{8}$$

Note that this measure is not exactly the mean of the reward-weight ratios over the scenarios where the item still fits as $\sum_{k \in \mathcal{K}_i} p^k < 1$ is possible. The exact mean would be obtained by dividing η_i^S by $\sum_{k \in \mathcal{K}_i} p^k$. The utility ratio does thus increase with the probability that item i still fits the capacity (given by $\sum_{k \in \mathcal{K}_i} p^k$).

We define two non-utility ratios. For half of the ants the first measure is applied and for the other half the second. The first non-utility ratio is defined to be the "average" ratio of second-stage penalty and weight over those instances where the item does not fit in the knapsack any more. Contrary to the utility ratio η^S, this first non-utility ratio increases with $\sum_{k \notin \mathcal{K}_i} p^k$. The second non-utility ratio equals the reward we would gain on average in the second stage if we do not add the item and assume that it can be added in any scenario in the

second stage.

$$\nu_i^{S_1} = \sum_{k \notin \mathcal{K}_i} p^k \frac{d_i}{\chi_i^k} \qquad \nu_i^{S_2} = \sum_{k=1}^{K} p^k \frac{\overline{r}_i}{\chi_i^k} \tag{9}$$

Difference measure: We compare what we would gain by adding an item in the first and not the second stage ($r_i - \overline{r}_i$) with what we would loose if we would have to remove the item in the second stage ($d_i - r_i$):

$$\eta_i^D = \sum_{k \in \mathcal{K}_i} p^k \frac{r_i - \overline{r}_i}{\chi_i^k} \qquad \nu_i^D = \sum_{k \notin \mathcal{K}_i} p^k \frac{d_i - r_i}{\chi_i^k} \tag{10}$$

3.4 Transition Probabilities

In this study we only consider the most traditional way of computing the transition probabilities from the pheromone level and utility ratio (see e.g. [15]):

$$p_{uv}^a(t) = \frac{\tau_{i(u,v)}^{\alpha}(t)\eta_v^{\beta}}{\sum_{w=1}^{n+1} \tau_{i(u,w)}^{\alpha}(t)\eta_w^{\beta}} \tag{11}$$

Here α and β are two parameters that control the relative importance of pheromone level and utility ratio and $i(u,v) = v$ (vertex pheromone) or $i(u,v) = (u,v)$ (arc or edge pheromone). In case we use the simple measure, the utility ratio η_v of an item v is computed using (8) while the utility ratio η_{n+1} of the termination vertex is computed as $\min_{v \in \{1,\dots,n\}} \nu_v^{S_1}$ for half of the ants and $\min_{v \in \{1,\dots,n\}} \nu_v^{S_2}$ for the other half (see (9)). Similarly, if we decide to use the difference measure, η_v is given by η_v^D for an item v and $\eta_{n+1} = \min_{v \in \{1,\dots,n\}} \nu_v^D$ (see (11)).

3.5 Decreasing the Computing Time

In order to decrease the computing time of the algorithm we only recompute the objective function values for those newly constructed solutions that had not been constructed by any ant so far. A binary tree was used as data structure to store already computed solutions. This procedure turned out to be especially efficient towards the end of the algorithm when generally a lot of ants reproduce the best solution(s) found so far. Compared to a preliminary design where only the globally best solution was not recomputed (see [13]) we were able to increase the computing time by around 33%.

In our preliminary setting of the *ACO*-algorithm we used CPLEX in order to evaluate the objective function while for the final implementation we chose to use the Minknap algorithm implemented by D. Piesinger (see [19]) and presented in [18]. This decreased the CPU time of the *ACO*-algorithm by another 90%.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

4 Numerical Tests

The *ACO*-algorithm has been implemented in C++ and the tests run on a Intel Core 2 with 3GHz and 4GB RAM.

Our test instances were generated from $MCKP$ test instances of the OR-library. For the second-stage rewards (penalties) we uniformly generated a factor on $[0.95, 1]$ ($[1, 1.1]$) and multiplied it with the first-stage reward. The probabilities of the scenarios have been independently generated.

After a first extensive test on an instance with 5 scenarios, only small changes were made in the parameters for each number of scenarios and utility measure: ρ and ρ_{loc} were always chosen as either 0.1, 0.3 or 0.5 and τ_{min} and τ_{init} on $[0.01, 0.1]$. In case of pheromone on vertices or edges best results were achieved with $\lambda = 0$, while in case of pheromone on arcs we always set $\lambda = 3$. While the choice of these parameters is not surprising compared to other studies, the choice of the parameters α and β is: most often, best results are obtained with $\alpha = 1$ and β taking an integer value strictly greater than 1 (see e.g. [9] or [14]). Moreover, early stagnation was reported in several studies whenever the relative importance of α was chosen too high ($\alpha > 1$, see e.g. [16]). In our tests it turned out that best results are achieved with $\beta = 1$ and $\alpha \in \{20, 30, 40\}$, with α increasing when K increases. For $\alpha \leq 10$ the algorithm did not converge even to a local optimum and kept repeating solutions far from the optimum. On one hand this reflects the aforementioned difficulty to define a suitable utility measure. On the other hand it shows that the pheromone level plays an important role and exploitation seems to be much more important to obtain good solutions when solving the $TSKP$ with an *ACO*-algorithm than it is when solving other combinatorial problems.

When using the simple measure globally best solutions were found by both ants that were using the first and ants using the second non-utility ratio.

4.1 Comparison of the 3 Different Variants to Lay Pheromone

For our algorithm placing pheromone on arcs showed much better results than placing it on vertices or edges. More precisely, we observed during our tests that in the latter two cases the ants had difficulties to reproduce the globally best solution and to search in its local neighborhood. As a consequence, the solution value of the best solution produced during an iteration was mostly strictly smaller than that of the the globally best solution. This caused severe problems for the convergence of our *ACO*-algorithm. On the contrary, with pheromone on arcs, the quality of the best solutions produced during a single iteration generally increased monotonically (however not strictly).

These observations can be explained as follows: In case of pheromone on arcs pheromone will mainly be accumulated on a particular path. An ant that reconstructs the globally best solution will probably do it by following this path. In case of pheromone on vertices there are several possibilities to reconstruct the globally best solution. Due to the accumulated pheromone an ant might choose with high probability any of the items that are contained in the globally best

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

solution as first (or next) vertex to go to. However, recall that both utility ratios rely on the order in which the items have been chosen as this order defines for the different scenarios which items still fit, and which do not. So assume that the items chosen so far by an ant are all part of the globally best solution but that the order in which they have been chosen is different. Then it is possible that the utility of an item that is not part of the globally best solution has now a much higher utility than the rest of the items. This explains also the superiority of the variant where pheromone is laid on *directed* and not on *indirected* edges, that could not be explained by the structure of the $TSKP$ that does not take the order in which the items are added into account.

Due to this preliminary observations further studies were only made with pheromone on arcs.

4.2 Comparison of the 2 Different Utility Measures (pheromone on arcs) and Performance Relative to CPLEX

For a representative comparison of the convergence behavior of our ACO-algorithm using the two different measures see Figure 1 in [13]. The figure shows two properties that we noticed in most runs: First, that the solution found in the initial iteration of the ACO-algorithm (where only the utility is taken into account) is better when using the difference measure than when using the simple measure. This can already be seen as a sign that the difference measure might be more suited as utility measure for the $TSKP$. Second, one sees that the algorithm converges much faster to near optimal solutions when using the difference measure and the quality of the best solution produced per iteration never decreases even when the globally best solution is already close to the optimum. It turned out that this latter behavior generally indicates if the algorithm will, on the given instance, find the optimal or at least a near optimal solution: On instances where the algorithm is (repeatedly) not able to reconstruct the globally best solution during several iterations, the solution returned by the algorithm is mostly of very poor quality. This again shows that exploitation seems to play an important role when solving the $TSKP$ with a metaheuristic.

The numerical results of our tests are displayed in Table 4.2. The first row gives the number of items n, the number of scenarios K and the tightness t (as defined in [4]) of the $MCKP$ instance used to construct the $TSKP$ instance. For each such triple, we randomly chose 3 instances that were especially hard to be solved by CPLEX, i.e. where the CPU time exceeded 2 hours or were the computation stops earlier due to lack of available memory space. On each instance, we made 50 independent runs of our algorithm. The first row shows the relative gap between the best solutions given by CPLEX and the greedy heuristic that one obtains from the proposed ACO-Algorithm when setting $\alpha = 0$. In this randomized heuristic only the utility measure counts for the computation of the transition probability. Note that in none of the runs the heuristic was able to find the best solution given by CPLEX (or a better one). The second row displays the number of instances where our algorithm was able to either find the best solution given by CPLEX (after 2h) in at least one of the 50 runs, or an

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Table 1. Numerical results using the two different utility measures

			Difference measure			**Simple measure**		
n-K-t	Heuristic (rel. gap)	Succesf. instances	Succesf. runs	Average rel. gap	Time in sec.	Succesf. runs	Average rel. gap	Time in sec.
100-5-0.25	0.53 %	3/3	57 %	0.02 %	35	13 %	0.05 %	30
100-5-0.5	0.21 %	2/3	28 %	0.01 %	57	1 %	0.03 %	52
100-5-0.75	0.15 %	1/3	1 %	0.02 %	69	0 %	0.02 %	71
100-10-0.25	0.50 %	3/3	93 %	0.06 %	47	63 %	0.01 %	34
100-10-0.5	0.35 %	2/3	23 %	0.01 %	72	0 %	0.03 %	63
100-10-0.75	0.13 %	1/3	15 %	0.02 %	85	0 %	0.04 %	85
100-30-0.25	0.61 %	2/3	58 %	0.02 %	147	0 %	0.12 %	107
100-30-0.5	0.28 %	3/3	63 %	0.01 %	232	8 %	0.02 %	179
100-30-0.75	0.08 %	1/3	25 %	0.01 %	295	0 %	0.03 %	183
250-30-0.25	0.63 %	0/3	0 %	0.04 %	414	N/T	N/T	N/T
250-30-0.5	0.37 %	0/3	0 %	0.06 %	592	N/T	N/T	N/T
250-30-0.75	0.13 %	0/3	0 %	0.06 %	835	N/T	N/T	N/T

even better solution. For each utility measure, the first row gives the percentage of runs where our algorithm found a solution as least as good as the best solution given by CPLEX. The average relative gaps given in the second row are computed *only over those runs* where the best solution found was worse than that given by CPLEX. The third row contains the average CPU time in seconds.

Instances with 100 **items:** For $K = 5$ and $K = 10$ we used a number of 100 ants and a maximum number of iterations of 300, while, in order to obtain good solutions for $K = 30$, the number of ants needed to be raised to 150.

The tests showed that, although the pheromone level can be very small due to our parametrization, without the influence of the accumulated pheromone the algorithm is unable to find the optimal solution and to obtain on average similarly small relative gaps as. This is mainly due to the lack of exploration when using the heuristic obtained by setting $\alpha = 0$.

The table confirms the superiority of the difference utility measure over the simple utility measure that has already been discussed above. The smaller running times in case of the simple measure are mainly due to the repetition of sub-optimal solutions: As the objective function value for already found solutions are not recomputed, the computing time needed is consequently smaller.

The increase of the running time with increasing number of scenarios is of cause due to the increase in computation that needs to be done especially to evaluate the objective function, i.e. to solve K knapsack problems.

Table 4.2 indicates that our algorithm has much more problems to find the optimal solutions (or at least the same solution as CPLEX after 2 hours) of instances with tightness 0.75 than of instances with tightness 0.25. This can be explained in a simplified manner as follows: In each iteration the ants have to decide whether to choose another item or to stop the construction. This decision is,

among others, based on a greedy ordering of the items already in the knapsack, and the more items added to the knapsack, the more "inaccurate" the ordering might become. Moreover, in a larger knapsack more items fit which means that the randomized construction procedure takes longer, and is thus naturally more susceptible to "false decisions". This clearly has an influence on the probability to find the optimal solution. Note however, that the average relative gaps do not significantly change with increasing tightness.

The longer construction times with higher tightness are also reflected in the longer running times of the algorithm.

Our algorithm was able to produce for most of the instances the same solutions as CPLEX, in much shorter running time. Moreover, the table shows that with increasing K the performance of our algorithm improves relatively to the performance of CPLEX. More precisely, as with increasing K CPLEX has more difficulty to find the optimal solution in 2h running time, the percentage of runs where our algorithm finds (in much less computing time) a solution equal or better than CPLEX increases. Moreover, while for both $K = 5$ and $K = 10$ our algorithm found for only one of nine instances a solution better than that given by CPLEX, we were able to produce better solutions for 4 of the nine instances with $K = 30$. Table 4.2 also shows, that the average relative gap between the solution produced by CPLEX and the best solution found by our algorithm is quite small (mostly between 0.02% and 0.01%), and does not significantly increase with K.

Our algorithm was able to produce for most of the instances the same solutions as CPLEX, in much shorter running time. Moreover, the table shows that with increasing K the performance of our algorithm improves relatively to the performance of CPLEX. More precisely, as with increasing K CPLEX has more difficulty to find the optimal solution in 2h running time, the percentage of runs where our algorithm finds (in much less computing time) a solution equal or better than CPLEX increases. Moreover, while for both $K = 5$ and $K = 10$ our algorithm found for only one of nine instances a solution better than that given by CPLEX, we were able to produce better solutions for four of the nine instances with $K = 30$. Table 4.2 also shows, that the average relative gap between the solution produced by CPLEX and the best solution found by our algorithm is quite small (mostly between 0.02% and 0.01%), and does not significantly increase with K.

Instances with 250 **items:** For instances with 250 items only the difference measure was tested (N/T = not tested). Table 4.2 shows that our algorithm already reaches its limit at this dimension concerning the CPU-time as well as the availability to find optimal solutions. Although the average relative gaps are still rather small, the algorithm was incapable of finding optimal solutions. Note that we needed to decrease the number of ants to 50 and increase the maximum number of iterations to 600 to obtain the shown results. This is clearly due to the importance of exploitation as discussed above that leads to the inability

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

of the algorithm to explore the much bigger solution space. To be able to solve problems with higher number of items better utility ratios will clearly be needed.

5 Future Work

Our numerical tests have confirmed what one would naturally expect, i.e. that the running time of the ACO-algorithm increases with the number of scenarios K. For instances with a high number of scenarios sampling should thus be considered. This means that at each iteration a set of scenarios is sampled whose cardinality is smaller than K. By increasing the number of sampled scenarios during the iterations convergence might be achieved. Moreover, one obtains a natural additional diversification of the produced solutions (see [3] for more details). However, the analysis of the algorithm would be completely different, which is why we thought this approach out of scope for this primary study.

A possibility to decrease running time for higher number of items is to use an approximate knapsack algorithm when evaluating the objective function, instead of an exact one. Once again this would entail an additional diversification.

Our numerical tests have shown that the ACO-algorithm proposed in this paper is able to produce, in much less computing time, solutions for instances of 100 items of similar quality as CPLEX in 2h. Moreover, with increasing number of scenarios, our algorithm clearly outperforms CPLEX concerning both running time and solution quality. Although we think that an ACO-algorithm is a natural choice to solve $TSKP$s (see introduction), the next step clearly consists in comparing it with other metaheuristics. Results from this study such as the difference measure might be useful for other algorithms as well.

References

1. Benoist, T., Bourreau, E., Rottembourg, B.: Towards stochastic constraint programming: A study of online multi-choice knapsack with deadlines. In: Walsh, T. (ed.) 7th International Conference on Principles and Practice of Constraint Programming (CP '01), pp. 61–76. Springer, London (2001)
2. Bhalgat, A., Goel, A., Khanna S.: Improved Approximation Results for Stochastic Knapsack Problems. In: D. Randall (ed.) SODA, 1647–1665. SIAM (2011)
3. Bianchi, L., Dorigo, M., Gambardella, L.M., Gutjahr, W.J.: A survey on metaheuristics for stochastic combinatorial optimization. Natural Computing: an international journal 8(2), 239–287 (2009)
4. Chu, P.C., Beasley, J.E.: A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics 4, 63–86 (1998)
5. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Trans Evolutionary Computing 1(1), 53–66 (1997)
6. Fidanova, S.: Ant Colony Optimization for Multiple Knapsack Problem and Model Bias. In: Margenov, S., Vulkov, L.G., Wasniewski, J. (eds.) Numerical Analysis and Its Applications. LNCS vol. 3401, pp. 280–287. Springer, Berlin Heidelberg (2005)
7. Gaivoronski, A.A., Lisser, A., Lopez, R., Hu, X.: Knapsack problem with probability constraints. Journal of Global Optimization 49(3), 397–413 (2010)

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

8. Goyal, V., Ravi, R.: A PTAS for the chance-constrained knapsack problem with random item sizes. Operations Research Letters 38(3), 161–164 (2010)
9. Ke, L., Feng, Z., Ren, Z., Wei, X.: An ant colony optimization approach for the multidimensional knapsack problem. Journal of Heuristics 16(1), 65–83 (2010)
10. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin Heidelberg (2004)
11. Kelly, T.: Combinatorial Auctions and Knapsack Problems. In: Proc. of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Vol. 3, 1280–1281. IEEE Computer Society, Washington, DC, USA (2004)
12. Kleywegt, A.J., Shapiro, A., Homem-de-Mello, T.: The sample average approximation method for stochastic discrete optimization. SIAM Journal on Optimization 12(2), 479–502 (2002)
13. Kosuch, S.: Towards an Ant Colony Optimization algorithm for the Two-Stage Knapsack problem. In: VII ALIO/EURO Workshop on Applied Combinatorial Optimization, pp. 89–92. http://paginas.fe.up.pt/~agomes/temp/alio_euro_2011/uploads/Conference/ALIO-EURO_2011_proceedings_v2.pdf#page=101 (Accessed 29. August 2011) (2011)
14. Leguizamon, G., Michalewicz, M.: A New Version of Ant System for Subset Problems. In: Evolutionary Computation 1999 (CEC 99), pp. 1459–1464 (1999)
15. Maniezzo, V., Gambardella, L.M., de Luigi, F.: Ant Colony Optimization. In: Onwubolu, G.C., Babu, B.V. (eds.) New Optimization Techniques in Engineering. Chapter 5, pp. 101–117. Springer, Berlin Heidelberg (2004)
16. Matthews, D.C.: Improved Lower Limits for Pheromone Trails in Ant Colony Optimization. In: Proceedings of the 10th international conference on Parallel Problem Solving from Nature, pp. 508–517. Springer, Berlin Heidelberg (2008)
17. Merzifonluoglu, Y., Geunes, J., Romeijn, H.: The static stochastic knapsack problem with normally distributed item sizes. Mathematical Programming (Online First) (2011)
18. Pisinger, D.: A Minimal Algorithm for the 0-1 Knapsack Problem. Operations Research 45(5), 758–767 (1997)
19. David Pisinger's optimization codes, http://www.diku.dk/hjemmesider/ansatte/pisinger/codes.html (Accessed 29. August 2011)
20. Puchinger, J., Raidl, G.R., Pferschy, U.: The Multidimensional Knapsack Problem: Structure and Algorithms. INFORMS Journal On Computing 22(2), 250–265 (2010)
21. Shapiro, A., Dentcheva, D., Ruszczynski, A.: Lectures on Stochastic Programming: Modeling and Theory. MPS/SIAM Series on Optimization 9. SIAM-Society for Industrial and Applied Mathematics (2009)

An Improved Memetic Algorithm for the Antibandwidth Problem*

Eduardo Rodriguez-Tello and Luis Carlos Betancourt

CINVESTAV-Tamaulipas, Information Technology Laboratory.
Km. 5.5 Carretera Victoria-Soto La Marina, 87130 Victoria Tamps., MEXICO
{ertello,lbetancourt}@tamps.cinvestav.mx

Abstract. This paper presents an Improved Memetic Algorithm (IMA) designed to compute near-optimal solutions for the antibandwidth problem. It incorporates two distinguishing features: an efficient heuristic to generate a good quality initial population and a local search operator based on a Stochastic Hill Climbing algorithm. The most suitable combination of parameter values for IMA is determined by employing a tunning methodology based on Combinatorial Interaction Testing. The performance of the fine-tunned IMA algorithm is investigated through extensive experimentation over well known benchmarks and compared with an existing state-of-the-art Memetic Algorithm, showing that IMA consistently improves the previous best-known results.

Key words: Memetic Algorithms, Antibandwidth Problem, Combinatorial Interaction Testing, Parameter Tunning

1 Introduction

The *antibandwidth* problem was originally introduced as the *separation number* problem by Leung et al. in connection with the multiprocessor scheduling problem [1]. Later, it has also received the name of *dual bandwidth* [2]. This combinatorial optimization problem consists in finding a labeling for the vertices of a graph $G(V, E)$, using distinct integers $1, 2, \ldots, |V|$, so that the minimum absolute difference between labels of adjacent vertices is maximized.

There exist practical applications of the antibandwidth problem which arise in various fields. Some examples are: radio frequency assignment problem [3], channels assignment and T-coloring problems [4], obnoxious facility location problem [5] and obnoxious center problem [6, 2].

The antibandwidth problem can be formally stated as follows. Let $G(V, E)$ be a finite undirected graph, where V ($|V| = n$) defines the set of vertices and $E \subseteq V \times V = \{\{i, j\} : i, j \in V\}$ is the set of edges. Given a bijective labeling

* This research work was partially funded by the following projects: CONACyT 99276, Algoritmos para la Canonización de Covering Arrays; 51623 Fondo Mixto CONACyT y Gobierno del Estado de Tamaulipas.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

function for the vertices of G, $\varphi : V \rightarrow \{1, 2, \ldots, n\}$, the antibandwidth for G with respect to the labeling φ is defined as:

$$\mathrm{AB}_{\varphi}(G) = \min\{|\varphi(i) - \varphi(j)| : (i, j) \in E\} \quad (1)$$

Then the antibandwidth problem consists in finding a labeling (solution) φ^* for which $\mathrm{AB}_{\varphi^*}(G)$ is maximized, i.e.,

$$\mathrm{AB}_{\varphi^*}(G) = \max\{\mathrm{AB}_{\varphi}(G) : \varphi \in \mathscr{L}\} \quad (2)$$

where $\mathscr{L}$ is the set of all possible labeling functions.

Leung et al. have shown that finding the maximum antibandwidth of a graph is NP-hard for general graphs [1]. Therefore, there is a need for heuristics to address this problem in reasonable time since it is unlikely that exact algorithms running in polynomial time exist for solving it in the general case.

This paper aims at developing an Improved Memetic Algorithm (IMA) for finding near-optimal solutions for the antibandwidth problem. To achieve this, the proposed IMA algorithm incorporates two distinguishing features: a fast heuristic to create a good quality initial population and a local search operator based on a Stochastic Hill Climbing algorithm. Through the use of a tunning methodology, based on Combinatorial Interaction Testing [7], the combination of both components and parameter values for IMA was determined to achieve the best trade-off between solution quality and computational effort. The performance of IMA is assessed with a test-suite, composed by 30 benchmark instances taken from the literature. The computational results are reported and compared with previously published ones, showing that our algorithm is able to consistently improve the previous best-known solutions for the selected benchmark instances.

The rest of this paper is organized as follows. In Sect. 2, a brief review is given to present some representative solution procedures for the antibandwidth problem. Then, the components of our Improved Memetic Algorithm are discussed in detail in Sect. 3. Two computational experiments are presented in Sect. 4. The first one is dedicated to determine the best parameter settings for IMA, while the second carries out a performance comparison of IMA with respect to an existing state-of-the-art Memetic Algorithm. Finally, the last section summarizes the main contributions of this work and presents some possible directions for future research.

2 Relevant Existing Procedures

Because of the theoretical and practical importance of the antibandwidth problem, much research has been carried out in developing effective heuristics for it. Most of the previous published work on the antibandwidth problem was devoted to the theoretical study of its properties for finding optimal solutions for specific cases. Polynomial time exact algorithms are known for solving some special instances of the antibandwidth problem: paths, cycles, special trees, complete and complete bipartite graphs, meshes, tori and hypercubes [8, 2, 9–13].

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

The work of Bansal and Srivastava [14] in an exception. They proposed a Memetic Algorithm, called MAAMP, which starts by constructing an initial population through the use of a label assignment heuristic, called LAH. It builds a level structure of a graph using a random breadth first search. Then, it randomly choses to start the labeling process either by the even or the odd levels of the structure. The vertices of the graph belonging to the same level are labeled one at a time in a greedy manner. MAAMP continues performing a series of cycles called generations. At each generation, selection for mating is done by applying a binary tournament operator in such a way that each individual in the parents population participates in exactly two tournaments. Children are generated using a unary reproduction operator that constructs a level structure of a graph by employing an intermediate breadth first search to produce a new labeling. Then, mutation based on swapping two randomly selected labels is performed over the solutions in the children population. Finally, the population is updated when each child competes with its respective parent and the best of them becomes the parent for the next generation. This process repeats until the Memetic Algorithm ceases to make progress, i.e., when a better solution is not produced in a predefined number of successive generations. The authors argued that MAAMP was able to obtain optimal solutions for standard graphs like paths, cycles, d-dimensional meshes, tori, hypercubes and complement of power graphs. However, they recognize that their algorithm did not reach the optimal antibandwidth in the case of unbalanced trees and complete binary trees. Furthermore, Bansal and Srivastava only present detailed results of their experiments for a set of 30 random connected graphs.

3 An Improved Memetic Algorithm

In this section we present the implementation details of the key components of an Improved Memetic Algorithm (IMA) for solving the antibandwidth problem. For some of these components different possibilities were analyzed (see Sect. 4.2) in order to find the combination which offers the best quality solutions at a reasonable computational effort.

3.1 Search Space, Representation and Fitness Function

Given a graph $G = (V, E)$ with vertex set V ($|V| = n$) and edge set E. The search space $\mathscr{L}$ for the antibandwidth problem is composed of all possible labelings (solutions) from V to $\{1, 2, ..., n\}$, i.e. there exist $n!/2$ possible labelings for a graph with n vertices[1].

In our IMA a labeling φ is represented as an array l of integers with length n, which is indexed by the vertices and whose i-th value $l[i]$ denotes the label assigned to the vertex i. The fitness $\mathrm{AB}_\varphi(G)$ of the labeling φ is evaluated by using (1).

[1] Because each one of the $n!$ labelings can be reversed to obtain the same antibandwidth.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

3.2 General Procedure

Our IMA implementation starts building an initial population P, which is a set of configurations having a fixed constant size $|P|$. Then, it performs a series of cycles called generations. At each generation, assuming that $|P|$ is a multiple of four, the population is randomly partitioned into ($|P|$ mod 4) groups of individuals. Within each group, the two most fit individuals are chosen to become the parents in a recombination operator. The resulting offspring are mutated, then they are improved by using a local search operator for a fixed number of iterations L. Finally, the population is updated by applying a survival selection strategy.

The iterative process described above stops when a predefined maximum number of generations ($maxGenerations$) is reached. Algorithm 1 presents the pseudo code of IMA.

Algorithm 1: Improved Memetic Algorithm (IMA).

```
IMA(A graph G(V, E))
begin
    P ← initPopulation(|P|)
    while not stopCondition() do
        for i ← 1 to offspring do
            // select two parents a, b ∈ P
            (a, b) ← selectParents(P)
            c ← recombineIndividuals(a, b)
            c' ← mutation(c)
            c'' ← localSearch(c', L)
            insertIndividual(c'', P)
        end
        P ← updatePopulation(P)
    end
    return The best solution found
end
```

3.3 Initializing the Population

After comparing different heuristics for constructing labelings for the antibandwidth problem we have decided to use a variant of the heuristic LAH reported in [14].

Our labeling heuristic constructs a level structure of a graph using a breadth first search procedure exactly like LAH does. Then, the vertices are labeled one at a time following the order of this level structure and starting randomly either by the even or the odd levels. The main difference of our labeling heuristic with respect to LAH is that we do not select the next vertex to label in a greedy manner.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

In our preliminary experiments we have found that a good balance between diversity and quality for the initial population is reached using a labeling built with our heuristic combined with $|P| - 1$ distinct randomly generated labelings.

3.4 Selection Mechanisms

In this implementation mating selection ($selectParents(P)$) prior to recombination is performed by tournament selection, while one of the following standard schemes is used for the survival selection ($updatePopulation(P)$): $(\mu + \lambda)$, (μ, λ) and (μ, λ) with elitism [15].

3.5 Recombination Operators

The recombination (crossover) operator plays a very important role in any Memetic Algorithm. Indeed, it is this operator that is responsible for creating potentially promising individuals. There are several crossover operators reported in the literature that can be applied to permutation problems [16–19].

In our computational experiments, described in Section 4.2, we compare the following three crossover operators in order to identify the most suitable one for the antibandwidth problem.

The *Cycle Crossover* (CX) operator [18], which preserves the information contained in both parents in the sense that all elements of the offspring are taken from one of the parents, i.e., CX does not perform any implicit mutation. The *Partially Matched Crossover* (PMX) operator [17], designed to preserve absolute positions from both parents. The Order Crossover (OX) operator [16], which is implemented to inherit the elements between two randomly selected crossover points, inclusive, from the first parent in the same order and position as they appeared in it. The remaining elements are inherited from the second parent in the order in which they appear in that parent, beginning with the first position following the second crossover point and skipping over all elements already present in the offspring.

3.6 Mutation Operator

In our IMA implementation the mutation operator was designed to introduce diversity into the population. It starts receiving a configuration (labeling) c produced by the recombination operator. Then, every label in c is exchanged with another randomly selected one with certain probability (mutation probability). Finally, these exchange operations allow to produce a new labeling c'.

3.7 Local Search Operator

The purpose of the local search (LS) operator $localSearch(c', L)$ is to improve a configuration c' produced by the mutation operator for a maximum of L iterations before inserting it into the population. In general, any local search method

can be used. In our implementation, we have decided to use a Stochastic Hill Climbing (SHC) algorithm because it only needs as parameter the maximum number of iterations.

In our SHC-based LS operator the neighborhood $\mathcal{N}(\varphi)$ of a configuration φ is such that for each $\varphi \in \mathscr{L}$, $\varphi' \in \mathcal{N}(\varphi)$ if and only if φ' can be obtained by exchanging the labels of any pair of vertices from φ. The main advantage of this neighborhood function is that it allows an incremental fitness evaluation of the neighboring solutions.

The LS operator starts from the current solution $c' \in \mathscr{L}$ and at each iteration randomly generates a neighboring solution $c'' \in \mathcal{N}(c')$. The current solution is replaced by this neighboring solution if the fitness of c'' improves or equals that of c'. The algorithm stops when it reaches a predefined maximum number of iterations L, and returns the best labeling found.

4 Computational Experiments

In this section two main experiments accomplished to evaluate the performance of the proposed IMA algorithm and some of its components are presented. The objective of the first experiment is to determine both a component combination, and a set of parameter values which permit IMA to attain the best trade-off between solution quality and computational effort. The purpose of the second of our experiments is to carry out a performance comparison of IMA with respect to an existing state-of-the-art Memetic Algorithm called MAAMP [14].

For these experiments IMA was coded in C and compiled with *gcc* using the optimization flag -*O*3. It was run sequentially into a CPU Xeon at 2.67 GHz, 1 GB of RAM with Linux operating system. Due to the non-deterministic nature of the algorithm, 30 independent runs were executed for each of the selected benchmark instances in each experiment.

4.1 Benchmark Instances and Comparison Criteria

The test-suite that we have used in our experiments is the same proposed by Bansal and Srivastava [14]. It consists of 30 undirected planar graphs taken from the Rome set which are employed in graph drawing competitions. All of them have a number of vertices between 50 and 100. These instances are publicly available at: `http://www.graphdrawing.org/data`.

The criteria used for evaluating the performance of the algorithms are the same as those used in the literature: the best antibandwidth found for each instance (bigger values are better) and the expended CPU time in seconds.

4.2 Components and Parameters Tunning

Optimizing parameter settings is an important task in the context of algorithm design. Different procedures have been proposed in the literature to find the most

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Table 1. Input parameters of the IMA algorithm and their selected values.

$\|P\|$	Cx	$ProbCx$	$ProbMuta$	$Survival$	$ProbLS$	L
40	CX	0.70	0.00	$(\mu + \lambda)$	0.05	1000
80	PMX	0.80	0.05	(μ, λ)	0.10	5000
120	OX	0.90	0.10	(μ, λ) with elitism	0.15	10000

suitable combination of parameter values [20–22]. In this paper we employ a tunning methodology, previously reported in [23], which is based on Combinatorial Interaction Testing (CIT) [7]. We have decided to use CIT, because it allows to significantly reduce the number of tests (experiments) needed to determine the best parameter settings of an algorithm. Instead of exhaustive testing all the parameter value combinations of the algorithm, it only analyzes the interactions of t (or fewer) input parameters by creating interaction test-suites that include at least once all the t-way combinations between these parameters and their values.

Covering arrays (CAs) are combinatorial designs which are extensively used to represent those interaction test-suites. A covering array, $\mathrm{CA}(N; t, k, v)$, of size N, strength t, degree k, and order v is an $N \times k$ array on v symbols such that every $N \times t$ sub-array includes, at least once, all the ordered subsets from v symbols of size t (t-tuples) [24]. The minimum N for which a $\mathrm{CA}(N; t, k, v)$ exists is the *covering array number* and it is defined according to the following expression: $\mathrm{CAN}(t, k, v) = min\{N : \exists\, \mathrm{CA}(N; t, k, v)\}$.

CAs are used to represent an interaction test-suite as follows. In an algorithm we have k input parameters. Each of these has v values or levels. An interaction test-suite is an $N \times k$ array where each row is a test case (i.e., a covering array). Each column represents an input parameter and a value in the column is the particular configuration. This test-suite allows to cover all the t-way combinations of input parameter values at least once. Thus, the costs of tunning the algorithm can be substantially reduced by minimizing the number of test cases N in the covering array. Next, we present the details of the tunning process, based on CIT, for the particular case of our IMA algorithm.

First, we have identified $k = 7$ input parameters used for IMA: population size $|P|$, crossover operator Cx, crossover probability $ProbCx$, mutation probability $ProbMuta$, survival selection strategy $Survival$, local search probability $ProbLS$ and maximum number of local search iterations L. Based on some preliminary experiments, $v = 3$ reasonable values (shown in Table 1) were selected for each one of those input parameters.

We have constructed the smallest possible covering array $\mathrm{CA}(40; 3, 7, 3)$, shown (transposed) in Table 2, by using the Memetic Algorithm reported in [25]. This covering array can be easily mapped into an interaction test-suite by replacing each symbol from each column to its corresponding parameter value. For instance, we can map 0 in the first column (the first line in Table 2) to $|P| = 40$, 1 to $|P| = 80$ and 2 to $|P| = 120$. The resulting interaction test-suite contains, thus, 40 test cases (parameter settings) which include at least once all the 3-way combinations between IMA's input parameters and their values[2].

[2] In contrast, with an exhaustive testing which contains $3^7 = 2187$ test cases.

Table 2. Covering array CA$(40; 3, 7, 3)$ representing an interaction test-suite for tunning IMA (transposed).

0	2	2	1	0	1	1	2	2	0	0	1	2	0	0	0	1	1	1	0	2	0	2	2	2	0	2	0	0	1	1	1	2	0	1	2	1	1	2	0
1	0	2	1	2	1	1	1	1	2	0	2	0	2	0	1	0	2	0	1	1	0	0	2	1	0	2	0	1	0	1	0	1	2	2	0	2	2	2	0
1	0	0	2	0	2	0	2	0	2	1	2	2	0	1	2	1	1	0	0	1	2	1	2	0	1	2	0	2	0	1	2	1	1	0	0	2	1	1	0
0	2	0	1	2	0	0	2	1	0	2	2	0	1	0	1	0	0	2	2	0	1	2	1	1	1	0	0	2	1	2	2	1	2	1	1	0	1	2	0
0	2	0	0	1	2	1	0	1	1	0	1	1	0	1	1	0	2	0	0	2	0	1	2	2	2	1	2	2	1	1	2	0	2	2	0	0	1	0	1
0	1	0	2	2	0	1	0	0	0	1	1	1	1	1	2	2	1	0	2	2	0	0	0	1	0	2	2	0	1	1	2	2	1	0	2	2	1	0	2	2
2	0	0	2	1	0	1	1	0	2	1	0	2	2	0	1	0	2	2	0	1	0	1	1	2	2	0	1	2	2	2	1	0	0	0	1	1	1	2	2

Each one of those 40 test cases was used to executed 30 times the IMA algorithm over a subset of 6 representative graphs,[3] selected from the benchmark instances described in Sect. 4.1. The data generated by these 7200 executions is summarized in Fig. 1, which depicts the average antibandwidth reached by each test case over the 6 selected graphs, as well as the average CPU time expended.

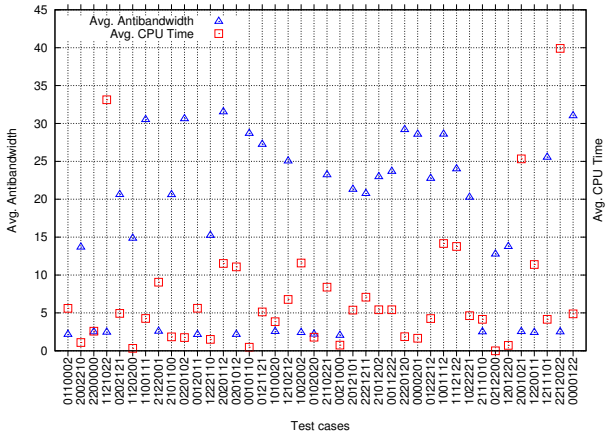


Fig. 1. Average results obtained in the tuning experiments using 40 parameter value combinations over a subset of 6 representative graphs.

From this graphic we have selected the 5 test cases which yield the best results. Their average antibandwidth and the average CPU time in seconds are presented in Table 3. This table allowed us to observe that the parameter setting giving the best trade-off between solution quality and computational effort corresponds to the test case number 40 (shown in bold). The best average antibandwidth with an acceptable speed is reached with the following input parameter values: population size $|P| = 40$, Cycle Crossover (CX) operator, crossover probability $ProbCx = 0.90$, mutation probability $ProbMuta = 0.00$, (μ, λ) survival selection strategy, local search probability $ProbLS = 0.15$ and maximum number of local search iterations $L = 10000$. These values are thus used in the experimentation reported in the next section.

[3] One graph for each size $50 \leq |V| \leq 100$ in the original set.

Table 3. Results from the 5 best parameter test cases in the tuning experiments.

Num.	Test case	Avg. antibandwidth	Avg. CPU time
13	2020112	31.527	11.802
40	**0000122**	**31.011**	**5.209**
10	0220102	30.616	2.112
7	1100111	30.511	4.611
27	2220120	29.183	2.223

4.3 Comparison Between IMA and MAAMP

In this experiment a performance comparison of the best bounds achieved by IMA with respect to those produced by the MAAMP algorithm [14] was carried out over the test-suite described in Sect. 4.1.

Table 4 displays the detailed computational results produced by this experiment. The first three columns in the table indicate the name of the graph as well as its number of vertices and edges. The theoretical upper bound (C^*) reported in [14] for those graphs is presented in Column 4. The best (C) and average ($Avg.$) antibandwidth attained by MAAMP in 10 executions and its average CPU time in seconds are listed in columns 5 to 7. These results were taken directly from [14], where a Pentium 4 at 3.2 GHz and 1 GB of RAM system was used to execute the algorithm. Next four columns provide the best (C), average ($Avg.$) and standard deviation ($Dev.$) of the antibandwidth found by IMA over 30 independent executions and the average CPU time (T) in seconds expended. The running times from MAAMP and IMA cannot be directly compared because they were executed on different computational platforms. Nevertheless, we have scaled, by a factor of 2.71, our execution times according to the Standard Performance Evaluation Corporation[4] in order to present them in a normalized form in Column 12 ($\overline{T}$). Finally, the difference (Δ_C) between the best result produced by our IMA algorithm and that achieved by MAAMP is depicted in the last column.

Analyzing the data presented in Table 4 lead us to the following main observations. First, the solution quality attained by the proposed IMA algorithm is very competitive with respect to that produced by the existing Memetic Algorithm called MAAMP [14], since IMA provides solutions whose costs (antibandwidth) are closer to the theoretical upper bounds (compare Columns 4, 5 and 8). Indeed, IMA consistently improves the best antibandwidth found by MAAMP, obtaining an average amelioration of $\Delta_C = -4.93$.

Second, one observes that for the selected instances the antibandwidth found by IMA presents a relatively small standard deviation (see Column $Dev.$). It is an indicator of the algorithm's precision and robustness since it shows that in average the performance of IMA does not present important fluctuations.

Third, we can notice that MAAMP is the most time-consuming algorithm. It uses an average of 257.79 seconds for solving the 30 selected instances, while IMA employs only 25.82 seconds (see column $\overline{T}$).

[4] `http://www.spec.org`

Table 4. Performance comparison between MAAMP and IMA over 30 undirected planar graphs from the Rome set.

				MAAMP			IMA					
Graph	$\|V\|$	$\|E\|$	C^*	C	$Avg.$	T	C	$Avg.$	$Dev.$	T	$\overline{T}$	Δ_C
ug1-50		64		18	17.40	65.28	21	20.97	0.03	8.51	23.07	-3
ug2-50		66		18	16.80	33.51	21	20.00	0.21	2.38	6.46	-3
ug3-50	50	63	24	17	16.80	52.04	21	21.00	0.00	3.61	9.77	-4
ug4-50		70		19	18.40	59.55	21	20.37	0.52	2.74	7.42	-2
ug5-50		62		20	19.80	83.44	22	21.40	0.25	12.33	33.41	-2
ug1-60		79		21	20.20	155.98	25	24.13	0.12	3.59	9.72	-4
ug2-60		81		23	21.80	21.80	24	23.53	0.26	2.41	6.54	-1
ug3-60	60	84	29	22	20.20	116.12	24	23.33	0.30	7.27	19.71	-2
ug4-60		79		22	21.20	235.20	25	24.27	0.20	6.32	17.12	-3
ug5-60		80		21	20.80	70.20	25	23.87	0.19	3.26	8.84	-4
ug1-70		98		26	23.80	354.52	28	26.83	0.35	6.13	16.60	-2
ug2-70		82		28	15.80	54.20	32	31.30	0.22	9.68	26.22	-4
ug3-70	70	82	34	27	26.20	116.31	31	30.20	0.17	5.72	15.49	-4
ug4-70		97		23	22.60	74.50	29	28.07	0.06	15.42	41.77	-6
ug5-70		88		26	25.50	323.45	29	28.47	0.26	13.03	35.31	-3
ug1-80		92		34	33.60	61.07	37	36.13	0.26	9.31	25.24	-3
ug2-80		93		31	30.20	226.90	35	34.07	0.34	14.26	38.64	-4
ug3-80	80	95	39	28	27.20	138.68	35	34.23	0.46	9.72	26.35	-7
ug4-80		101		27	25.20	582.50	34	32.80	0.23	11.05	29.95	-7
ug5-80		94		27	25.80	190.52	34	33.10	0.51	10.74	29.11	-7
ug1-90		102		32	29.80	150.40	39	38.27	0.41	8.78	23.78	-7
ug2-90		114		35	34.60	469.62	38	36.73	0.34	7.90	21.41	-3
ug3-90	90	108	44	31	29.80	316.29	40	38.83	0.49	16.84	45.64	-9
ug4-90		99		34	33.30	381.20	40	39.23	0.25	11.77	31.90	-6
ug5-90		104		34	31.80	815.47	39	38.50	0.47	10.50	28.46	-5
ug1-100		114		34	32.20	499.20	44	43.07	0.34	17.61	47.71	-10
ug2-100		114		40	38.40	715.90	44	42.80	0.23	11.22	30.40	-4
ug3-100	100	116	49	33	31.60	256.40	43	42.33	0.57	15.10	40.92	-10
ug4-100		122		35	33.80	419.80	43	42.13	0.33	15.23	41.27	-8
ug5-100		125		31	30.60	693.65	42	40.60	1.14	13.41	36.33	-11
	Avg.			27.23		257.79	32.17		0.32	9.53	25.82	-4.93

The outstanding results achieved by IMA are better illustrated in Fig. 2. The plot represents the studied instances (ordinate) against the best solution (antibandwidth) attained by the compared algorithms (abscissa). The theoretical upper bounds for these graphs are shown with squares, the previous best-known solutions provided by MAAMP [14] are depicted as circles, while the bounds computed with our IMA algorithm are shown as triangles. From this figure it can be seen that IMA consistently outperforms MAAMP, achieving for certain instances, like the graph *ug5-100*, an important increase in solution cost (Δ_C up to -11).

Thus, as this experiment confirms, our IMA algorithm is more effective than the existing Memetic Algorithm, called MAAMP.

5 Conclusions and Further Work

In this paper, an Improved Memetic Algorithm (IMA) designed to compute near-optimal solutions for the antibandwidth problem was presented. IMA's components and parameter values were carefully determined, through the use of a

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

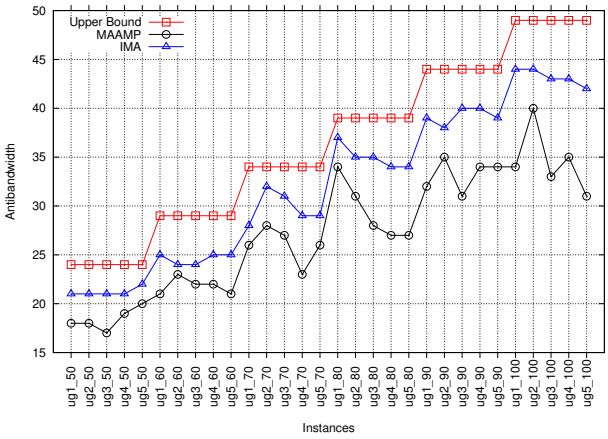


Fig. 2. Performance comparison between MAAMP and IMA with respect to the theoretical upper bounds.

tunning methodology based on Combinatorial Interaction Testing [7], to yield the best solution quality in a reasonable computational time.

The practical usefulness of this fine-tunned IMA algorithm was assessed with respect to an existing state-of-the-art Memetic Algorithm, called MAAMP [14] over a set of 30 well-known benchmark graphs taken from the literature. The results show that our IMA algorithm was able to consistently produce labelings with higher antibandwidth values than those furnished by MAAMP. Furthermore, IMA achieves those results by employing only a small fraction (10.01%) of the total time used by MAAMP.

This work opens up a range of possibilities for future research. Currently we are interested on developing a Multimeme Algorithm [26] based on the Memetic Algorithm presented here in order to efficiently solve the antibandwidth problem for bigger graphs with different topologies.

References

1. Leung, J., Vornberger, O., Witthoff, J.: On some variants of the bandwidth minimization problem. SIAM Journal on Computing **13**(3) (1984) 650–667
2. Yixun, L., Jinjiang, Y.: The dual bandwidth problem for graphs. Journal of Zhengzhou University **35**(1) (March 2003) 1–5
3. Hale, W.K.: Frequency assignment: Theory and applications. Proceedings of the IEEE **68**(12) (December 1980) 1497–1514
4. Roberts, F.S.: New directions in graph theory. Annals of Discrete Mathematics **55** (1993) 13–44
5. Cappanera, P.: A survey on obnoxious facility location problems. Technical report, Uni. di Pisa (1999)
6. Burkard, R.E., Donnani, H., Lin, Y., Rote, G.: The obnoxious center problem on a tree. SIAM Journal on Computing **14**(4) (2001) 498–509

7. Cohen, D.M., Dalal, S.R., Parelius, J., Patton, G.C.: The combinatorial design approach to automatic test generation. IEEE Software **13**(5) (1996) 83–88
8. Miller, Z., Pritikin, D.: On the separation number of a graph. Networks **19** (1989) 651–666
9. Yao, W., Ju, Z., Xiaoxu, L.: Dual bandwidth of some special trees. Journal of Zhengzhou University Natural Science Edition **35** (2003) 16–19
10. Calamoneri, T., Missini, A., Török, L., Vrt'o, I.: Antibandwidth of complete k-ary trees. Electronic Notes in Discrete Mathematics **24** (2006) 259–266
11. Török, L.: Antibandwidth of three-dimensional meshes. Electronic Notes in Discrete Mathematics **28** (March 2007) 161–167
12. Raspaud, A., Schröder, H., Sykora, O., Török, L., Vrt'o, I.: Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discrete Mathematics **309**(11) (2009) 3541–3552
13. Wang, X., Wu, X., Dimitrescu, S.: On explicit formulas for bandwidth and antibandwidth of hypercubes. Discrete Applied Mathematics **157**(8) (2009) 1947–1952
14. Bansal, R., Srivastava, K.: Memetic algorithm for the antibandwidth maximization problem. Journal of Heuristics **17**(1) (2011) 39–60
15. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. 1st edn. Springer (2007)
16. Davis, L.: Applying adaptive algorithms to epistatic domains. In: Proceedings of the International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1985) 162–164
17. Goldberg, D.E., Lingle, R.: Alleles, loci, and the travelling salesman problem. In: Proceedings of the 1st. International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum Associates (1985) 154–159
18. Oliver, I.M., Smith, D.J., Holland, J.R.C.: A study of permutation crossover operators on the travelling salesman problem. In: Proceedings of the 2nd. International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum Associates (1987) 224–230
19. Freisleben, B., Merz, P.: A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In: Proceedings of IEEE International Conference on Evolutionary Computation, IEEE Press (1996) 616–621
20. Adenso-Diaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental design and local search. Operations Research **54**(1) (2006) 99–114
21. de Landgraaf, W.A., Eiben, A.E., Nannen, V.: Parameter calibration using meta-algorithms. In: In proceedings of the IEEE Congress on Evolutionary Computation, IEEE Press (2007) 71–78
22. Gunawan, A., Lau, H.C., Lindawati: Fine-tuning algorithm parameters using the design of experiments. Lecture Notes in Computer Science **6683** (2011) In press
23. Gonzalez-Hernandez, L., Torres-Jimenez, J.: MiTS: A new approach of tabu search for constructing mixed covering arrays. Lecture Notes in Artificial Intelligence **6438** (2010) 382–392
24. Colbourn, C.J.: Combinatorial aspects of covering arrays. Le Matematiche **58** (2004) 121–167
25. Rodriguez-Tello, E., Torres-Jimenez, J.: Memetic algorithms for constructing binary covering arrays of strength three. Lecture Notes in Computer Science **5975** (2010) 86–97
26. Krasnogor, N.: Towards robust memetic algorithms. In: Recent Advances in Memetic Algorithms. Volume 166 of Studies in Fuzziness and Soft Computing. Springer (2004) 185–207

Improving State-of-the-Art 3-SAT Solvers using Automatic Design of Algorithms through Evolution

Roland Olsson[1] and Arne Løkketangen[2]

[1] Østfold Unversity College
Halden, Norway
Roland.Olsson@hiof.no
[2] Molde University College
Molde, Norway
Arne.Lokketangen@himolde.no

Abstract. Automatic generation of code is becoming an increasingly important field. This also includes code for combinatorial optimization. We show how such an automated code generation system, ADATE, is able to improve significantly on a state-of-the-art SAT-solver, G2WSAT, by automatically evolving and improving the original program in a controlled fashion. We have chosen SAT - the Satisfiability Problem - as our target problem, as this is a central problem in many real-world applications, and because improving SAT-solvers might have significant economic impact. Our computational testing clearly shows the code improvement, as well as showing that the resulting code can be generalized and used outside its original learning domain.

1 Introduction

There is an increasing focus on generating optimization code automatically. Most efforts in this area consist of composing new programs from known, high-level, components(see [1]). This is in general a limitation as there will be constructs that cannot be generated. (E.g. the system in [6] cannot synthesize the heuristic R-Novelty, see [7]).

The ADATE system is somewhat different, in that programs are synthesized from general low-level programming components.(See [11]). The ADATE system can also be regarded as a hyper-heuristic. According to [2], a *hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve hard computational search problems.* For a recent survey on hyper-heuristics, see [3]. The ADATE system can in this framework be defined as an off-line, heuristic generating, learning hyper-heuristic. A brief description of ADATE is in Section 4. Previously, ADATE has been applied to a tabu search meta-heuristic for boolean optimization problems with great success. (See [9]).

We have chosen to apply ADATE to a state-of-the-art heuristic for Satisfiability (SAT) problems, called G2WSAT. The ADATE system is applied to

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

the source-code of G2WSAT, coded in ML [10], and evolved with a focus on better performance. The reason for choosing SAT is due to its high practical applicability and the abundance of very good solvers.

This introduction is followed in the next section by a description of the Satisfiability problem and some solution methods. Section 3 describes the G2WSAT heuristic. ADATE is described in Section 4 with the necessary adaptations for G2WSAT. Section 5 gives the computational results of the generated code.

2 The Satisfiability Problem

The satisfiability problem is of great practical significance, and good solvers can be of great value. SAT also has significance in that it was the first NP-complete problem, as shown in [4]. For a thorough treatment of the SAT-problem, see [5].

Formally, a SAT problem in propositional logic is to find a set of truth assignments to the variables in a formula consisting of logical variables and the operators *and*, *or* and *not*. In practice, one often constrains the formula to be in CNF (Conjunctive Normal Form). This states that a formula is a conjunction of clauses, where each clause is a disjunction of literals, and a literal is a (possibly negated) logical variable.

We restrict our attention in this paper to random 3-SAT, that is randomly generated SAT-problems where each clause has exactly 3 literals.

3 G2WSAT

There is a plethora of different algorithms and heuristics for SAT-problems. Many of these are tailored to function best on 3-SAT. We have chosen the heuristic G2WSAT as a basis for evolution by ADATE, as it is among the state-of-the-art heuristics for 3-SAT. (see [8]).

G2WSAT works by maintaining two views on the problem. It keeps a list of *promising decreasing variables*, pdv, that is variables that would lead to a decrease in the number of unsatisfied clauses, if flipped. It also keeps a list of currently unsatisfied clauses. If there are any pdv's, choose the one with biggest improvement, preferring the least recently flipped in case of a tie. Otherwise, select an unsatisfied clause at random, and select a variable there according to the heuristic Novelty++. (see again [8]).

4 On ADATE

Ever since its inception 20 years ago [11], ADATE has been inspired by evolution as described by Charles Darwin, whereas no parts of the system are derived from or inspired by other evolutionary computational methodologies or systems.

A key property of Darwinian evolution is that complex entities may be built from very simple ones through a vast number of incremental changes over long

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

periods of time. In ADATE, these changes are transformations of purely functional programs using program transformations specifically designed for program evolution rather than using so-called mutation and crossover that are common in other methods for evolutionary computation. ADATE thus works by systematic program transformations, as described in [11].

ADATE may start with an empy initial program and develop that into a possibly complex final program that contains invented recursive help functions. However, the evolution often runs much faster if it starts from one of the best known programs for the task at hand, in this paper G2WSAT for the SAT problem.

4.1 ADATE for G2WSAT

The ADATE system is used to improve on the very good heuristic G2WSAT. To achieve this, G2WSAT is recoded in ML, and submitted to ADATE as the initial program. This initial program is then subjected to a systematic series of transformations. The performance of each new program is measured by subjecting it to a set of training instances. Promising programs are kept, failures thrown away. To identify the best programs, the runners-up have to be submitted to a separate, larger, validation set of other instances.

When recoding G2WSAT in ML, we used the random 3-SAT instances from the international SAT competition arranged in conjunction with SAT'09 as benchmarks to compare our G2WSAT code with the one provided in the UBC-SAT toolbox developed in [12].

The following three design choices had to be made in order to develop an ADATE specification for improvement of G2WSAT.

1. How many variables should the training instances contain? Should the number of variables be the same for each training instance?
2. Should a synthesized program be run from a randomly chosen starting point or start from a point generated by first running G2WSAT for a number of iterations? How many iterations should we use?
3. How many training instances should there be?

Initially, we tried out several different answers to these questions and after a series of preliminary ADATE runs we arrived at the following design choices.

1. In order to save computation time, the number of variables should be small, but not so small that the instances become trivially easy for G2WSAT. We also considered the hypothetical possibility that automatically generated programs might come to contain parameters that depend on the number of variables and the number of iterations. If that were to be the case, we would choose to find a formula for these parameters a posteriori and not burden ADATE with generating that formula. Therefore, we chose the number of variables to be constant and equal to the smallest number used in the SAT'09 competiton, namely 360.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

After randomly generating a training instance candidate, we ran G2WSAT 100 000 times with 50 000 iterations on the candidate and rejected it if none of these runs could satisfy it.
2. We found that 50 000 G2WSAT iterations were enough to solve reasonably many 360-variable instances. We generated such instances along with a random starting point and then ran G2WSAT for 30 000 iterations. If an instance then was solved, we rejected it, but else we used the instance together with the point found after 30 000 iterations as a training example to be solved with 20 000 more iterations by a synthesized program.
3. Since ADATE basically is a deterministic system, our preliminary experiments employed 500 training examples that were run with the same random seed for each program. However, we then noted severe overfitting problems and decided to evaluate programs with a different random seed each time, which reduced overfitting to such an extent that we could reduce the number of training examples to just 100 and still mostly avoid overfitting.

The best synthesized program, called ADSAT160, that we report on in this paper was found after a wall clock time of 726197 seconds when running 256 INTEL Xeon X5355 CPU cores at very close to 100% CPU utilization the entire time.

5 Computational results

To check the validity and quality of the best generated program, ADSAT160, its performance should be compared to that of G2WSAT.

The program evolution had training and validation instances with 360 variables. Our first set of comparative tests used the same instance size. We used a total of 20.000 instances of random 3-SAT, with 360 variables, with a clause/variable ratio of 4.26 (the transition region).

Some of these test instances are unsatisfiable, and some are too easy to distinguish. We therefore ignored the instances where none of the programs found any solutions in 1000 runs (10.444, presumed infeasible), and those instances where all the 1000 runs of both programs found satisfiable solutions (2.800, presumed too easy). This left 8.356 instances for comparison. See Table 1.

To see if the generated code also would work outside the learning framework, we also generated 20.000 instances with 500 and 1000 variables. Similar statistics as for the 360 variable instances are also shown in Table 1.

Quantitative results. As the program evolution was done with 360 variables, we tested on this first. To see if the generated program also is competitive outside the constraints imposed during generation, we also tested the program on 500 and 1000 variable instances.

We ran all the tests 1000 times, with different random seed, on each of the test instances with both programs. Each test was run for 50,000 iterations.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Table 1. Domination of ADSAT160 over G2WSAT

Variables	#Tests	Unsat	Easy	Remaining	Avg. Dist.	Avg. std. dev.	Total Deviation
360	20,000	10,444	2800	8356	5.7	9.4	67
500	20,000	10,812	185	9002	6.4	10.7	62
1000	20,000	11,097	0	8903	6.7	6.8	41

By counting and comparing the number of satisfied runs (out of 1000 runs) over all the remaining instances for the two programs, we found the results summarized in Table 1.

The #Tests column states the number of test instances for the indicated test instance size. "Avg. Dist" gives the average number of the differences in the number of instances solved (out of the 1000 runs per instance). This shows that ADSAT160 solves around 0.6% more solvable instances than G2WSAT, which is quite good in this highly competitive area. "Avg. std. dev." is the average standard deviation in number of runs (out of 1000) solved per instance for G2WSAT. (The number for ADSAT160 is very similar). Finally. the "Total Deviation" column gives the number of standard deviations of significance required for the two distributions (of the solvers) to be similar. In other words, ADSAT160 is better than G2WSAT with a very high statistical probability. One reason for this high probability is due to the very large number of computational tests run. We recommend other researchers to increase their testing significantly whenever possible, too often the tests are too few to give statistically significant results.

Qualitative results. Since the results of the ADATE transformations are programs, it is usually possible to decipher the generated code. The transformations can be tracked as changes to ancestral programs, and the final program can thus be understood. The number of transformations from the start program to the final one is usually from twenty to several hundred. In the current set of transformations from G2WSAT to ADSAT160, there are two significant transformations.

1. One is a small increase in the diversification probability for the Novelty++ part.
2. The other is in the selection of pdv's. Instead of choosing according to score and time, it selects among these variables more randomly, and with bias towards variables that have most recently become part of the pdv set because they belonged to a previously satisfied clause that became unsatisfied. In other words, it is a top priority to satisfy such a clause in a new way, which obviously is a diversification strategy.

It is not so easy to see why these transformations should result in a superior solver. By analyzing the detailed workings of the generated programs, one

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

might gain new insights into algorithm design, and gain knowledge that can be generalized and applied in other settings.

6 Conclusions

We have shown that state-of-the-art optimization code can be significantly improved by the ADATE system. Of interest is also the transparency of the resulting code, meaning that the resulting mechanisms can be decoded and understood, and thus be used in a larger context.

At the same time, the ADATE system is able to find new search mechanisms, that might be difficult to find for humans, as our creativity is limited by prior experience.

Acknowledgements. We would like to thank *NOTUR - The Norwegian metacenter for computational science* for supplying this project with CPU-time on one of its clusters.

References

1. Burke E.K. , G. Kendall, J. Hart, P. Ross and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology, In: F. Glover and G. Kochenberger (Eds.), Handbook of meta-heuristics, chap 16,, 457 - 474.
2. Burke E.K., M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward. A Classification of Hyper-heuristics Approaches, School of Computer Science and Information Technology, University of Nottingham Computer Science Technical Report No. NOTTCS-TR-SUB-0907061259-5808.
3. Burke E.K., M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. A Survey of Hyper-heuristics, School of Computer Science and Information Technology, University of Nottingham, Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747.
4. Cook S.A. The complexity of theorem-proving procedures, Proceedings of the Third ACM Symposium on Theory of Computing, pp 151 - 158.
5. Du D., J. Gu and P. Pardalos. Satisfiabilitry Problem: Theory and Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol 35.
6. Fukunaga A.S.. Automated Discovery of Local Search Heuristics for Satisfiability Testing, Evolutionary Computation 16(1), pp 31 - 61.
7. Hoos H. On the run-time behaviour of stochastic local search methods for SAT, Proceedings of AAAI, 661 - 666.
8. Li, C.M. and W.Q. Huang. Diversification and determinism in local search for satisfiability. Proceedings of SAT-2005.
9. Løkketangen, A. and R. Olsson. Generating Meta-heuristic Optimization Code using ADATE, Journal of Heuristics.
10. Milner R., M. Tofte, R. Harper and D. MacQueen. The Definition of Standard ML (Revised), MIT Press.
11. Olsson R. Inductive functional programming using incremental program transformation, Artificial Intelligence 1, 55–83.
12. Tompkins D. and H. Hoos. UBCSAT: An Implementation and Experimentation Environment for SLS Algorithms for SAT and MAX-SAT. Theory and Applications of Satisfiability Testing Lecture Notes in Computer Science, 2005, Volume 3542/2005, Springer Verlag.

Adaptive Play in a Pollution Bargaining Game

Vincent van der Goes

Vrije Universiteit Amsterdam, The Netherlands,
vgoes@feweb.vu.nl

Abstract. We apply adaptive play to a simplified pollution game with two players. We find that agents with longer memory paradoxically perform worse in the long run. We interpret this result as an indication that adaptive play may be too restrictive as a model of agent behaviour in this context, although it can serve as a starting point for further research on bounded rationality in pollution games.

Keywords: evolutionary economics, game theory, adaptive play

1 Introduction

Global environmental problems, such as pollution, provide real-world examples of a *social dilemma*. Countries involved in an environmental issue could often benefit from mutual cooperation. However, if they choose to do so, any country in the cooperation might benefit from *free riding*. That is, each country could possibly improve net welfare by defecting and profiting from the efforts at reducing pollution, without reducing its own pollution.
So it is in the mutual interest of countries involved in such an environmental problem to find a way to facilitate cooperation. To this end, countries have signed treaties known as *International Environmental Agreements* (IEA's).
In particular, the case of greenhouse gas emissions is receiving a lot of attention today. Greenhouse gas emission is a form of *transboundary pollution*, i.e. the impacts of emissions are global and do not depend on the location where they originate. As reduction of greenhouse gas emissions is very costly, it provides a good example of a social dilemma. Agreeing on an appropriate IEA has proven difficult in practice.
In the literature this has sparked a discussion about the question why negotiations on reduction of emissions are so difficult and what kind of policy instruments could help on reaching an agreement. For a recent overview of the literature in this area we refer the reader to [2] and [3].
A common way of modelling emission reductions is as a repeated game. The dynamics of a realistic model are highly complex, as there are several complicating factors at play. The game changes over time, as greenhouse gas concentrations build up and new technologies become available.
Also, there is a high degree of uncertainty, especially considering the long time scale at which the climate responds [6]. First of all *systematic uncertainty*: damage as a function of emissions is hard to predict. Secondly *strategic uncertainty*:

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

agents can not predict the behaviour of other agents very well.
In this paper we explore the possibility of modelling bargaining for reduction of emissions of a transboundary pollutant, while assuming only bounded rationality and imperfect information. We focus on the question whether agents will be able to coordinate on an efficient outcome under these conditions. In particular, we look at the effect of asymmetry in information.
The agents in our model will bargain on pollution abatement levels, using the *adaptive play* mechanism [9] to optimize their results. In adaptive play, agents are *myopic*: payoff in future rounds is not taken into account. Instead, the agents remember the earlier choices of other agents, assume they will make similar choices at present and search for a best response. The agents will also make occasional errors. This is to account for the fact that agents in the real world will occasionally act irrational, or there might be an unexpected external factor playing a role.
The bargaining game in our model is defined in two steps. The first step in the definition of our model is a toy model of transboundary pollution, the *pollution game*. The model is attractive for study and has been widely used for analysis with perfect information and rational agents, e.g. [1] [8]. We make a further simplification to the model, as we limit it to two agents.
Within the context of this pollution game, we assume that agents can make binding agreements on abatement for one round. At the beginning of each round, the agents will play a bargaining game over their abatement levels during that round. In the bargaining game both agents simultaneously make a proposal for the abatement levels. If the two proposals are mutually compatible, an agreement is made for that round. If the proposals are not compatible with each other, the agents have failed to make an agreement in that round and they will both choose their abatement levels according to the Nash equilibrium of the pollution game, instead.
We investigate the behavior of this model by running simulations. We find that agents are able to reach Pareto efficient outcomes, even though this is not true for all parameter settings. However, when one agent has a longer memory of past actions than the other, our behaviour model predicts that he will be worse off in the long run. This is a rather paradoxal finding. There is very little difference between our model and the bargaining model in [10], except that the underlying pollution game has a different payoff structure. Yet, in that study agents with longer memory were consistently better off.
This leads to two conclusions. First, it shows that the result in [10] does not generalize very well to other classes of bargaining games. Secondly, it is an indication that adaptive play may compromise the rationality of agents too strongly to be suitable as a model for this particular bargaining game.
The remainder of this paper is organised as follows. Section 2 describes the bargaining game. The pollution game is defined and analyzed in section 2.1 and in section 2.2 it is extended with a bargaining mechanism. Section 2.3 defines the behaviour of the agents. Section 3 contains the experiments. The simulations

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

are motivated and detailed in section 3.1. The experimental results are given in section 3.2 and section 4 concludes and suggests paths for further research.

2 Model

2.1 The Pollution Game

We model negotiations between two agents on emissions of a pollutant. The pollutant is transboundary and damage functions depend only on total emissions. Agents have the options to *abate*, lowering their emission for one period at a cost. The emphasis of the study is on learning dynamics. For this purpose we employ a strongly simplified pollution model, known as the standard model [1] [8]. We assume that costs of abatement are independent between agents and do not change over time. Further, it is assumed that the pollutant does not accumulate in the environment, even though greenhouse gases are in reality a stock pollutant. Also, all agents have the same damage function.
Agents are labelled $i = 1, 2$. They have an fixed unabated emission level $a/2$, where a is the aggragate emission level. They can spend on abatement, resulting in benefits for every agent. Hence, the benefits to agent i depend on the total abatement level Q. We assume linear marginal abatement benefits $B_i(Q)$. This leads to a quadratical benefit function, as specified in equation 1.

$$B_i(Q) = b(aQ - Q^2/2)/2 \tag{1}$$

The marginal benefit of the first unit of abatement is $ba/2$, with b a positive parameter. When total abatement equals total emission, the marginal benefit of further abatement is zero.
The costs of abatement $C_i(q_i)$ for agent i depends only on the abatement q_i of that agent, where $Q = q_1 + q_2$. The marginal costs of abatement are assumed to rise linearly, leading to the quadratic cost function in equation 2.

$$C_i(q_i) = cq_i^2/2 \tag{2}$$

where c is a positive parameter. The resulting net payoff vector or *utility* vector is shown in equation 3.

$$u_i(q_1, q_2) = B_i(q_1 + q_2) - C_i(q_i) \tag{3}$$

The Nash Equilibrium In the *Nash equilibrium*, both agents optimize their own payoff, without considering the possibility of mutual benefits from cooperation. This is known as *non-cooperative* behaviour.
In the pollution game of section 2.1, the Nash equilibrium can be calculated by taking the partial derivatives of the utility functions of both agents with respect to their own abatement level. Solving for the first order conditions yields a single symmetric Nash equilibrium, given in equation 4.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

$$q_i = q_D = \frac{a}{2(1 + c/b)} \tag{4}$$

Here the Nash equilibrium is denoted as q_D, because it will serve as the *disagreement point* in the bargaining game in section 2.2. The Nash equilibrium yields equal utility $u_D = u_1(q_D, q_D)$ to both players.

The Pareto Frontier The Nash equilibrium of this game is not Pareto optimal. Both agents could increase their payoff by cooperation. Therefore, this game is an example of a *social dilemma*, similar to the *prisoner's dilemma*.
If the agents cooperate, they will ideally choose Pareto optimal abatement levels. Additionally, both agents should receive greater payoff than in the Nash equilibrium. The symmetric solution on the Pareto frontier, the *Nash bargaining solution* (q^*, q^*) (see equation 5), satisfies both conditions. In [7], it is shown that under certain assumptions, the Nash bargaining solution will be the result of negotiations between two agents equal bargaining power.

$$q^* = \frac{a}{2 + c/b} \tag{5}$$

However, it should be noted that in our simulations the agents will not have perfect information, nor will they necessarily have identical bargaining power. The complete Pareto frontier can be characterized by maximizing the weighted average of u_1 and u_2 in equation 6, with weight α.

$$\alpha u_1(q_1, q_2) + (1 - \alpha) u_2(q_1, q_2) \tag{6}$$

The function in equation 6 has a single maximum at $(q_1^*(\alpha),\ q_2^*(\alpha))$ in 7.

$$\begin{aligned} q_1^*(\alpha) &= \frac{(1 - \alpha)a}{1 + 2\alpha(1 - \alpha)c/b} \\ q_2^*(\alpha) &= \frac{\alpha a}{1 + 2\alpha(1 - \alpha)c/b} \end{aligned} \tag{7}$$

In the special case that $\alpha = 1/2$, the solution in equation 7 reduces to the Nash bargaining solution (q^*, q^*). The extreme points of the Pareto front are the cases $\alpha = 0$ and $\alpha = 1$. In these cases, one agent does not abate and the other agent chooses an abatement level of a.

2.2 The Bargaining Game

The pollution game described in section 2.1 is assumed to be repeated for an unlimited number of rounds. Under such conditions, cooperation is possible, according to the folk theorem [5]. However, the incentive for agents to cooperate,

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

rather than to play the Nash equilibrium, is that cooperation in this round may be rewarded by the other agent in later rounds.
In this work we will look at agents with no forward looking abilities. With such limited rationality, agents will not be able to cooperate unless some bargaining mechanism is added to the game.
Therefore, we add a simple bargaining system. Let agent $-i$ denote the opposite agent from agent i. Instead of choosing abatement levels directly, both agents simultaneously make a proposal $p_i^t = (\hat{q}_{i,1}^t, \hat{q}_{i,2}^t)$ for the abatement levels in round t. Here $\hat{q}_{i,i}^t$ is the level of abatement that agent i is offering himself. However, in exchange, agent i demands an abatement level of at least $\hat{q}_{i,-i}$ from agent $-i$. Only if the offers and demands of both agents are compatible, the negotiations result in cooperation for round t. Otherwise, the agents fail to reach an agreement in round t and they will play the Nash equilibrium of the pollution game instead. Denote the boolean value of this requirement as $\Delta \in [\mathbf{true}, \mathbf{false}]$:

$$\Delta(p_i^t, p_{-i}^t) = \forall i : \hat{q}_{i,i}^t \geq \hat{q}_{i,-i}^t \tag{8}$$

If condition 8 is met, both agents are willing to abate at least as much as the other agent demands. But it is not yet clear what the final agreement will be. Will q_i^t be set to the level $\hat{q}_{i,i}^t$ that i offered himself, or to the level $\hat{q}_{-i,i}^t$ that the other agent demanded? In general, we could imagine the final result of negotiations to be any value in between the two. We assume that the agents will share the difference in a manner agreed upon prior to negotiations. The final abatement level of agent i will be a weighted average of the abatement level that agent i offered and the abatement level that was required, as in equation 9. The effect of varying λ will be analysed in sections 3 and 4.

$$q_i^t = q_i(p_i^t, p_{-i}^t) = \begin{cases} \lambda \hat{q}_{i,i}^t + (1-\lambda)\hat{q}_{-i,i}^t & \text{if } \Delta(p_i^t, p_{-i}^t) \\ q_D & \text{otherwise} \end{cases} \tag{9}$$

Note that if requirement 8 fails, the agents abort negotiations and instead revert to the disagreement point, which is the Nash equilibrium of the pollution game.
The final utility level of agent i in round t, $u_i^t(q_i^t, q_{-i}^t)$, then follows by inserting these values into equation 3:

$$u_i^t = u_i(q_i^t, q_{-i}^t) = B_i(q_i^t, q_{-i}^t) - C_i(q_i^t) \tag{10}$$

Nash Equilibria of the Bargaining Game The negotiation process thus defined constitutes a new game, where the actions agents have to choose from are the proposals they make. In order to distinguish this game from the pollution game itself, we will refer to it as the *bargaining game*. Where the pollution game has only a single Nash equilibrium, the bargaining game has many.
Consider the solution where, for $i = 1, 2$, $\hat{q}_{i,i}^t = 0$ and $\hat{q}_{i,-i}^t = a$. This is a degenerate Nash equilibrium. Since requirement 8 is not met, the result is that both

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

players play the Nash equilibrium $q_i^t = q_D$ in the pollution game. Neither agent can improve his own utility by deviating unilaterally.
The game also has many non-degenerate equilibria, where both players end up with higher utility than in the Nash equilibrium of the pollution game. For example, consider $\hat{q}_{i,i}^t = \hat{q}_{i,-i}^t = q^*$. This leads to the Nash Bargaining Solution of the pollution game. Both agents have improved their utility over u_D and neither agent could improve his utility further by a unilateral deviation from his negotiation strategy.
Any non-degenerate equilibrium of the bargaining game must be Pareto dominant over the Nash equilibrium of the pollution game. If it does not, then at least one agent is worse off than u_D and therefore has an incentive to let the negotiations fail instead, by lowering his abatement offer.
If $\lambda < 1$, any non-degenerate Nash equilibrium, with at least one player ending up with a higher utility than u_D, must have $\hat{q}_{i,i}^t = \hat{q}_{-i,i}^t$. If $\hat{q}_{i,i}^t < \hat{q}_{-i,i}^t$ then the negotiations fail. But if $\hat{q}_{i,i}^t > \hat{q}_{-i,i}^t$, then agent $-i$ has an incentive to increase $\hat{q}_{-i,i}^t$, since by equation 9 this would increase the abatement of i, but leave his own abatement unchanged. As an increase in the abatement of i increases the benefits for $-i$ but not his costs, the utility of agent $-i$ would increase.
Provided that $0 < \lambda < 1$, any solution of the bargaining game that satisfies both of these conditions is a Nash equilibrium of the bargaining game. Neither player has an incentive to change his own proposed abatement $\hat{q}_{i,i}^t$, because it would either let the negotiations fail or it would cause him to increase his abatement level further above q_D, lowering his utility. Similarly, neither player has an incentive to change the abatement level he demands from the opponent, $\hat{q}_{i,-i}^t$.
Now that we identified the non-degenerate Nash equilibria (at least for $0 < \lambda < 1$), we can calculate the corresponding range of values for q_i^t.
The extreme points of the region that Pareto dominates q_D are on the Pareto frontier of the pollution game, as defined by $q_i^*(\alpha)$ in equation 7. We have the constraint that $q_i^t \leq q_D$. Solving for α in equation 7, we find that the region of non-degenerate Nash equilibria is bounded by the values q_{min} and q_{max} in equation 11.
We will assume that agents only consider values for their proposals in the range $[q_{min}, q_{max}]$. It could be argued that the agents have insufficient information to know q_{min} and q_{max} ahead of time. However, if our agent model is applied to the Pollution Game itself, rather than to the Bargaining Game, it will converge to the unique Nash equilibrium (q_D, q_D), so it is not altogether unreasonable to postulate that agents are aware of $q_{min} = q_D$, at least.

$$\begin{aligned} \alpha_{min} &= \frac{\sqrt{b^2 + 2bc} - b}{2c} \\ \alpha_{max} &= 1 - \alpha_{min} \\ q_{min} &= q_D \\ q_{max} &= q_D * \frac{\alpha_{max}}{\alpha_{min}} \end{aligned} \tag{11}$$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

2.3 Adaptive Play

In this section we describe how agents decide what proposals to make in the negotiations of section 2.2. Agents are assumed to have bounded rationality. They do not have any forward looking abilities and do not know each others payoff functions. Instead, they form a model of the behaviour of the other agents, in order to find the best response to their actions.
We apply the *adaptive play* mechanism [9]. In adaptive play, agents assume that the other agents draw their proposals from a fixed probability distribution. In other words, they assume that the other agent plays a mixed strategy that doesn't change over time.
Agents have a limited memory m_i. Only actions chosen by the other agent in the latest m_i rounds are remembered. The agents then assume that the other agent will play any of the actions that he played during the last m_i rounds, each with equal probability.
Based on this internal model of the opponent, the agents search for an optimal response. They will maximize the expected final utility of their proposal p_i^t. Let (x, y) be a candidate proposal for p_i^t. It is evaluated by the expected value of the utility that it could yield:

$$f_i^t(x,y) = \frac{1}{m_i} \sum_{t'=t-1}^{t-m_i} u_i(q_i((x,y), p_{-i}^{t'}), q_{-i}((x,y), p_{-i}^{t'})) \tag{12}$$

Where q_i is calculated as in equation 9 and u_i as in equation 3. The function $f_i^t(x, y)$ serves as a fitness function.
The agent will then make the proposal that maximizes this fitness function:

$$p_i^t = \underset{(x,y)\in[q_{min},q_{max}]^2}{\operatorname{argmax}} f_i^t(x,y) \tag{13}$$

In case there are multiple values of (x, y) that maximize $f(x, y)$, one value is drawn from the set with a uniform random distribution.
However, in adaptive play, agents are allowed to make occasional errors. These are introduced to take into account that agents in the real world occasionally behave unpredictably, because of factors external to the game. With a small probability ε, equation 13 is ignored and p_i^t is instead drawn at random from $[q_{min}, q_{max}]^2$ with uniform distribution. This is known as a mutation.

Of course, equation 12 is undefined during the first few rounds of play. For $t = 1, \ldots, \max(m_1, m_2)$, p_i^t is again drawn at random from $[q_{min}, q_{max}]^2$ with uniform distribution.

2.4 Adaptive Play as an Evolutionary Algorithm

The whole process defined so far can be interpreted as an evolutionary algorithm. In this interpretation, the rounds of the repeated bargaining game are the generations and the chosen proposals p_i^t form the parent population of generation t.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

The offspring population is the entire domain $[q_{min}, q_{max}]$; the fitness function f_i^t depends both on the parent individual and on time. Mutations occur at the mutation rate ε.

2.5 Stochastic Stability

As shown in section 2.2, the bargaining game has many Nash equilibria. It is easily verified that all of the non-degenerate Nash equilibria of the game are also evolutionary stable strategies. Hence, the question is how to define or identify the most likely outcome of the game.
We are interested in finding the *stochastically stable strategies* of the game. Stochastically stable strategies are a refinement of the concept of evolutionary stable strategies. Any stochastically stable strategy is also an evolutionary stable strategy, but the reverse is not true.
In an evolutionary system such as our multi-agent model, an evolutionary stable strategy is stable in the short run, as neither agent has an incentive to deviate unilaterally. However, over increasingly long time scales there is an increasing probability that mutations disturb the equilibrium and cause the system to enter a different equilibrium. On sufficiently long timescales, the system behaves ergodic, switching back and forth between different evolutionary stable strategies. The timescale at which this kind of behaviour is typical is called the *long run*, or sometimes the *ultra-long run* [10].
Stochastically stable strategies are the evolutionary stable strategies that occur the most frequently in the long run, when the mutation rate ε approaches zero. For a full definition we refer the reader to [4].

3 Simulations

We will use simulations to gain insight into the stochastically stable strategies under different parameter settings of the model. The model will be run for a large but finite number of rounds T_{MAX}, under different parameter settings.
We will then analyze the results, using autocorrelation in the series under different time lags to find empirically what timescale constitutes the long run, i.e., at what timescale the system behaves ergodic.

3.1 Experimental Setup

The primary question that we wish to address is the role of information: what happens when one agent has an information advantage over the other? What if one agent has a longer memory? We have set up a series of five experiments (E_3 and $E_6 \dots E_9$) to compare the behaviour of the system under varying differences in memory.
The second question we wish to address is: how sensitive are the outcomes to a change in λ? For this purpose, we have set up a second series of experiments,

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

$E_1 \dots E_5$, where λ was varied. Experiment E_3 serves as the benchmark, as it has equal memory for both players and a value of λ of 0.5, splitting the remainder in equation 9 equally.
Table 1 provides an overview of all the parameter settings that have been used in the experiments. The ratio of b and c has been chosen such that it maximizes the potential gain from cooperation [1]. The mutation rate ε has been chosen as small as feasible to approximate the stochastically stable equilibria.

Table 1. Overview of Parameter Values

parameter	experiment								
	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
a	1								
b	1.64								
c	1								
ε	0.01								
T_{MAX}	10^7								
λ	0	0.25	0.5	0.75	1	0.5	0.5	0.5	0.5
m_1	8								
m_2	8	8	8	8	8	6	4	2	1

3.2 Results

In this section we will analyze the output, series of resulting utility values u_i^t. They are not time independent, but strongly autocorrelated. That is also to be expected, since a stochastic equilibrium is defined as the equilibrium that the is dominant in the long run. In order to find out what the long run is in this game, we need to find out at what time lag the autocorrelation between subsequent proposals approaches zero.
Autocorrelation can be expressed as a function of timelag, using formula 14:

$$R_i(\tau) = \frac{E[(u_i^t - \bar{u_i})(u_i^{t+\tau} - \bar{u_i})]}{\sigma^2} \tag{14}$$

In figure 1 all the autocorrelation plots of the experiments are shown. As expected, autocorrelation vanishes at sufficiently long timescale. From the figure we estimate a time lag of $\tau = 50.000$ as the typical long run.

Now we can sample measurements from each of the series at intervals of 50.000 rounds and treat them as approximately independent. Figure 2 shows a histogram of the resulting sample for benchmark experiment E_3.

As can be seen from figure 2, the measurements follow a roughly bell shaped curve, with a group of outliers at the disagreement utility U_D. The outliers result

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

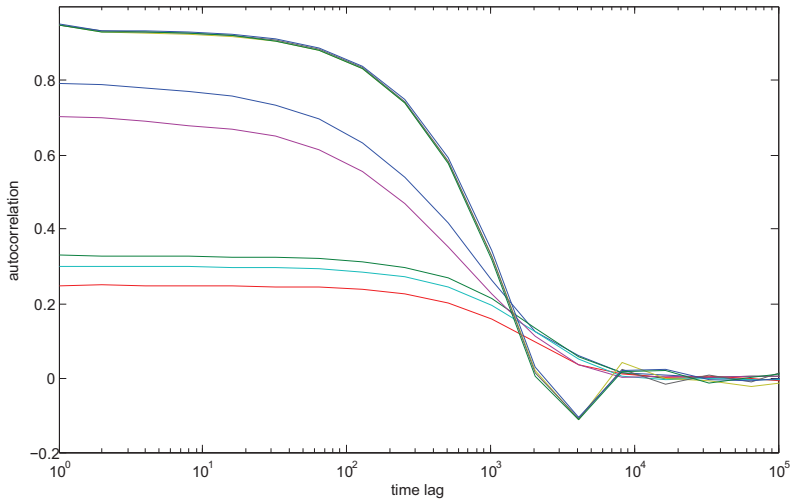


Fig. 1. Autocorrelation as a function of timelag, for both agents in each of the experiments.

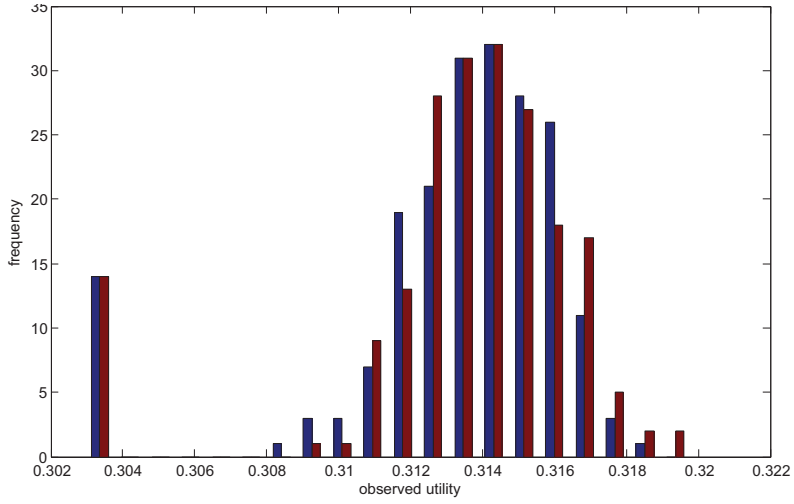


Fig. 2. Histogram of utility values sampled from E_3.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

from rounds where the negotiations failed. Since the interval $[q_{min}, q_{max}]$ does not contain degenerate equilibria, failed negotiations are not part of a stochastichally stable equilibrium. Therefore, we will ignore those measurements in the remainder of the analysis.
We assume by the bell shaped form of the rest of the distribution that there is a single stochastically stable equilibrium and estimate it by taking the mean value. The mean values for each experiment, after dismissing outliers, can be found in table 2.

In the benchmark experiment E_3, with equal memory and $\lambda = 0.5$, agents have on average been following the Nash Bargaining Solution ($u^* = 0.3141$) nearly exactly. Indeed, a t-test shows no statistically significant difference from a normal distribution with mean u^*.
When λ differs from 0.5, as in expirements E_1, E_2, E_4 and E_5, it clearly negatively affects the results of negotiations. For each of those experiments, average utility remained significantly below u^*, all at p-values below 10^{-5}.

Table 2. Average observed utility values (outliers removed)

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
λ	0	0.25	0.5	0.75	1	0.5	0.5	0.5	0.5
m_2	8	8	8	8	8	6	4	2	1
u_1	0.3098	0.3129	0.3140	0.3131	0.3101	0.3125	0.3125	0.3131	0.3129
u_2	0.3099	0.3128	0.3142	0.3133	0.3102	0.3149	0.3148	0.3141	0.3141

Agents with shorter memory length have a clear advantage in the long run. Experiments E_6 to E_9 all seem to favour the less informed agent 2. However, the results from those experiments show a wide spread, with long tails. This might explain why a t-test shows no statistical significant difference between both agents in E_8 and E_9. In E_6 and E_7, however, the difference is significant at the 1% confidence level.
The average utilities observed in the experiments with asymmetric information are all below the Pareto frontier. For example, in E_6 agent 1 had an average utility of 0.3125. It follows from 7 that a pareto optimal solution with this u_1 should have $u_2 = 0.3157$. The actual average u_2 was lower, even though a t-test shows no statistal significance. The same holds for E_7 to E_9.

4 Conclusions and Outlook

Even though our model is strongly simplified, especially as it is a two agent model, it nonetheless offers important insights into myopic bargaining on pollution games.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

As often in adaptive play, agents are able to coordinate on a Pareto efficient outcome, without knowing each others payoff functions. However, this is not guaranteed and depends on parameter values of the model, moreso than in many other applications of adaptive play.
Surprisingly, it turns out that having a shorter memory is an advantage in the long run. This must be due to the structure of the pollution game, as adaptive play has been applied in this fashion to a similar bargaining game [10], where adaptive play favours the agents remembering more of the past under any parameter settings.
This result indicates that adaptive play may be too limiting for studying pollution games. While we believe that evolutionary economics is generally a promising venue for studying bounded rationality, the challenge for future research is to understand the behaviour of agents that are imperfect, yet less short sighted than in adaptive play.

5 Acknowledgements

The author would like to thank Cees Withagen for reviewing the manuscript. Also, thanks to Guszti Eiben and Daan van Soest for their helpful suggestions.

References

1. Barrett, S.: Self-Enforcing International Environmental Agreements. Oxford Economic Papers 46, 804–878 (1994)
2. Barrett, S.: Environment & Statecraft: the Strategy of Environmental Treaty-Making. Oxford University Press (2005)
3. Finus, M.: Game Theoretic Research on the Design of International Environmental Agreements: Insights, Critical Remarks, and Future Challenges. International Review of Environmental and Resource Economics 2, 2967 (2008)
4. Foster, D., Young, P.: Stochastic evolutionary game dynamics. Theoretical Population Biology 38:2, 219-232 (1990)
5. Fudenberg, D., Maskin, E.: The folk theorem in repeated games with discounting or with incomplete information. Econometrica 54, 533-554 (1986)
6. Kolstad, C.D.: Systematic Uncertainty in Self-enforcing International Environmental Agreements. Journal of Environmental Economics and Management 53, 68-79 (2007)
7. Nash, J.: The Bargaining Problem. Econometrica 18, 155-162 (1950)
8. Rubio, S., Ulph, A.: Leadership and Self-Enforcing International Environmental Agreements with Non-Negative Emissions. unpublished manuscript, university of Valencia (2003)
9. Young, P.: The evolution of conventions. Econometrica 61, 57-84 (1993)
10. Young, P.: Individual strategy and social structure. Princeton University Press, 113-129 (1998)

Learn-and-Optimize: a Parameter Tuning Framework for Evolutionary AI Planning

Mátyás Brendel[1] and Marc Schoenauer[2]

[1] Projet TAO, INRIA Saclay & LRI
matthias.brendel@lri.fr
[2] Projet TAO, INRIA Saclay & LRI
marc.schoenauer@inria.fr

Abstract. Learn-and-Optimize (LaO) is a generic surrogate based method for parameter tuning combining learning and optimization. In this paper LaO is used to tune Divide-and-Evolve (DaE), an Evolutionary Algorithm for AI Planning. The LaO framework makes it possible to learn the relation between some features describing a given instance and the optimal parameters for this instance, thus it enables to extrapolate this relation to unknown instances in the same domain. Moreover, the learned knowledge is used as a surrogate-model to accelerate the search for the optimal parameters. The proposed implementation of LaO uses an Artificial Neural Network for learning the mapping between features and optimal parameters, and the Covariance Matrix Adaptation Evolution Strategy for optimization. Results demonstrate that LaO is capable of improving the quality of the DaE results even with only a few iterations. The main limitation of the DaE case-study is the limited amount of meaningful features that are available to describe the instances. However, the learned model reaches almost the same performance on the test instances, which means that it is capable of generalization.

1 Introduction

Parameter tuning is basically a general optimization problem applied off-line to find the best parameters for complex algorithms, for example for Evolutionary Algorithms (EAs). Whereas the efficiency of EAs has been demonstrated on several application domains [25, 14], they usually need computationally expensive parameter tuning. Being a general optimization problem, there are as many parameter tuning algorithms as optimization techniques. However, several specialized methods have been proposed, and the most prominent ones today are Racing [4], REVAC [16], SPO [2], and ParamILS [10]. All these approaches face the same crucial generalization issue: can a parameter set that has been

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

optimized for a given problem be successfully used for another one? The answer of course depends on the similarity of both problems.

However, until now, in AI Planning, sufficiently accurate features have not been specified that would allow to describe the problem, no design of a general learning framework has been proposed, and no general experiments have been carried out. This paper makes a step toward a framework for parameter tuning applied generally to AI Planning and proposes a preliminary set of features. The Learn-and-Optimize (LaO) framework consists of the combination of optimizing and learning, i.e., finding the mapping between features and best parameters. Furthermore, the results of learning will already be useful to further the optimization phases, using the learned model similarly, but also in a different way as in standard surrogate-model based techniques (see e.g., [1] for a Gaussian-process-based approach).

In this paper, the target optimization technique is an Evolutionary Algorithm (EA), more precisely the evolutionary AI planner called Divide-and-Evolve (DaE). However, DaE will be here considered as a black-box algorithm, without any modification for the purpose of this work compared to its original version described in [13].

The paper is organized as follows: AI Planning Problems and the Divide-and-Evolve algorithm are briefly introduced in section 2. Section 3 introduces the original, top level parameter tuning method, Learn-and-Optimize. The case study presented in Section 4 applies LaO to DaE, following the rules of the International Planning Competition 2011 – Learning Track. Finally, conclusions are drawn and further directions of research are proposed in Section 5.

2 AI Planning and Divide-and-Evolve

An Artificial Intelligence (AI) planning problem is defined by the triplet of an initial state, a goal state, and a set of possible actions. An action modifies the current state and can only be applied if certain conditions are met. A solution plan to a planning problem is an ordered list of actions, whose execution from the initial state achieves the goal state.

Domain-independent planners rely on the Planning Domain Definition Language PDDL2.1 [6]. The domain file specifies object types and predicates, which define possible states, and actions, which define possible state changes. The instance scenario declares the actual objects of interest, sets the initial state and provides a description of the goal. A solution plan to a planning problem is a consistent schedule of grounded actions whose execution in the initial state leads to a state that contains the goal state, i.e., where all atoms of the problem goal are true.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

A planning problem defined on domain D with initial state I and goal G will be denoted in the following as $\mathcal{P}_D(I, G)$.

Early approaches to AI Planning using Evolutionary Algorithms directly handled possible solutions, i.e. possible plans: an individual is an ordered sequence of actions see [20, 15, 22, 23, 5]. However, as it is often the case in Evolutionary Combinatorial optimization, those direct encoding approaches have limited performance in comparison to the traditional AI planning approaches. Furthermore, hybridization with classical methods has been the way to success in many combinatorial domains, as witnessed by the fruitful emerging domain of memetic algorithms [9]. Along those lines, though relying on an original "memetization" principle, a novel hybridization of Evolutionary Algorithms (EAs) with AI Planning, termed Divide-and-Evolve (DaE) has been proposed [18, 19]. For a complete formal description, see [12].

The basic idea of DaE in order to solve a planning task $\mathcal{P}_D(I, G)$ is to find a sequence of states $S_1, \ldots, S_n$, and to use some embedded planner to solve the series of planning problems $\mathcal{P}_D(S_k, S_{k+1})$, for $k \in [0, n]$ (with the convention that $S_0 = I$ and $S_{n+1} = G$). The generation and optimization of the sequence of states $(S_i)_{i\in[1,n]}$ is driven by an evolutionary algorithm. The fitness (makespan or total cost) of a list of partial states $S_1, \ldots, S_n$ is computed by repeatedly calling the external 'embedded' planner to solve the sequence of problems $\mathcal{P}_D(S_k, S_{k+1})$, $\{k = 0, \ldots, n\}$. The concatenation of the corresponding plans (possibly with some compression step) is a solution of the initial problem. Any existing planner can be used as embedded planner, but since guarantee of optimality at all calls is not mandatory in order for DaE to obtain good quality results [12], a sub-optimal, but fast planner is used: YAHSP [21] is a lookahead strategy planning system for sub-optimal planning which uses the actions in the relaxed plan to compute reachable states in order to speed up the search process.

One-point crossover is used, adapted to variable-length representation in that both crossover points are independently chosen, uniformly in both parents. Four different mutation operators have been designed, and once an individual has been chosen for mutation (according to a population-level mutation rate), the choice of which mutation to apply is made according to user-defined relative weights. Because an individual is a variable length list of states, and a state is a variable length list of atoms, the mutation operator can act at both levels: at the individual level by adding (addState) or removing (delState) a state; or at the state level by adding (addAtom) or removing (delAtom) some atoms in the given state.

3 Learn-and-Optimize for Parameter Tuning

3.1 The General LaO Framework

As already mentioned, parameter tuning is actually a general global optimization problem, thus facing the routine issue of local optimality. But a further problem arises in parameter tuning, and this is the generality of the tuned parameters. Generalizing parameters learned on one instance to another instance might be problematic, because there are instances with very different complexity in the same domain. For example in [3] per-domain tuning was performed with the most difficult, largest instance, considered as a representative of the whole domain. However, it is clear from the results that these parameters were often suboptimal for the other instances. One workaround to this generalization issue is to relax the constraint of finding a single universally optimal parameter-set, that certainly does not exist, and to focus on learning a complex relation between instances and optimal parameters. The proposed Learn-and-Optimize framework (LaO) aims at learning such a relation by adding learning to optimization. The underlying hypothesis is that there exists a relation between some features describing an instance and the optimal parameters for solving this instance which can be learned, and the goal of this work is to propose a general methodology to do so.

Suppose for now that we have n features and m parameters, and we are doing per-instance parameter tuning on instance $\mathcal{I}$. For the sake of simplicity and generality, both the fitness, the features and the parameters are considered as real values. Parameter tuning is the optimization (e.g., minimization) of the fitness function $f_{\mathcal{I}} : \mathbf{R}^m \to \mathbf{R}$, the expected value of the stochastic algorithm DaE executed with parameter $p \in \mathbf{R}^m$. The optimal parameter set is defined by $p_{opt} = argmin_p\{f_{\mathcal{I}}(p)\}$. For each instance $\mathcal{I}$, consider the set $F(\mathcal{I}) \in \mathbf{R}^n$ of the features describing this instance. Two relations have to be taken into account: each planning instance has features, and it has an optimal parameter-set. In order to be able to generalize, we have to get rid of the instance, and collapse both relations into one single relation between feature-space and parameter-space. For the sake of simplicity let us assume that there exists an unambiguous mapping from the feature space to the optimal parameter space.

$$p : \mathbf{R}^n \to \mathbf{R}^m, p(F) = p_{opt} \tag{1}$$

.

The relation p between features and optimal parameters can be learned by any supervised learning method capable of representing, interpolating and extrapolating $\mathbf{R}^n \to \mathbf{R}^m$ mappings, provided sufficient data are available.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

The idea of using some surrogate model in optimization is not new. Here, however, there are several instances to optimize, and only one model is available, that maps the feature-space into the parameter-space. A significant conceptual difference is that while in standard surrogate techniques the model is trained and evaluated for fitness, while in our approach we directly evaluate it for optimal parameter candidates.

Nevertheless, there is no question about how to use our model of p in optimization: one can always ask the model for hints about a given parameter-set. It seems reasonable that the stopping criterion of LaO is determined by the stopping criterion of the optimizer algorithm. After exiting one can also do a re-training of the learner with the best parameters found.

3.2 An Implementation of LaO

A simple multilayer Feed-Forward Artificial Neural Network (ANN) trained with standard backpropagation was chosen here for the learning of the features-to-parameters mapping, though any other supervised-learning algorithm could have been used. The implicit hypothesis is that the relation p is not very complex, which means that a simple ANN may be used. In this work, one mapping is trained for each domain. Training a single domain-independent ANN is left for future work. The other decision for LaO implementation is the choice of the optimizer used for parameter tuning. Because parameter optimization will be done successively for several instances, the simple yet robust (1+1)-Covariance Matrix Adaptation Evolution Strategy [8], in short (1+1)-CMA-ES, was chosen, and used with its robust own default parameters, as advocated in [3].

One original component, though, was added to some direct approach to parameter tuning: gene-transfer between instances. There will be one (1+1)-CMA-ES running for each instance, because using larger population sizes for a single instance would be far too costly. However, the (1+1)-CMA-ES algorithms running on all training instances form a population of individuals. The idea of gene-transfer is to use something like a crossover between the individuals of this population. Of course, the optimal parameter sets for the different instances are different; However, good 'chromosomes' for one instance may at least help another instance. Thus it may be used as a hint in the optimization of that other instance. Therefore random gene-transfer was used in the present implementation of LaO, by calling the so-called *Genetransferer*. This is similar to the migration operator of the so called Island Model Genetic Algorithm [24], and the justification is similar: parallelism and separability. There are however considerable differences: in our case, When the Genetransferer is requested for a hint for one instance, it returns the so-far best parameter of a different instance chosen with uniform random distribution (preventing, of course, that the default parameters are tried twice). Another benefit of Genetransferer is that it may smoothen out

the ambiguities between instances, by increasing the probability for instances with the same features to test the same parameters, and thus the possibility to find out that the same parameters are appropriate for the same features. Figure 1 shows the LAO framework with the described implementations.

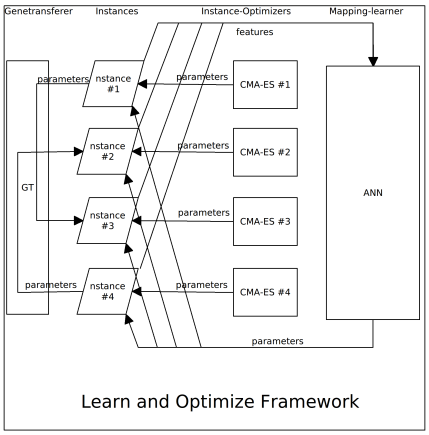


Fig. 1. Flowchart of the Lao framework, displaying only 4 instances.

One additional technical difficulty arose with CMA-ES: each parameter is here restricted to an interval. This seems reasonable and makes the global algorithm more stable. Hence the parameters of the optimizer are actually normalized linearly onto the [0,1] interval. It is hence possible to apply a simple version of the box constraint handling technique described in [7], with a penalty term simply defined by $||p^{feas} - p||$, where p^{feas} is the closest value in the box. Moreover, only p^{feas} was recorded as a feasible solution , and later passed to the ANN. Note that the GeneTransferer and the ANN itself cannot return hints outside of the box. In order to not to compromise too much CMA-ES, several iterations of this were carried out for one hint of the ANN and one Genetransferer.

The implementation of LaO algorithm uses the Shark library [11] for CMA-ES and the FANN library for ANN [17]. To evaluate each parameter-setting with each instance, a cluster was used, that has approximately 60 nodes, most of them with 4 cores, some with 8. Because of the heterogeneity of the hardware architecture used here, it is not possible to rely on accurate predicted running times. Therefore, for each evaluation, the number of YAHSP evaluations in DaE is fixed. Moreover, since DaE is not deterministic, 11 independent runs were carried out for each DaE experiment with a given parameter-set, and the fitness of this parameter set was taken to be the median.

4 Results

In the Planning and Learning Part of IPC2011 (IPC), 5 sample domains were pre-published, with a corresponding problem-generator for each domain: Ferry, Freecell, Grid, Mprime, and Sokoban. Ferry and Sokoban were excluded from this study since there were not enough number of instances to learn any mapping. For each of the remaining 3 domains, approximately 100 instances were generated, with the published generators and distribution (ranges) of generator-parameters 100 instances per domain seemed to be appropriate for a running time of 2-3 weeks. The competition track description fixes running time as 15 minutes. However, many instances were never solved within 15 minutes, and those instances were dropped from the rest of experiment. The remaining instances were used for training.

The real IPC competition domains of the same track were released later. These domains were much harder, meaning that most of the official train instances could not be solved at all by DaE in 15 minutes. Therefore the published instance-generators were used, but with a lower range of the generator-parameters. Even this way, we can only present one domain: Parking. For the other domains, the number of training instances, or the iterations of LaO carried out until the deadline is not sufficient to take the case-study seriously.

Domain Name	# of training instances	# of test instances
Freecell	108	230
Grid	55	124
Mprime	64	152
Parking	101	129

Table 1. Description of the domains

Table 1 presents the train- and testsets for each domain. The Mean Square Error (MSE) of the trained ANN is shown for each domain. Note that because the fitness takes only few values, there can be multiple optimal parameter sets for the same instance, resulting in an unavoidable MSE. So we do not expect this error to converge to 0. One iteration of LaO amounts to 5 iterations of CMA-ES, followed by one ANN training and one Genetransferer. Due to the time constraints, only a few iterations of LaO were run. For example the 10 iterations in domain Grid amounts to 500 CMA-ES calls in total.

The controlled parameters of DaE are described in table 2. For a detailed description of these parameters, see [3]. The feature-set consists of 12 features which are presented in table 3. The first 5 features are computed from the domain file, after the initial grounding of YAHSP: number of fluents, goals, predicates,

Name	Min	Max	Default
Probability of crossover	0.0	1	0.8
Probability of mutation	0.0	1	0.2
Rate of mutation add station	0	10	1
Rate of mutation delete station	0	10	3
Rate of mutation add atom	0	10	1
Rate of mutation delete atom	0	10	1
Mean average for mutations	0.0	1	0.8
Time interval radius	0	10	2
Maximum number of stations	5	50	20
Maximum number of nodes	100	100 000	10 000
Population size	10	300	100
Number of offsprings	100	2 000	700

Table 2. DaE parameters that are controlled by LaO.

Name	minimum	mean	maximum
# initial fluents	28	31.17	34
# goals	2	2	2
# predicates	7	7	7
# objects	32	35.17	38
# types	3	3	3
mutexdensity	0.14	0.15	0.17
# lines domain	198	198	198
# words domain	640	640	640
# bytes domain	5729	5729	5729
# lines instance	139	153.33	166
# words instance	379	427.42	469
# bytes instance	2017	2265.75	2479

Table 3. Minimum, mean and maximum values are given for the Freecell domain.

objects and types. One further feature we think could even be more important is called mutex-density, which is the number of mutexes divided by the number of all fluent-pairs. We also kept 6 less important features: number of lines, words and byte-count - obtained by the Linux-command "wc" - of the instance and the domain file. These features were kept only for historical reasons: they were used in the beginning as some "dummy" features. Note that some features take only one values, they however had meaning when training an inter-domain model.

The ANN had 3 fully connected layers, the layers had all 12 neurons, corresponding to the number or parameters and features, respectively. Standard back-propagation algorithm was used for learning (the default in FANN). In one iteration of LaO, the ANN was only trained for 50 iterations (aka epochs) without reseting the weights, in order to i- avoid over-training, and ii- making a gradual transition from the previous best parameter-set to the new best one, and eventually try some intermediate values. Hence, over the 10 iterations of LaO,

500 iterations (epochs) of the ANN were carried out in total. However, note that the best parameters were trained with much fewer iterations, depending on the time of their discovery. In the worst case, if the best parameter was found in the last iteration of LaO, it was trained for only 50 epochs and not used anymore. This explains why retraining is needed in the end.

LaO has been running for several weeks on a cluster. But this cluster was not dedicated to our experiments, i.e. only a small number of 4 or 8-core processors were available for each domain on average. After stopping LaO, retraining was made with 300 ANN epochs with the best data, because the ANN's saved directly from LaO may be under-trained. The MSE error in retraining of the ANN did not decrease using more epochs, which indicates that 300 iterations are enough at least for this amount of data and for this size of the ANN. Tests with 1000 iterations did not produce better results and neither did the training of the ANN uniquely, i.e. only with the first found best parameters.

Domain Name	# of iterations	ANN error	quality-ratio in LaO	quality-ratio ANN on train	quality-ratio ANN on test
Freecell	16	0.1	1.09	1.05	1.04
Grid	10	0.09	1.09	1.05	1.03
Mprime	8	0.08	1.11	1.05	1.04
Parking	11	0.12	1.49	1.41	1.14

Table 4. Results by domains (only the actually usable training instances are shown). ANN-error is given as MSE, as returned by FANN. The quality-improvement ratio in Lao is that of the best parameter-set found by LaO.

Since testing was also carried out on the cluster, the termination criterion for testing was also the number of evaluations for each instance. For evaluation the quality-improvement the quality-ratio metric defined in IPC competitions was used. The baseline qualities come from the default parameter-setting obtained by tuning for some representative domains with global racing in previous work see [3]. The ratio of the fitness value for the default parameter and the tuned parameter was computed and average was taken over the instances in the train or test-set.

$$Q = \frac{Fitness_{baseline}}{Fitness_{tuned}} \tag{2}$$

Table 4 presents several quality-improvement ratios. Label "in LaO" means that the best found parameter is compared to the default. By definition, this ratio can never be less than 1. This improvement indicated by high quality-ratio is already useful if the very same instances used in training have to be optimized. Quality-improvement ratios for the retrained ANN on both the training-set and the

test-set are also presented. In these later cases, numbers less then 1 are possible (the parameters resulting from the retrained ANN can have worse results than the ones given by the original ANN), but were rare.

The first 3 domains in Table 4 have similar results: some quality-gain in training was consistently achieved, but the transfer of this improvement to the ANN-model was only partial. The phenomenon can appear because of the unambiguity of the mapping, or because the ANN is not complex enough for the mapping, or, and most probably, because the feature-set is not representative enough. On the other hand, the ANN model generalizes excellently to the independent test-set, at least for the first 3 domains. Quality-improvement ratios dropped only by 0.01, i.e. the knowledge incorporated in the ANN was transferable to the test cases and usable almost to the same extent than for the train set. The size of the training set seems not to be so crucial. For example for Freecell all the instances (108 out of 108 generated) could be used, because they were not so hard. On the other hand, only few Grid instances (55 out of 107 generated) could be used. However, both performed well. The explanation for this may be that both the 32 and 108 instances covered well the whole range of solvable instances. The situation is somewhat different for the last domain: Parking. Here we have a high quality-gain in training (almost 50%), even much of this could be learned by the ANN, however, here we have a huge drop for the test-set. The reason for this is that when using the ANN as an extrapolation, there is a considerable number of instances which get unsolvable. Still, we get some overall gain in average, in fact we get the highest gain for this difficult domain. The main issue for such hard domains will be to avoid much more effectively unfeasible parameters when extrapolating the ANN-model for unknown instances.

5 Conclusions and Future Work

The LaO method presented in this paper is a surrogate-model based combined learner and optimizer for parameter tuning. LaO was demonstrated to be capable of improving the quality of the DaE algorithm over tuning with global racing consistently, even though it was run only for a few iterations. Ongoing work is concerned with running LaO for an appropriate number of iterations. A clearly visible result is also that some of this quality-improvement can be incorporated into an ANN-model, which is also able to generalize excellently to an independent test-set.

The most important experiment to carry out in the future is simply to test the algorithm with more iterations and on more domains – and this will take several months of CPU even using a large cluster. Since LaO is only a framework, as indicated other kind of learning methods, and other kind of optimization techniques may be incorporated. If an ANN is used, the optimal structure has

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

to be determined, or a more sophisticated solution is to apply one of the so-called Growing Neural Network architectures. Also the benefit of gene-transfer and/or crossover should be investigated further. Gene-transfer shall be improved so that chromosomes are transfered deterministically, measuring the similarity of instances by the similarity of their features. Present results indicate that the current feature set is too small and should be extended for better results. Also a more effective mechanism to avoid unfeasible parameters in the ANN-model has to be developed, especially for hard domains and instances.

6 Acknowledgements

This work is funded through French ANR project DESCARWIN ANR-09-COSI-002.

References

1. R. Bardenet and B. Kégl. Surrogating the surrogate: accelerating gaussian-process-based global optimization with a mixture cross-entropy algorithm. In *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, 2010.
2. T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential parameter optimization. In B. McKay, editor, *Proc. CEC'05*, pages 773–780. IEEE Press, 2005.
3. J. Bibai, P. Savéant, M. Schoenauer, and V. Vidal. On the generality of parameter tuning in evolutionary planning. In J. B. et al., editor, *Genetic and Evolutionary Computation Conference (GECCO)*, pages 241–248. ACM Press, July 2010.
4. M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A Racing Algorithm for Configuring Metaheuristics. In *GECCO '02*, pages 11–18. Morgan Kaufmann, 2002.
5. A. H. Brié and P. Morignot. Genetic Planning Using Variable Length Chromosomes. In *Proc. ICAPS*, 2005.
6. M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *JAIR*, 20:61–124, 2003.
7. N. Hansen, S. Niederberger, L. Guzzella, and P. Koumoutsakos. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1):180–197, 2009.
8. N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.
9. W. Hart, N. Krasnogor, and J. Smith, editors. *Recent Advances in Memetic Algorithms*. Studies in Fuzziness and Soft Computing, Vol. 166. Springer Verlag, 2005.
10. F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009.

11. C. Igel, T. Glasmachers, and V. Heidrich-Meisner. Shark. *Journal of Machine Learning Research*, 9:993–996, 2008.
12. Jacques Bibai, Pierre Savéant, Marc Schoenauer, and Vincent Vidal. An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning. In *ICAPS 2010*, pages 18–25. AAAI press, 2010.
13. Jacques Bibai, Pierre Savéant, Marc Schoenauer, and Vincent Vidal. On the benefit of sub-optimality within the divide-and-evolve scheme. In P. Cowling and P. Merz, editors, *EvoCOP 2010*, number 6022 in Lecture Notes in Computer Science, pages 23–34. Springer-Verlag, 2010.
14. F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*. Springer, Berlin, 2007.
15. I. Muslea. SINERGY: A Linear Planner Based on Genetic Programming. In *Proc. ECP '97*, pages 312–324. Springer-Verlag, 1997.
16. V. Nannen, S. K. Smit, and A. E. Eiben. Costs and benefits of tuning parameters of evolutionary algorithms. In *Proceedings of the 20th Conference on Parallel Problem Solving from Nature*, 2008.
17. N. Nissen. Implementation of a Fast Artificial Neural Network Library (FANN). Technical report, Department of Computer Science University of Copenhagen (DIKU), 2003.
18. M. Schoenauer, P. Savéant, and V. Vidal. Divide-and-Evolve: a New Memetic Scheme for Domain-Independent Temporal Planning. In J. Gottlieb and G. Raidl, editors, *Proc. EvoCOP'06*. Springer Verlag, 2006.
19. M. Schoenauer, P. Savéant, and V. Vidal. Divide-and-Evolve: a Sequential Hybridization Strategy using Evolutionary Algorithms. In Z. Michalewicz and P. Siarry, editors, *Advances in Metaheuristics for Hard Optimization*, pages 179–198. Springer, 2007.
20. L. Spector. Genetic Programming and AI Planning Systems. In *Proc. AAAI 94*, pages 1329–1334. AAAI/MIT Press, 1994.
21. V. Vidal. A lookahead strategy for heuristic search planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, pages 150–159, Whistler, BC, Canada, June 2004. AAAI Press.
22. C. H. Westerberg and J. Levine. "GenPlan": Combining Genetic Programming and Planning. In M. Garagnani, editor, *19th PLANSIG Workshop*, 2000.
23. C. H. Westerberg and J. Levine. Investigations of Different Seeding Strategies in a Genetic Planner. In E.J.W. Boers et al., editor, *Applications of Evolutionary Computing*, pages 505–514. LNCS 2037, Springer-Verlag, 2001.
24. D. Whitley, S. Rana, and R. B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–47, 1998.
25. T. Yu, L. Davis, C. Baydar, and R. Roy, editors. *Evolutionary Computation in Practice*. Studies in Computational Intelligence 88, Springer Verlag, 2008.

Multi-Problem Parameter Tuning using BONESA

S.K. Smit, A.E. Eiben

Vrije Universiteit Amsterdam

Abstract. We introduce a parameter tuning method designed to tune an evolutionary algorithm (EA) on multiple problems in one go. Our method has two special features. It is based on a 'double loop' scheme, consisting of an intertwined searching and learning loop. The searching loop is seeking EA parameter vectors with a high utility, while the learning loop creates a model to predict the utility of parameter vectors. This reduces computational efforts by using the model instead of performing expensive EA runs. Furthermore, our method uses Gaussian filtering and significance testing to establish Pareto dominance in the presence of noisy data. We demonstrate and validate this approach with experiments on an artificial utility landscape. The results show that our method can estimate the utility values of parameters very well and collects much useful information on EA robustness.

1 Background and objectives

In recent years, there has been a growing interest in automated parameter tuning methods in Evolutionary Computing [8]. New tuning algorithms, such as SPO [2–4], REVAC [18, 21], and Meta-ES [26], F-RACE [5], ParamILS [14] have been developed and their success has been demonstrated in several case studies. These methods can focus the search for good parameter values, to the most promising areas of the parameter space and can therefore deliver high performing parameter values for the evolutionary algorithm (EA) to be tuned.

In this paper we argue and demonstrate that such tuning algorithms can do more than finding good parameter values. This stance is based on noting that tuning algorithms are in essence search algorithms that generate much data while traversing the space of parameter vectors. This data contains information about all visited parameter vectors and their 'utility', that is, the performance of the EA using the given parameter vector. Often this data is lost, because one is solely interested in the best found parameter values. However, this data can be stored and utilized for more general purposes. To this end, we adopt the view of Hooker [12] distinguishing competitive and scientific testing, and observe that parameter tuning can be used for

- configuring an evolutionary algorithm by choosing parameter values that optimize its performance, and
- analyzing an evolutionary algorithm by studying how its performance depends on its parameter values.

Furthermore, let us also note that the behavior of an EA depends on three factors: its parameter values, the problem instance it is solving, and random effects. Thus, the above list can be extended by:

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

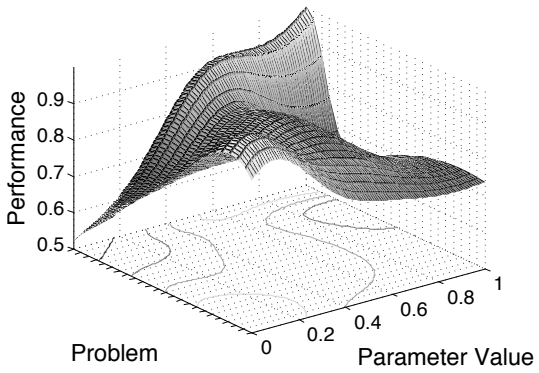


Fig. 1. Illustration of the grand performance landscape showing the performance (z) of EA instances belonging to a given parameter vector (x) on a given problem instance (y). Note: The "cloud" of repeated runs is not shown.

- analyzing an evolutionary algorithm by studying how its performance depends on the problems (problem instances) it is solving, and
- analyzing an evolutionary algorithm by studying how its performance varies when executing multiple independent repetitions of its run,

It is easy to see that a detailed understanding of algorithm behavior has a great practical relevance. Knowing the effects of problem characteristics and algorithm characteristics on algorithm behavior, users can make well-informed design decisions regarding the (evolutionary) algorithm they want to use. Figure 1 shows the kind of information that can be gathered if tuning data is kept and integrated into what we call a *grand performance landscape*. For the sake of this illustration, we restrict ourselves to a single EA parameter. Thus, we obtain a 3D landscape with axis x representing the values of the parameter and axis y representing the different problems investigated.[1] (In the general case of n parameters, we have $n + 1$ axes here.) The third dimension z shows the performance of the EA instance belonging to a given parameter vector on a given problem instance. It should be noted that for stochastic algorithms, such as EAs, this landscape is blurry if the repetitions with different random seeds are also taken into account. That is, rather than one z-value for a pair $\langle x, y \rangle$, we have one z for every run, for repeated runs we get a "cloud".

The left-hand-side of Figure 2 shows 2D slices of the grand performance landscape corresponding to specific parameter vectors. This provides information on robustness to changes in problem specification. Such data are often reported in the EC literature, be it with a different presentation. A frequently used option is to show a table containing the experimental outcomes (performance results) of one or more EA instances on a

[1] We specifically refer to them as being different problems, since we do not want to imply that these belong to the same type of problem, e.g., multiple TSP or 3SAT instances, that can be solved with similar parameter values. In general, to get the best insights into the effect of parameter values, the test-suite should contain a diverse set of problems.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

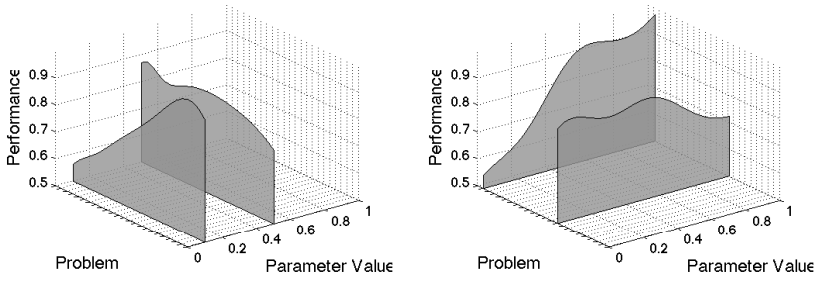


Fig. 2. Illustration of parameter-wise slices (left) and problem-wise slices (right) of the grand utility landscape shown in Figure 1. (The "cloud" of repeated runs is not shown.)

predefined test suite, e.g., the five DeJong functions, the 25 functions of the CEC 2005 contest, etc.

The right-hand-side of Figure 2 shows 2D slices corresponding to specific problem instances. On such a slice we see how the performance of the given EA depends on the parameter vectors it uses. This discloses information regarding robustness to changes in parameter values. Such data is hardly ever published in evolutionary computing, mainly because they are not even generated: parameter values are mostly selected by conventions, ad hoc choices, and very limited experimental comparisons.

One of the main messages of this paper is that, by the increased adoption of tuning algorithms, this practice could change and knowledge about EA parameterization could be collected and disseminated. This changes the question of 'which parameter-values should I choose?' to 'when should I choose which parameter values?'.

However, the parameter-tuning problem is characterized by two specific issues that makes the realization this vision of generating knowledge on EA behavior on the large scale very difficult:

- The cost of testing a single parameter vector $\bar{p}$ is high. Namely, to establish its performance, the given EA must be executed using the values in $\bar{p}$ and this can be very time consuming.
- The presence of noise. Noise is caused by the stochastic nature of EAs, implying that different runs with $\bar{p}$ can (and will) deliver different results. This mandates that a parameter vector is tested repeatedly, thus amplifying the cost issue possibly by orders of magnitude (depending on the number of required repetitions).

The main contributions of this paper can be listed as follows:

- We introduce a multi-problem parameter tuning approach, called Bonesa that can deal with the specific issues of parameter tuning
- We validate this method by experiments on an artificial utility landscape.

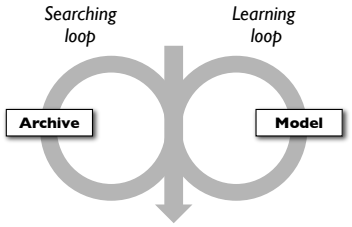


Fig. 3. The generic 'double loop' scheme of an intertwined searching and learning procedure. The searching loop is a generate-and-test procedure iteratively bulding an archive that contains all visited points and their quality. The learning loop is iteratively building a model of the quality surface, based on data in the archive.

2 Multi-Problem Parameter Tuning using Bonesa

Bonesa is, in essence, an iterative model based search procedure much like Sequential Parameter Optimization [2], or in general, any search method using surrogate models[16, 9]. The generic scheme is illustrated in Figure 3 that shows an intertwined searching and learning procedure. The searching loop is a generate-and-test procedure that is iteratively generating new points in the search space and is assessing the quality of the most promising ones. The learning loop is using the information generated by the searching loop to build a model of the quality surface (here: utility landscape), which allows for estimating the quality of previously unseen points. Notice the two-way interaction between the two loops: information generated by the searching loop is used to update the model, while estimations based on the model are used to direct the search.

Such an iterative model based search procedure, in which a model is used to quickly pre-assess the generated points, can highly reduce the costs of the tuning session with respect to uniform sampling. Only points that are expected to perform reasonably well are tested and used to update the model. Furthermore, this approach does not only contribute to the reduction in costs, it also a natural way to derive a model of the grand performance landscape, since the procedure is iteratively increasing its accuracy.

However, the fact that the points that pass the model-based test need several expensive EA runs to establish their real performance still leads to high tuning-costs. Therefore, we have chosen to use a Gaussian filter to reduce the effects of noise when assessing the performance of a parameter vector. A Gaussian filter is a common method of reducing noise in spacial data, which does not depend on repetitive testing of the same point but uses proximity data [11]. Therefore, every parameter vector has to be evaluated only once.

Furthermore, we adopt an implicit racing approach to determine which of two selected parameter vectors performs best [17]. The outcome of such a test is either 'A is better than B', or 'B is better than A', or that more data is needed to determine the order. Therefore, by iteratively increasing the number of data-points, only the minimal amount of tests are used, needed for comparing the two parameter-vectors.

Finally, the most distinguishing difference between the currently known iterative model based search procedures and Bonesa is its multi-objective approach. Rather than optimizing on a single problem or on a weighted sum of performance values [22, 19],

Bonesa uses the Pareto-strength approach from SPEA2[27] to optimize on a whole range of different problems in one go. Therefore, one is ultimately not only able to select the best parameter-values for a single problem, but also for a class of problems. It is important to note that this approach of multi-problem tuning differs from other multi-problems approaches [6, 15, 5] by means of the problems that are assessed. Rather than optimizing over a set of instances of the same type of problem. A true multi-objective approach allows for assessing instances of completely different problems. Since such a method terminates with a whole set of parameter-vectors, an experimenter can choose which of those suits his needs best. For example, certain parameter vectors could be well-suited for a subset of problems, rather than the whole set. Bonesa can identify these and deliver a (small) set of good parameter vectors, each with a distinct subset of covered problems. Such a set of vectors can support a much higher overall performance than a single vector, delivered by usual tuners.

These four features form the core ingredients of Bonesa, in particular, of its learning loop. The searching loop on the other hand uses a straight-forward generate-and-test approach adopted from REVAC[18].

2.1 Learning Loop

The learning loop is iteratively updating the statistics and the prediction model of the utility landscape, using the set of currently known parameter vectors and corresponding performances. In each iteration, a new model is built that can be used to:

- Derive the relative distance between two parameter-vectors
- Predict the performance of an unknown parameter vector on each of the problems
- Estimate the cumulative Pareto strength of an unknown parameter vector

Relative Distance In order to predict the utility values for unknown parameter vectors, a common approach is to model the utility landscape. There are two main approaches for modeling, surface modeling and nearest-neighbor classification. The first one fits a model (a regression model, Kriging model, classification tree, etc.) to the data that can directly be used to predict the utility values. The second type estimates the utility of a new vector by a weighted average of the utilities of the closest known neighbors This method is very robust –for large neighborhood-sizes– even if measurements are very noisy. However, the problem with this approach is that there is no generic definition of being ‘close to’ a neighbor. In particular, it depends on the size of the sweet pot, i.e., the range where the good parameter values are. To illuminate this let us consider two parameters, x and y, with the same range between 0 and 100. If the good values for parameter x are all between 50 and 100, then two values for x with a distance of 10 can be considered ‘close’. However, if the good values for parameter y are all between 70 and 71, then two values for y with a distance of 10 should be considered ‘far’. Therefore, we propose to use the standardized distance, which is obtained by dividing the distance between two points on a certain axis by the standard deviation of the parameter values of all ’good vectors’ in the archive A. Here, ‘good vectors’ are defined as vectors that perform above average on at least one of the problems.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Definition 1 *Let GV be the set of l-dimensional 'good vectors' from the archive A and $\bar{\sigma}$ its standard deviation vector. Then the standardized distance between two l-dimensional vectors $\bar{x}$ and $\bar{y}$ is:*

$$r(\bar{x},\bar{y}) = \sqrt{1/l \cdot \sum_{i=1}^{l} \left((\bar{x}_i - \bar{y}_i)/\bar{\sigma}_i\right)^2} \tag{1}$$

Noise Reduction and Prediction The quality of a specific parameter vector is defined by the performance of the EA on a collection of problems $P = \{p_1, \ldots, p_n\}$. By the stochastic nature of EAs, this performance is a noisy observable. In tuning terms, this means that the utility of a parameter vector $\bar{x}$ can only be estimated. The usual way of improving these estimates is to repeat the measurements[24, 13, 23, 10, 7], that is, to do multiple EA runs using $\bar{x}$, however, this is clearly an expensive way of gaining more confidence. Therefore, Bonesa test each vector only once, and uses a Gaussian filtering technique to reduce the noise.

To this end, we use the concept of perceived similarity, as proposed in [20]. The estimated utility $\hat{u}_{\bar{x},p}$ of a certain parameter vector $\bar{x} \in A$ on problem p is calculated by a weighted average of the test-results (u_p) of all vectors (including $\bar{x}$ itself) in the archive A on the same problem.

Definition 2 *The estimated utility $\hat{u}_{\bar{x},p}$ of a vector $\bar{x}$ on problem p is:*

$$\hat{u}_{\bar{x},p} = \left(\sum_{\bar{y}\in A} u_{\bar{y},p} \cdot \omega_{\bar{x},\bar{y}}\right)/\left(\sum_{\bar{y}\in A} \omega_{\bar{x},\bar{y}}\right) \text{ , where} \tag{2}$$

$$\omega_{\bar{x},\bar{y}} = e^{c \cdot r(\bar{x},\bar{y})} \qquad \forall\, \bar{y} \in A \tag{3}$$

The term c is a scaling factor, chosen in such a way that that the average value of $\sum_{\bar{y}} \omega_{\bar{x},\bar{y}}$ is equal to 50, if the $|A|$ points are random uniformly distributed in a l-dimensional hypercube. This means that the more points there are in the archive, the smaller the contribution is of points far from $\bar{x}$. Note that, since A increases over time, c also changes each time A is updated. The same approach of similarity based weighted average can be used to estimate the variance of the utility of $\bar{x}$ on problem p and the support ($\hat{\rho}$) of $\bar{x}$.

Definition 3 *The estimated variance $\hat{\sigma}^2_{\bar{x},p}$ of a vector $\bar{x} \in A$ on problem p is:*

$$\hat{\sigma}^2_{\bar{x},p} = \left(\sum_{\bar{y}\in A} \left(u_{\bar{y},p} - \hat{u}_{\bar{x},p}\right)^2 \omega_{\bar{x},\bar{y}}\right)/\left(\left(\sum_{\bar{y}\in A} \omega_{\bar{x},\bar{y}}\right) - 1\right) \tag{4}$$

Definition 4 *The support $\hat{\rho}$ of a vector $\bar{x} \in A$ is:*

$$\hat{\rho}_{\bar{x}} = \sum_{\bar{y}\in A} \omega_{\bar{x},\bar{y}} \tag{5}$$

Obviously, the same approach can be used to predict the utility, variance, and support of unseen vectors. Namely, the predicted utility $\hat{u}_{\bar{z}}$ of a certain seen or unseen vector $\bar{z}$ is derived using Gaussian interpolation over all vectors in the archive (Equation 3). The same can be done for the predicted variance (Equation 4) and support (Equation 5).

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Pareto Strength In general, the quality of a vector $\bar{z}$ (seen or unseen) in a multi-objective setting is based on the notion of Pareto dominance. In our case, parameter vectors must be compared based on their utility, which is an inherently noisy function. Therefore, we cannot simply compare the utility of $\bar{z}$ with that of $\bar{y}$ on a problem p. Instead we need to test the hypothesis that the average value of the utility distribution of $\bar{z}$ on problem p is higher than the average value of the utility distribution of $\bar{y}$ on problem p. There are various known tests relying on samples from each distribution. However, Bonesa does not have such samples, because each parameter vector is only tested once. Therefore, Bonesa uses an adaptation of the Welch's T-test[25] to compare utilities:

$$\tau_{\bar{z},\bar{y},p} = (\hat{u}_{\bar{z},p} - \hat{u}_{\bar{y},p}) / \sqrt{\hat{\sigma}^2_{\bar{z},p}/\hat{\rho}_{\bar{z}} + \hat{\sigma}^2_{\bar{z},p}/\hat{\rho}_{\bar{z}}} \tag{6}$$

$$\nu_{\bar{z},\bar{y},p} = \frac{\left(\hat{\sigma}^2_{\bar{z},p}/\hat{\rho}_{\bar{z}} + \hat{\sigma}^2_{\bar{y},p}/\hat{\rho}_{\bar{y}}\right)}{\hat{\sigma}^4_{\bar{z},p}/(\hat{\rho}^2_{\bar{z}} \cdot (\hat{\rho}_{\bar{z}} - 1)) + \hat{\sigma}^4_{\bar{y},p}/(\hat{\rho}^2_{\bar{y}} \cdot (\hat{\rho}_{\bar{y}} - 1))} \tag{7}$$

$$D_{\bar{z},\bar{y},p} = \mathbb{T}\left(-\tau_{\bar{z},\bar{y},p}, \nu_{\bar{z},\bar{y},p}\right) \tag{8}$$

where $\mathbb{T}$ is the cumulative t-distribution. If the support for one of the two vectors is lower than 10, then this test can lead to unreliable results. Therefore we use the following equations to calculate the D given $\hat{\rho}_x > 10$ and $\hat{\rho}_y \leq 10$:

$$\tau_{\bar{z},\bar{y},p} = (\hat{u}_{\bar{z},p} - \hat{u}_{\bar{y},p}) / \sqrt{\hat{\sigma}^2_{\bar{z},p} \cdot 2} \tag{9}$$

$$D_{\bar{z},\bar{y},p} = \mathbb{T}\left(-\tau_{\bar{z},\bar{y},p}, 10\right) \tag{10}$$

If both of the two have a support lower than 10, then there is not enough evidence to decide which dominates the other, and no domination is assumed. In the other cases, $1 - D_{\bar{z},\bar{y},p}$ indicates the level of confidence in the hypothesis that average value of the utility distribution of $\bar{z}$ on problem p, is better or equal than the average value of the utility distribution of $\bar{y}$ on problem p. If this confidence is higher a certain threshold $1 - \epsilon$, we assume that $\bar{z}$ *is better than* $\bar{y}$ on problem p. The definition of dominance can therefore naturally be extended to:

Definition 5 *A parameter vector $\bar{z}$ dominates parameter vector $\bar{y}$ if and only if:*

1. $\exists p \in P \ : \ D_{\bar{z},\bar{y},p} \leq \epsilon$, *and*
2. $\forall g \ (g \neq p) \in P \ : \ D_{\bar{y},\bar{z},g} > \epsilon$.

Using this definition of dominance we can define the number of 'slaves', the number of vectors from the archive that are dominated by a vector $\bar{z}$, as:

Definition 6 *The number of slaves $S(\bar{z})$, of any parameter vector $\bar{z}$ is equal to:*

$$S(\bar{z}) = \sum_{\bar{y} \in A} \begin{cases} 1 & \text{if } \bar{z} \text{ dominates } \bar{y} \\ 0 & \text{if } \bar{z} \text{ does not dominate } \bar{y} \end{cases} \tag{11}$$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Finally, we use the Pareto Strength $\hat{S}(\bar{z})$ [27] as a quality metric for previously unseen vectors. Pareto Strength is defined as the -1 times, the sum of the number slaves over all vectors that dominate $\bar{z}$ ('masters'):

The Pareto Strength $Q(\bar{z})$, of a newly generated parameter vector $\bar{z}$ is:

$$Q(\bar{c}) = \sum_{\bar{y} \in A} \begin{cases} -1 \cdot S(\bar{y}) & \text{if } \bar{z} \text{ is dominated by } \bar{y} \\ 0 & \text{if } \bar{z} \text{ is not dominated by } \bar{y} \end{cases} \tag{12}$$

Candidates that are not dominated by any of the vectors in the archive have a strength of 0, all others have a value lower than zero. The more vectors from the archive dominate the new candidate, the lower the strength. This pushes the archive towards the Pareto-front.

2.2 Searching Loop

As mentioned before, the task of the searching loop is to provide new vectors and performance values to the learning loop. The search loop of Bonesa is initialized with a randomly generated set of K parameter vectors. For each of them a single run of the EA to be tuned is executed on each problem from the test-suite, to determine its utility. Therefore, if the test-suite contains n problems, this yields into a total number of $K \cdot n$ tests executed in this step. After all testa are finished, the vectors are added to the archive A together with the utility values obtained by these tests. This first initial set is fed into the learning loop, and when that is finished, the iterative search procedure is started.

Each iteration starts with generating k candidate parameter-vectors. These are generated using the REVAC approach, [18, 21]. To generate one new parameter vector, for each of the parameters ($i = 1, \ldots, l$) we have to randomly draw a value x_i from the corresponding parameter values in the archive. The second step is to generate a new parameter value, based on the one drawn from the archive. In Bonesa, this step is simplified w.r.t. the original REVAC scheme. Instead of calculating the so-called smoothening interval and drawing a random value from it, we simply add some Gaussian noise to x_i in such a way that the new value is expected to be within the top 50 closest values. To this end, the level of noise s_i has to be defined as follows.

$$s_i = \sqrt{12} \cdot \bar{\sigma}_i \cdot (\Gamma(0.5 \cdot l + 1) \cdot (50 \cdot 1/|A|))^{1/l} / \sqrt{\pi \cdot |A|} \tag{13}$$

where $\bar{\sigma}$ is the standard deviation vector of the archive A and Γ is the gamma function. This process is repeated until k new parameter vectors are created. For each of these, the Pareto strength is determined using the model (cf. Section 2.1), and the corresponding support is calculated (Section 2.1); the higher the support, the higher the confidence in the Pareto strength. Next we select the m vectors with the highest Pareto strength. In case of a tie, the candidate with the lowest support ($\hat{\rho}$) is selected. This ensures a sampling bias towards areas with only a few vectors (low support), which raises the support of the vectors in this area and refines the model. Finally, the set of m vectors is tested on each of the problems and added to the archive. If the maximum number of tests is not yet reached, a new iteration is started. Observe that by increasing k or lowering m Bonesa can be made more explorative or exploitative.

3 Validation experiments

For a rigorous evaluation of Bonesa and its derived models, we should apply it to a combination of an evolutionary algorithm and a test-suite is for which the complete performance landscape is known. However, to our knowledge there is no such combination in evolutionary computing. So to be able to compare the models derived by Bonesa with the true utility landscape, we have created artificial performance landscapes in which we can control all aspects. We decided to take the mean best fitness performance measure (and not the EA speed, or whatever else) to be reflected by the artificial utility. Furthermore, for the sake of visualizing the results, we have chosen to limit the test-suite to only two artificial utility landscapes, based on two parameters, leading to three-dimensional robustness and four-dimensional utility landscapes. The performance $H^p(\bar{x})$ of a parameter vector $\bar{x}$ on an artificial utility-landscape p is defined as:

$$H^p(\bar{x}) = \sqrt{((\bar{x}_1 - B_1^p)/C_1^p)^2 + ((\bar{x}_1 - B_2^p)/C_2^p)^2}$$

$$\varepsilon(\bar{x}) = 0.05 \cdot log(4 + \bar{x}_1) \cdot (2 - \bar{x}_2)$$
$$U^p(\bar{x}) = e^{-1 \cdot Q^p(\bar{x})} + N(0, \varepsilon(\bar{x}))$$

The vectors B^p and C^p can be used to control the landscape. B^p indicates for which parameter-values the utility (mean best fitness) is optimal. C^p indicates how much the utility drops when moving away from the optimal vector. Finally, ε controls the amount of noise that we add to the outcome of each 'run'. This is to include the "cloud of repeated runs" (cf. Figure 2 in our artificial test data. In our two landscapes, we have chosen to make ε dependent on the parameter values of the 'run'. High values for parameter 2 yield stable results, high values for parameter 1 yield very noisy results. The values for B and C used in this experiment, are given in Table 1. The setup of Bonesa is given in Table 2. The full code of the software used in these experiments as well as the data sets containing the results of the experiments are available from `http://tuning.sourceforge.net`.

3.1 Results

The main objective of Bonesa is to provide a cost-efficient method to accurately model the grand utility landscape. Since this landscape is a high dimensional surface, the ac-

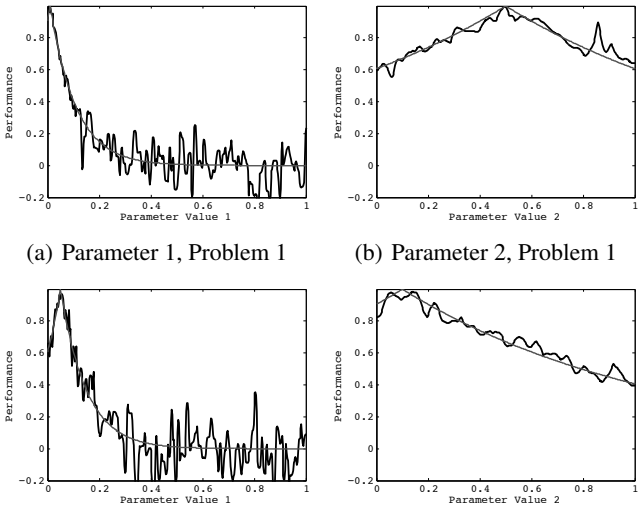
Table 1. Setup of the artificial utility-landscapes

Parameter	Landscape 1	Landscape 2
B_1	0.01	0.05
B_2	0.50	0.10
C_1	0.10	0.10
C_2	1.00	1.00

Table 2. Setup of Bonesa

Parameter	Value
K	1000
k	1000
m	10
Dominance ϵ	0.05
Maximum number of tests	4000

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



(a) Parameter 1, Problem 1 (b) Parameter 2, Problem 1

(c) Parameter 1, Problem 2 (d) Parameter 2, Problem 2

Fig. 4. Real utility values (smooth curves) vs. utility values estimated by the Bonesa-made models (rugged curves).

curacy is illustrated in Figure 4 using several slices. Each subfigure shows the known and predicted utility (EA performance) for a certain parameter value.

It is clear that the accuracy greatly differs between different areas of the grand performance landscape. The higher the performance, the better the accuracy. Figure c) shows an almost perfect fit, when predicting the performance for the 10% best parameter-values. Furthermore, it is clear that the tuneability of a parameter also influences the accuracy of the model. The predictions for parameter 1 are much more accurate near the optimum, than the predictions for parameter 2. So, the more important a correct parameter-value is, the better the accuracy of the prediction. Furthermore, since the grand performance landscape contains a "cloud" around the average performance, we also look at accuracy in predicting the variance in performance. These results are shown in Figure 5 for parameter one and problem one. All other graphs are very similar to this one, therefore omitted here. Figure 4 and 5 demonstrate that Bonesa is able to accurately model the grand performance landscape, especially in the 'interesting' areas.

The next question we pose concerns the efficiency of Bonesa. To this end, we inspect the space of all possible performance values for problem 1 and problem 2, a 2D square with values ranging from 0 to 1 on both axes, cf. Figure 6. On this square, the Pareto front is indicated by the grey line representing the optimal performance based on a certain combination of weights for both problems. The grayscale values of the rest of the plot show the computing time spent in a given area (black is much time, white is little time). The data exhibited by Figure 6 shows that Bonesa is able to identify and explore the interesting areas very effectively. 80% of the computing-time is spent in the area close to the Pareto front.

4 Conclusions

In this paper we have introduced a new parameter tuning method called Bonesa. It is designed to tune an (evolutionary) algorithm on multiple problems in one go and to provide useful insights about the algorithm. Thus, Bonesa not only helps obtain superior parameter values, but it also provides much information about different problem instances, parameter values, and algorithm performance. This information can be used to analyze EAs and to obtain a deeper understanding of them. Furthermore, information about the grand performance can help EA designers to choose a good EA off-the-shelf, without executing costly tuning sessions themselves. Using fully known artificial utility landscapes we have experimentally verified that Bonesa is able to approximate the utility landscape fast, and is therefore very well suited for scientific testing of algorithms.

In the near future, we will apply Bonesa to real utility landscapes belonging to existing test-suites and evolutionary algorithms, to illustrate its use in practice. Furthermore, we will make it available online in an easy to use package, which will not only contain the algorithm but also the visualizations needed for a unified report of scientific testing.

References

1. *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, Edinburgh, UK, 2-5 Sept. 2005. IEEE Press.
2. T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation* [1], pages 773–780 Vol.1.
3. T. Bartz-Beielstein and S. Markon. Tuning search algorithms for real-world applications: A regression tree based approach. Technical Report of the Collaborative Research Centre 531 Computational Intelligence CI-172/04, University of Dortmund, March 2004.
4. T. Bartz-Beielstein, K. Parsopoulos, and M. Vrahatis. Analysis of Particle Swarm Optimization Using Computational Statistics. In Chalkis, editor, *Pr. of the International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2004)*, pages 34–37. Wiley, 2004.
5. M. Birattari. *Tuning Metaheuristics*. Springer, 2005.
6. S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7:77–97, 2001.

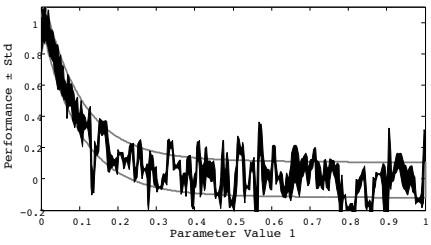


Fig. 5. Real noise shown by the real performance plusminus standard deviation (two smooth curves) vs. noise estimated by the Bonesa (rugged curves).

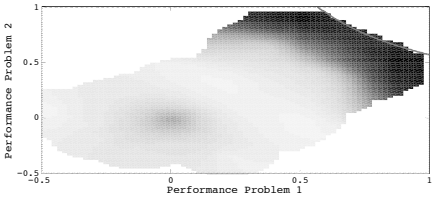


Fig. 6. Illustration of the search effort in relation to the predicted utilities

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

7. K. Deb and H. Gupta. Searching for robust pareto-optimal solutions in multi-objective optimization. *Evolutionary Multi-Criterion Optimization*, pages 150–164, 2005.
8. A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):to appear, 2011.
9. M. El-Beltagy, P. Nair, and A. Keane. Metamodeling techniques for evolutionary optimization of computationally expensive problems: Promises and limitations. In W. Banzhaf et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pages 196–203. Morgan Kaufmann, San Francisco, 1999.
10. H. Eskandari and C. D. Geiger. Evolutionary multiobjective optimization in noisy problem environments. *Journal of Heuristics*, 15(6):559–595, 2009.
11. R. Haralick and L. Shapiro. *Computer and Robot Vision*, volume 1, chapter 7. Addison-Wesley Publishing Company, 1992.
12. J. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, 1995.
13. E. Hughes. Evolutionary multi-objective ranking with uncertainty and noise. In *EMO '01: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, pages 329–343, London, UK, 2001. Springer-Verlag.
14. F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stutzle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.
15. F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proc. of the 21th Conference on Artifical Intelligence (AAAI '07)*, pages 1152–1157, 2007.
16. Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.
17. O. Maron and A. Moore. The racing algorithm: Model selection for lazy learners. In *Artificial Intelligence Review*, volume 11, pages 193–225. Kluwer Academic Publisher, 1997.
18. V. Nannen and A. E. Eiben. Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In M. M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1034–1039. Hyderabad, India, 2007.
19. M. E. H. Pedersen and et al. Parameter tuning versus adaptation: proof of principle study on differential evolution. Technical report, Hvass Laboratories, 2008.
20. R. N. Shepard. Toward a universal law of generalization for psychological science. *Science*, 237(4820):1317–1323, 1987.
21. S. K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, pages 399–406, Trondheim, May 18-21 2009. IEEE Press.
22. S. K. Smit and A. E. Eiben. Parameter tuning of evolutionary algorithms: Generalist vs. specialist. In C. e. a. Di Chio, editor, *Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 542–551. Springer, 2010.
23. J. F. University and J. E. Fieldsend. Multi-objective optimisation in the presence of uncertainty. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation* [1], pages 476–483.
24. T. Voß, H. Trautmann, and C. Igel. New uncertainty handling strategies in multi-objective evolutionary optimization. In R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, editors, *PPSN*, volume 6239 of *Lecture Notes in Computer Science*, pages 260–269. Springer, 2011.
25. B. L. Welch. The generalization of "student's" problem when several different population variances are involved. *Biometrika*, 34(1–2):28–35, 1947.
26. B. Yuan and M. Gallagher. Combining Meta-EAs and Racing for Difficult EA Parameter Tuning Tasks. In F. Lobo, C. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, pages 121–142. Springer, 2007.
27. E. Zitzler, M. Laumanns, and L. Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

A Model based on Biological Invasions for Island Evolutionary Algorithms

I. De Falco[1], A. Della Cioppa[2], D. Maisto[1], U. Scafuri[1], and E. Tarantino[1]

[1] ICAR–CNR, Via P. Castellino 111, 80131 Naples, ITALY
{ivanoe.defalco,domenico.maisto, umberto.scafuri,
ernesto.tarantino}@na.icar.cnr.it
[2] DIEII, University of Salerno, Via Ponte don Melillo 1, 84084 Fisciano (SA), ITALY
adellacioppa@unisa.it

Abstract. Migration strategy plays an important role in designing effective distributed evolutionary algorithms. Here, a novel migration model inspired to the phenomenon known as biological invasion is adopted. The migration strategy is implemented through a multistage process involving large invading subpopulations and their competition with native individuals. In this work such a general approach is used within an island parallel model adopting Differential Evolution as the local algorithm. The resulting distributed algorithm is evaluated on a set of well known test functions and its effectiveness is compared against that of a classical distributed Differential Evolution.

Keywords: massive migration, biological invasion, distributed EAs.

1 Introduction

Evolutionary Algorithms (EAs) [1] have shown to be very effective to solve complex search and optimization problems in different fields. Their main drawback is related to the convergence speed. A popular way to enhance EA performance is to implement parallel versions in which the individuals are distributed over several subpopulations (*islands*) which evolve independently and interact by a migration operator used to exchange individuals among them. This distributed framework is based on the classical coarse-grained approach to EAs [2]. It consists in a locally-linked strategy, known as *stepping stone-model* [3], in which each EA instance is connected to a number of other instances equal to the degree of connectivity of the topology beneath. Thus, each subpopulation is kept relatively 'isolated' from all the other ones in the sense that it can be implicitly influenced by them only through communications with its own neighbors.

The selection function, deciding what individuals migrate, and the replacement function, indicating the individuals to be substituted by the immigrants, play a key role for the effectiveness of the model. Different strategies can be devised: the migrants can be selected either according to better fitness values or randomly, and they might replace the worst individuals or substitute

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

them only if better, or they might replace any individual (apart from the best ones) in the neighboring subpopulations. The exchange of the solutions is determined by the *migration rate* defining the number of individuals that are sent to (received from) other islands, a *replacement function* implementing the substitution policy and by the *migration interval* indicating the exchange frequency among neighboring subpopulations. An unsuitable number of migrants can be disruptive when using swap, substitution or copy as replacement function [4, 5]. Finally, the island topology determines the number of the neighboring subpopulations and its diameter gives an indication of the speed at which information spreads through them [6].

Due to interaction among subpopulations, island EAs have a dynamic behavior totally different from that of their sequential versions. They are likely to explore a search space more evenly and that they may face population stagnation thanks to their ability to maintain population diversity [6]. Another advantage is that generally these distributed versions allow speedup in computation and better solution quality [7] in comparison with a single EA evolution.

The idea underlying this paper is to device a novel migration model inspired to the biological phenomenon known as *biological invasion*, consisting in the introduction, the establishment and the spread of non–native organisms in an environment [8]. Although this process has inherently ecological impacts, it has also some evolutionary consequences. Essentially, when organisms are introduced to a new habitat they may experience new selective pressures due to biotic (i.e., indigenous species) and abiotic (non–living chemical and physical environmental components) factors, and simultaneously act as novel selective agents on native individuals in the invaded ecosystem. Such conditions are extremely favorable for a rapid evolution of both the invaders and the natives they interact with in the new environment [9]. This frequently induces a quick adaptation of both native and alien organisms to prevent their extinction [9, 8, 10]. According to the literature produced in this field, biological invasions can be divided into different stages [11, 12]. Firstly, a *propagule*, i.e. a group of individuals involved in the invasion, is formed in its native habitat. Secondly, that propagule is transported to a new range where forms a *founding subpopulation*. Finally, the establishment and the spread of the propagule in the new habitat take place by means of a competition process with native individuals. Each stage of this process acts as a filter, passing only individuals that own the characteristics needed to survive to novel selection pressures. The impact of a biological invasion is assessed through a composite measure named as *propagule pressure*. It relies on two key components: the propagule size (the number of individuals composing a propagule) and the propagule number (the rate at which propagules are introduced per unit of time). Thus, propagule pressure is the distribution of propagule sizes and the frequency in which propagules arrive [13]. Both theoretical and empirical studies suggest that the propagule pressure is often positively correlated with the success of invasion process [14, 12]. Genetic studies conducted recently have confirmed this hypothesis showing that large invading populations and multiple introductions are required for the process to succeed in each stage [14].

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

With the aim to exploit the above mentioned features of *biological invasion*, we have introduced into a generic island EA, instead of the canonical exchange process, a migration model inspired to the invasion process, structured in three distinct stages. With reference to each island, these stages are: the formation of the propagules in neighboring islands, the creation of a founding subpopulation and, finally, the generation of a new local subpopulation. In particular, the individuals to migrate from a subpopulation are those which are fitter than the current local average fitness. Then a founding subpopulation is formed by collecting the immigrants from all the neighbors and the native individuals. Finally, to simulate the establishment and the spread of the propagule and, hence, to generate a new local subpopulation, a competition replacement mechanism is adopted, i.e., the new subpopulation is created by choosing, through a fitness-proportionate selection, among all the individuals in the founding subpopulation. Such a novel general Invasion–based Distributed Evolutionary Algorithm (IDEA), can be conceived for use with any EA. In this paper, simulations have been carried out by choosing as evolutionary algorithm the Differential Evolution (DE) [15, 16]. To assess the effectiveness of IDEA a comparison is carried out on a set of diverse benchmark problems with respect to a distributed version of the DE algorithm, *i.e.*, DDE [17].

Paper structure is as follows: Section 2 outlines the related works; Section 3 presents the details of the parallel framework with the novel migration scheme. In Section 4 the experimental findings are discussed, and behavior of IDEA is described. Finally, Section 5 contains conclusions and future works.

2 Related works

Several island EA models differing in the selection function, the replacement function, the migration rate and the topology have been proposed in literature.

Cantú–Paz [18] investigates how the policy used to select migrants and individuals to replace affects the selection pressure in parallel EAs. The four possible combinations of random and fitness–based migration and replacement of existing individuals in the subpopulations are taken into account. The paper shows that the migration policy that chooses both the migrants and the replacements according to their fitness increases the selection pressure and may cause a significatively faster convergence.

In [5] different migration intervals and migration sizes are investigated. In general, the authors suggest that the best performance is achieved with small migration sizes and moderate migration intervals. Their experiments indicate that the reasons for this migration policy are the necessity of maintaining diversity in the system to produce novel results and the need for enough information exchange among islands to combine partial results.

Considered that we have chosen DE for our investigation, some distributed DEs are reported. In [19] the migration mechanism as well as the algorithmic parameters are adaptively coordinated according to a criterion based on genotypical diversity. An adaptive DE is executed on each subpopulation for

a fixed number of generations. Then a migration process, based on a random connection topology, is started: each individual in each subpopulation can be probabilistically swapped with a randomly selected individual in a randomly chosen subpopulation (including the one containing the initial individual).

Tasoulis et. al [20] propose a distributed DE characterized by a ring topology, a selection function that picks up the individuals with the best performance and a replacement function that substitutes random individuals of the neighboring subpopulation. A similar approach is shown in [21] with the same topology whereas the migration consists in the best individual replacing the oldest member of another subpopulation in the topological neighborhood.

In Apolloni et al. [22] a distributed version which modifies the one suggested in [20] is presented: the migration policy is based on a probabilistic criterion depending on five parameters. The individuals to migrate are randomly selected and the individuals arriving from other islands replace randomly chosen local individuals only if the former ones are fitter. The topology is a unidirectional ring in which the individuals are exchanged with the nearest neighbors.

In [23] a distributed DE algorithm with subpopulations grouped into two families is described. The first family has the role of exploring decision space and constituting an external evolutionary framework. In the second family, subpopulations reduce progressively their size, in order to increase highly exploitation and to quickly detect solutions with a high performance. Then the solutions generated by the second family migrate to the first family.

Weber et al. [24] propose a distributed DE that employs a novel self–adaptive scheme. Subpopulations are disposed in a ring topology. Each subpopulation has its own scale factor value. With a probabilistic criterion, the individual with the best performance migrates to the neighboring island and replaces a pseudo-randomly selected individual of the target subpopulation. The target subpopulation inherits also the scale factor if it is promising at the current stage of the evolution. A perturbation mechanism is introduced to improve the exploration ability of the algorithm.

Ishimizu and Tagawa [25] present a structured DE approach still based on the island model. Different network topologies are taken into account. The migration takes place every fixed number of generations and the exchange involves only the best individual which migrates towards only one of the adjacent subpopulations on the basis of the topological neighborhood and randomly replaces an individual, except for the best one, in the receiving subpopulation.

3 IDEA scheme

The proposed algorithm distinguishes itself from other parallel evolutionary algorithms by both the number of individuals migrating from a node to another and by the way such individuals are used to update the receiving population.

The philosophy underlying this novel migration scheme is inspired by biological invasion phenomena. As in the biological context as in the computational one, if the number of new individuals introduced in a population

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

is commensurable with the size of receiving population, evolutionary pressure increases [18]. In reality, the arrival of alien individuals alters the balance of the new habitat insofar as they compete with internal individuals to survive.

Our model makes use of a locally connected topology where each node, i.e., each processor, hosts an instance of an EA, and is connected to other $|\mathscr{C}(p)|$ nodes, where $\mathscr{C}(p)$ is the set of p neighbors of the node.

Execution of the algorithm starts by means of an initialization process. Let us suppose there are $\mathscr{N}$ nodes in the chosen topology; $\mathscr{N}$ subpopulations $\{P_1, \ldots, P_p, \ldots, P_{\mathscr{N}}\}$, each one composed of n individuals, are randomly sampled and each of them is assigned to a different node.

At each generation, each subpopulation P_p performs a sequential EA. As a consequence, each subpopulation is updated from a generation to the next one by means of the steps typical of the EA chosen. This scheme is repeated for each individual and for a number of generations $\mathscr{T}$, named as *invasion time*. This corresponds to the migration interval usually defined for distributed EAs. Every $\mathscr{T}$ generations, neighboring subpopulations exchange individuals. The set of individuals each subpopulation P_p sends to its neighbors is called *propagule* and is indicated as $\mathscr{M}_{P_p}$. $\mathscr{M}_{P_p}$ is determined by collecting the individuals of P_p which are fitter than its current average fitness:

$$\mathscr{M}_{P_p} = \left\{ x_i^p \in P_p \;\middle|\; f(x_i^p) \succ \frac{1}{n}\sum_{j=1}^{n} f(x_j^p) \right\} \tag{1}$$

where $f(x_i^p)$ is the fitness associated to the individual x_i^p and "$\succ$" is a binary relation stating that the left member is fitter than the right member. All of the chosen individuals are sent to all of the neighboring subpopulations $P_{\tilde{p}}$, with $\tilde{p} \in \mathscr{C}(p)$, so, at each invasion time $\mathscr{T}$, each subpopulation receives a total number of $\sum_{\tilde{p}} \left|\mathscr{M}_{P_{\tilde{p}}}\right|$ elements where $\left|\mathscr{M}_{P_{\tilde{p}}}\right|$ is the cardinality of the set $\mathscr{M}_{P_{\tilde{p}}}$. By considering that each propagule originated in any of the neighboring nodes has a size averagely equal to circa $n/2$, the number of individuals introduced to each node is $|\mathscr{C}(p)| \cdot n/2$, approximately.

Afterwards, in each node, a larger subpopulation, formed by both native and exotic individuals, is constructed by adding this massive number of alien individuals to its own subpopulation. This new subpopulation is called *founding subpopulation* $\mathit{\Pi}_p$ and is defined as:

$$\mathit{\Pi}_p \equiv \left\{ \bigcup_{\tilde{p} \in \mathscr{C}(p)} \mathscr{M}_{P_{\tilde{p}}} \right\} \cup P_p \tag{2}$$

with a size equal to $n + \sum_{\tilde{p}} \left|\mathscr{M}_{P_{\tilde{p}}}\right|$. Fundamentally, $\mathit{\Pi}_p$ is an archive of potential solutions and is composed of both internal and external candidate solutions. Hence, $\mathit{\Pi}_p$ constitutes a source of heterogeneity exploitable by the algorithm to improve its ability to search new evolutionary paths.

Subsequently, a new subpopulation P'_p is built up for the current node. Firstly, elitism is performed by copying the best individual of $\mathit{\Pi}_p$ in P'_p. Secondly,

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Algorithm 1 Pseudo-code of IDEA on a generic node

```
randomly generate an initial subpopulation P_p of n individuals
evaluate fitness of each individual
while halting conditions are not satisfied do
  for each generation do
    generate the new subpopulation through the classical actions of the chosen EA
    evaluate fitness of individuals in the new subpopulation
    if invasion time then
      create the propagule M_{P_p} for the current subpopulation
      send the propagule M_{P_p} to the neighboring subpopulations
      receive the propagules M_{P_p̃} from the neighboring subpopulations
      construct the founding subpopulation Π_p
      copy the best individual of Π_p in P'_p
      select n − 1 individuals in Π_p through stochastic universal sampling
      insert the n − 1 selected individuals into the new population P'_p
    end if
  end for
end while
```

stochastic universal sampling [26] (SUS) is applied on Π_p and $n-1$ individuals selected in this way are inserted into P'_p. SUS permits to choose without bias potentially useful solutions from Π_p via a fitness-proportionate selection. This mechanism, interpretable as an additional reproduction induced by introduction of elements arriving from neighboring nodes, enhances both the exploitation and the exploration of the original algorithm and guarantees their trade-off.

The described procedure, whose pseudo-code is represented in Algorithm 1, is repeated on each node until the halting conditions are satisfied.

At this point, it is worth stressing some remarkable differences between the particular 'migration' mechanisms characterizing IDEA and the other common distributed EAs based on the stepping stone-model. Usually, this kind of distributed EAs exchanges individuals, selected by some specific characteristic, between the islands with a ratio equal to one: each immigrated individual substitutes an individual in the receiving subpopulation. Besides, such a substitution mostly occurs by either a swap or a copy-and-replacement.

IDEA acts in a very different way. Here, exchange is performed in three distinct stages: 1) the formation of the propagules in neighboring islands according to equation (1); 2) the creation of a founding subpopulation as specified in equation (2) with a size much larger than that of the local subpopulation; 3) the generation of a new local subpopulation through the application of an elitism mechanism and SUS on the founding subpopulation. Each stage exerts a selective pressure not traceable in other distributed evolutionary algorithms where substitutions generally take place without any actual competition among individuals.

Ultimately, in our opinion, the model of exchange of the solutions among subpopulations, inspired to biological invasion phenomenon and presented in

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

this work might be considered as an extra evolutionary mechanism with respect to the main one implemented by the specific evolutionary algorithm adopted.

4 Experiments

To investigate the effectiveness of IDEA scheme, we have decided to compare its behavior against that of DDE on a set of 12 benchmark functions. Namely, the 500–sized versions of Ackley, Griewangk, Michalewicz, Rastrigin, Rosenbrock and Schwefel functions, as well as those of their rotated versions, have been taken into account [23].

DDE consists of a set of classical DE schemes, running in parallel, assigned to different processing elements arranged in a torus topology [17], in which each generic DE instance has four neighboring communicating subpopulations, and migration rate and migration interval have been set equal to 1 and 5, respectively. The individual sent is the best one and it randomly replaces an individual in the neighboring subpopulation, except for the local current best one. The mutation mechanism used in DDE is $DE/rand/1/bin$ [15, 16].

To carry out the comparison, we have implemented an IDEA containing instances of DE, hence we have named it IDEA–DE. Throughout the experiments, the values for the parameters *crossover ratio* (CR) and *scale factor* (F) have been set to 0.3 and 0.7, respectively, as suggested in [23], for both algorithms, and the DE operator in IDEA-DE is the same of DDE. A 4×4 folded torus has been chosen as topology for the network of subpopulations, resulting in a total of 16 nodes. The total population size has been chosen as 200, which results in eight subpopulations with 12 individuals and eight containing 13. The number of generations has been set to $2,500$.

A first phase of our investigation has aimed at finding the best possible value for the migration interval $\mathscr{T}$ for any function, and this for both algorithms. We have considered a given range of possible values, i.e., 5, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. For any such value 25 runs have been effected for each function and each algorithm, and the averages A_v of the best final fitness values over the 25 runs have been computed. Table 1 reports the best values of $\mathscr{T}$, together with the corresponding values of A_v.

A very important remark from Table 1 is that for any problem, apart from Rastrigin, Rosenbrock, and Schwefel rotated, the best $\mathscr{T}$ for IDEA–DE is lower than that for DDE. This seems to represent algorithmically the correlation between biological invasion success and propagule pressure, namely, large invading subpopulations and frequent introductions lead to a faster adaptation of both native and alien individuals.

Examination of the results shows that in several cases the best value for $\mathscr{T}$ is obtained at the lowest tested migration interval. For the remaining problems, it happens that the results are better and better as the migration interval increases, and this holds true until a given value for $\mathscr{T}$ is reached; after this value, the performance worsens more and more as $\mathscr{T}$ further increases.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Table 1. Best migration interval and related average final value for each problem.

	IDEA–DE		DDE		FACPDE
Problem	$\mathscr{T}$	A_v	$\mathscr{T}$	A_v	A_v
Ackley	5	$1.26 \cdot 10^{-2}$	10	$6.65 \cdot 10^{-2}$	$2.67 \cdot 10^{-2}$
Griewangk	5	$1.77 \cdot 10^{+0}$	10	$1.65 \cdot 10^{+1}$	$5.04 \cdot 10^{+2}$
Michalewicz	20	$-3.44 \cdot 10^{+2}$	30	$-3.21 \cdot 10^{+2}$	$-3.54 \cdot 10^{+2}$
Rastrigin	20	$1.46 \cdot 10^{+3}$	20	$1.89 \cdot 10^{3}$	$9.91 \cdot 10^{+2}$
Rosenbrock	10	$6.38 \cdot 10^{+2}$	10	$1.59 \cdot 10^{+3}$	$1.63 \cdot 10^{+3}$
Schwefel	20	$-1.52 \cdot 10^{+5}$	30	$-1.43 \cdot 10^{+5}$	$-1.54 \cdot 10^{+5}$
Ackley rotated	5	$1.74 \cdot 10^{-2}$	10	$1.66 \cdot 10^{-1}$	$6.92 \cdot 10^{-2}$
Griewangk rotated	5	$1.69 \cdot 10^{+0}$	10	$1.82 \cdot 10^{+1}$	$5.10 \cdot 10^{+2}$
Michalewicz rotated	30	$-7.67 \cdot 10^{+1}$	60	$-8.62 \cdot 10^{+1}$	$-1.92 \cdot 10^{+2}$
Rastrigin rotated	20	$1.66 \cdot 10^{+3}$	30	$2.51 \cdot 10^{+3}$	$1.10 \cdot 10^{+3}$
Rosenbrock rotated	5	$5.32 \cdot 10^{+2}$	10	$9.76 \cdot 10^{+2}$	$9.74 \cdot 10^{+2}$
Schwefel rotated	30	$-9.45 \cdot 10^{+4}$	30	$-9.26 \cdot 10^{+4}$	$-1.67 \cdot 10^{+5}$

Table 2. Results of Wilcoxon test for all the faced functions.

Problem	*Pr*	*H*	*result*
Ackley	$1.41 \cdot 10^{-9}$	1	+
Griewangk	$1.41 \cdot 10^{-9}$	1	+
Michalewicz	$1.41 \cdot 10^{-9}$	1	+
Rastrigin	$1.41 \cdot 10^{-9}$	1	+
Rosenbrock	$1.41 \cdot 10^{-9}$	1	+
Schwefel	$5.21 \cdot 10^{-9}$	1	+
Ackley rotated	$1.41 \cdot 10^{-9}$	1	+
Griewangk rotated	$1.41 \cdot 10^{-9}$	1	+
Michalewicz rotated	$7.12 \cdot 10^{-1}$	0	=
Rastrigin rotated	$1.41 \cdot 10^{-9}$	1	+
Rosenbrock rotated	$1.41 \cdot 10^{-9}$	1	+
Schwefel rotated	$5.60 \cdot 10^{-1}$	0	=

From the table, it can be seen that IDEA–DE performance seems superior to that of DDE on any problem, apart from Michalewicz rotated. Namely, on five problems IDEA–DE reaches results better by one order of magnitude.

To ascertain whether or not those figures allow us to conclude the superiority of the approach here proposed from a statistical viewpoint, Wilcoxon test [27] has been performed for any function by taking into account the 25 best final values achieved by both algorithms. The test is performed by taking as null hypothesis the one that the two algorithms are equivalent, at the 0.05 significance level. Results of the test are shown in Table 2 where, for each function, the values of Pr (probability of observing equivalence by chance if the null hypothesis is true) and H (1 means acceptance of the null hypothesis, 0 rejection) are reported. The last column contains a '+' if IDEA–DE is better than DDE, a '−' in the opposite case, and a '=' if the two algorithms are equivalent.

The results indicate that the null hypothesis can be rejected for all the functions but Michalewicz rotated and Schwefel rotated, for which the results of

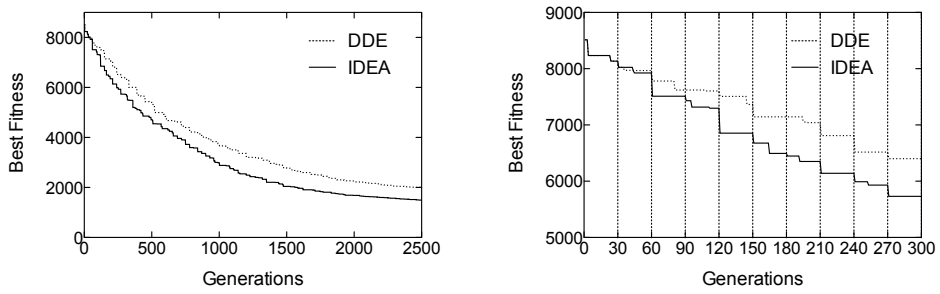


Fig. 1. Evolution of a typical run on the Rastrigin function. Left panel: the whole evolution. Right panel: zoom on the first 300 generations.

IDEA–DE and DDE should be considered as equivalent. Table 2 also says that for the only function for which DDE seems to be slightly better than IDEA–DE, i.e., Michalewicz rotated, there is actually a substantial equivalence.

Finally, in Table 1 we have also reported the average performance of FACPDE [24], one of the most performing distributed DE. As it is evident, IDEA–DE is very competitive and, in many cases, exhibits better performance.

4.1 IDEA–DE behavior and its motivations

A first remark about the behavior of our algorithm can be made by looking at Fig. 1. The left panel shows the evolution of a typical run on the Rastrigin function for both algorithms with $\mathscr{T} = 30$. It is worth noting that the general dynamics in terms of evolution is similar for both the algorithms, the difference being that IDEA–DE gets advantage over DDE in the early generations, and keeps it during the whole evolution. It looks like IDEA–DE somehow 'anticipates' DDE. The first part of this figure evidences the rapid adaptation of both native and alien individuals already described above. The right panel, instead, displays the first 300 generations only: noticeable improvements in best fitness can be noted after almost every migration.

A second important feature of IDEA–DE can be realized by taking a closer look at the behavior of our algorithm as provided in Fig. 2 that makes reference again to a typical run on the Rastrigin function for $\mathscr{T} = 30$. The figure reports the number of immigrants accepted in P'_p on node 0 (P'_0) at each migration. It is worth noting that this number varies within a minimum of 5 and a maximum of 12, and is usually between 6 and 10. So, the number of accepted migrants is noticeably higher in IDEA–DE than it is in DDE. Since the subpopulation under investigation has 12 individuals, this means that at generations $1,350$ and $1,620$ the local population is completely replaced by immigrants.

A third extremely important feature of IDEA–DE resides in the genotypic diversity. Figure 3 shows the genotypic variance for each gene in P_0 (in

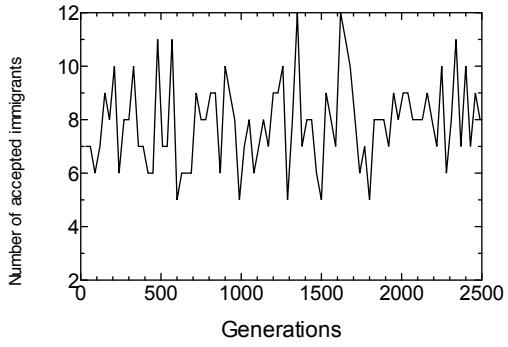


Fig. 2. The number of immigrants accepted in subpopulation P_0' at each migration.

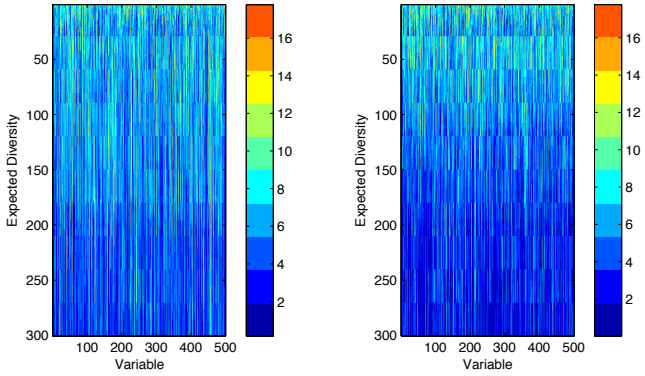


Fig. 3. Genotypic diversity in a subpopulation for DDE (left) and IDEA–DE (right).

horizontal) as a function of generations (in vertical) for a typical run of both algorithms for Rastrigin function. Only the first 300 generations are reported. Migration takes place every 30 generations. A clear difference in genotypic variance before and after any migration can be appreciated in IDEA–DE, since horizontal lines can be distinctly perceived at migrations. This says that the occurrence of each migration actually yields a net difference in genotypes, which results in a different regime in the subpopulation. The effects of migration are, instead, much less evident for DDE, for which a higher genotypic continuity is reported in figure, if we look at each gene, and horizontal lines are harder to detect at migrations.

As a further remark, we wish to say that we have also made some preliminary investigations about the use of DE operators other than the $DE/rand/1/bin$. They seem to confirm both the superiority of IDEA–DE with respect to DDE in terms of higher solution quality and the above described IDEA–DE behavior.

Finally, it should be noted that the highest computational cost of the migration model introduced by us is related to the sorting needed by SUS and is $O(n \log n)$. As regards the communication overhead, if Δt is the time required by MPI for sending just one individual, the time needed to send m individuals is between $m/2 \cdot \Delta t$ and $m \cdot \Delta t$, where $m \sim 6$ in our experiments.

5 Conclusions and future works

A new distributed scheme for EAs, IDEA, has been introduced, based on the concept of mimicking massive immigration phenomena. It has been used here to implement a distributed DE scheme. The resulting IDEA–DE algorithm has been evaluated on a set of classical test functions. Its results, compared against those of DDE, have shown the effectiveness of the proposed approach.

Since the dynamics of IDEA–DE is faster than that of DDE in achieving solutions with satisfactory quality, our algorithm could be profitably used to find suboptimal solutions to problems requiring quasi–realtime answers.

Future works will aim at carrying out a wider evaluation phase. This will be accomplished by performing sets of experiments with other DE operators, so as to ascertain IDEA–DE performance with respect to that of DDE independently of the specific operator chosen, and to understand if some operators are, on average, better than the others for IDEA–DE. Moreover we plan to make use of a wider set of test functions, and to take into account for comparison a larger number of other distributed versions of DE available in literature. Furthermore, dependence on the values of F and CR will be investigated.

Finally, since the scheme introduced is general and could be favorably adopted for any evolutionary algorithm, we aim at using it to implement distributed versions for other evolutionary algorithms as well.

References

1. Holland, J.: Adaptation in natural and artificial systems. (1975)
2. Cantú-Paz, E.: A summary of research on parallel genetic algorithms. Technical Report 95007, University of Illinois, Urbana–Champaign, USA (1995)
3. Mühlenbein, H.: Evolution in time and space - the parallel genetic algorithm. In Rawlins, G., ed.: Foundation of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA (1992) 316–337
4. Grajdeanu, A.: Parallel models for evolutionary algorithms. Technical report, ECLab, Summer Lecture Series, George Mason University, 38 (2003)
5. Skolicki, K., De Jong, K.: The influence of migration sizes and intervals on island models. In: Proceedings of the Conference of Genetic and Evolutionary Computation, Association for Computing Machinery, Inc. (ACM (2005) 1295–1302
6. Tomassini, M.: Spatially structured evolutionary algorithms. Springer (2005)
7. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Trans. on Evolutionary Computation **6**(5) (2002) 443–462
8. Shigesada, N., Kawasaki, K.: Biological invasions: theory and practice. Oxford University Press, USA (1997)

9. Suarez, A., Tsutsui, N.: The evolutionary consequences of biological invasions. Molecular Ecology **17**(1) (2008) 351–360
10. Strayer, D., Eviner, V., Jeschke, J., Pace, M.: Understanding the long-term effects of species invasions. Trends in Ecology & Evolution **21**(11) (2006) 645–651
11. Catford, J., Jansson, R., Nilsson, C.: Reducing redundancy in invasion ecology by integrating hypotheses into a single theoretical framework. Diversity and Distributions **15**(1) (2009) 22–40
12. Kolar, C., Lodge, D.: Progress in invasion biology: predicting invaders. Trends in Ecology & Evolution **16**(4) (2001) 199–204
13. Simberloff, D.: The role of propagule pressure in biological invasions. Annual Review of Ecology, Evolution, and Systematics **40** (2009) 81–102
14. Lockwood, J., Cassey, P., Blackburn, T.: The role of propagule pressure in explaining species invasions. Trends in Ecology & Evolution **20**(5) (2005) 223–228
15. Price, K., Storn, R.: Differential evolution. Dr. Dobb's Journal **22**(4) (1997) 18–24
16. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4) (1997) 341–359
17. De Falco, I., Della Cioppa, A., Maisto, D., Scafuri, U., Tarantino, E.: Satellite image registration by distributed differential evolution. In: Applications of evolutionary computing. Volume 4448 of Lectures notes in computer science., Springer (2007) 251–260
18. Cantú-Paz, E.: Migration policies, selection pressure, and parallel evolutionary algorithms. Journal of Heuristics **7**(4) (2001) 311–334
19. Zaharie, D.: Parameter adaptation in differential evolution by controlling the population diversity. In D. Petcu et alii, ed.: Proceedings of the International Workshop on Symbolic and Numeric Algorithms for Scientific Computing. (2002) 385–397
20. Tasoulis, D., Pavlidis, N., Plagianakos, V., Vrahatis, M.: Parallel differential evolution. In: Proceedings of the Congress on Evolutionary Computation. Volume 2. (19-23 June 2004) 2023–2029
21. Kozlov, K.N., Samsonov, A.M.: New migration scheme for parallel differential evolution. In: Proceedings of the International Conference on Bioinformatics of Genome Regulation and Structure. (2006) 141–144
22. Apolloni, J., Leguizamón, G., García-Nieto, J., Alba, E.: Island based distributed differential evolution: an experimental study on hybrid testbeds. In: Proceedings of the Eight International Conference on Hybrid Intelligent Systems, IEEE Press (2008) 696–701
23. Weber, M., Neri, F., Tirronen, V.: Distributed differential evolution with explorative-exploitative population families. Genetic Programming and Evolvable Machines **10**(4) (2009) 343–371
24. Weber, M., Neri, F., Tirronen, V.: Scale factor inheritance mechanism in distributed differential evolution. Soft Computing **14** (2010) 1187–1207
25. Ishimizu, T., Tagawa, K.: A structured differential evolution for various network topologies. International Journal of Computers and Communications **4**(1) (2010) 2–8
26. Baker, J.: Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the Second International Conference on Genetic Algorithms and their application, L. Erlbaum Associates Inc. (1987) 14–21
27. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin **1**(6) (1945) 80–83

A Multi-Objective Particle Swarm Optimizer Enhanced with a Differential Evolution Scheme

Jorge S. Hernández Domínguez[1], Gregorio Toscano Pulido[1] and Carlos A. Coello Coello[2]

[1]Information Technology Laboratory, CINVESTAV - Tamaulipas, Parque Científico y Tecnológico TECNOTAM. Km. 5.5, carretera Cd. Victoria-Soto La Marina. Cd. Victoria, Tamaulipas, 87130, MÉXICO
jhernandez@tamps.cinvestav.mx, gtoscano@cinvestav.mx
[2]CINVESTAV-IPN (Evolutionary Computation Group), Depto. de Computación, Av. IPN No. 2508, San Pedro Zacatenco, México, D.F. 07360, MÉXICO
ccoello@cs.cinvestav.mx

Abstract. Particle swarm optimization (PSO) and differential evolution (DE) are meta-heuristics which have been found to be very successful in a wide variety of optimization tasks. The high convergence rate of PSO and the exploratory capabilities of DE make them highly viable candidates to be used for solving multi-objective optimization problems (MOPs). In previous studies that we have undertaken [2], we have observed that PSO has the ability to launch particles in the direction of a leader (i.e., a non-dominated solution) with a high selection pressure. However, this high selection pressure tends to move the swarm rapidly towards local optima. DE, on the other hand, seems to move solutions at smaller steps, yielding solutions close to their parents while exploring the search space at the same time. In this paper, we present a multi-objective particle swarm optimizer enhanced with a differential evolution scheme which aims to maintain diversity in the swarm while moving at a relatively fast rate. The goal is to avoid premature convergence without sacrificing much the convergence rate of the algorithm. In order to design our hybrid approach, we performed a series of experiments using the ZDT test suite. In the final part of the paper, our proposed approach is compared (using 2000, 3500, and 5000 objective function evaluations) with respect to four state-of-the-art multi-objective evolutionary algorithms, obtaining very competitive results.

1 Introduction

Particle swarm optimization (PSO) [4] is a meta-heuristic that mimics the behavior of bird flocks by "searching" based on social and personal knowledge acquired by a set of particles. Due to its effectiveness in single-objective optimization, PSO has been extended to multi-objective optimization problems (MOPs). In addition, differential evolution (DE) [6] is another meta-heuristic which has been particularly successful as a single-objective optimizer. DE works by mutating solutions based on the population's variance and this strategy has been

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

found to be a very powerful optimizer in continuous search spaces. Similar to PSO, several DE algorithms have been migrated to multi-objective optimization. Even though both meta-heuristics (PSO and DE) are simple to conceptualize and have shown competitive results on a variety of MOPs, relatively few research has been performed in regards to comparing and contrasting these two meta-heuristics in a multi-objective context. We believe that obtaining more detailed knowledge about the search capabilities of multi-objective evolutionary algorithms (MOEAs) such as these, is of utmost importance to design more powerful algorithms. For example, the few studies currently available indicate that several multi-objective particle swarm optimizers (MOPSOs) have difficulties to deal with multifrontal problems [3], while multi-objective differential evolution (MODE) approaches have shown better results on this type of problems [8,2]. The different behavior of MOPSO and MODE on this type of problems may serve as indicative that a better understanding of these two meta-heuristics will be fruitful.

In this article, we propose a hybrid MOEA that attempts to combine the advantages of MOPSO and MODE. Some of the features adopted for our hybrid approach were obtained from a series of experiments (some of these experiments are included in this document while others were obtained from our previous research [2]) performed on the Zitzler-Deb-Thiele (ZDT) test suite. As a result, aiming to adopt the mechanisms that promote desirable effects (in MODE and MOPSO), we have devised a MOEA which shows competitive results (using the IGD [9] performance measure) when compared to four state-of-the-art MOEAs.

The remainder of this paper is organized as follows. Section 2 introduces some basic concepts related to multi-objective optimization as well as the PSO and DE algorithms. In Section 3, we show a series of experiments that allowed us to better understand the way in which a MODE and a MOPSO algorithms perform the search. Then, in Section 4, we introduce the designed algorithm whose performance is assessed in Section 5. Finally, our conclusions and some possible paths for future research are provided in Section 6.

2 Basic Concepts

In this study, we assume that all the objectives are to be minimized and are equally important. We are interested in solving the general *multi-objective optimization problem* with the following form:

$$\begin{aligned} &\text{Minimize} \;\; \mathrm{f}(\mathbf{X}_i) = (f_1(\mathbf{X}_i), f_2(\mathbf{X}_i), \ldots, f_M(\mathbf{X}_i))^T \\ &\text{subject to} \;\; \mathbf{X}_i \in \mathcal{F} \end{aligned} \tag{1}$$

where $\mathbf{X}_i$ is a *decision vector* (containing our *decision variables*), $\mathrm{f}(\mathbf{X}_i)$ is the M-dimensional *objective vector*, $f_m(\mathbf{X}_i)$ is the m-th objective function, and $\mathcal{F}$ is the feasible region delimited by the problem's constraints.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

2.1 Particle Swarm Optimization

The flight of particles in PSO is typically directed by the following three components: i) ***velocity*** - this component is conformed by a velocity vector which aids in moving the particle to its next position. Moreover, an inertia weight w is used to control the amount of previous velocity to be applied. ii), ***cognitive*** - this component represents the "memory" of a particle. This is done with a *personal best* vector (we will refer to this vector as pbest) which remembers the best position found so far by a particle. iii) ***social*** - this component represents the position of a particle known as the *leader*. The leader is the particle with the best performance on the neighborhood of the current particle. These three components of PSO can be seen on its flying formula. When a particle is about to update its position, a new velocity vector is computed using:

$$\mathbf{v}_i(t+1) = w\mathbf{v}_i(t) + c_1 r_1(t)(\mathbf{y}_i(t) - \mathbf{x}_i(t)) + c_2 r_2(t)(\hat{\mathbf{y}}_i(t) - \mathbf{x}_i(t)) \quad (2)$$

Thereafter, the position of the particle is calculated using the new velocity vector:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \quad (3)$$

In Equations (2) and (3), $\mathbf{x}_i(t)$ denotes the position of particle i at time t, $\mathbf{v}_i(t)$ denotes the velocity of particle i at time t, w is the inertia weight, c_1 and c_2 are the cognitive and social factors, respectively, and r_1, $r_2 \sim$ U(0,1). Additionally, $\mathbf{y}_i$ is the best position found so far by particle i (pbest), and, $\hat{\mathbf{y}}_i$ is the neighborhood best position for particle i (leader).

2.2 Differential Evolution

Differential evolution was proposed under the idea that a convenient source for perturbation is the population itself. Therefore, in DE, the step size is obtained from the current population. In this manner, for each parent vector $\mathbf{x}_i$, a difference vector $\mathbf{x}_{i_1} - \mathbf{x}_{i_2}$ is used to perturb another vector $\mathbf{x}_{i_3}$. This can be seen in the following mutation equation,

$$\mathbf{z}_i(t) = \mathbf{x}_{i_3} + F * (\mathbf{x}_{i_1} - \mathbf{x}_{i_2}) \quad (4)$$

where $\mathbf{x}_{i_1}$, $\mathbf{x}_{i_2}$, $\mathbf{x}_{i_3}$ are pairwise different random vectors, and F is a scaling factor.

Recombination in DE is achieved by combining elements from a parent vector $\mathbf{x}_i(t)$ with elements from $\mathbf{z}_i(t)$,

$$\mu_{i,j}(t) = \begin{cases} z_{i,j}(t) \text{ if } U(0,1) < Pr \text{ or } j = r \\ x_{i,j}(t) \text{ otherwise.} \end{cases} \quad (5)$$

where r is a random integer from $[1, 2, \ldots, Dim]$, Pr is the recombination probability and j is used as index for the Dim dimensions of a solution.

3 Analysis of MOPSO and MODE

3.1 Velocity on MOPSO

Most of the current MOPSOs have premature convergence in the presence of multifrontal problems [3]. Some researchers have analyzed the behavior of the velocity of MOPSOs in multifrontal problems and have reported that the reason for their premature convergence is that velocity values grow too much [5,3]. An experiment was developed to study the velocity of MOPSO in more detail. For this experiment, we selected two test problems: ZDT1 (which can be solved relatively easily by a MOPSO), and ZDT4 (which is a multifrontal problem that is very hard to solve for several MOPSOs). For our study, we will adopt OMOPSO[1] which obtained the best results in [3] but still was unable to solve ZDT4. In order to observe the behavior of velocity, we have used boxplots to show the values reached in 30 executions using 50 particles and 100 generations (see Figures 1(a) and 1(b)). In the figures, the y axis shows the velocity values, while the x axis shows the iterations. Please note that the actual boxes is where the majority of observations are located while the dark region (composed of many $+$ symbols) shows the outliers. Figure 1(a) shows that the velocity values for ZDT1 are (approximately) in the range $[-2,2]$. For the case of ZDT4, Figure 1(b) shows that the velocity values are in the range $[-20,20]$. As previously noted by other researchers, the velocity values for ZDT4 are much bigger than for ZDT1. Nonetheless, it should be pointed out that the velocity values shown are proportional to the range of the decision variables of each problem [2]. Moreover, it is important to note that the velocity values in Figure 1(b) fall close to 0 around iteration 60. Since velocity depends on the difference of the leader and the pbest with the current particle's position, we infer that, at some point during the run, particles get really close to the leader and the pbest producing very small velocities. This premise led us to the following experiment.

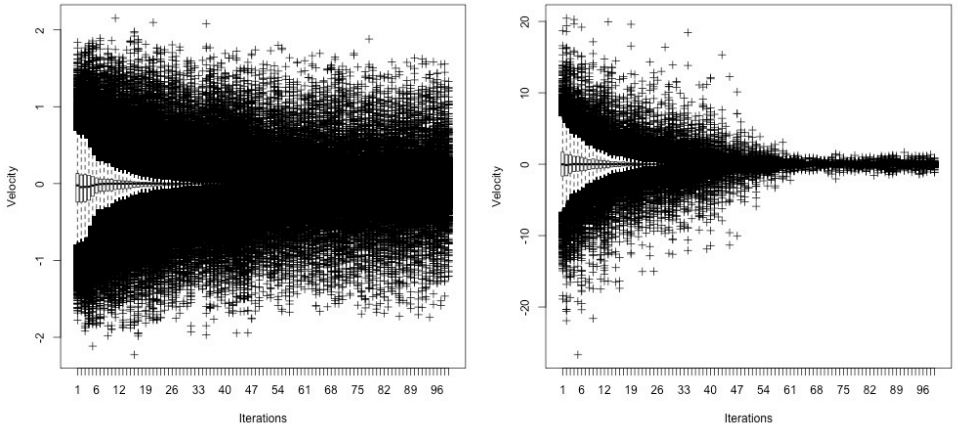
3.2 Distance of Movement

Our hypothesis is that the observed velocity values are really a consequence, rather than a cause for the premature convergence of OMOPSO in ZDT4. To validate our hypothesis, we will now try to see why is that MODE can achieve much better results on ZDT4. In this experiment we have used the same structure of OMOPSO to design a MODE algorithm. The variant used is DE/Best/1/Bin which showed the best results in previous research. In Figures 2(a) and 2(b), we plotted the Euclidean distance traveled by particles of OMOPSO and solutions of MODE in ZDT4. For OMOPSO, this is the distance between the previous and the new position while in MODE this is the distance between the parent and the candidate position.

[1] We have removed the turbulence operator included in this algorithm, since we aim to observe its raw behavior.

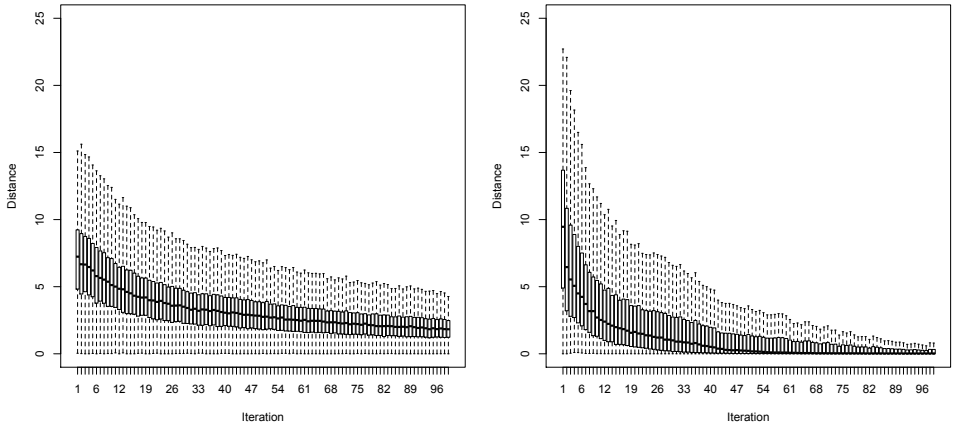
[2] ZDT1 has its variables in the range $[0,1]$ while ZDT4 has the first variable in the range $[0,1]$ while the rest are in the range $[-5,5]$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



(a) ZDT1 (b) ZDT4

Fig. 1. Velocity of OMOPSO for ZDT1 (left) and ZDT4 (right)



(a) MODE (b) OMOPSO

Fig. 2. Euclidean distance traveled from parent to offspring for MODE (left) and OMOPSO (right) on ZDT4

Figures 2(a) and 2(b) show different behaviors. OMOPSO reaches bigger distance values than MODE at the beginning of the execution but as the iterations elapse distances in OMOPSO fall rapidly to zero (meaning all particles are landing very close to its previous position). Based on our previous research

[2] and from the information seen at the presented plots, we argue that this is due to the following: OMOPSO uses only leaders and pbests to move particles while MODE uses information from the entire population. Indeed, in OMOPSO all particles are heavily attracted towards their leaders. If these leaders are diverse enough, then OMOPSO seems to move fast towards improvement. However, if leaders at some point get stuck in a local front, then all solutions will rapidly move towards that front making it harder for OMOPSO to escape (as diversity is very limited). This is not the case for MODE, in which every solution of the population can be used for perturbation and, therefore, more of the search space is explored. Moreover, solutions will land close to their parent (as the recombination operator allows certain variables to pass intact from the parent to the candidate), thus preventing that the whole population moves quickly towards local Pareto fronts.

4 Our Proposal

The two previous experiments led us to conjecture that, if properly integrated, a MOPSO combined with a MODE could be a very powerful yet efficient MOEA. Specifically, our aim is to use a MOPSO scheme, but to incorporate a DE mechanism that places a few particles very close to the leaders (rather than trying to strongly dominate the leaders as in MOPSO). We hypothesized that this sort of hybrid scheme might provide a fast convergence rate, while preserving the required diversity to avoid premature convergence. The next paragraphs describe our proposed approach.

4.1 Leader Selection Scheme

We have developed a scheme which attempts to select a leader that is diverse enough (with respect to the rest of the leaders). Our proposed mechanism works as follows. First, we obtain the centroid of all the available leaders. Then, we use roulette wheel selection to pick a leader such that the probability for each leader to be selected is proportional to the distance of that leader to the centroid (the bigger the distance of the leader to the centroid, the greater its chance of being selected). We believe that this scheme should be able to provide enough diversity in cases in which most of the leaders move towards the same region.

Here, one can argue that a leader selection scheme based on crowding would give similar results to ours. There is, however, an important difference. Crowding will favor both boundaries of the search space at all times, whereas our proposed scheme will favor the boundary opposite to the location of most of the leaders (see Figure 3).

4.2 The Use of the Velocity

It has been a common practice in PSO to decrease the previous velocity using a factor w. Moreover, this factor has traditionally been set using one of three

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

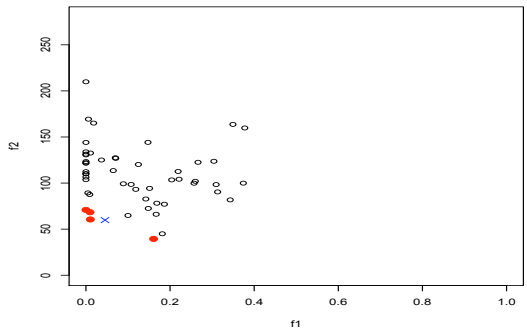


Fig. 3. Leader selection scheme for our proposal. Leaders are represented by filled red dots, solutions are black circles and the "X" symbol is the centroid of the leaders

schemes: i) w adopts a constant value, ii) w depends on the current iteration, and iii) w is randomly selected from a range. Here, we have adopted a different scheme in which w depends on the previous success of the particle. We believe that if a particle succeeded at previous iterations, then it should be beneficial to use a bigger portion of its previous velocity. On the contrary, if the particle has not been successful, then the previous velocity portion should be decreased giving a higher chance to the social and cognitive factors to take action. In short, Equation (6) works in the following way[3]. If a particle is the selected leader (meaning the particle has found a very good position at its previous iteration), then we use all of its previous velocity ($w = 1$). If a particle's current position is its pbest (the particle found its best position of the whole execution in the previous iteration), then we use a high range of its previous velocity ($w = U \sim (0.5, 1.0)$). In none of the two previous cases occur, then we use a lower percentage of the velocity ($w = U \sim (0.1, 0.5)$).

$$w = \begin{cases} 1 & \text{if } \mathbf{x}_i(t) = \hat{\mathbf{y}}_i(t) \text{ and } flip(0.5) = true \\ U \sim (0.5, 1.0) & \text{if } \mathbf{x}_i(t) = \mathbf{y}_i(t) \text{ and } flip(0.5) = true \\ U \sim (0.1, 0.5) & \text{otherwise.} \end{cases} \quad (6)$$

4.3 Moving Towards an Specific Leader

We adopt a DE mechanism to place some particles close to the selected leader rather than finding a position that strongly dominates the leader but might be moving "deeper" into some local optima. Thus, using a low probability we adopt a mechanism which will make several variables (with a high probability) to be equal to the selected leader. The aim is that the obtained particle will be close to the selected leader and that region of the search space is not lost.

[3] $flip(0.5)$ refers to a function which returns true with 50% probability

Moreover, a few variables will be obtained from a mutation based on three different random pbests. Equation (7) describes this mechanism.

$$\mathbf{x}_{i,j}(t) = \begin{cases} \mathbf{y}_{i_3,j} + F * (\mathbf{y}_{i_1,j} - \mathbf{y}_{i_2,j}) & \text{if } U(0,1) < Pr \text{ or j} = \text{r} \\ \hat{\mathbf{y}}_{p,j} & \text{otherwise.} \end{cases} \quad (7)$$

4.4 The Algorithm of Our Proposed Approach

Algorithm 1 describes our proposal, called **Multi-Objective Particle Swarm Optimizer Enhanced with a Differential Evolution Scheme** (***MOPEDS***).

Algorithm 1 Proposed algorithm (MOPEDS)

```
Initialize Population
Find non-dominated solutions (Leaders)
g = 0
while g < gMax do
    for each Particle i do
        Select leader (from non-dominated solutions) using mechanism from Section 4.1
        if U ~ (0, 1) > Pm then
            Select a w value using mechanism from Section 4.2
            Update velocity
            Update position
        else
            Select three random (different) pbest
            Move particle using DE scheme from Section 4.3
        end if
        Evaluate particle i
        if Particle i position dominates its pbest then
            update pbest to current position
        end if
    end for
    Update non-dominated solutions (Leaders)
end while
```

5 Experimental Study

Next, we compare our proposal with four state-of-the-art MOEAs: OMOPSO[4] [3], SMPSO [5], NSGA-II[5] [1], and DEMO [6] [8]. The following parameters were adopted. For NSGA-II: 0.9 for the crossover rate, $1/Dim$ for the mutation rate, 15 for the distribution index for crossover, and 20 for the distribution index for mutation. DEMO uses $Pr = 0.3$, and $F = 0.5$. OMOPSO uses $C1, C2 = U \sim (1.5, 2.0)$, and $w = U \sim (0.1, 0.5)$. Finally, SMPSO uses $C1, C2 = U \sim (1.5, 2.5)$, and $w = U \sim (0.1, 0.5)$. MOPEDS adopted $Pm = 0.2$, $F = G(0.5, 0.5)$, $C1, C2 = U \sim (1.2, 2.0)$ and $Pr = 0.2$, since these parameters provided the best behavior in our preliminary tests. The w parameter is used as described in Section 4.2. We have compared all algorithms with respect to the IGD performance measure using 2000, 3500, and 5000 objective function evaluations in order to obtain more detailed information about their behavior. There is one plot for each problem

[4] The implementation of OMOPSO used in our experiments differs from its original proposal [7] in that ϵ-dominance is not adopted.

[5] We took the code available at: `http://www.iitk.ac.in/kangal/codes.shtml`

[6] Please do not confuse DEMO with MODE. DEMO is an specific implementation of MODE which refers to multi-objective differential evolution.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

in which each algorithm is presented at the three different numbers of function evaluations indicated above (see Figure 4).

ZDT1 - In this problem it can be observed that MOPEDS is ahead of all the other algorithms at 2000 function evaluations. This shows the fast convergence rate of our proposal on this problem. Moreover, at 3500 evaluations MOPEDS is still ahead and DEMO is just a little behind. Finally, when reaching 5000 evaluations MOPEDS and DEMO show IGD values very close to 0 (the ideal value).

ZDT2 - Again, MOPEDS has the best performance at 2000 function evaluations. In fact, MOPEDS is capable of obtaining considerable advantage over the rest of the algorithms in this number of evaluations. At 3500 evaluations, IGD values for MOPEDS are already very close to 0 while OMOPSO and DEMO are a little behind. Finally, at 5000 function evaluations, OMOPSO and DEMO have reached values as good as MOPEDS.

ZDT3 - MOPEDS shows the best IGD values at 2000 and 3500 evaluations. Nonetheless, at 3500 evaluations, DEMO is very competitive also. Furthermore, at 5000 evaluations MOPEDS and DEMO have a very similar performance.

ZDT4 - In this problem, it is clear the SMPSO shows much better results than any other algorithm, while OMOPSO shows the poorest. Acknowledging this, we will omit OMOPSO and SMPSO from the discussion of this problem. At 2000 evaluations, our proposal is ahead of DEMO and a little bit behind NSGA-II. At 3500 evaluations, MOPEDS and NSGA-II show similar results. Nonetheless, the box is bigger for MOPEDS indicating a bigger dispersion than NSGA-II. Finally, at 5000 evaluations, our proposal is again competitive with NSGA-II.

ZDT6 - It is clear that all algorithms (except one) get close to the true Pareto front very early in the search (after only 2000 function evaluations). In this problem, NSGA-II requires a larger number of function evaluations to reach the true Pareto front.

Since SMPSO achieved such excellent results in ZDT4, we decided to analyze this algorithm in more detail. SMPSO is actually a modification of OMOPSO in which a velocity constriction mechanism is used[7]. Basically, this constriction factor limits the values that the velocity can take before moving a particle. The velocity is limited using Equations (8) and (9):

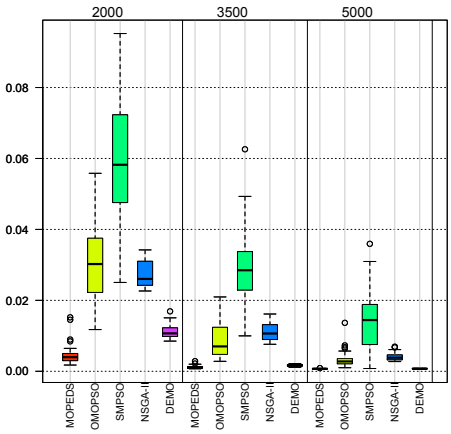
$$v_{i,j} = \begin{cases} delta & \text{if } v_{i,j} > delta_j \\ -delta_j & \text{if } v_{i,j} \leq -delta_j \\ v_{i_j} & \text{otherwise.} \end{cases} \tag{8}$$

$$delta_j = \frac{(upper\_limit_j - lower\_limit_j)}{2} \tag{9}$$

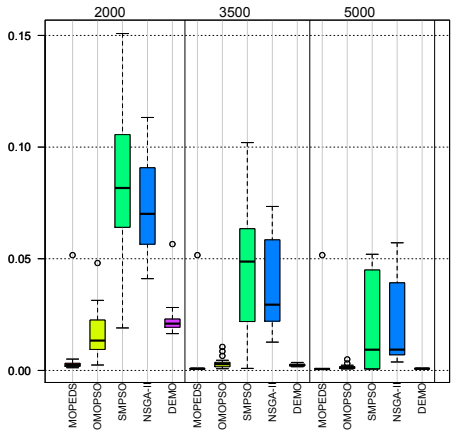
This constriction factor is the main difference with respect to the original OMOPSO. After analyzing the behavior of this approach in ZDT4, we reached the following conclusions. For ZDT4 (from the second to the tenth variables) $\delta = 5$ and $-\delta = -5$. Moreover, SMPSO (as many other MOEAs) truncates variables to their upper and lower limits when these go beyond their predefined

[7] Please refer to [5] for further details on this algorithm

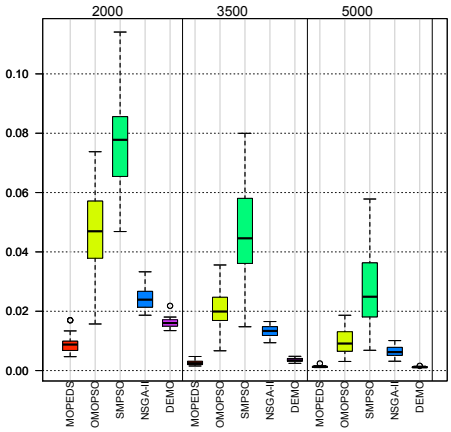
Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



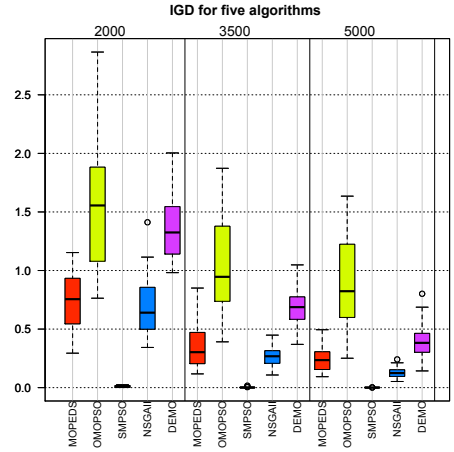
(a) Five algorithms on ZDT1



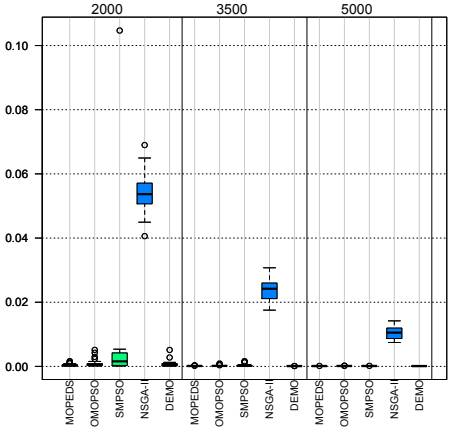
(b) Five algorithms on ZDT2



(c) Five algorithms on ZDT3



(d) Five algorithms on ZDT4



(e) Five algorithms on ZDT6

Fig. 4. Comparison of 5 algorithms using the IGD performance measure at 2000, 3500, and 5000 function evaluations in the ZDT test suite.

bounds. For ZDT4, these bounds are -5 and 5. Therefore, if a variable at iteration t lands above its upper limit, this variable will be truncated to 5. Then, at iteration $t+1$, if it happens that the velocity goes below -5 (which is $-\delta$), this velocity will be truncated to $-\delta = -5$. Therefore, when we add the velocity to the current particle's position $5 + (-5)$, we end up with 0 which is precisely in the region where the Pareto optimal set for this test problem is located. Even when this is a very clever mechanism and works perfectly in ZDT4, we decided to observe the robustness of SMPSO when the ranges of ZDT4 are changed. Thus, we moved the lower limit (again from the second to the tenth variable) to -2. The upper limit was not changed. Moreover, we also tested our proposal using these modifications with ZDT4. Both algorithms were run using 25000 function evaluations (100 particles and 250 iterations). Results are shown in Figure 5.

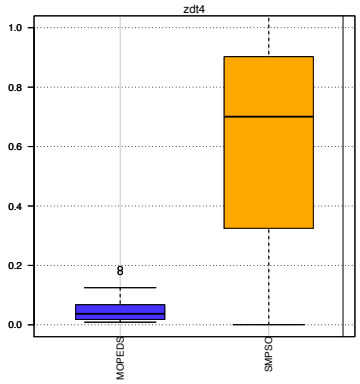


Fig. 5. MOPEDS and SMPSO at a modified version of ZDT4, using $25,000$ evaluations

From Figure 5, we can observe that the performance of SMPSO is clearly deteriorated when changing the ranges of the variables for the ZDT4 problem. We can see that SMPSO works very well some times while reporting poor results at other executions. Regarding our proposal, we observe that the modification did not have a significant impact on its performance. In fact, some improvements were achieved with regards to its execution at 2000, 3500, and 5000 evaluations. It is important to note, however, that our proposal was never able to reach the true Pareto front but could only closely approximate it.

6 Conclusions and Future Work

In conclusion, our proposal shows competitive results when compared with other state of the art MOEAs using a small number of function evaluations. Moreover, it is important to note that our proposal is able to reach competitive results in the multifrontal problem ZDT4 which has found to be quite difficult for most current MOPSOs. Therefore, we believe this indicates that MOPEDS is benefitting from the high convergence rate of MOPSO while maintaining diversity using the DE scheme. It is important to note, however, that the number of

evaluations adopted (except for 25000) are relatively small and, therefore, the plots presented here do not give further information about the performance of MOPEDS if we extend its execution. This seems important since our algorithm was not able to reach the true Pareto front at 25000 evaluations. This is certainly an issue that deserves some further work. Finally, we also believe that further investigation on mechanisms to fine tune the parameters of our proposed approach are a promising research path.

Acknowledgements

The first author acknowledges support from CONACyT through a scholarship to pursue graduate studies at CINVESTAV-Tamaulipas. The second author gratefully acknowledges support from CONACyT through project 105060. The third author acknowledges support from CONACyT project no. 103570

References

1. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (April 2002)
2. Dominguez, J.S.H., Pulido, G.T.: A comparison on the search of particle swarm optimization and differential evolution on multi-objective optimization. In: Evolutionary Computation (CEC), 2011 IEEE Congress on. pp. 1978 –1985 (june 2011)
3. Durillo, J.J., García-Nieto, J., Nebro, A.J., Coello Coello, C.A., Luna, F., Alba, E.: Multi-Objective Particle Swarm Optimizers: An Experimental Comparison. In: Ehrgott, M., Fonseca, C.M., Gandibleux, X., Hao, J.K., Sevaux, M. (eds.) Evolutionary Multi-Criterion Optimization. 5th International Conference, EMO 2009, pp. 495–509. Springer. Lecture Notes in Computer Science Vol. 5467, Nantes, France (April 2009)
4. Kennedy, J., Eberhart, R.: Particle Swarm Optimization. In: IEEE International Conference on Neural Networks. vol. 4, pp. 1942–1948 (1995)
5. Nebro, A.J., Durillo, J.J., Garcia-Nieto, J., Coello Coello, C.A., Luna, F., Alba, E.: SMPSO: A New PSO-based Metaheuristic for Multi-objective Optimization. In: 2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM'2009). pp. 66–73. IEEE Press, Nashville, TN, USA (March 30 - April 2 2009), iSBN 978-1-4244-2764-2
6. Price, K., Storn, R.: Differential Evolution - a simple evolution strategy for fast optimization (April 1997)
7. Reyes Sierra, M., Coello Coello, C.A.: Improving PSO-Based Multi-objective Optimization Using Crowding, Mutation and ϵ-Dominance. In: Coello Coello, C.A., Aguirre, A.H., Zitzler, E. (eds.) Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005. pp. 505–519. Springer. Lecture Notes in Computer Science Vol. 3410, Guanajuato, México (March 2005)
8. Robič, T., Filipič, B.: DEMO: Differential Evolution for Multiobjective Optimization. In: Coello Coello, C.A., Aguirre, A.H., Zitzler, E. (eds.) Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005. pp. 520–533. Springer. Lecture Notes in Computer Science Vol. 3410, Guanajuato, México (March 2005)
9. Veldhuizen, D.A.V., Lamont, G.B.: On Measuring Multiobjective Evolutionary Algorithm Performance. In: 2000 Congress on Evolutionary Computation. vol. 1, pp. 204–211. IEEE Service Center, Piscataway, New Jersey (July 2000)

A Novel Particle Swarm Optimization Algorithm

Shahriar Asta[1] and A. Şima Uyar[1],

[1] Computer & Informatics Faculty
Istanbul Technical University, Istanbul, Turkey

{asta, etaner}@itu.edu.tr

Abstract. In this study a novel memory based particle swarm optimization algorithm is presented. This algorithm utilizes external memory. A set of globally found best and worst positions, along with their parameters are stored in two separate external memories. At each iteration, a coefficient, based on the distance of the current particle to the closest best and closest worst particles is calculated. When updating the velocity component, this coefficient is added to the current velocity of the particle with a certain probability. Also randomized upper and lower bound values have been defined for the inertia component. The algorithm is tested on benchmark functions and it is shown empirically that it converges faster to the optima. It also outperforms the PSO and a recent improved PSO, as well as maintaining a superior precision in comparison. Convergence speed is particularly important since the method will be used in a realistic robot motion simulator environment in which the simulation time is long enough to make convergence speed a primary concern.

Keywords: Particle Swarm Optimization, External Memory.

1 Introduction

Particle Swarm Optimization (PSO) is a nature inspired meta-heuristic method. This method was first introduced by Kennedy and Eberhart in 1995 [1]. It is inspired by the swarm behavior of birds flocking, and utilizes this behavior to guide the particles to search for globally optimal solutions.

Basically, in PSO, a population of particles is spread randomly throughout the search space. The particles are assumed to be flying in the search space. The velocity and position of each particle is updated iteratively based on personal and social experiences. Each particle possesses a local memory in which the best so far achieved experience is stored. Also a global memory keeps the best solution found so far. The sizes of both memories are restricted to one. The local memory represents the personal experience of the particle and the global memory represents the social experience of the swarm. The balance between the effect of the personal and social experiences are maintained using randomized correction coefficients. The philosophy behind the velocity update procedure is to reduce the distance between the particle

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

and the best personal and social known locations. PSO is very easy to implement and there have been many successful implementations in several real world applications.

PSO is a population based heuristic approach. It can get stuck in local optima when dealing with complex multimodal functions. This is why accelerating the convergence speed as well as avoiding the local optima are two primary goals in PSO research. Multiple methods and approaches have been suggested to improve the performance of the original PSO in terms of these goals.

In [2], these efforts have been divided into four categories. The first category includes the parameter selection methods. Introducing inertia and constriction factors into the basic velocity expression or developing strategies for time independent variation of algorithm parameters are among the many methodologies in this category. The method presented in this paper fits best within this category. Other categories given in [2] are: Applications of PSO to different problem areas (second category); Generation of different algorithm strategies and analysis of convergence (third category); Hybridization (fourth category). Although, the novel approach presented in this paper focuses on parameter selection, it also tries to generate a different strategy and attempts to reduce the iteration count. This is why, it can also be included in the third category.

The proposed algorithm utilizes external memory. A set of globally found best and worst positions, along with their parameters are stored in two separate external memories. At each iteration, a coefficient, based on the distance of the current particle to the closest best and closest worst particles is calculated. When updating the velocity component, this coefficient is added to the current velocity of the particle with a certain probability. Also randomized upper and lower bound values have been defined for the inertia component. The algorithm is tested on benchmark functions and it is shown empirically that it converges faster to the optima. It also outperforms the PSO and a recent improved PSO, as well as maintaining a superior precision in comparison. Convergence speed is particularly important since the method will be used in a realistic robot motion simulator environment in which the simulation time is long enough to make convergence speed a primary concern.

This paper is organized as follows. First a brief review of PSO is given (section 2). Then some recent versions of PSO utilizing additional memory are discussed (section 3). The proposed approach is introduced in the subsequent section (section 4). The comparative experimental results are shown in section 5 and finally the conclusion and future works are presented in the last section (section 6).

2 Particle Swarm Optimization

In the basic PSO, each particle is considered as a potential solution to the numerical optimization problem in a D dimensional space. Every particle has a position in this search space and a velocity assigned to it. The position of the particle is represented by $X_i = x_{i1}, x_{i2}, \dots, x_{iD}$. The velocity of a particle is given as $V_i = v_{i1}, v_{i2}, \dots, v_{iD}$.. Also, each particle has a local memory (*pBest*) which keeps the best position that is experienced by the particle so far. A globally shared memory (*gBest*) keeps the best

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

global position found so far. This information contributes to the flying velocity of each particle, using the following equation:

$$v_i = v_i + \varphi_1 \times rand \times (pBest_i - x_i) + \varphi_2 \times rand \times (gBest - x_i) \quad (1)$$

$$x_i = x_i + v_i \quad (2)$$

where, $\varphi_1$and φ_2 are constants determining the relative influences of the personal and social experiences. Defining an upper bound for the velocity component increases the performance of the approach. Eq.2 gives the particle position update.

In [3], it has been shown that the introduction of an inertia factor to Eq.1 improves performance, since it adjusts the velocity over time and improves the search precision of the particles. Eq.1 can be rewritten as:

$$v_i = \theta \times v_i + \varphi_1 \times rand \times (pBest_i - x_i) + \varphi_2 \times rand \times (gBest - x_i) \quad (3)$$

where θ is the inertia factor and *rand* is a uniformly distributed random number in [0,1]. Later Clerc[4] introduced a constriction factor K for more efficient control and constraining of velocities. Then Eq.1 was modified as:

$$v_i = K \times \left(v_i + \varphi_1 \times rand \times (pBest_i - x_i) + \varphi_2 \times rand \times (gBest - x_i)\right) \quad (4)$$

Here *K* can be expressed as:

$$K = \frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|} \quad (5)$$

where $\varphi = \varphi_1 + \varphi_2$. The value $\varphi > 4$, prevents the explosion of the system [5], which can occur when particle velocities increase without control. According to [2], successes of inertia and constriction factor equations are problem dependent. Therefore, in this paper both equations are used and the best result is taken.

3 Related Work

As mentioned earlier, a lot of effort has been put into improving the quality of the basic PSO. Since the main focus of this work is on utilizing additional memory, we mainly try to mention studies in which additional external and internal memory (repository) have been employed in order to improve the basic PSO.

Memory based PSO proposals are mainly concentrated on various methods, by which the local and global best positions are selected and used. The work of Coello Coello et al. [6] is one of the first in this respect. In this work, the global best is determined by selection of a non-dominated solution from the external memory. Local best is updated with respect to Pareto dominance. Hu et al. [7] extended their work using dynamic neighborhoods and employed an external memory to memorize all potential Pareto optimal solutions. In their work the global best is selected among

the candidates in the external memory by defining a neighborhood objective and an optimization objective. The global best is found among the particle's neighbors, subject to the defined neighborhood objective.

One other attempt to employ additional memory in PSO is the work of Wang [8]. In this method, The Triggered Memory-Based PSO, they try to adjust the parameters of the PSO in dynamic environments where the characteristics of the search space change over time. However, this method is novel and successful in terms of presenting the effects of utilizing additional memory in PSO. In their work a certain, predefined number of globally best positions, are kept and re-injected into the search population when necessary. This method is particularly successful when the location of the optima change over time.

One other successful work which utilizes external memory is the work of Acan [9]. In their work a single globally best position is kept along with a certain number of globally worst positions. A crossover operator is used to replace a randomly chosen set of particles after each iteration.

Yet in another work, Acan [2] introduced a hybrid method where there is a global memory and a local memory for each particle. A colony, consisting of the local and global positions is then constructed and at each evaluation, members of the colony are used to update the velocity and position of the particle in process. Then the new positions are evaluated where the best replaces the particle's current velocity.

There are some other approaches, which employ additional memory and/or hybridization or other techniques. Additional information can be found in [2,10]. The main idea in almost all of these memory utilizing approaches is to re-inject the globally best position into the population during the search. In our study, however, the main idea is to deduce information from the contents of the external memory in order to affect the velocity of the particles towards the global optima.

4 The Proposed Approach

Based on observations of the application of evolutionary algorithms on robot motion, one can say that sometimes, parts of the new individuals are close to those of the optimal solution. Only a few parameters in the current individual are different and are far from being a part of the optimum solution. Although the number of these parameters may be small, their effect may cause the performance of the individual to be closer to that of the worst case.

Considering a landscape in which we have many local minima and maxima points, one can also say that, while updating the position of the individual, keeping it away from the maxima (in a minimization problem) will increase the speed of convergence to the minima. But this alone might not be sufficient, since there are problems for which both minima and maxima are in close neighborhoods or when there are many local minima. Then we need to equip the optimization algorithm with the ability to escape from the maxima and move towards the minima even when the minima are close to both the current position of the particle and the maxima.

In this work, we propose an algorithm, in which the particles are guided away from the closest worst location by correcting their positions to the location which they are supposed to be, prior to updating their positions. In order to do this, we construct two

lists for the so far found global best and global worst positions. Each of the two lists is in fact, separate external memories of the PSO. Before updating the velocity of the particles, we scan the external memory that keeps the best positions and determine the best location which is closest to the particle. We call this CB. Then we scan the external memory that keeps the global worst positions and in a similar way, we find the global worst location closest to the particle. We call this position CW. In our experiments Euclidean distances are used. However, any other distance measure may also be used for this purpose.

After choosing the closest best and worst elements from the external memories with respect to the particle, we measure the similarity of the particle to each of these elements. Eqs.7 and 8 calculate the similarity between the closest best and the particle (C_1) and the similarity between the closest worst and the particle (C_2) respectively.

$$c_1 = \frac{\sqrt{\sum_{i=1}^{D}(CB - x_i)^2}}{(x_u - x_l)} \tag{7}$$

$$c_2 = \frac{\sqrt{\sum_{i=1}^{D}(CW - x_i)^2}}{(x_u - x_l)} \tag{8}$$

Here $x_u$and x_l are the upper and lower bound values for each dimension. In our experiments this value range has been considered to be equal for all dimensions. The position correction coefficient (*C*) can be defined as in Eq.9.

$$C = \frac{|c_1 - c_2|}{(c_1 + c_2)} \tag{9}$$

We are now able to present our algorithm as follows. First, we define two external memories: one contains a fixed number of global best positions and the other contains a fixed number of global worst positions. The sizes of these two external memories are not necessarily equal. To initiate both memories, we first run the basic PSO until the iteration count is equal to the maximum size of the two external memories. At each iteration, we insert the global best and worst positions into the corresponding external memories.

After initialization, we can calculate the *C* coefficient in Eq. 9. In other words, after each evaluation of all the particles, we refresh the external memory contents and based on the information available in the external memories we calculate the coefficient. Then according to a particular probability, we either use the basic equation or the following equations which are modified forms of Eqs. 3 and 4 respectively:

$$v_i = \omega.(v_i + C) + \varphi_1.rand.(lBest - x_i) + \varphi_2.rand.(gBest - x_i) \tag{10}$$

$$v_i = K.\big((v_i + C) + \varphi_1.rand.(lBest - x_i) + \varphi_2.rand.(gBest - x_i)\big) \tag{11}$$

In Eqs. 10 and 11, we add the coefficient *C* to the current velocity of the particle, in order to move the particle away from the worst position towards the closest best. When we use Eq.10 for velocity update, based on observations, we can say that

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

decreasing the inertia linearly, will decrease the effect of the coefficient and yield early convergence.

The pseudo code of the algorithm is shown in Fig. 1. In the algorithm, ω is the inertia which is only applied to Eqs. 10 and 11. Eqs. 3 and 4 use a fixed inertia value of 0.99. $\Delta\omega$ is the inertia decreasing factor.

```
1-  Algorithm CBCW_PSO
2-  begin
3-  Randomly initialize the particles within their ranges and set all velocities to zero
4 – Define external memory for best positions (bests) . Initialize to all zeros
5 - Define external memory for worst positions (worsts). Initialize to all zeros
6 - Until a certain number of iterations is reached or convergence do
7 -   for each particle x_i do
8 -   x_i = x_i + v_i
9 -   if particle position at each dimension exceeds its allowed range
10-       x_i = random from range
11-   val = evaluate particle
12-   update the local best
13-   replace the worst in bests array with gbest
14-   replace the best in worsts array with gworst
15-   if iteration count > max(size(bests), size(worsts))
16-     for each particle x_i
17-       CB = closest best to particle
18-       CW = closest worst to particle
19-       Calculate c_1 and c_2 according to Eqs. 7 and 8
20-       calculate C according to Eq. 9
21-       for each dimension
22-           with probability p
23-           update the position like the basic PSO in Eqs. 3 and 4
24-           otherwise
25-           update the position using Eqs. 10 and 11
26-       if ω ≥ lower bound + ε
27-           ω = ω − Δω − ε          // decrease the inertia
28-       else
29-           ω = upper bound – ε     // resetting the inertia
30- end
```

Fig 1. The pseudo code for CBCW-PSO algorithm

Since this algorithm is based on the distance of the particle to its globally closest best and globally closest worst, we refer to it as CBCW-PSO from here on. Decreasing the inertia (ω in Fig. 1) means that during this period we increase the importance of the social and cognitive factors and pay less attention to the actual velocity of the particle. In cases where the particle is stuck in local optima, using a linearly decreasing inertia leaves the particle with a velocity less than what the particle requires to escape those optima. This is why we decrease the inertia value linearly until a certain threshold is reached. At such a point, the inertia is reset to its upper bound. Here we defined a lower and upper bound value for the inertia. In order

to achieve diversification, some randomness is added to the upper and lower bounds and the decreasing factor ($\Delta\omega$ in Fig. 1).

5 Experiments

The performance of the proposed algorithm is compared with the performance of the improved PSO presented in [2] called CLPSO. This work has been chosen due to its success over the performance of the basic PSO and the extent of the range of problems with which it has been tested. Since an extensive comparison between PSO and CLPSO has been presented in [2], we do not compare our proposed algorithm with PSO. To see the comparison results between the CLPSO and the basic PSO, please refer to [2].

Several unimodal and multimodal benchmark problems have been adopted from [11] and [12]. The list of the test functions and some of their characteristics can be seen in Table 1. In the table "Min" column gives the optimum value of the function, the "Range" column gives the defined range of the parameters. The range is same for all dimensions. Functions $f_1 - f_8$ are unimodal. Function $f_1$is the shifted sphere function. Function f_2 produces hyper ellipsoids; it is continuous and convex. The f_3 function is another simple unimodal function. Function f_4 consists of plateaus. Function f_5 is referred to as sum of different power functions. Function f_6 is a noisy problem. Function f_7 is the high conditioned elliptic function. Function f_8 is the Rosenbrock function. The global optimum lies inside a long narrow parabolic shaped valley. Finding the valley itself is trivial. However, finding the global optima is considered to be a difficult task. Function f_9 is the Schwefel function. This function is deceptive, in that the global minima are geometrically distant over the parameter space from the next best local minima. Therefore the search algorithms are potentially prone to converge in the wrong direction. Function f_{10} or the Rastrigin function, is based on the function of De Jong with the addition of cosine modulation in order to produce frequent local optima. This function is highly multimodal, however the location of local minima are regularly distributed. Function $f_{11}$is the Griewank function and has similar properties to the Rastrigin function. f_{12} is the Ackley function and is multimodal. $f_{13}$is the Weierstrass function. This function is a difficult, multimodal function with multiple local optima. Functions $f_{14} - f_{16}$ are easy to solve, two dimensional functions.

5.1 Experimental Design

The algorithm settings for which the experiments have been done are as follows:
$\Delta\omega = 10^{-5}$, $\varepsilon = 10^{-6}$, $lower\ bound = 0.95\ (0.8\ for\ f_9)$, $upper\ bound = 0.99$. The upper and lower bound of ω as well as $\Delta\omega$ are randomized with a small number, ε. The value of ε has been chosen to be smaller than the value of $\Delta\omega$ to prevent ω from exceeding its boundaries. Choosing a small value for ε also makes sure that ω's value maintains a smooth and slow motion between its boundaries as if $\Delta\omega$ where not

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

randomized. The value for $\Delta\omega$ is chosen such that it shows a decrement rate close to that of [2] for sake of comparison. The upper bound value is used as suggested in several publications and its validity has been verified in our experiments. The lower bound, however, is experimentally tuned.

Table 1. Different benchmark functions with which the proposed algorithm has been tested.

Benchmark Problems	Min	Range
$f_1(\vec{x}) = \sum_{i=1}^{D} x_i^2$	0	$-5.12 \le x_i \le 5.12$
$f_2(\vec{x}) = \sum_{i=1}^{D} \sum_{j=1}^{i} x_i^2$	0	$-65 \le x_i \le 65$
$f_3(\vec{x}) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} x_i$	0	$-10 \le x_i \le 10$
$f_4(\vec{x}) = \sum_{i=1}^{D} \left(\left\lfloor x_i + \frac{1}{2} \right\rfloor \right)$	0	$-100 \le x_i \le 100$
$f_5(\vec{x}) = \sum_{i=1}^{D} \lvert x_i^{i+1} \rvert$	0	$-1 \le x_i \le 1$
$f_6(\vec{x}) = \left(\sum_{i=1}^{D} (i+1) x_i^4 \right) + rand[0,1[$	0	$-5.12 \le x_i \le 5.12$
$f_7(\vec{x}) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} x_i^2$	0	$-100 \le x_i \le 100$
$f_8(\vec{x}) = \sum_{i=1}^{D} (100 \times (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	0	$-30 \le x_i \le 30$
$f_9(\vec{x}) = \sum_{i=1}^{D} -x_i \times \sin(\sqrt{\lvert x_i \rvert})$	-12569.5	$-500 \le x_i \le 500$
$f_{10}(\vec{x}) = \sum_{i=1}^{D} (x_i^2 - 10 \times \cos(2\pi x_i) + 10)$	0	$-5.12 \le x_i \le 5.12$
$f_{11}(\vec{x}) = \frac{1}{4000} \left(\sum_{i=1}^{D} x_i^2 \right) + \left(\prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i+1}} \right) \right) + 1$	0	$-600 \le x_i \le 600$
$f_{12}(\vec{x}) = -20 \times \exp\left(-0.2 \times \sqrt{\frac{1}{n} \sum_{i=1}^{D} x_i^2} \right) - \exp\left(\frac{1}{n} cos(2\pi x_i) \right) + 20 + e$	0	$-32 \le x_i \le 32$
$f_{13}(\vec{x}) = \sum_{i=1}^{D} \left(\sum_{k=0}^{Kmax} a^k \cos\left(2\pi b^k (x_i + 0.5) \right) \right) - D \sum_{k=0}^{Kmax} (a^k \cos(2\pi b^k \times 0.5))$	0	$-0.5 \le x_i \le 0.5$
$f_{14}(\vec{x}) = 4x_0^2 - 2.1x_0^4 + \frac{1}{3} x_0^6 + x_0 x_1 - 4x_1^2 + 4x_1^4$	1.0316	$-5 \le x_i \le 5$
$f_{15}(\vec{x}) = \left(x_1 - \frac{5.1}{4\pi^2} x_0^2 + \frac{5}{\pi} x_0 - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) \cos(x_0) + 10$	0.398	$-5 \le x_i \le 15$
$f_{16}(\vec{x}) = \frac{-(1 + \cos(12\sqrt{x_0^2 + x_1^2}))}{\frac{1}{2}(x_0^2 + x_1^2) + 2}$	0	$-5.12 \le x_i \le 5.12$

The value of the probability p, which determines whether to use Eqs. 3 and 4 or Eqs. 10 and 11, has a direct effect on the convergence speed of the algorithm. For lower values, the algorithm switches to Eqs. 10 and 11. This adds an extra amount to the velocity of the particle in the direction of its closest best position and opposite the direction of its closest worst position. However, there is a trade-off. Frequently using the extra velocity for a long time, results in convergence in a wrong direction. This is due to the fact that local optima may be scattered and far from each other but at a relatively equal distance with respect to the particle. This is why a value above 0.5 has been considered for the probability p (0.8 in general and 0.5 for the f_9 problem). Our experiments show that setting the probability p proportional to $(1 - \omega)$, increases the convergence speed, since binding p to the value of ω, which is a randomized value,

provides more diversification. Replacing the linearly decreasing inertia in the original form of Eq. 10 with an inertia value, which is subject to multiple restarts, also shows to be a very influencing factor on convergence speed. This is the methodology of choice in our experiments.

The maximum number of swarm evaluations is set to 40000 for the CBCW-PSO and equivalently to 2000 for the CLPSO. CLPSO runs an internal loop for $m \times n$ number of elements inside the colony. Here *m* represents the local memory size and *n* represents the global memory size. Referring to the work of Acan [2], the recommended local and global memory sizes are 5 and 4 respectively. Then setting the maximum evaluation to 2000 does in fact perform a total of 40000 swarm evaluations which is equal to that of CBCW-PSO. The size of the external memory for best and worst global positions is 2 and 4 respectively. The swarm size in all the experiments is constant and set to be equal to 30, which is the dimension of the functions. All the experiments are performed over 10 runs for each problem and algorithm. The test platform is an 8 core (3.2 GHz) system running on Ubuntu 10.04 with 4 GB RAM. The test results can be seen in Table 2. Outcomes less than or equal to 10^{-5}, is considered as zero.

Table 2. Result of applying both CLPSO and CBCW-PSO on various benchmark functions. The bold numbers, represent the best results.

Functions	Dim	CLPSO			CBCW-PSO			t-test
		Min	Max	Avg	Min	Max	Avg	
$f_1(\vec{x})$	30	0	0	0	0	0	0	S
$f_2(\vec{x})$	30	0	0	0	0	0	0	S
$f_3(\vec{x})$	30	0	0	0	0	0	0	S
$f_4(\vec{x})$	30	0	0	0	0	0	0	S
$f_5(\vec{x})$	30	0	0	0	0	0	0	S
$f_6(\vec{x})$	30	0	0	0	0	0	0	S
$f_7(\vec{x})$	30	0	0	0	0	0	0	S
$f_8(\vec{x})$	30	0.033	7.93	3.899	**0.001**	**4.001**	**0.911**	**S+**
$f_9(\vec{x})$	30	-12451	-11622	-12107.5	**-12569.5**	**-12332**	**-12438**	**S+**
$f_{10}(\vec{x})$	30	48.915	237.94	173.8	**0**	**1.989**	**0.994**	**S+**
$f_{11}(\vec{x})$	30	0	0.11	0.042	**0**	**0.041**	**0.012**	**S+**
$f_{12}(\vec{x})$	30	2.2209	5.22	3.318	**0**	**0**	**0**	**S+**
$f_{13}(\vec{x})$	30	2.7555	7.19	5.922	**0**	**1.57**	**0.15**	**S+**
$f_{14}(\vec{x})$	2	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	S
$f_{15}(\vec{x})$	2	0.398	0.398	0.398	0.398	0.398	0.398	S
$f_{16}(\vec{x})$	2	0	0	0	0	0	0	S

5.2 Results

In Table 2, the minimum, maximum and average values obtained over 10 runs are given. The "dim" column shows the number of dimensions used for each function. Better results are shown in bold in the table. A t-test at a significance level of 0.95 has been performed to test for statistical significance of the differences. In the last

column, S+ means that CBCW-PSO results are statistically significantly better than those of CLPSO and S means no statistically significant differences were found.

As it is shown in Table 2, the CBCW-PSO, outperforms the CLPSO in functions$f_8 - f_{13}$. These functions are all multimodal functions for which CLPSO and PSO are unable to converge to the optima in the experiments. The performance of the CBCW-PSO is better also in terms of the maximum value and the average fitness.

As mentioned earlier, the real purpose and motivation behind the design of CBCW-PSO is to achieve acceptable results in lower iteration counts. In other words, there is a need for an optimization algorithm as good as PSO or improved versions of it, like CLPSO, which yields at least the same results in a fewer number of evaluations. We have shown that CBCW-PSO achieves better results, compared to CLPSO in a fixed number of iterations. Although this alone indicates a better convergence speed, still presenting this superiority in an analytical form is needed. There are functions in Table 1 for which both algorithms find the global optima. In fact, looking at Table 2, one can see that in functions $f_8 - f_{13}$, CLPSO is not able to find the optima, given a fixed limit on maximum evaluation counts. Again Table 2 shows that CBCW-PSO is able to find the optima in almost all of the benchmark functions (in f_8, although both algorithms are unable to find the optima, CBCW-PSO still gives a closer value to the optimum). The question is how much faster CBCW-PSO is able to find the optima.

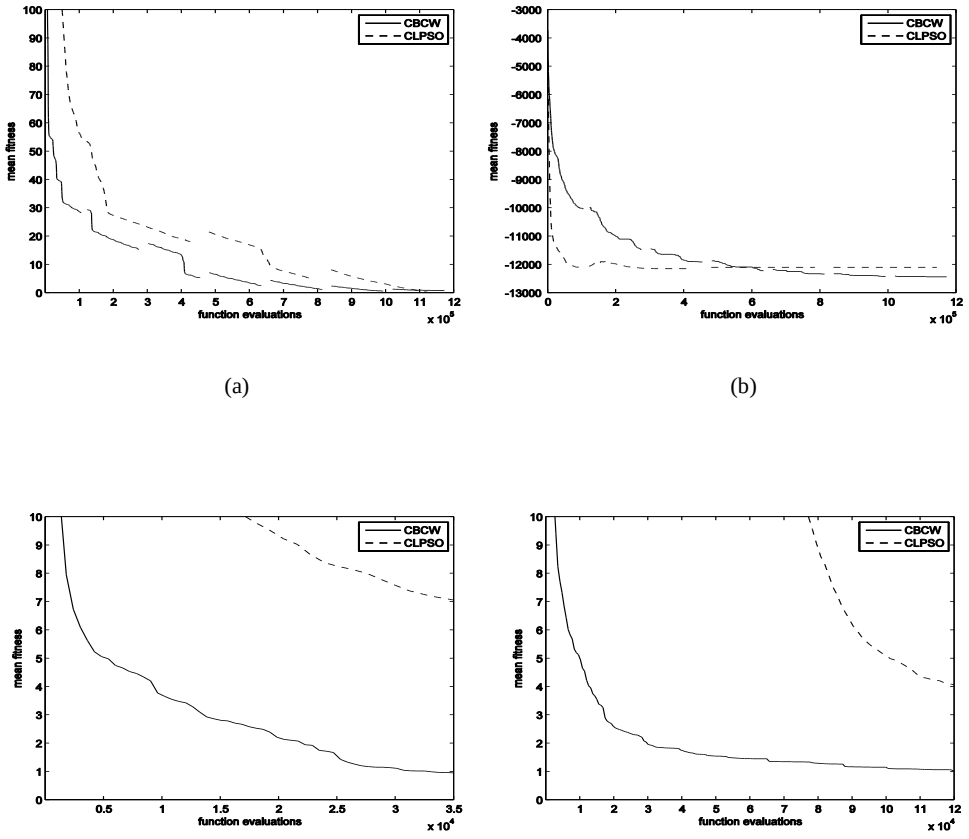
Table 3 shows the average number of evaluations until the first hit, where the global best in the algorithm is the same as the location of the optima, if found. The last column of this table represents the 95% significance t-test results with first hit time as the test parameter.

Table 3.Comparing CBCW-PSO and CLPSO in terms of average number of evaluations until the first hit for each of the benchmark functions for which at least one of the two algorithms finds the optima. For f_8 none of the methods located the optimum.

Functions	**CLPSO**	**CBCW-PSO**	**t-test**
$f_1(\vec{x})$	122820	**15255**	**S+**
$f_2(\vec{x})$	165900	**31419**	**S+**
$f_3(\vec{x})$	299880	**122265**	**S+**
$f_4(\vec{x})$	1199400	**77832**	**S+**
$f_5(\vec{x})$	100920	**2340**	**S+**
$f_6(\vec{x})$	122280	**7605**	**S+**
$f_7(\vec{x})$	247860	**33582**	**S+**
$f_8(\vec{x})$	-	**-**	
$f_9(\vec{x})$	-	**1109850**	**S+**
$f_{10}(\vec{x})$	-	**1076010**	**S+**
$f_{11}(\vec{x})$	1094340	**963669**	**S**
$f_{12}(\vec{x})$	-	**57153**	**S+**
$f_{13}(\vec{x})$	-	**646980**	**S+**
$f_{14}(\vec{x})$	3180	**531**	**S+**
$f_{15}(\vec{x})$	2700	**489**	**S+**
$f_{16}(\vec{x})$	2120	**775**	**S+**

Also, convergence plots of four functions, namely f_8, f_9, f_{12} and f_{13}, given as best fitness plots, averaged over 10 runs has been shown in Fig 2. Functions f_9, f_{12}, f_{13} show how CBCW-PSO performs better than CLPSO since in all three cases CLPSO is unable to converge to the optimum while CBCW-PSO converges precisely. Function f_8 is chosen since it demonstrates the fact that although both algorithms fail to find the optimum, CBCW-PSO maintains a better converging behavior.

As can be seen (both in Table 3 and Fig. 2), CBCW-PSO, needs a shorter number of evaluations to find the optima. One interesting point is that, although CLPSO seems to converge faster than CBCW-PSO in case of function f_9, due to its random choice of best position in local memory, it jumps to some other local optima in later evaluations. This shows that CBCW-PSO maintains a monotonically decreasing behavior with respect to the best fitness.



(a) (b)

Fig 2.The evolution of the best fitness averaged over ten runs for each function. (a) f_8 (b) f_9 (c) f_{12} and (d) f_{13}

6 Conclusions and Future Work

In this paper, we proposed a new and novel PSO, based on external memory for both global best and worst positions. We showed that the proposed algorithm outperforms a recent improvement of PSO, CLPSO, in almost all of the tested benchmark functions, in terms of precision and convergence speed.

However, the algorithm settings may be problem dependent. In case of function f_9, the value chosen for some of the parameters are different than the general settings. Although applying the general settings of the algorithm in this case (function f_9), also yields good results, there exists settings which yield better results. Based on this, we will be introducing adaptive parameter settings for the algorithm in our future studies. Also, after improving our algorithm, we will use it in a realistic robot motion simulator environment.

References

[1] J. Kennedy, R. C. Eberhart, "A New Optimizer Using Particle Swarm Theory", Sixth International symposium on Micro Machine and Human Science, IEEE, 1995

[2] A. Acan, "A Memory-Based Colonization Scheme for Particle Swarm Optimization", IEEE Congress on Evolutionary Computation (CEC) , 2009.

[3] Y. Shi, R. C. Eberhart, "Parameter Selection in Particle Swarm Optimization", 7th International Conference on Evolutionary Programming VII. (EP), 1998

[4] M. Clerc, "The Swarm and The Queen: Towards a deterministic and Adaptive Particle Swarm Optimization", IEEE Congress on Evol. Comp. (CEC), 1999.

[5] M. Clerc, J. Kennedy, "The Particle Swarm – Explosion, Stability, and Convergence in a Multidimensional Complex Space", IEEE Transactions on Evol. Comput., vol. 6, no. 1, pp. 58–73, Feb. 2002.

[6] C. A. Coello Coello, M. S. Lechuga,"MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization", IEEE Congress on Evol. Comp. (CEC), 2002.

[7] X. Hu, R. C. Eberhart, Y. Shi,"Particle Swarm with Extended Memory for Multiobjective Optimization", Proceedings of the IEEE Swarm Intelligence Symposium (SIS), 2003.

[8] H. Wang, D. Wang, S. Yang, "Triggered Memory-Based Swarm Optimization in Dynamic Environments", Applications of Evolutionary Computing, 2007

[9] A. Acan , A. Gunay, "Enhanced Particle Swarm Optimization Through External Memory Support", IEEE Congress on Evolutionary Computation (CEC), 2005.

[10] X. Hu, Y. Shi, R. C. Eberhart, "Recent Advances in Particle Swarm", IEEE Congress on Evolutionary Computation (CEC), 2004.

[11] R.Thomsen, J. Vesterstrom, "A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems", IEEE Congress on Evolutionary Comp. (CEC), 2004.

[12] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. -P. Chen, A. Auger, S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization", IEEE Congress on Evol. Comp. (CEC), 2005.

Evolution of multisensory integration in large neural fields

Benjamin Inden[1], Yaochu Jin[2], Robert Haschke[3], and Helge Ritter[3]

[1] Research Institute for Cognition and Robotics, Bielefeld University
binden@cor-lab.uni-bielefeld.de
[2] Department of Computing, University of Surrey
[3] Neuroinformatics Group, Bielefeld University

Abstract. We show that by evolving neural fields it is possible to study the evolution of neural networks that perform multisensory integration of high dimensional input data. In particular, four simple tasks for the integration of visual and tactile input are introduced. Neural networks evolve that can use these senses in a cost-optimal way, enhance the accuracy of classifying noisy input images, or enhance spatial accuracy of perception. An evolved neural network is shown to display a kind of McGurk effect.

1 Introduction

Animals take advantage of different sensory facilities to react appropriately to their environment. Among these facilities are visual, auditory, tactile, olfactory, gustatory, and electric senses. Researchers are typically interested in questions like: How are percepts from different modalities perceived as a single entity? What happens if there is conflicting input from different modalities? How can top-down influences (attention) bias the process of integration? [1,2]

Here we are particularly interested in evolutionary aspects of multisensory integration. Advantages of multisensory integration may include increased spatial accuracy of perception, increased classification accuracy, and minimizing costs associated with different forms of perception. The purpose of this study is to introduce a number of sufficiently understandable tasks that require multisensory integration to be solved, to verify that artificial neural networks can evolve to solve these tasks, and to get a rough look at how these networks do it.

Studying the integration of input from different modalities has already been possible for some time in the context of evolutionary robotics. One important difference between earlier studies and what we present here is that the techniques for evolving neural networks have advanced significantly in the meantime [3]. Previously, only a few contact or distance sensors were typically used. If there was camera input, it was usually pre-processed to reduce its dimension [4]. It is now possible to evolve very large neural networks that take input from not just a few sensors, but from whole sensor arrays, e.g. cameras. The NEAT-fields method used here for neuroevolution is particularly suited to these tasks

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

because it evolves not just the connectivity between single neurons, but also the connectivity between complete neural fields. How this method solves a range of different problems and how its implementation details and default parameter settings were determined is discussed in detail elsewhere [5,6,7]. Of course, multisensory integration has also previously been studied using large hand designed networks (e.g. [8]). However, an advantage of evolving neural networks is that often solutions are found that are free from human bias, or beyond human imagination.

2 Methods

2.1 The NEATfields method

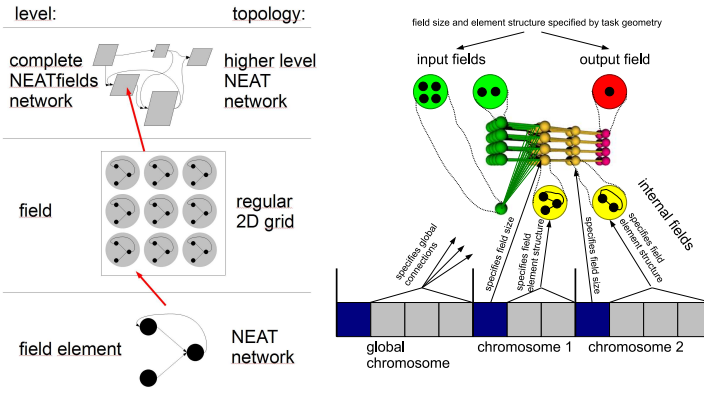


Fig. 1. (a) The three levels of architecture in NEATfields. (b) Construction of a NEATfields network from its genome (bottom). The balls in the central network represent field elements. Their contents are shown in the circles above and below. In these circles, black dots represent the individual (input, output or hidden) neurons.

Neural networks NEATfields is based on the well known NEAT method [9] that simultaneously evolves the topology and the connection weights of neural networks. NEATfields networks have three levels of architecture (see fig. 1 (a)). At the highest level, a network consists of a number of fields that are connected just like individual neurons are in a NEAT network. At the intermediate level, fields are collections of identical (or similar) subnetworks with a two dimensional topology. At the lowest level, these subnetworks, termed field elements, are recurrent NEAT networks composed of individual neurons. A complete NEATfields network consists of at least one internal field as specified by the genome, and fields for network input and output as specified by the given task. There can be

several input and output fields with different dimensions. For example, a bias input can be provided as an input field of size 1×1. Within the NEATfields network, connections can be local (within a field element), lateral (between field elements of the same field), or global (between two fields). It should be noted that connections between field elements or fields are in fact connections between individual neurons in these field elements or fields. How they are established will be described below. Evolution is started with a single internal field of size 1×1 that is connected to all input and output fields. So there is initially only one field element, and it contains one neuron for each different output.

The activation of the neurons in NEATfields is a weighted sum of the outputs of the neurons $j \in J$ to which they are connected, and a sigmoid function is applied on the activation: $o_i(t) = \tanh(\sum_{j \in J} w_{ij} o_j(t-1))$. Connection weights are constrained to the range $[-3, 3]$. There is no explicit threshold value for the neurons. Instead, a constant bias input is available in all networks.

Encoding The parameters for an individual field are encoded in the genome on a corresponding chromosome (see fig. 1 (b)). The first gene in a chromosome specifies the field size in x and y dimensions. After that node and connection genes for one field element are specified. All genes contain a unique reference number that is assigned once the gene is generated through a mutation. In addition, connection genes contain a connection weight, a flag indicating whether the connection is active, and the reference numbers of the source and target neurons.

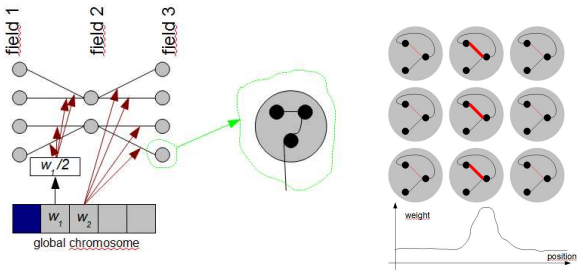


Fig. 2. (a) An example of how the global connections between neurons in fields of different sizes (shown here as one dimensional) are created using a deterministic and topology preserving method. The genetically specified weights are automatically scaled if necessary. As shown in detail on the right side, connections go in fact to individual neurons within the field elements as specified by the connection gene. (b) Dehomogenization of neural fields by the focal areas technique. The thickness of connections here symbolizes their weights.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

There is a special chromosome that contains genes coding for global connections. Global connections contain reference numbers of a source and a target. These can be reference numbers either of neurons or of inputs and outputs as specified by the task description. They must be in different fields — global connections between two neurons in the same field are never created by the NEATfields method. Due to the higher level architecture of NEATfields, a neuron with a given reference number will be present n times in a field with n field elements. A single global connection gene implicitly specifies connections for all these neurons. If a global connection is between neurons in fields with the same sizes, every field element in the target field will get a connection from the field element in the source field that has the same relative position (in x and y dimension) in its field. Their connection weights are all the same because they are all derived from one gene. If field sizes are different in a dimension, then the fields will still be connected using a deterministic and topology preserving method (see fig. 2 (a)): if the source field is smaller than the target field, each source field neuron projects to several adjacent target field neurons, whereas if the source field is larger than the target field, the target field neurons get input from a number of adjacent source field neurons, while the genetically specified connection weight is divided by that number. That way, field sizes can mutate without changes in the expected input signal strength.

Mutation operators for the field elements The NEATfields method uses mutation operators that are very similar to those of the original NEAT implementation for evolving the contents of the field elements. The most common operation is to choose a fraction of connection weights and either perturb them using a normal distribution with standard deviation 0.18, or (with a probability of 0.15) set them to a new value. The application probability of this weight changing operator is set to 1.0 minus the probabilities of all structural mutation operators, which amounts to 0.8815 in the experiments reported here. In general, structural mutations are applied rarely because they will cause the evolutionary process to operate on larger neural networks and search spaces. A structural mutation operator to connect neurons is used with probability 0.02, while an operator to insert neurons is used with probability 0.001. The latter inserts a new neuron between two connected neurons. The weight of the incoming connection to the new neuron is set to 1.0, while the weight of the outgoing connection keeps the original value. The existing connection is deactivated but retained in the genome where it might be reactivated by further mutations. There are two operators that can achieve this: one toggles the active flag of a connection and the other sets the flag to 1. Both are used with probability 0.01.

Evolving network topology on higher levels For evolving higher level topology, NEATfields introduces some new operators. At the level of a single field, one operator doubles the field size along one dimension (with a probability of 0.001) and another changes the size (for both dimensions independently) to a random value between its current size and the size of the largest field it is connected to

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

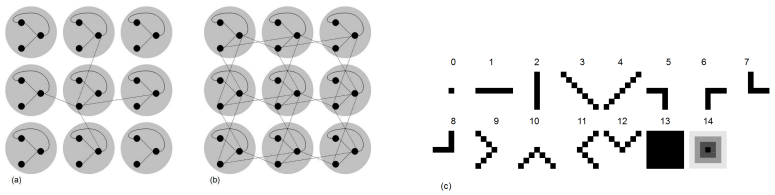


Fig. 3. (a) Lateral connections are established between a neuron in a field element and another neuron in each of the up to four neighbor field elements. There are less neighbors if it is at the border of the field. Lateral connections are only shown for the central element as dotted lines here for clarity. (b) All lateral connections constructed from a single gene. (c) Handdesigned patterns for local feature detectors. Black denotes a connection weight of 1.0, white a connection weight of -1.0, and the gray scales denote connection weights between 0.0 and 1.0.

(with a probability of 0.005). NEATfields networks can also have lateral connections between field elements of the same field. Lateral connections are like local connections: they are between two neurons in the NEAT network that describes the field elements. However, the connection from the source neuron does not go to a neuron in the same field element, but to the corresponding neurons in the up to four neighbor field elements instead (see fig. 3). The gene coding for a lateral connection specifies source and target neuron reference numbers just as genes coding for local connections do; it is also located in the same chromosome. However, it has a lateral flag set to 1, and is created by a lateral connect operator (with a probability of 0.02).

By default, corresponding connections in different field elements all have the same strength so they can be represented by one gene. The same is true for the global connections between field elements of two fields. For some tasks, it is useful to have field elements that are slightly different from each other. One way to realize this is to make a connection weight larger in a neighborhood of some center coordinates on the field. The connection weight is scaled according to $\exp(-\epsilon(\frac{distance}{field\,size})^2)$ (in our implementation, this is done separately for the x and y dimensions), where $\epsilon = 5.0$ (see figure 2 (b)). The center coordinates are specified in the genome. There is a mutation operator that (at a probability of 0.03) sets these properties for a single connection gene.

At the level of the complete NEATfields network, there is an operator that inserts global connections (with a probability of 0.01) and an operator that inserts a new field into an existing global connection (with a probability of 0.0005). The size of the new field is set randomly to some value between 1 and the larger of the sizes of the two fields between which it is inserted. This is done independently for both dimensions. An existing field can also be duplicated, where all elements of the new field receive new reference numbers. The new field can either be inserted parallel to the old one (in this case, the outgoing connection weights will be halved to prevent disruption of any existing function)

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

or in series with the old one (in this case, every neuron in every field element in the new field gets input from the corresponding neuron in the corresponding field element in the old field, while the output from the new field goes to where the output from the old field went previously). The serial duplication operator is also applied on input fields, in which case an internal field is created that contains one neuron for every input in the input field. Both mutations occur with probability 0.0005.

Building local feature detectors NEATfields also has another kind of global connections that serve to detect local patterns by connecting to a number of adjacent field elements in another field. For example, if the input field contains camera data, they can detect lines or edges in particular orientations. These global connections have an additional number on the connection gene that refers to one of at most 16 prespecified connection weight patterns. The patterns have been designed by hand with the aim of providing useful building blocks for local feature detection (see fig. 3 (c)). A value from the pattern is multiplied by the original connection weight specified in the gene to get the weight of the connection from the field element at the respective position. The patterns have a size of 7×7. The central element of the pattern corresponds to the weight modification of the connection between exactly corresponding field elements. The feature detectors do not have to use an entire pattern. Instead, the maximal offsets between the field element positions in the source and target fields can be between 0 and 3 separately for the x and y dimensions, and are also specified on the genome. There is a mutation operator (used with a probability of 0.02) that mutates the pattern reference and maximal offsets of an already existing connection. Here we use only patterns 1 to 4 from figure 3 (c) with equal probabilities, and the maximal offsets in x and y direction are 0 or 1 with equal probabilities. Comparisons to other parameter settings and a number of alternative approaches can be found in earlier work [7].

2.2 Selection methods

We use speciation selection based on genetic similarity as described previously [7] for the spatial accuracy task. For the other three tasks, we use a hybrid selection method that combines fitness based tournament selection and a recent approach known as "novelty search". Half of the population is chosen based on fitness, and the other half is chosen based on novelty. Each half has its own elite of size 10 that is copied to the next generation unchanged. This method has also be described elsewhere [6]. The behavioral novelty measures used are described in the sections that describe the respective task to be solved. In all cases, a population of 1000 individuals is used.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

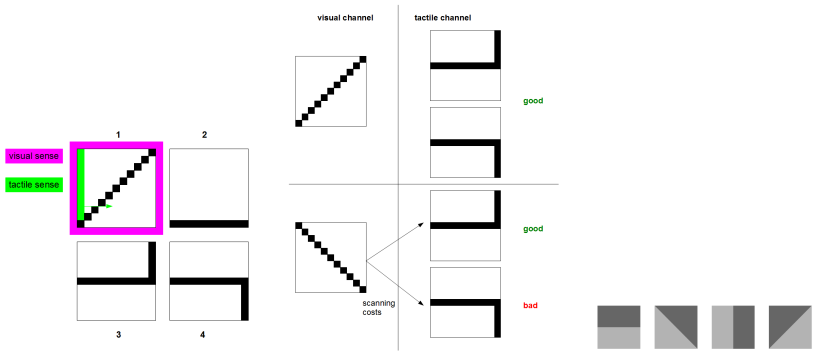


Fig. 4. Input patterns for multisensory integration tasks. (a) The Scan/Lift task. There are four episodes in this task with four different images. The visual sense sees the whole image after a lift action was executed, whereas the tactile sense can scan the image column-wise. (b) The Conditional Scan task. There are four episodes with different combinations of visual and tactile input. Tactile scanning is only required in the lower two episodes to disambiguate the input. (c) The four classes of grayscale images presented to neural networks in the third task.

3 Tasks

3.1 The "Scan/Lift" task

This classification task can be optimally solved by integrating information from different sensory channels. The network sees a different image of size 11×11 in each of the four episodes (see fig. 4a). These patterns have to be assigned to two classes depending on whether the line goes up or down on the right side of the image. The neural network has four outputs: one classification output for each class, and two outputs for active perception. Every 10 time steps, the output with the highest value is considered to be an action. If it is a classification action, the episode terminates, and the reward is determined. The active perception outputs control one of the two sensory modalities each. The first controls the "visual" modality. If it is active, a "lift" command is executed, which means the whole pattern becomes visible in the visual input field. At the beginning of an episode, the pattern is not visible, so all visual inputs are 0.0. The second active perception input controls the "tactile" input, which can look at the same image in a different way. If it is active, a "scan" command is executed, which means that the scanning apparatus (which provides a single column of the image as input) moves one column to the right. At the beginning of an episode, it shows the leftmost column. If it has already been moved to the rightmost column, it will not move further.

The neural network gets 35 fitness points for every correct classification. In addition, it gets $15 - 5a_L - a_S$ fitness points in every episode, where a_L denotes whether the lift action was invoked (0 or 1), and a_S how often the scan

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

action was invoked. So lifting is more expensive than scanning a row, but it is cheaper than scanning the whole image. This means that a network can get the optimal fitness if it uses scanning once for images 1 and 2 (where the line begins in the bottom of the image), but lifting for images 3 and 4 (where the line begins in the middle of the image). In that case, it gets a total fitness of 188 points. The behavioral distance between individuals is calculated as follows: If the corresponding commands of the two individuals at time step t are unequal, $\frac{10}{t+1}$ points are added to the distance. Times where only one of the individuals is still alive are disregarded. So this measure assigns the greatest weights to commands given at the beginning.

3.2 The "Conditional Scan" task

This task (see fig. 4b) is like the previous one but now the "visual" and "tactile" channel both have a size of 11×11, and they are provided with different input images. The visual channel is always active, but the tactile channel is active only after a tactile scan is performed. Evolution starts with a network that has two completely separate partitions, one consisting of the visual input field, an internal field, and the output field, and the other partition consisting of the tactile channel, another hidden field, and another output field, the state of which is ignored. Evolution can subsequently add more fields or connect them. There are three possible actions for the network: scan, assign to the "good" class, or assign to the "bad" class. "Scan" makes the whole image at appear at once on the tactile channel. The neural network gets 10 fitness points for correct classification of a "bad" pattern, 5 points for correct classification of a "good" pattern, and 1 point if the scan action was not used in an episode. So the highest obtainable fitness is 27. Behavioral distance is calculated as for the previous task.

3.3 A multisensory classification task

In this task the networks have a visual input channel of size 11×11. 32 patterns of 4 different classes are displayed in separate episodes (see fig. 4c). However, the patterns are very noisy. The pixels in the area with the brighter color are generated from a normal distribution with mean 0.1 and variance 0.3, while those in the darker area are generated from a normal distribution with mean -0.1 and variance 0.3. The network has four classification outputs and an output to generate a scan command. Its tactile sense is of size 11×1 and looks at a single row of another image of the same class. Initially, the scanner is at position $x = -1$, so no input is perceived. Scanning does not incur any costs. There is a control condition were all rows are empty (i.e., the input is 0.0). The neural network gets 1.0 fitness points for every episode in which it classifies the image correctly. If the classification is wrong, the network can get up to 0.5 fitness points according to $f = 0.5 - 0.25(o_a - o_c)$, where $o_a$is the highest output activation, and $o_c$is the activation of the correct output. Behavioral distance is calculated as for the other tasks.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

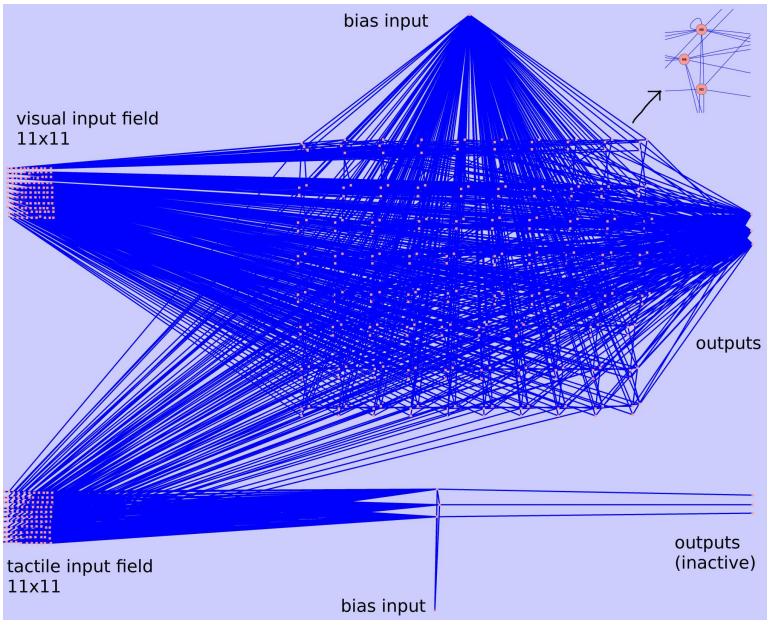


Fig. 5. A neural network solving the "Conditional Scan" task. In the upper left corner, a field element of the large internal field is shown magnified.

3.4 Increasing spatial accuracy by sensorimotor integration

In this task 64 randomly generated images each containing a spatial cue are presented to a network in separate episodes. The input value is 1.0 at the position of the cue, and 0.0 elsewhere. The network can perceive the images through a visual and a tactile channel, each of size11×11. The positions of the cues on the input fields randomly deviate from the true position of the cue by up to ± 2 in both dimensions. For both input fields there is each an internal field and an output field. These two pathways are completely separated in the common ancestor. However, the network can gain higher fitness if it has evolved connections to average over the positions of the cues in the two input fields. The output fields are also of size 11×11. In each episode, the network gets a reward of $2 - (d(c, o_1)^2 + d(c, o_2)^2)/s^2$, where d is Euclidean distance, c is the true position of the cue, and the o_i are the positions on the two output fields with the highest activation. Behavioral distance between two individuals is defined as the sum over all episodes of the squared distances between the positions with highest activation on the corresponding output fields.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

4 Results

20 runs of each of the four tasks are performed. The "Scan/Lift" task is solved perfectly in 75% of the runs using 349883 evaluations on average. In 15% of the runs, the classification is done correctly by the winner, but the "lift" action is used in all episodes, which results in a fitness of 180 points only. In 10% of the runs, the fitness is between 180 and the maximum 188 points. The "Conditional Scan" task is solved perfectly in 85% of the runs using 329705 evaluations on average. The other 15% runs converge on a solution that never uses the "scan" action and always classifies the ambiguous visual pattern as "bad", which results in a fitness of 24 points. One of the networks solving the "Conditional Scan" task perfectly can be seen in fig. 5. The internal field that is connected to the visual input has grown to a size of 10×8 by evolution, while the internal field connected to the tactile input has not grown. Instead, evolution has connected the first internal field to the tactile input as well. It can be seen that evolution uses one recurrent connection within each field element. Further recurrence is achieved through mutual lateral connections between some neurons in different field elements. It should be noted that solutions from other runs look very much different. In particular, some solutions for these tasks use several internal fields.

In the multisensory classification task, 31.55 fitness points are reached on average, while in its unisensory control, 30.74 fitness points are reached on average. This difference is slight but significant ($p < 0.001$, t-test). Looking at the 9 evolved solutions from the multisensory condition that solved the task perfectly, a number of different strategies can be found. One solution does not use tactile scans at all. Therefore, it still works perfectly when it is evaluated under control conditions. Seven solutions use tactile scans in between 8 and 28 of the 32 episodes. When they are evaluated under control conditions, they never classify the patterns that they normally used the scanner on, but performs scans until their lifetime is over. One solution uses the scanner in 17 episodes. Under control conditions, it stills classifies correctly in 10 of these episodes.

In the spatial accuracy task, 125.1 fitness points are reached on average, while in its unisensory control, 123.4 fitness points are reached on average. This is a significant difference ($p < 10^{-7}$, t-test). When having a closer look at the evolved network with the highest fitness, we found that it displayed a kind of McGurk effect. This effect originally refers to the perception of an intermediate phoneme when a subject simultaneously sees a video of a person producing one phoneme and hears a recording of that person producing another phoneme. It demonstrates that speech perception is based on multimodal information [1]. Our network was examined under two conditions. The first was to provide it with a new set of 100 randomly chosen locations and record the distances to the cues and the distances to another set of randomly chosen cues that were not displayed. The second condition was to test it with the same 100 locations again, but this time the tactile cues were from the other random set, so they were different from the visual cues. The distance to both cues was recorded. The network displayed a kind of McGurk effect because the distances to the cues in the second condition both were significantly larger than the distance to the cue

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

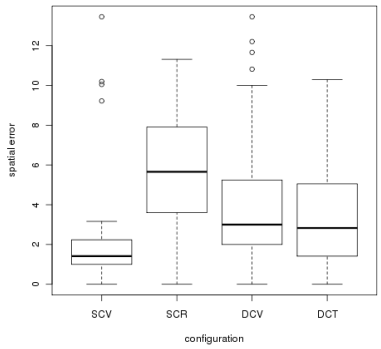


Fig. 6. Errors in the spatial accuracy task for four configurations: SCV - same cue condition / visual cue; SCR - same cue condition / suppressed cue; DCV - different cue condition / visual cue; DCT - different cue condition / tactile cue. All differences, except between DCV and DCT, are significant.

in the first condition, and were both significantly smaller than the distance to the non-visible cue in the first condition (see fig. 6) That means that the response was intermediate between both cues. The network is shown in fig. 7.

5 Discussion

We have shown that multisensory integration can evolve for cost-optimal use of sensors or increased classification or spatial accuracy. Importantly, this has been done not just using a few sensors, but using complete images. Further research will be on using real world input data. Besides, tasks with more emphasis on the temporal dimension on multisensory integration could be studied. If learning mechanisms are added to neuroevolution methods, the interaction of evolution and learning in multisensory integration can also be considered. In general, evolved neural networks that can process high-dimensional input in non-trivial ways are not just relevant for the goal of automatic construction of controllers, but also provide a useful tool for neuroscience research [10].

Acknowledgments Benjamin Inden gratefully acknowledges the financial support from Honda Research Institute Europe.

References

1. Spence, C.: Multisensory integration, attention and perception. In: Signals and Perception — The Fundamentals of Human Sensation. palgrave macmillan (2002) 345–354

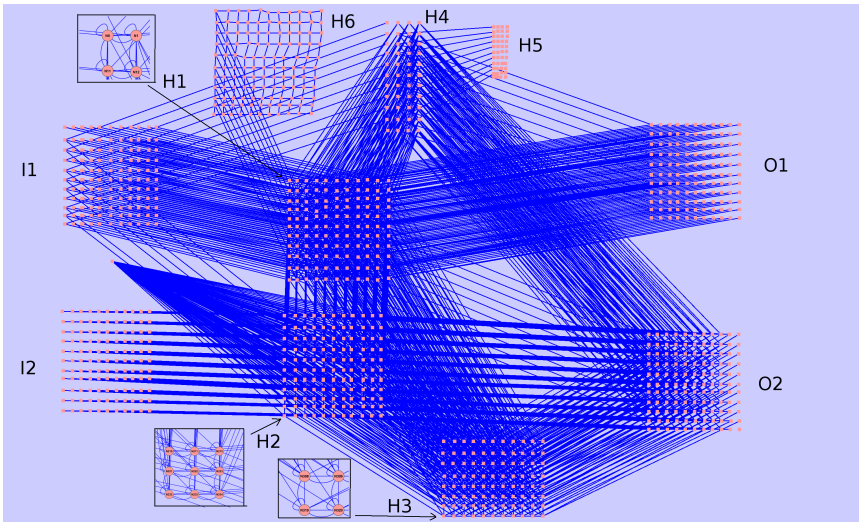


Fig. 7. An evolved network for the spatial accuracy task. "I" denotes an input field, "H" a hidden/Internal field, and "O" an output fields. The fine structure of three internal fields is shown in magnification. Field H4 only produces constant output, while fields H5 and H6 do not produce outputs because the respective connections have been deactivated. Multisensory integration is achieved by mutual connections between H1 and H2. Furthermore, H3 processes inputs from both sensor fields, but only contributes output for O2.

2. Stein, B.E., Stanford, T.R.: Multisensory integration: current issues from the perspective of the single neuron. Nature Rewievs Neuroscience **9** (2008) 255–266
3. Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning. Evolutionary Intelligence **1** (2008) 47–62
4. Nolfi, S., Floreano, D.: Evolutionary Robotics — The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press (2000)
5. Inden, B., Jin, Y., Haschke, R., Ritter, H.: Neatfields: Evolution of neural fields. In: Proceedings of the Conference on Genetic and Evolutionary Computation. (2010)
6. Inden, B., Jin, Y., Haschke, R., Ritter, H.: How evolved neural fields can exploit inherent regularity in multilegged robot locomotion tasks. In: Third World Congres on Nature and Biologically Inspired Computation. (2011)
7. Inden, B., Jin, Y., Haschke, R., Ritter, H.: Evolving neural fields for problems with large input and output spaces. submitted (2011)
8. Westermann, G.: A model of perceptual change by domain integration. In: Proceedings of the 23rd Annual Conference of the Cognitive Science Society. (2001)
9. Stanley, K., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10** (2002) 99–127
10. Ruppin, E.: Evolutionary autonomous agents: A neuroscience perspective. Nature Reviews Neuroscience **3** (2002) 132–141

Handling Diversity in Estimation of Distribution Algorithms

[1]S. Ivvan Valdez, [1]Eduardo Sánchez-Soto, [1]Eduardo Ortíz, [2]Arturo Hernández, and [2]Salvador Botello

[1]University of the Papaloapan, Computer Sciences Departement
Av Ferrocarril s/n, Col. Ciudad Universitaria, 68400 Loma Bonita, Oax., MEXICO
svaldez,esanchez,eortiz@unpa.edu.mx,
[2]Centre for Research in Mathematics, CIMAT A.C. Computer Sciences Département
C. Jalisco s/n, Mineral de Valenciana, A.P.36000, Guanajuato, Gto., MEXICO
artha,botello@cimat.mx,
http://www.cimat.mx/

Abstract. Estimation of Distribution Algorithms were derived from probabilistic modeling of Genetic Algorithms, the crossover and mutation operators were replaced by estimating and sampling from a probability distribution. In this work, we argue that the estimation and sampling steps in EDAs perform a similar role than the crossover operator in Genetic Algorithms, but EDAs lack (in general) of a diversity preservation mechanism such as mutation in Genetic Algorithms. In order to enhance the EDA framework we propose a novel mechanism called: the Synthetic Mutation. By using graphical comparisons, and hypothesis tests, we show that the Synthetic Mutation procedure consistently improves the EDA performance.

1 Introduction

Estimation of distribution algorithms (EDAs) is a core area of research in evolutionary computation (EC). EDAs were derived from probabilistic modeling of genetic algorithms (GAs). They replace the crossover and mutation operators by estimating and sampling from a probability distribution. The main EDA framework steps are shown in the left side of Table 1. In Step 1, the distribution $Q^0(x)$ usually is initialized as an uniform distribution, then the population X^t is simulated from $Q^t(x)$ in Step 2. In Step 3, the population is evaluated, and in Step 4, called elitism, the best known solution is stored. When the search process finishes, such best solution is returned as the optimum approximation. Step 5 verifies the stopping criteria which usually are related to: the number of function evaluations, the number of generations, and/or the population variance. The selection, in Step 6, returns a subset of the fittest individuals (the so called *selected set*), and finally, Step 7 estimates a new probabilistic model.

The distribution $Q^t(x)$ is named as *the search distribution.* Usually, the search distribution is a parametric distribution which intends to approximate the underlying distribution of the selected set. Notice three important issues:

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

- The selected set S^t is a subset of X^t, and the selected individuals in S^t usually share characteristics (have near values of the decision variables), thus the variance of S^t usually is less than the variance of X^t.
- The search distribution $Q^t(x)$ usually is proposed as a single-peak (unimodal) distribution, such as a Gaussian model, then it promotes that the new individuals being collapsed in a single region.
- The search distribution $Q^t(x)$ intends to represent as better as possible the characteristics in the selected set, but it is remarkable that: the better $Q^t(x)$ fits to the selected set, the smaller is the probability of sampling individuals different than those we had already known.

In consequence, even if the current population have an adequate variance for performing the search, it will be lost in the selection and estimation steps. Also, consider the case when the search distribution $Q^t(x)$ accurately represents the selected set, if the population gets trapped in a local minimum, there is not a way to get out of it, because of the lack of an operator that inserts diversity (new characteristics) into the population. As a result of the mentioned issues, EDAs suffer of premature convergence.

Researchers have noticed this EDAs deficiency since the very firsts approaches. For instance, in the population based incremental learning (PBIL)[1], which uses a univariate model, it is proposed to apply a mutation operator to the probability distribution, in order to increase the exploration capacity. Some drawbacks about this approach are: the lost of generality of the proposed method, since it is proposed just for the univariate model of the PBIL; the explicit probabilistic model is altered, which implies it is not possible to have an adequate variance control; and the behavior of the algorithm becomes unclear. A recent approach [5,2] introduces a variance scaling for Gaussian models, it computes a measure to determine whether the scaling must be applied, the scaling basically is performed by modifying the covariance matrix. Nevertheless these recent studies [5,2,3] have shown its capacity to outperform standard EDAs results, they are not general approaches. A remark about this last approach is that it maintains the variance control, and the explicit probabilistic model is well defined during the whole search process, as a consequence the algorithm has a predictable behavior.

Desirable features for the EDAs' diversity enhancement method in order to increase their exploratory capacity (our proposal fulfills these criteria):

- It must be general enough to work in real and discrete domains and use different search distributions.
- It must not affect the probabilistic model building to preserve the direct relationship between the population and the search distribution.
- It must be robust. The expected performance of the whole search process should not be negatively affected by a few outliers or undesired input inserted by the method.
- It must be a simple manner to regulate the diversity (increase or decrease exploration).

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

The outline of the article is as follows: the second section introduces the synthetic mutation concept, the third section presents an example of application, the fourth section contrasts the proposal with a well known standard EDA. Finally, the fifth section presents the main conclusions and future work.

2 The synthetic mutation

As mentioned, the standard EDA intends to approximate the optimum by estimating and sampling a probability distribution. One of the most important issues in EDAs is the so called: *selection distribution*, $P_s^t(X)$. The selection distribution is the underlying distribution of the selected set when an infinite sized population is considered. It is proved [14,9] that the EDA converges to the optimum when the exact selection distribution is used as search distribution. Unfortunately, the selection distribution can not be used for this purpose, because it requires to know the objective function value in the whole search space (recall that it uses an infinite sized population). Thus, researchers have proposed to approximate the selection distribution by the search distribution. As in practice EDAs work with a finite sized population, the search distribution usually is a parametric distribution which fits the selected set. Consequently, we expect that the search distribution codifies or represents as well as possible the characteristics in the selected set. If so, the optimum will be reached if the current selected set represents the region where the optimum is confined, otherwise it is quite possible that the EDA gets trapped in a local minimum, resulting in the premature convergence problem.

As can be noticed, the estimation step in EDAs performs a similar role than crossover in genetic algorithms (GAs), both are operators that preserves the characteristics of the current best known individuals (selected set). In GAs the premature convergence problem is (usually) tackled by increasing the mutation probability in order to insert new characteristics (diversity) into the population. On the other hand, EDAs do not have a diversity-preservation or insertion mechanism, if the estimation step fails to locate the optimal region, there is not a way of inserting diversity into the population.

This work is based on emulating the mutation effect in estimation of distribution algorithms, by obtaining a probabilistic model for the search distribution which emulates both: crossover and mutation effects. In order to introduce the synthetic mutation let us recall how GA performs, and contrasts it with the estimation of distribution algorithm in Table 1. With the intention of clarifying the comparison we obviate the step of probabilistic model initialization. So, consider an initial population (given by an initial probabilistic model, as well as the genetic algorithm does). Steps 1 to 5 can be applied without distinction for both algorithms and in Step 6, both algorithms contemplate to preserve characteristics of the best individuals, the GA by applying the crossover operator on S^t, and the EDA by approximating the underlying distribution of S^t. In spite of the fact that both algorithms perform almost the same kind of operations, the GA also carries out the mutation operator, which introduces new characteristics

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Table 1. EDA (left) vs GA (right) comparison

0 $t \leftarrow 0$	0 $t \leftarrow 0$
1 Propose an initial population X^t	1 Propose an initial population X^t
2 Evaluate X^t, $F^t \leftarrow f(X^t)$	2 Evaluate X^t, $F^t \leftarrow f(X^t)$
3 Store the elite $x^t_{Best} \leftarrow f(X^t)$	3 Store the elite $x^t_{Best} \leftarrow f(X^t)$
4 If the stopping criteria is reached, **return** x^t_{Best}	4 If the stopping criteria is reached, **return** x^t_{Best}
5 Select a subset of the best individuals $S^t \leftarrow selection(X^t)$	5 Select a subset of the best individuals $S^t \leftarrow selection(X^t)$
6 Estimate $Q^{t+1}(x)$ by using S^t and sample it to obtain X^{t+1}	6 Apply the crossover on S^t to obtain X^{t+1}
7	7 **Mutate** $\hat{X}^{t+1} \leftarrow X^{t+1}$
8 $t \leftarrow t+1$	8 $t \leftarrow t+1$
9 Go to 2	9 Go to 2

in the population, while EDA does not consider any variational step to insert diversity into the population, as can be noticed in step 7.

The main goal of this approach is to build a probabilistic model for the search distribution which represents the underlying distribution of $\hat{X}^t$ (the mutated set). EDAs intend to approximate the selection distribution $P^t_S(x)$ of the selected set S^t. Let us assume that the selection distribution $P^t_S(x)$ is accurately computed, then the search distribution equals the selection distribution, $Q^t(x) = P^t_S(x)$, thus, the new population will be simulated from the exact selection distribution, $X^{t+1} \sim P^t_S(x)$. Notice, that under this assumptions the new population X^{t+1} and the selected set S^t are identically distributed, with an underlying distribution $P^t_S(x)$. Now we integrate a new assumption, as mentioned, we can argue that the estimation and sampling steps in EDAs performs a similar role than the crossover step in GAs. Thus, X^{t+1} is the new population before mutation. In addition, considering the assumption that S^t and X^{t+1} are identically distributed, a mutated new population $\hat{X}^{t+1}$ will be identically distributed than a mutated $\hat{S}^t$. Thus, the underlying distribution of $\hat{S}^t$ is the desired distribution $\hat{P}^t_S(x)$ equipped with the diversity-preservation component. Finally, as well as we approximate the search distribution $Q^t(x)$ to $P^t_S(x)$ by using the selected set, we can approximate a search distribution $\hat{Q}^t(x)$ which approximates the mutated selection distribution $\hat{P}^t_S(x)$, by using the ***mutated*** *selected set* $\hat{S}^t$. The EDA equipped with synthetic mutation is independent of the search distribution model and the kind of domain (discrete or continuous). Hence, this general framework can be adopted for continuous and discrete spaces, with univariate as well as multivariate search distribution models. For instance, marginal univariate models for discrete domains such as the Univariate Marginal distribution algorithm (UMDA) [10], multivariate discrete models[11] such as the Bayesian Optimization Algorithm (BOA) [12], univariate continuous models such as the Gaussian Univariate Marginal Distribution Algorithm (UMGA$_G$) [8] and multivariate continuous models such as the Estimation of Multivariate Normal Algorithm (EMNA) [8]. To apply the synthetic mutation the only change is to mutate the selected set before estimating the parameters of the search dis-

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

tribution, in order to avoid a wrong (non desired) bias, a re-sampling procedure could be applied as explained in the next subsection.

Table 2. EDA with the Synthetic Mutation

Step 0	$t \leftarrow 0$
1	Initialize a probability model $Q^0(x)$.
2	Simulate a Population X^t, $X^t \sim Q^t(x)$.
3	Evaluate the objective function $F^t \leftarrow f(X^t)$.
4	Store the elite individual $x^t_{Best} \leftarrow f(X^t)$.
5	If the stopping criteria is reached, return x^t_{Best}
6	Select a subset of the best individuals $S^t \leftarrow selection(X^t)$
7	**Mutate** n **times** S^t to obtain $\hat{S}^t$, of size $\lvert\hat{S}^t\rvert = n\lvert S^t\rvert$
8	Estimate a probability distribution $Q^{t+1}(x)$, by using $\hat{S}^t$.
9	$t \leftarrow t + 1$
10	Go to 2.

2.1 Synthetic mutation with resampling

Mutating the selected set inserts diversity into the population, but at the same time, it could induce a wrong bias. For example, consider the case when most of the population is mutated, the valuable information in the selected set will be lost. Even with a low mutation probability this case is possible. Also, recall that the purpose is to estimate the underlying distribution of the mutated selected set $\hat{S}^t$, it worths noticing that we can obtain as many samples as we want of $\hat{S}^t$, just by applying the mutation step several times, so we can obtain a set $\hat{S}^t$, which number of elements $|\hat{S}^t|$ is greater than the number of elements of the original selected set $|S^t|$. In this way we can obtain as many samples as desired, to estimate more accurately the underlying distribution of $\hat{S}^t$ than simply using a single mutated set. The EDA modified with the synthetic mutation with re-sampling is shown in Table 2. The next section applies the algorithm in the well known UMDA$_G$ [8].

3 UMDA$_G$ with synthetic mutation

This section presents a modified EDA: the UMDA$_G$ modified with the synthetic mutation and contrasted with the original implementation. The main purpose of this section is not to present a new estimation of distribution algorithm, but to show how an EDA can be improved by using the synthetic mutation. The UMDA$_G$ and the modified UMDA$_G$ are shown in Table 3

4 Experiments

This section presents a set of experiments for contrasting the standard UMDA$_G$ with UMDA$_G$ with synthetic mutation. As it is not possible, in a single article, to

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Table 3. $UMDA_G$ with the Synthetic Mutation

step 0	
1	Simulate a uniform-random initial population X^t
2	Evaluate the objective function $F^t \leftarrow f(X^t)$.
3	Store the elite individual $x^t_{Best} \leftarrow f(X^t)$.
4	If the stopping criterion is reach, return x^t_{Best}
5	Select a subset of the best individuals $S^t \leftarrow selection(X^t)$
6	**Mutate** n **times** S^t to obtain $\hat{S}^t$, of size $\lvert\hat{S}^t\rvert = n\lvert S^t\rvert$
7	Estimate an univariate Gaussian distribution $Q^{t+1}(x)$, by using $\hat{S}^t$.
8	$t \leftarrow t+1$
9	Go to 2.
^{Step6}	For experiments $n = 4$.

present many comparisons, we decide to use the well known UMDA$_G$. A remarkable reason of using UMDA$_G$ is that it is well known [8] that it has problems to approximate the optimum even with convex functions that present high variable correlation, such as the Rosenbrock function. UMDA$_G$ has a poor performance that is explained because of premature convergence issues, the variance is reduced too fast and the population is condensed on a non-optimal region. Thus, we are not proposing a new UMDA$_G$ version, but a way to improve UMDA$_G$ performance as well as to insert diversity to any EDA which operates according the framework in the left side of Table 1. Remember that we are only looking for the evidence of the UMDA$_G$ improvement with the synthetic mutation, so, we do not tune any other parameter such as the number of function evaluations (stopping criterion), or the population sizes, we intend to show that in spite of the used parameters the synthetic mutation can improve the EDA performance. In this case we use the *uniform mutation* to perform the experiments, notice that it is not a particular way of mutating the population according the framework in Table 2. The uniform mutation is used for the sake of simpicity and in order to show the advantages of the synthetic mutation, even with a simple mutation operator. Let us define the $i-th$ candidate solution in the selected set (individual) as $s_i \in S^t$, and let us define the value in the $j-th$ position of the vector s_i (a variable instance) as $s_{i,j}$, p_m =mutation probability, $minx_j, maxx_j$ =minimum and maximum limits for the $j-th$ variable. $rand(a,b)$ generates a uniform-real random number between a and b and $irand(m)$ generates a uniform-integer random number between 0 and m. According to this nomenclature, the uniform mutation procedure is as follows:

1. **for** each s_i in S^t **do**
2. **if** $rand(0,1) < p_m$ **then** $s_{i,j} = rand(minx_j, maxx_j)$
3. **endfor**

4.1 Experiment 1. Searching for a parameter tunning policy

By using empirical data, this experiment graphically shows that it exists a correlation between the mutation probability, the number of variables and the

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

population size. Using the well known Rosenbrock function, the UMDA$_G$ with synthetic mutation and 30 runs with 300000 function evaluations per run, we test all the possible combinations, in order to determine an adequate probability mutation parameter over the following:

Set of mutation probabilities $P_m = \{0.2, 0.1, 0.05, 0.02, 0.01, 0.005, 0.002, 0.001, 0.0005, 0.0002, 0.0001, 0.00005\}$.
Set of population sizes $S_p = \{100, 200, 400, 800, 1000, 2000, 4000, 5000\}$
Set of number of variables $N_{var}\{2, 4, 8, 10, 20, 40\}$

Proposal on mutation probability parameter tunning According to our experiments the mutation probability must be increased with the population size and must be decreased with the number of variables. Then, we propose the parameter $\theta = \frac{n_{pop}}{n_{var}}$, as the independent variable of a model for the mutation probability. Therefore, we have as many θ parameters as combinations of population sizes with number of variables. A linear adjustment (in the log scale) is performed, say: $log(pm) = \beta_0 + \beta_1 log(\theta)$, where pm is the mutation probability, and β_0, β_1 are coefficients which can be found by a least squares adjustment. The adjusted linear model (in log scale) is defined by Equation 1.

$$\log(pm) = -8.90479 + 0.4126 \log(\theta) \tag{1}$$

Notice that such model in log scale, is non-linear in the normal scale, hence, writing Equation 1 in the normal scale, we obtain $pm = e^{-8.90479}\theta^{0.4126} = 0.00013573760^{0.4126}$. Therefore, in the normal scale the mutation probability depends on $(n_{pop}/n_{var})^{0.4126}$. In the sake of simplicity we approximate $0.4126 \approx 0.5$, and looking for a general rule based on the obtained results we propose a new model given by

$$pm = k\sqrt{\theta}, \qquad for \quad \theta = \frac{n_{pop}}{n_{var}} \tag{2}$$

The constant k, in Equation 2, can be straightforward found by using a least squares approximation in Equation 3. Where θ_i are observations (if one has some runs with different probability mutations and population sizes and/or number of variables), for $i = 1, 2, 3, .., n_r$ and n_r is the number of training runs. Since we are training k for the best performance, n_r and the others parameters are referred to the *best* runs. For our experiments $k = 0.0001155$.

$$k = \frac{\sum_i^{n_r} \sqrt{\theta_i} pm_i}{\sum_i^{n_r} \theta_i} \tag{3}$$

Finally, to show that the proposed model fits the data, we present the adjustment in Figure 1. The continuous line represents the proposed model $pm = k\sqrt{\frac{n_{pop}}{n_{nvar}}}$, for k computed according Equation 3. The plot is in log scale.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Fig. 1. Population size vs mutation probability.

4.2 Experiment 2. Contrasting UMDA$_G$ versus UMDA$_G$ with synthetic mutation

By using a Boostrap non-parametric statistical tests [4], we show that there exist statistical evidence to say that the EDA equipped with the synthetic mutation outperforms the standard EDA. These experiments are conducted by using a set of different objective functions, populations sizes, and numbers of variables.
As has been discussed during the whole article until this point, the main purpose of this approach is not to propose a novel particular EDA, but to present a **general** framework in order to improve *any* EDA according to the framework in Table 2.
Thus, the following settings are used for the experiments:
We test population sizes of: $\{100, 200, 400, 800, 1000, 2000, 4000, 5000, 10000\}$, and 300000 function evaluations for all cases. The mutation probability was tunned by using Equation 2 and Equation 3. According to these equations the computed k value for all cases is $k = 0.0001155$.

4.3 Graphical comparison

The plot in Figure 3 contrast the standard UMDA$_G$ with the UMDA$_G$ with synthetic mutation, each $\boxtimes$ represents the mean of the best objective function found of 30 runs with the parameters in the $x - axis$ for the standard UMDA$_G$, the synthetic-mutation UMDA$_G$ counter part is represented with $\triangledown$ in the same graph. It can be observed that in most of the cases the UMDA$_G$ equipped with synthetic mutation clearly outperforms the standard UMDA$_G$. Figure 3 presents the comparison with the objective function ($y - axis$) in log scale.

The Figure 3 presents a qualitative comparison, on the other hand Table 4 presents a quantitative comparison: according to the Boostrap methodology [4] a non-parametric hypothesis test is applied to each set of 30 runs whith 20000 resamples (Boostrap parameter).

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

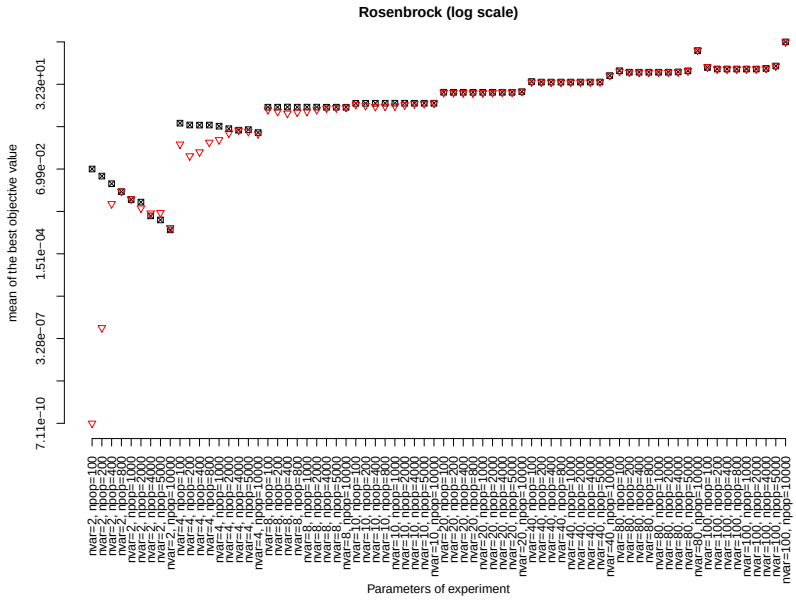


Fig. 2. Comparison of UMDA$_G$ ⊠ and UMDA$_G$ with synthetic mutation ▽, means of elite objective function values for 30 runs. The plot is in log scale in the $y-axis$. This is a minimization case. The lower the better objective function value.

Thus, with a significance of 5% we test if the mean value of the elite individual returned by the UMDA$_G$ with synthetic mutation is less than the returned by the standard UMDA$_G$, and vice versa. If there is statistical evidence to say that the UMDA$_G$ with **synthetic mutation** is the best we put a **SM**, otherwise we put a **St**. As can be noticed, most of the times there is statistical evidence to say that UMDA$_G$ equipped with synthetic mutation is the best. Other cases can be explained by the dimensionality of the problems, there are too many decision variables, thus more function evaluations are needed to get a better approximation to the optimum than the current. This possible explanation is supported by the Figure 3. It is well known (and reasonably) that, the higher is the number of decision variables the greater the population size must be, in order to get an adequate optimum approximation. Thus. for higher dimensions (80 and 100 variables), we require more function evaluations than for such problems with low dimensionality (less than 80).

Remark. It is reasonable to think that the greater the diversity is the slower the converge is, and the greater the population size is the slower the convergence is. Hence, the hypothesis test results for 40, 80 and 100 variables in Table 4 can be well explained because of convergence issues, the diversity is greater in the UMDA$_G$ with Synthetic Mutation than the standard UMDA$_G$, so the converge is slower. Hence, we can show that the last statement is true (or at least reasonably), by using more generations in order to achieve convergence in the

Table 4. The letters represent which algorithm has the best performance: SM=Synthetic Mutation, based UMDA$_G$, St= Standard UMDA$_G$, N=Neither of them. Top: 3e5 function evaluations, bottom: 15e5 function evaluations

Rosenbrock	n_{pop}	100	200	400	800	1000	2000	4000	5000	10000
2 var		SM	SM	SM	N	N	SM	N	St	N
4 var		SM	SM	SM	SM	SM	SM	N	SM	N
8 var		SM	SM	SM	SM	SM	SM	SM	SM	N
10 var		SM	SM	SM	SM	SM	SM	N	St	St
20 var		SM	SM	SM	SM	SM	SM	St	St	N
40 var		SM	SM	SM	SM	SM	SM	St	St	N
80 var		SM	N	SM	SM	SM	St	St	St	St
100 var		N	N	SM	SM	SM	St	St	St	N

Rosenbrock	n_{pop}	4000	5000	10000
40 var		SM	SM	St
80 var		SM	SM	St
100 var		SM	St	St

UMDA$_G$ with synthetic mutation. The bottom of Table 4 shows the experiments with $\{40, 80, 100\}$ variables, and {4000,5000,10000} when using 1500000 function evaluations, *as can be noticed, UMDA$_G$ with synthetic mutation outperforms the standard UMDA$_G$ for most of the cases*, and the cases the Standard UMDA$_G$ is not outperformed are such cases with the greatest population size.

4.4 Experiment 3. Diversity measure

In this section we contrast a diversity measure UMDA$_G$ with and without synthetic mutation. The diversity metric has been widely used mainly in multiobjective evolutionary optimization [6] but can be used to measure the diversity in the decision variable space. The diversity is measured according Equation 4.

$$S = \sqrt{\frac{1}{|X|}\sum_{i=1}^{|X|}(d_i - \bar{d})^2} \tag{4}$$

5 Conclusions

From the very firsts EDAs [1] to the most recent approaches [5,3] researchers have noticed the lack of an exploration operator in EDAs. This work proposes a general method to equip EDAs with a diversity-preservation operator. The approach is named: Synthetic Mutation. In this proposal we argue that synthetic mutation is equivalent to estimate the underlying distribution of a mutated population before knowing such mutated population. This is achieved by resembling that the new population must be identically distributed than the selected set, thus a mutated selected set must be identically distributed than the new population. Thus, the underlying distribution of the new mutated population is equal to the underlying distribution of the current selected set. According to

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

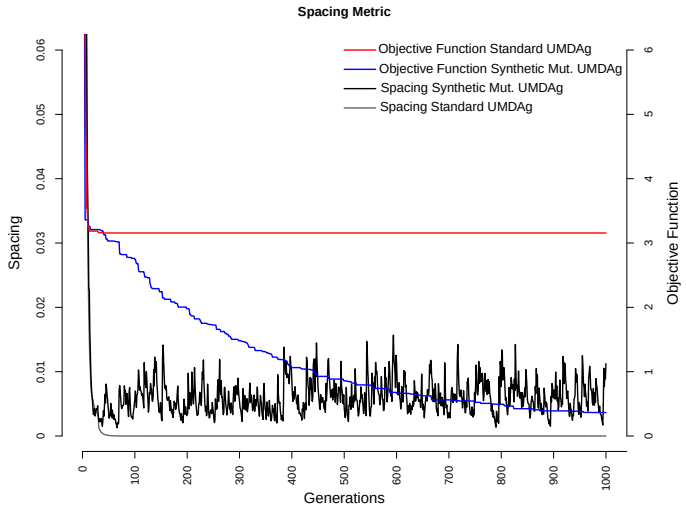


Fig. 3. Comparison of UMDA$_G$ and UMDA$_G$ with synthetic mutation. Spacing measure during 1000 generations, the very fisrts (5 generations) values are not shown (because a scale problem to observe with more detail the behavior during most of the generations). Also, the objective function is plotted (rigth-side scale).

this assumption, we propose to apply the mutation to the selected set. A new parameter is inserted with this approach: the mutation probability, this possible drawback is circumvented by using statistical information of sufficient runs to fit a mathematical model which relates the number of variables and the population size with the *mutation probability*. Thus, a general proposal to tune the *mutation probability* is presented, and once a constant k is found, by using the given formula the *mutation probability* parameter is obviated. $72 \times 30 = 2160$ experiments statistically validate the proposal by the support of hypothesis tests and graphical comparisons. Nevertheless the promissory results obtained, a more general validation is needed, hence, future work will contemplate the insertion of the synthetic mutation in different state of the art EDAs in discrete as well as continuous spaces, and with and without considering variable correlations. In addition, as it is shown in our experiments, the synthetic mutation can lead the algorithm to a slower convergence than the original implementation, thus, if the search ditribution is performing an adequate search the synthetic mutation must not be applied, in this vein a self-adapted mutation rate (mutation probability) is an important issue to study. Another issue, is to combine the synthetic mutation method with other state of the art EDA approaches such as the *empirical selection distribution* [13], which is a general method for improving the exploitation of the information in EDAs, thus, the combination of the exploration capacity of the synthetic mutation with an exploitation-of-the-information method, could derive to a new generation of enhanced EDAs.

References

1. S. Baluja. Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.
2. P. A. N. Bosman, J. Grahl, and F. Rothlauf. SDR: A Better Trigger for Adaptive Variance Scaling in Normal EDAs. In *GECCO '07: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 516–522. ACM, 2007.
3. P. A. N. Bosman and D. Thierens. Expanding from Discrete to Continuous Estimation of Distribution Algorithms: The IDEA. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 767–776, London, UK, 2000. Springer-Verlag.
4. B. Efron and R. J.Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall,Monographs on Statistics and Applied Probability, New York, 1993.
5. J. Grahl, P. A. Bosman, and F. Rothlauf. The correlation-triggered adaptive variance scaling IDEA. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 397–404, New York, NY, USA, 2006. ACM.
6. K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms.* John Wiley & Sons. USA, 2004
7. J. Grahl, P. A. N. Bosman, and S. Minner. Convergence Phases, Variance Trajectories, and Runtime Analysis of Continuos EDAs. In *GECCO '07: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 516–522. ACM, 2007.
8. P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, Norwell, MA, USA, 2001.
9. H. Mühlenbein. The Equation for Response to Selection and Its Use for Prediction. *Evolutionary Computation*, 5(3):303–346, 1997.
10. H. Mühlenbein and G. Paaβ. From Recombination of Genes to the Estimation of Distributions I. Binary Parameters. In *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 178–187, London, UK, 1996. Springer-Verlag.
11. M. Pelikan and H. Mühlenbein. The Bivariate Marginal Distribution algorithm. *Advances in Soft Computing Engineering Design and Manufacturing*, pages 521–535, 1999.
12. M. Pelikan, K. Sastry, and E. Cantú-Paz, editors. *Scalable Optimization via Probabilistic Modeling*, volume 33 of *Studies in Computational Intelligence*. Springer, 2006.
13. S. Ivvan Valdez, A. Hernández, and S. Botello Approximating the Search Distribution to the Selection Distribution in EDAs. In *GECCO '09: 11th annual conference on Genetic and evolutionary computation*, pages 461–468. ACM, 2009.
14. Q. Zhang and H. Mühlenbein. On the Convergence of a Class of Estimation of Distribution Algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):127–136, April 2004.

Reducing the Learning Time of Tetris in Evolution Strategies

Amine Boumaza

Univ. Lille Nord de France, F-59000 Lille, France,
ULCO, LISIC, F-62100 Calais, France
boumaza@lisic.univ-littoral.fr

Abstract. Designing artificial players for the game of Tetris is a challenging problem that many authors addressed using different methods. Very performing implementations using evolution strategies have also been proposed. However one drawback of using evolution strategies for this problem can be the cost of evaluations due to the stochastic nature of the fitness function. This paper describes the use of racing algorithms to reduce the amount of evaluations of the fitness function in order to reduce the learning time. Different experiments illustrate the benefits and the limitation of racing in evolution strategies for this problem. Among the benefits is designing artificial players at the level of the top ranked players at a third of the cost.

1 Introduction

Designing artificial game players has been, and still is, studied extensively in the artificial intelligence community. This is due to games being generally interesting problems that provide several challenges (difficult or large search spaces, important branching factors, randomness etc.) Many conferences organize special tracks every year where game playing algorithms are put in competition.

Among these games, Tetris [8] is a single player game where the goal is to place randomly falling pieces onto a 10×20 game board. Each completed horizontal line is cleared from the board and scores points to the player and the goal is to clear as much lines as possible before the board is filled.

Among the challenges in learning Tetris is the prohibitive size of the search space that approaches 10^{60}, and which can only be tackled using approximations. Interestingly enough one cannot play Tetris for ever, the game ends with probability one [7]. Furthermore, it has been shown that the problem of finding strategies to maximize the score in Tetris is NP-Complete [6].

Many authors proposed algorithms to design artificial Tetris player (see below), however to this day evolutionary algorithms outperform all other methods by far. And among these the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [10] and the noisy cross entropy[3] hold the record (several million lines on average).

In the present paper we will describe the use of racing procedures in the case of CMA-ES on the Tetris learning problem in order to reduce the learning time

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

which as it will be clear shortly, can be very problematic. We will begin with a description of the problem of learning Tetris strategies and review some of the existing work. We present the algorithm with the racing methods used, and then present some experiments. Finally we will discuss the results and give some conclusions.

1.1 Learning Tetris Strategies

In the literature, artificial Tetris players use evaluation functions that evaluate the game state by assigning numerical values. The agent will decide which action to take based on the values of this function.

When a piece is to be placed, the agent simulates and evaluates all the possible boards that would result from all the possible moves of the piece, and then chooses the move that leads to the best valued game board. Note that since the agent follows a greedy strategy, the evaluation function is the only component that enters into the decision making process. Hence, designing a Tetris player amounts to designing an evaluation function. Such a function should synthesize the game state giving high values to "good boards" and low values to "bad" ones. The evaluation function for the game of Tetris may be considered as mimicking the evaluation a human player makes when deciding where to drop the piece and which orientation to give to it. It should rank higher game states that increase the chance to clear lines in subsequent moves. For example, such boards should be low in height and should contain as less holes as possible. These properties (the height, the number of holes) are called *features* and an evaluation function is a *weighted linear combination of these features*.

Let f_i denotes a feature function, f_i maps a state (a board configuration) to a real value. The value of a state s is then defined as:

$$V(s) = \sum_{i=1}^{N} w_i f_i(s) \tag{1}$$

where N is the number of features and w_i are weights which, without loss of generality can sum to one.

Let P be the set of all pieces and D the set of all possible decisions. We note $d_i(p) \in D$ the i^{th} decision applied on piece $p \in P$. A decision is a rotation (4 possibilities) and a translation (10 possibilities[1]) for the current piece. Furthermore, we note γ the function that maps a pair $(s, d_i(p))$, a state s and a decision $d_i(p)$, to a new state of the game after taking decision d_i for the piece p. Given all possible decisions $d_i(p) \in D$ for a current piece p in state s, the player will choose the highest valued one:

$$\hat{d}(s,p) = \arg \max_{d_i(p) \in D} V(\gamma(s, d_i(p))).$$

The function $\hat{d}(s,p)$ is the player's decision function. As stated above the only component that enter into the decision process is the value V, in other words

[1] The number of translations may be different depending on the game width.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

the behavior of the player is conditioned by the value of the feature functions $f_{1...N}$. For a comprehensive review of the existing features the reader can see [19]. Learning tetris strategies amounts at (1) choosing a set of feature functions and (2) fixing their weights in eq. 1. In the present work we choose the set of eight features used in [19] and [5].

There are many papers that address the problem of learning Tetris strategies using different methods ranging from reinforcement learning to evolutionary computation. Dellacherie [8] fixed the weights by hand proposing good performing players. Different authors have used reinforcement learning some of which are : feature based value iteration [21], λ-policy iteration [2] and LS-λ-policy iteration [20], and approximate dynamic programming [9].

Other authors considered the problem differently and proposed to fix the feature weights using optimization techniques. The noisy cross entropy has been used by [18, 19]. Evolutionary algorithms have also been used with some success. For instance [16] proposed to use genetic programming. [4] also used an evolutionary algorithm and different combinations of value function. Finally [5] proposed to use CMA-ES to optimize the weights.

At this stage the best performing player are the one proposed by [19] and [5]. They both score on average around 35 millions lines.

In all these studies, authors agree on the challenges that the problem of learning Tetris strategies introduces. First of all, the score distribution of a given strategy on a series of games follows a long tailed distribution[2] which requires to evaluate (compute the fitness) the performance of the players on a series of games rather that just one. Secondly, as the artificial player learns during the run of an algorithm and thus get better, the games it plays last longer and lengthen the evaluation time. The number of games that is required for an accurate assessment of the performance is usually a parameter of the algorithm. It is fixed empirically to obtain an enough sample to compute statistics of the performance and in the same time keep the evaluation time reasonable[3].

Different methods have been proposed to reduce the learning time in Tetris. On the one hand one can train a player on small instances of the game, in this case the possibilities to place the pieces are reduced and the player looses rapidly. On the other hand, one can increase the frequency of certain (hard to place) pieces. Here again the player looses more quickly since these pieces will disturb the games board. All these methods do not take into account the number of games played in order to estimate the score of the player which is usually a fixed number. In what follows we will describe the use of racing procedures to dynamically adapt the number of required games to evaluate the players.

1.2 Racing for Stochastic Optimization

On stochastic objective functions, the fitness values of each individual follow a distribution $\mathcal{D}$ usually unknown. We have thus $f(x) \sim \mathcal{D}_x\left(\mu_x, \sigma_x^2\right)$, where μ_x, σ_x^2

[2] Some authors argue that this distribution is exponential [8]

[3] Many authors reported weeks length and some times months of learning time.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

are respectively the mean and the variance of the $\mathcal{D}_x$. On such problems, one cannot rely on a single evaluation of the offspring which is usually not a reliable measure, but needs to make several ones and compute statistics to estimate the true fitness.

In algorithms that rely only on the ranking of the population as it is the case for CMA-ES [10], ranking based on fitness values is not appropriate if the variability of the values is not taken into account. In such cases the ranking may not be correct and may lead to badly selected parents.

Different methods were proposed to overcome this problem. For example [11] proposes UH-CMA-ES which, among other things, increases the population variance when a change in the ranking of the population is detected. [17, 15] proposed methods to reduce the number evaluations applicable for certain types distribution $\mathcal{D}$. Another way to overcome this problem is to take into account confidence bounds around the estimated fitness value using racing methods [13, 1]. These methods adapts the number of evaluations dynamically until the algorithm reaches a reliable ranking in the population. [12] proposed to use such methods with CMA-ES and reported some success on different machine learning problems.

Basically these methods reevaluate only individuals where confidence intervals (CI) overlap. Reevaluation in this case is used to reduce the CI. Once enough individuals with non overlapping CI are found, the procedure stops. When the distribution $\mathcal{D}$ is unknown, the confidence intervals are computed using empirical methods using for example Hoeffding or Bernstein bounds which will be described below.

2 Learning Tetris with CMA-ES and Racing

We will begin this section by describing the CMA-ES algorithm and the racing method after which we describe their use in learning Tetris strategies. The covariance matrix adaptations evolution strategy [10] is an evolutionary algorithms that operates in continuous search spaces where the objective function f can be formulated as follows:

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} f(\mathbf{x}) \;\text{ with }\; f : \mathbb{R}^n \to \mathbb{R}$$

It can be seen as continuously updating a search point $m \in \mathbb{R}^n$ that is the centroid of a normally distributed population. The progress of the algorithm controls how the search distribution is updated to help the convergence to the optimum.

Alg. 1 describes the general procedure in CMA-ES. At each iteration, the algorithm samples λ points (the offspring) from the current distribution (line 5) : $\mathbf{x}_i^{t+1} \sim \mathbf{m}^t + \sigma^t \mathbf{z}_i(t)$ and $\mathbf{z}_i^t \sim \mathcal{N}(0, \mathbf{C}^t)$ $(i = 1 \cdots \lambda)$, where $\mathbf{C}^t$ is the covariance matrix and $\sigma > 0$ is a global step-size. The $\mu \leq \lambda$ best of the sampled points (the parents) are recombined by a weighted average (line 8) where $w_1 \geq \cdots \geq w_\mu > 0$ and $\sum_{i=0}^{\mu} w_i = 1$ to produce the new centroid $\mathbf{m}^{t+1}$.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Algorithm 1 Procedure $(\mu/\mu, \lambda)$-CMA-ES

Input $(\lambda, \mathbf{m}, \sigma, f, n)$

1: $\mu := \lfloor \lambda/2 \rfloor$, $\mathbf{m}^0 := \mathbf{m}$, $\sigma^0 := \sigma$, $\mathbf{C}^0 := \mathbf{I}$, $t := 1$, $\mathbf{p}_c^0 = \mathbf{p}_\sigma^0 = 0$
2: **repeat**
3: **for** $i := 1$ **to** λ **do**
4: $\mathbf{z}_i^t \sim \mathcal{N}_i\left(0, \mathbf{C}^t\right)$ /* Sample and evaluate offspring */
5: $\mathbf{x}_i := \mathbf{m}^t + \sigma^t \times \mathbf{z}_i^t$
6: **end for**
7: $\texttt{race}(\mathbf{x}_1 \dots \mathbf{x}_\lambda)$ /* Race the offspring */
8: $\mathbf{m}^{t+1} := \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}$ /* Recombine μ best offspring, $f(\mathbf{x}_{1:\lambda}) \leq \dots \leq f(\mathbf{x}_{\mu:\lambda})$ */
9: $\mathbf{p}_\sigma^{t+1} := (1 - c_\sigma)\, \mathbf{p}_\sigma^t + \sqrt{c_\sigma\,(2 - c_\sigma)\,\mu_{eff}}\, \left(\mathbf{C}^t\right)^{\frac{1}{2}} \frac{\mathbf{m}^{t+1} - \mathbf{m}^t}{\sigma^t}$ /* Update σ */
10: $\sigma^{t+1} := \sigma^t \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma^{t+1}\|}{\mathbb{E}_{[\|\mathcal{N}(0,\mathbf{I})\|]}} - 1\right)\right)$
11: $\mathbf{p}_c^{t+1} := (1 - c_c)\mathbf{p}_c^t + \sqrt{c_c\,(2 - c_c)\,\mu_{co}}\frac{\mathbf{m}^{t+1} - \mathbf{m}^t}{\sigma^t}$ /* Update $\mathbf{C}$ */
12: $\mathbf{C}^{t+1} := (1 - c_{co})\, \mathbf{C}^t + \frac{c_{co}}{\mu_{co}} \mathbf{p}_c^{t+1} \left(\mathbf{p}_c^{t+1}\right)^T + c_{co}\left(1 - \frac{1}{\mu_{co}}\right) \sum_{i=1}^{\mu} w_i \left(\mathbf{z}_{i:\lambda}^t\right) \left(\mathbf{z}_{i:\lambda}^t\right)^T$
13: $t := t + 1$
14: **until** stopping_criterion
15: **return** $\mathbf{m}^t$

Adaptation of the search distribution in CMA-ES takes place in two steps, first updating the mutation step and then updating the covariance matrix. The step-size adaptation is performed using cumulative step-size adaptation [14], where the evolution path accumulates an exponentially fading path[4] of the mean m^t (line 9) where the backward time horizon is determined by c_σ^{-1}. The adaptation of the covariance matrix C^t takes into account the change of the mean (rank-1 update), and the successful variations in the last generation (rank-μ update) (respectively lines 11 and 12). An in depth discussion on setting the parameters c_σ, d_σ, c_c, μ_{eff}, μ_{co}, c_{co} and their default values can be found in [11].

2.1 The Racing Procedure

The racing procedure we used is inspired from [12]. At each iteration of alg. 1, the λ offspring undergo multiple evaluations until either : 1) μ outstanding individuals get out of the race, in which case we are sure (with probability $(1 - \delta)$) that they are distinct. Or 2) the number of evaluations reaches some upper limit r_{limit}. Furthermore, if the actual number of evaluations necessary to distinguish μ individuals is less than the limit r_{limit}, this one is reduced:

$$r_{limit} := \max(1/\alpha\, r_{limit}, 3) \tag{2}$$

with $\alpha > 1$, and 3 is the minimum number of evaluations. On the other hand if r_{limit} evaluations where not enough to distinguish μ individuals, then r_{limit} is increased:

$$r_{limit} := \min(\alpha\, r_{limit}, r_{max}) \tag{3}$$

[4] The comparison of the travelled path with a path under random selection is used to adjust the step-size.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

where r_{max} is a maximum number of evaluations. Initially $r_{limit} = 3$ and it is adapted at each iteration of alg. 1 using the above two rules.

After r re-evaluations, the empirical bounds $c_{i,r}$ around the mean fitness of each individual $\hat{X}_{i,r} = \frac{1}{r}\sum_{j=1}^{r} f^j(x_i)$ where $i = 1 \dots \lambda$, are computed with:

$$c^h_{i,r} = R\sqrt{\frac{\log(2n_b) - \log(\delta)}{2r}} \tag{4}$$

using the Hoeffding bound and

$$c^b_{i,r} = \hat{\sigma}_{i,r}\sqrt{2\frac{\log(3n_b) - \log(\delta)}{r}} + 3R\frac{\log(3n_b) - \log(\delta)}{r} \tag{5}$$

using Bernstein. Where $n_b \leq \lambda r_{limit}$ is the number of evaluations in the current race, $\hat{\sigma}^2_{i,r} = \frac{1}{r}\sum_{j=1}^{r}\left(f^j(x_i) - \hat{X}_{i,r}\right)^2$ is the standard deviation of the fitness for individual x_i, and $(1-\delta)$ is the level of confidence we fix. $R = |a - b|$ such that the fitness values of the offspring are almost surely between a and b. These two constant are problem dependent.

After each evaluation in the race the lower bounds $lb_{i,r}$ and the upper bounds $ub_{i,r}$ around the mean of each offspring are updated to the tightest values: $lb_{i,r} = max\left(lb_{i,r-1}, \hat{X}_{i,r} - c^{h/b}_{i,r}\right)$ and $ub_{i,r} = min\left(lb_{i,r-1}, \hat{X}_{i,r} + c^{h/b}_{i,r}\right)$

Beside the above empirical bounds, there exists for the game of Tetris estimated bounds on the mean score of the player [19] which we also used in the racing procedure. We have:

$$|X - \hat{X}|/\hat{X} \leq 2/\sqrt{n} \tag{6}$$

with probability 0.95, where $\hat{X}$ is the average score of the player on n games and X is the expected score. In the remainder we will refer to this bound as the Tetris bound.

3 Experiments

In the following, we present few experiments we conducted in different settings to compare the effect of the racing procedures described above. In all experiments the initial search point x_0 was drawn randomly with $\|x_0\| = 1$ and the initial step-size $\sigma_0 = 0.5$. We let the algorithm run for 25 iterations (improvements for larger values were not noticeable), we set $r_{max} = 100$ and $\delta = 0.05$. There are eight feature functions in the evaluation function therefore the dimension of the search space is $n = 8$. CMA-ES parameters c_σ, d_σ, c_c, μ_{eff}, μ_{co} and c_{co} were set to their default values as presented in [11].

In order to reduce the learning time, we adopt the same scheme as in [5] and evaluate the offspring on harder games, in which "S" and "Z" appear four times more frequently than in the standard game. Experiments showed that learning on this setting does not impair the performance of the player on the standard game.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

The algorithm maximizes the score of the game i.e. the fitness function is the average of lines scored by a search point on a number of games. When racing is applied, this number of games can go from 3 to r_{max}. When it is not applied it is r_{max}. The initial bounds a and b used to compute R (eq. 5) were fixed experimentally to $a = 150$ and $b = 2000$.

The games were simulated on the MDPTeris platform[5]. All the expriments were repeated 50 times and the curves represent median values.

An important issue (incidentally ignored by many authors) about learning Tetris players and in general about learning the weights of evaluation functions for games, is that the weight should be normalized. This fact is described in [5] where empirical evidence show how in CMA-ES the step-size and the principal axes of the covariance matrix diverge. In these experiments we follow the same steps and normalize the offspring after the sampling step.

Furthermore, in order to provide a coherent ranking of the offspring, we evaluate them on the same series of games. At each iteration, a series of r_{max} independent games (random seeds) are drawn randomly and each individual of the λ offspring plays the same games. This way we can assess of the offspring's performance on the same setting.

3.1 Results and Discussions

The effect of racing is clear on the learning time of the algorithm. The number of evaluations is indeed reduced without a loss of performance. Fig. 1 on the right shows the fitness function of the mean vector (the centroid of the distribution) for the different racing procedures and without racing. All algorithms reach the same performance at different evaluations costs. Hoeffding bounds perform the best reaching the performance for the smallest evaluation time, followed by Tetris bounds and Bernstein. Fig. 1 on the left shows the median rate of evaluations (r_{limit} being 100%) used in the races. With Tetris bounds this number of evaluation is reduced at the beginning of the learning process and increases towards the end reaching r_{limit}. For Hoeffding bounds the number of evaluations oscillates between r_{limit} and lower values. Finally, Bernstein bounds were not efficient compared to Hoeffding bounds, the number of evaluations is not reduced and at each race each individual is evaluated r_{limit} times. This is due to the fact that the standard deviation of the fitness is of the same order of its average. This is the case for Tetris, the standard deviation of the score distribution is almost equal to its mean. In other words $\hat{\sigma_{i,t}}$ is large compared to R in eq. 5. When this is the case Hoeffding bounds are tighter than Bernstein's, and this is the case in all our experiments.

Recall that r_{limit} is adapted using equations 2 and 3 and is initially set to 3, which explains why in the right of fig. 1, Bernstein races used less evaluations than in the case without racing.

[5] MDPTetris simulator is available at http://mdptetris.gforge.inria.fr

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

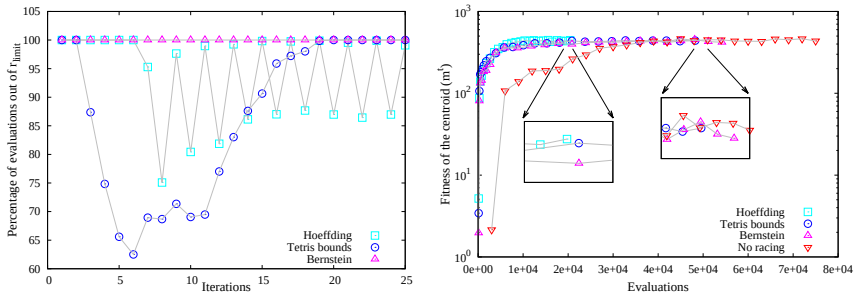


Fig. 1. On the left, the number of evaluations with racing using different bounds versus iterations. On the right, log of the fitness of the mean vector m^t versus the number of evaluations. Median values over 50 runs.

We noticed also that there is a large variability in the racing procedure. Fig. 2 shows the rate of evaluation out of r_{limit} of all 50 runs for Hoeffding and Tetris bounds.

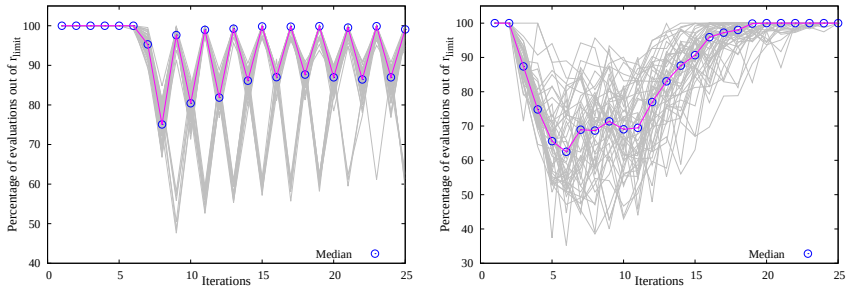


Fig. 2. The number of evaluations for 50 runs thick line represent the median value. With Hoeffding races (left). Races with Tetris bounds(right).

It is interesting to notice that the racing procedures reduce the number of evaluations early in the process and do not perform as well at the end. At this stage the racing procedure cannot distinguish between the offspring and selection is made only based on the fitness values (averaged over the r_{limit} reevaluation). The CI around the offspring's fitness overlap indicating that they perform equally well. This also suggest (see below) that the ranking of the offspring might not be correct. Incidentally, this correspond to the time where the convergence of step-sizes start to "flatten" (figures 4 and 5).

The effect of the population size on the racing procedure can be seen on fig. 3 where is shown the number of evaluations out of r_{limit} for different population sizes. Apparently increasing the population size reduces the number of evalua-

tions within the race using Hoeffding bounds. On the other hand the opposite can be seen when using Tetris bounds.

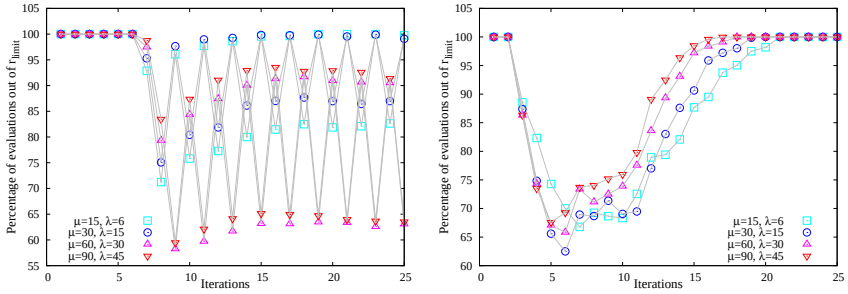


Fig. 3. The median number of evaluations over 50 runs for different population sizes. Using Hoeffding races (left). Using Tetris bounds (right)

Furthermore the population size does not effect the performance of the algorithms, the median value of the mean vector fitness reach similar values for different population sizes. This is the case for both Tetris and Hoeffding bounds (figures 4 and 5).

Fig. 6 shows the effect of the confidence value on the evaluation time. It presents four setting with different confidence $(1 - \delta)$ levels. Using Hoeffding bounds, if we reduce the confidence value the algorithm stops earlier. This is the expected behavior: reducing the confidence reduces the bounds and races are decided faster. On the other hand, Bernstein bounds are not affected at all. The races are not able to distinguish statistical differences between the offspring even with low confidence values.

In order to assess the results of the racing procedure, we test the learned players on a standard game[6] and compare the scores with previously published scores. We follow the same scheme as in [5] to construct the player. We record the last mean vector m^t of each run of the algorithm and compute the mean of all these vectors (in our setting 50). This is merely a convention and others are possible[7]. The score of a player is the average score on 100 games. Taking into account variability of the score distribution, a confidence bound of $\pm 20\%$ is to be taken into account (see eq.6).

Table 1 present the scores of the players learned on CMA-ES with racing compared to the score reported by [5]. We notice that for both games instances the scores are statistically equivalent. Therefore we can conclude that the racing procedures do not affect the performance of the player. Furthermore, the same

[6] A game of size 10×20 with all seven pieces equally probable.

[7] One could also choose the best performing vector out of the whole set. However testing them all on a reasonable number of games might be long. For example testing one player on 10 games took 15 hours on a 2 Ghz CPU.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

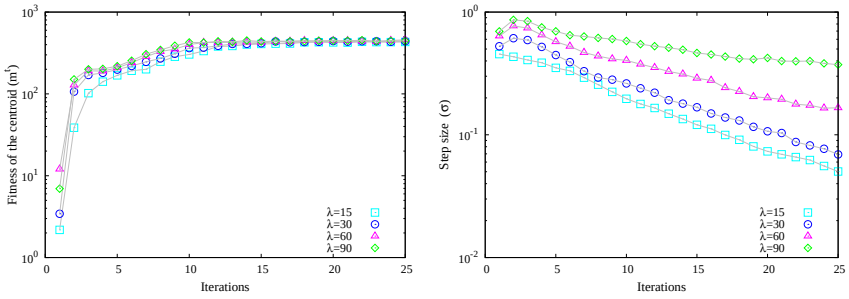


Fig. 4. CMA-ES and Tetris bounds with multiple population sizes. Log of the fitness of the mean vector m^t (left) and the step-size σ (right). Median values over 50 runs.

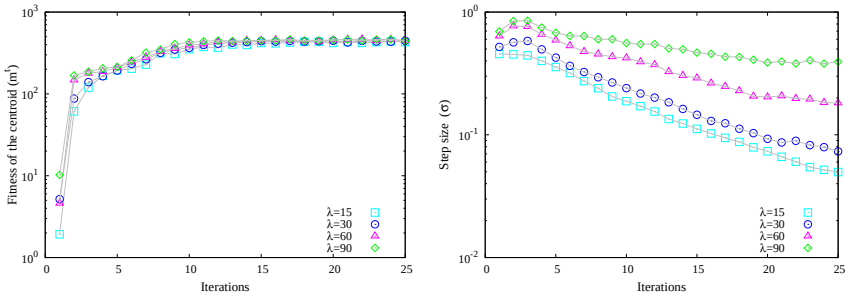


Fig. 5. CMA-ES and Hoeffding races with multiple population sizes. Log of the fitness of the mean vector m^t (left) and the step-size σ (right). Median values over 50 runs.

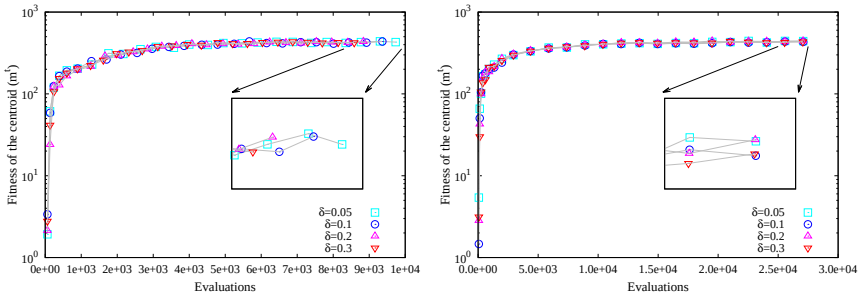


Fig. 6. Log of the fitness of the population centroid for different confidence values. Hoeffding races (left) and Bernstein races (right). Median values over 50 runs.

Table 1. Comparison of the scores of the learned strategies on two game sizes

Strategy	10×16	10×20	Evaluations
No racing [5]	8.7×10^5	36.3×10^6	100%
Tetris bounds	8.1×10^5	31.5×10^6	67%
Hoeffding	7.8×10^5	33.2×10^6	27%

performance was obtained using, in the case of Hoeffding bounds two thirds less evaluations and in the case of Tetris bounds one third less evaluations.

3.2 Further Thoughts

Even though there are benefits in using racing, there are also some disadvantages. First of all, the choice of a value for r_{max}, is crucial for time consuming problems. A too low value leads to short races and thus to no statistical soundness. On the other hand giving a hight value may lead to too many evaluations once the algorithm converges.

As said earlier, when the algorithm converges, the confidence intervals overlap and the racing procedure cannot distinguish statistical differences. This could be considered as a limitation in problems where the cost of an evaluation is not the same at the beginning of the evaluation and towards the end, as it is the case for Tetris. If given the choice, one would prefer to have less costly evaluations rather than the converse. In problems where the cost of the evaluation is constant, this issue is not as severe.

Furthermore, in the event the offspring converge, they all have the same fitness (again statistically). This could raise problems in algorithms that select based on ranking. In the experiments we performed, we noticed that in some cases the ranking (based only on the fitness values) of the offspring changes at each reevaluation during the race. This indicates that at no time the ranking of the population is correct. Selection in such cases becomes random which, and this is purely speculative, might explain why the step-size increases at that stage (figures 4 and 5).

One way to circumvent this limitation, would be to use this convergence as an indication to restart procedure. Once the population converges and we cannot distinguish different individuals, start the algorithm over. It could also be used as a stopping criterion.

4 Conclusions

In the present work, we have described the use of racing methods to reduce the evaluation time in learning artificial Tetris players. Designing such players was done in the past by performing the learning on reduced instances of the game. The addition of racing methods can reduce significantly the learning time without loss of performance, as it has been shown in the experiments.

These experiments also showed that the population size does not affect the racing procedure and its performance. Using Hoeffding and Tetris bounds allowed to reduce the evaluation time, on the other hand Bernstein bounds were inefficient in all our problem instances due the properties of the Tetris fitness function. This also was the case when the confidence level was lowered.

Testing on the standard game instances allowed to show that players designed using CMA-ES with racing have the same performance as the best existing players. Racing also raises few questions that we leave for further investigations.

References

1. Audibert, J.Y., Munos, R., Szepesvári, C.: Tuning bandit algorithms in stochastic environments. In: et. al., M.H. (ed.) Algorithmic Learning Theory, LNCS, vol. 4754, pp. 150–165. Springer (2007)
2. Bertsekas, D., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific (1996)
3. de Boer, P., Kroese, D., Mannor, S., Rubinstein, R.: A tutorial on the cross-entropy method. Annals of Operations Research 1(134), 19–67 (2004)
4. Böhm, N., Kókai, G., Mandl, S.: An Evolutionary Approach to Tetris. In: University of Vienna Faculty of Business; Economics, Statistics (eds.) Proc. of the 6th Metaheuristics International Conference. p. CDROM (2005)
5. Boumaza, A.: On the evolution of artificial tetris players. In: Proc. of the IEEE Symp. on Comp. Intel. and Games CIG'09. pp. 387–393. IEEE (June 2009)
6. Burgiel, H.: How to lose at Tetris. Mathematical Gazette 81, 194–200 (1997)
7. Demaine, E.D., Hohenberger, S., Liben-Nowell, D.: Tetris is hard, even to approximate. In: Proc. 9th COCOON. pp. 351–363 (2003)
8. Fahey, C.P.: Tetris AI, Computer plays Tetris (2003), on the web `http://colinfahey.com/tetris/tetris_en.html`
9. Farias, V., van Roy, B.: Tetris: A study of randomized constraint sampling. Springer-Verlag (2006)
10. Hansen, N., Müller, S., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation 11(1), 1–18 (2003)
11. Hansen, N., Niederberger, S., Guzzella, L., Koumoutsakos, P.: A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. IEEE Trans. Evol. Comp. 13(1), 180–197 (2009)
12. Heidrich-Meisner, V., Igel, C.: Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In: Proc. of the 26th ICML. pp. 401–408. ACM, New York (2009)
13. Maron, O., Moore, A.W.: Hoeffding races: Accelerating model selection search for classification and function approximation. In: In Proc. Advances in neural information processing systems. pp. 59–66. Morgan Kaufmann (1994)
14. Ostermeier, A., Gawelczyk, A., Hansen, N.: A derandomized approach to self-adaptation of evolution strategies. Evolutionary Computation 2(4), 369–38 (1994)
15. Schmidt, C., Branke, J., Chick, S.: Integrating techniques from statistical ranking into evolutionary algorithms. In: Rothlauf, F.e.a. (ed.) Applications of Evolutionary Computing, LNCS, vol. 3907, pp. 752–763. Springer (2006)
16. Siegel, E.V., Chaffee, A.D.: Genetically optimizing the speed of programs evolved to play tetris. In: Angeline, P.J., Kinnear, Jr., K.E. (eds.) Advances in Genetic Programming 2, pp. 279–298. MIT Press, Cambridge (1996)
17. Stagge, P.: Averaging efficiently in the presence of noise. In: et. al., A.E. (ed.) Proc. of PPSN 5, LNCS, vol. 1498, pp. 188–197. Springer (1998)
18. Szita, I., Lörincz, A.: Learning tetris using the noisy cross-entropy method. Neural Comput. 18(12), 2936–2941 (2006)
19. Thiery, C., Scherrer, B.: Building Controllers for Tetris. International Computer Games Association Journal 32, 3–11 (2009)
20. Thiery, C., Scherrer, B.: Least-Squares λ Policy Iteration: Bias-Variance Trade-off in Control Problems. In: Proc. ICML. Haifa (2010)
21. Tsitsiklis, J.N., van Roy, B.: Feature-based methods for large scale dynamic programming. Machine Learning 22, 59–94 (1996)

Model-guided Evolution Strategies for Dynamically Balancing Exploration and Exploitation

Edgar Reehuis[1,2], Johannes Kruisselbrink[1], Markus Olhofer[2], Lars Graening[2], Bernhard Sendhoff[2], and Thomas Bäck[1]

[1] Natural Computing Group, LIACS, Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
{ereehuis,jkruisse,baeck}@liacs.nl
http://natcomp.liacs.nl
[2] Honda Research Institute Europe GmbH
Carl-Legien-Straße 30, 63073 Offenbach/Main, Germany
{markus.olhofer,lars.graening,bernhard.sendhoff}@honda-ri.de

Abstract. Wide exploration of high-dimensional, multimodal design spaces is required for uncovering alternative solutions in the conceptual phase of design optimization tasks. We present a general framework for balancing exploration and exploitation during the course of the optimization that induces sequential exploitation of different optima in the search space by selecting on a solution's fitness and a dynamic criterion termed interestingness. We use a fitness approximation model as a memory representing the parts of the search space that have been visited before. It guides the optimizer toward those areas that require additional sampling to be correctly modeled, and are hence termed interesting. Next to applying the prediction error of the model as a measure of interestingness, we consider the statistical variation in the predictions made by multiple parallel models as an alternative approach to quantify interestingness. On three artificial test functions we compare these setups running on a canonical ES to the same ES extended with either archive-based novelty, niching, or restarting, and to simply evaluating a Latin Hypercube set of sample points.

Keywords: Multimodal optimization, interestingness, novelty, niching, multiobjective selection, prediction error, variation in prediction

1 Introduction

Conventional design optimization of real-world problems aims for efficient adaptation of free system parameters to improve an existing system on given quality criteria. Following such a procedure often results in only a marginal improvement of a principally known solution. Methods for finding innovative and conceptually new solutions therefore are of increasing interest, allowing development of competitive products that are sufficiently distinct from rival products. Determining

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

alternative solutions is generally part of the conceptual design phase where normally a search is performed on a simplified problem with a reduced parameter space, thereby decreasing the level of solution detail to facilitate efficient traversal of the design space. The simplified solutions found serve as a starting point in a following optimization phase on the fully parametrized problem. Ideally the initial search would however be performed on the actual problem as well since potentially good products may get obscured through the parameter reduction.

Evolutionary Algorithms (EAs) are generally capable of exploration but rely largely on random modifications, making the probability of identifying optimal solutions relative to the problem dimensionality. Evolution Strategies (ESs), featuring sophisticated adaptations guiding the sampling of new solutions, quickly result in local searches around good solutions found in an initial exploration phase, on a multimodal problem usually converging to a single, randomly picked optimum. Relying solely on a quality function to select new solutions mainly causes this rapid shift toward exploitation of a single optimum.

An effective way to overcome this problem is *restarting* the optimization process from random initial positions in order to identify multiple optima [1]. This has the chance however of repeatedly zooming in on the same optima while others are never found, depending on the basins of attraction. In [3] an archive of *novel* solutions is kept between restarts that contains solutions exceeding a novelty threshold according to some difference measure [9]. The most novel solution found (i.e., viewed from within the optimization process, not necessarily from a designer's point of view) is used to position the next restart in the least explored region of the search space. *Niching* methods [17, 14] and derivates [18, 15] on the other hand run on top of the optimizer and induce parallel local searches for a predefined maximum number of optima, but as such are not very suitable for highly multimodal landscapes [14]. Another method is *Continuous Tabu Search* that is centered around lists of recently visited and promising solutions that (temporarily) get excluded from the search [10], but is therefore dependent on appropriate continuous neighborhood definitions. We propose an approach that sequentially jumps from optimum to optimum by alternating between states of exploration and exploitation. Next to the fitness function, an additional criterion that is subject to change during optimization is used to guide the random search.

Recent papers report on the use of such dynamic criteria based on a memory of earlier seen solutions parallel to the static fitness function in a *Pareto*-based *multiobjective* selection scheme. Mouret [13] uses an archive-based method, like in [3] of solutions that were highly novel upon discovery [9] and calculates the *novelty* of a solution as the average distance to the nearest solutions in this archive and the current population. Graening et al. [5] use a *fitness model* (i.e., surrogate model of the objective space) to represent the current knowledge of the search space and calculate a solution's *interestingness* as the maximization of the *prediction error* made by the fitness model as compared to the actual fitness value of a solution. As the optimization proceeds the model improves in

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

the area that is being sampled and the Pareto selection will gradually steer the optimizer into an area where the prediction error remains larger.

The current study continues on the model-based approach by introducing an alternative expression for interestingness, namely the *variation in prediction* between multiple fitness submodels trained on the same data. Furthermore, a straightforward approach for attaining global memory is tested. The behavior of the different methods is analyzed on three artificial test functions. We include archive-based novelty *multiobjectivization* [13], *fixed radius* and *self-adaptive radius* niching [17], and restarting with increased population size [1] as benchmarks. All methods run on top of a simple, single-stepsize, comma ES, and use the same evaluation budget. Furthermore, we compare to the basic approach of evaluating a *Latin Hypercube* set [11] of sample points.

In the following section the suggested framework for balancing exploration and exploitation comprising optimizer and interestingness measure is laid out, after which the results of the experiments involving the artificial test functions are presented in Section 3. In Section 4 we conclude with discussion and outlook.

2 Model-guided Framework for Dynamically Balancing Exploration and Exploitation

We propose a model-guided optimization framework for the search of optimal, conceptually different solutions in multimodel fitness functions. The usage of the model differs from other model-based and model-assisted strategies [8, 7] in the sense that the model does not replace the fitness function but is used to provide an additional criterion to guide the search. Based on previous work [5], the framework targets the efficient handling of the trade-off between explorative and exploitative search. The aim is to identify many conceptually different solutions by means of exploration, while providing an optimal configuration for each of the identified concepts, referred to as exploitation. In an alternating pattern the suggested algorithm zooms in on a single optimum until enough data has become available to properly model that part of the fitness landscape, after which it continues exploring for alternative optima.

Fig. 1 depicts the overall framework of the model-guided optimization process, building upon a population-based optimization strategy such as an EA. The main difference to traditional optimization is that the search process is additionally guided by the interestingness of a solution, a dynamic additional criterion that gets taken into account next to the static fitness, in a multiobjective selection scheme. Applying interestingness comes with two major challenges, the construction of an adequate memory of already generated solutions and the definition of the indicator that quantifies interestingness. A universal approximation model predicting solutions' fitness is adopted to form an abstract representation of the solution space. A solution should be qualified as interesting if the solution vector is different from the already generated solutions, or if it resides in a rugged, hard-to-model area of the fitness landscape. Both requirements are

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

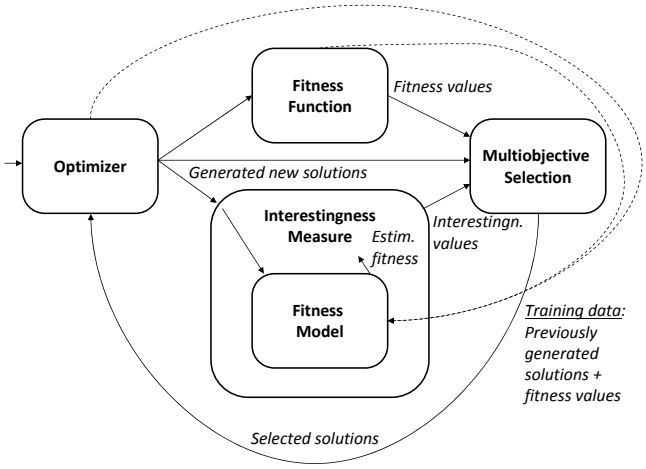


Fig. 1. Model-guided Framework for Dynamically Balancing Exploration and Exploitation.

inherently accounted for in using an interestingness indicator that is based on the prediction error of a fitness approximation model.

2.1 Framework Instantiation

The current study employs a $(\mu, 7\mu)$-ES with self-adaptation based on a single stepsize per individual ($\mu : \lambda$ should at least be $1 : 7$ [16]) as optimizer. The model of the fitness landscape is built using feed-forward neural networks with linear outputs and a single hidden layer of 10 sigmoidally activated nodes. The weights of the networks are trained with the *improved Rprop* backpropagation algorithm by Igel et al. [6] for a maximum of 1000 epochs. The networks are fully connected, including direct connections from input to output nodes, and contain a bias node. All connection weights are initialized in the interval $[-0.1, 0.1]$. The fitness model should represent the current knowledge of the function landscape best, hence there is no risk of overfitting and therefore no validation set is used (i.e., all available solution data is used for the actual training).

Two model-based interestingness indicators are investigated (defined further on), the prediction error (PE) of the fitness model directly and the variation in prediction (ViP) of multiple approximation submodels, the latter implicitly depending on the prediction errors of the different submodels. The question remains whether the aimed for dynamics are best induced by either a memory involving all generated solutions or only of the most recent ones. Therefore two types of models are considered: a *local* fitness model that per iteration is re-initialized and trained on the solution data from the *previous* γ iterations, and a *global* memory consisting of all *distinct* previous local models, that is, every γ-th model. We term the latter a "horizontal ensemble" of fitness models referring

to the distinct training data used per model. In evaluating the interestingness the distinct models as well as the new local model are taken into account, for each evaluated solution adopting the model with the best prediction and lowest variation in case of PE and ViP respectively.

Multiobjective selection based on Pareto dominance is used, as proposed in [5] and [13], implemented using non-dominated sorting (and crowding distance sorting) as featured in NSGA-II [4]. Note that we apply this sorting procedure in combination with (μ, λ)-selection (i.e., the new parent population is selected from the offspring only), as the aim is not to obtain and refine a Pareto front of optimal solutions but to steer the optimizer away from "sufficiently" exploited optima. An initial comparison between plus (i.e., taking parents into account as well) and comma selection showed slower dynamics and the chance of relapsing to standard ES convergence behavior using the first, identifying only one optimum. A disadvantage of using Pareto-based multiobjective selection is that the required degree of exploitation cannot be indicated. To what extent a certain optimum is exploited before the optimizer shifts the focus to a different region of the search space therefore depends on the shape of the fitness landscape and the capabilities of the applied modeling technique, together with the characteristics of the used interestingness measure.

Prediction Error. The interestingness of a real-valued solution vector $\mathbf{x} = (x_1, \ldots, x_n)$ with n indicating the problem's dimensionality is calculated as the absolute difference between the model's fitness estimation $\tilde{f}(\mathbf{x})$ and the actual fitness value $f(\mathbf{x})$ [5],

$$\mathrm{PE}(\mathbf{x}) = \left| \tilde{f}(\mathbf{x}) - f(\mathbf{x}) \right|. \tag{1}$$

Variation in Prediction. The interestingness is taken as the variation between the predictions of multiple submodels that are trained on exactly the same data (as the training data is equal, this could be termed a "vertical ensemble" of submodels). In the current study multiple neural networks of the same architecture and type are trained. If sufficient data is available to correctly model a certain solution the fitness estimations of the different submodels should largely align. Otherwise, the predictions will be more diverse. Let $\tilde{f}_i(\mathbf{x})$ be the fitness estimation by submodel i and $\mathrm{num}_{\mathrm{subm}}$ be the number of different submodels, then the variation in prediction is calculated using the *interquartile range* (IQR, the difference between the third and first quartiles of the data) of the fitness estimations,

$$\mathrm{ViP}(\mathbf{x}) = \mathrm{IQR}(\tilde{f}_1(\mathbf{x}), \ldots, \tilde{f}_{\mathrm{num}_{\mathrm{subm}}}(\mathbf{x})). \tag{2}$$

The IQR is a robust statistic in the sense that it is not prone to outliers. If most submodels "agree" but one is considerably different, this can have a great impact when for instance the sample variance is used as measure of variation.

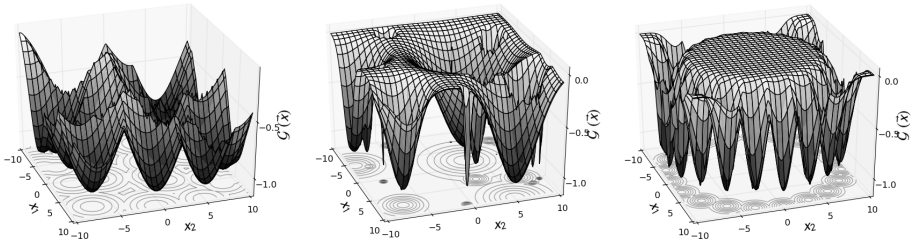
Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

3 Experiments

We run experiments on three artificial test functions, visualized in Fig. 2, that are variants of the *Gaussians* test function [5], which we define as

$$\mathcal{G}(\mathbf{x}) = -\max_{i\in\{1,\ldots,20\}} \exp(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)'\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)), \tag{3}$$

with $\mathbf{x} \in [-10,10]^n$, $\boldsymbol{\mu}_i \sim \mathcal{U}([-10,10]^n)$, $\boldsymbol{\Sigma}_i = \mathrm{diag}((\sigma_1^2,\ldots,\sigma_{20}^2))$, and $\sigma_i = 2\sqrt{n}$. All variants feature 20 global optima regardless of the dimensionality, and problem instances with dimension 2, 5, and 10 are considered. *GaussiansEqual* uses the definition listed above, involving 20 kernels with equal standard deviations positioned randomly in the fitness landscape. *GaussiansVary* uses the default standard deviation for the first kernel and decreases it for the remaining $i \in \{2,\ldots,20\}$ via $\sigma_i = \sigma_{i-1} \cdot 0.8$. *GaussiansPlane* places 20 kernels with equal standard deviations $\sigma_i = 1$ on a two-dimensional plane in a circle with radius 10; in case $n > 2$ the remaining $n-2$ object variables in the kernel means $\boldsymbol{\mu}_i$ are set equal to the same random vector $\mathcal{U}([-10,10]^{n-2})$. This setup reflects the real-world case in which certain parameters (here the first two) can take various values while the remaining ones should be set precisely.



(a) GaussiansEqual (b) GaussiansVary (c) GaussiansPlane

Fig. 2. Test Functions in 2D.

For performance assessment of the different methods we use the *Maximum Peak Ratio* (MPR) measure that reflects the quality and the quantity of the approximated optima in the set of all generated solutions. MPR was originally proposed in the context of niching and maximization problems with strictly positive fitness range [12], therefore we define a slightly adjusted version:

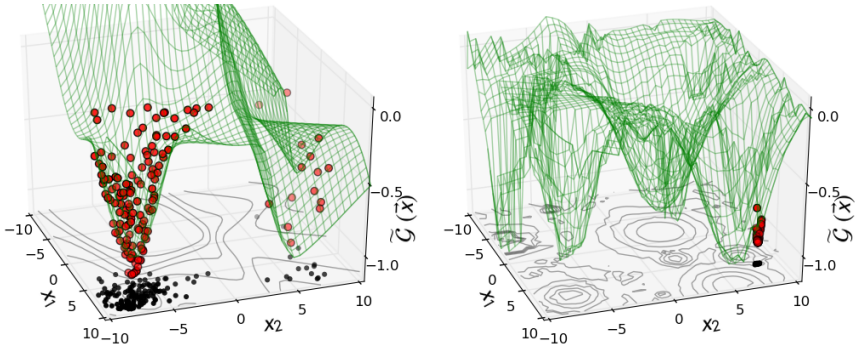
1. Normalize the Gaussians variants to maximization problems, strictly positive, using $f_{\mathrm{norm}}(x) = \frac{f(x)-\max}{\min-\max}$ with $\min = -1.0 - 1\% \cdot \mathrm{range}$, $\max = 0.0 + 1\% \cdot \mathrm{range}$, and $\mathrm{range} = 1.0$;
2. Assign each generated solution to the closest optimum with respect to Euclidean distance;
3. Per optimum, from the solutions assigned to it, use the fitness of the best solution; if it is less than 80% [12] of the actual fitness of the optimum, use 0;

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

4. Divide the sum of the selected fitness value per optimum by the sum of the actual fitness values of the optima.

The maximum MPR score is therefore 1.0, indicating that all optima were closely sampled.

The model-based methods PE and ViP are either guided by γ-*local* models as in [5] or the proposed horizontal *ensemble* of γ-local models. ViP uses 10 submodels per γ-local model. We take $\gamma = 5$ and at the heart of the optimization is a $(5, 35)$-ES applying single-stepsize self-adaptive mutation with learning rate $\tau = \frac{1}{\sqrt{2n}}$ [2]. The initial parent individuals are sampled uniformly within the initialization interval, and their initial stepsize σ_{init} is set to $\frac{1}{4}$ of the initialization interval [17]. Each offspring is the product of 2 parents, using discrete recombination for the object variables x_i and intermediary recombination for the stepsize [2].



(a) Local (b) Ensemble

Fig. 3. Modeling Setups Compared. Plots of the final fitness models obtained in a single run of PE on GaussiansVary 2D, using the 5-local fitness model setup in (a) and the ensemble of 5-local fitness models in (b). The dots represent the solution vectors used to train the last generated 5-local fitness model, in both cases.

As first benchmark we include archive-based novelty multiobjectivization [13], a similar Pareto setup that uses an archive-based novelty criterion to steer the optimizer into sparsely sampled regions, aiming for a wide sampling of the search space that becomes finer grained as the optimization continues. It runs on the same ES as the model-based methods. The novelty of a solution is calculated as the average distance to the nearest 15 solutions with respect to Euclidean distance, selected from an archive of novel solutions and the current population. A solution is added to the archive if its novelty exceeds the novelty threshold, which value is self-adapted during the optimization: if 1% of the fitness evaluations was used and no new solutions were added, the threshold is decreased by 5%, and if more than 4 solutions were added in one generation, the threshold is increased by 5%. Presumably the initial threshold value is to be set relative

to the size of the initialization domain but this relation is not clear. After some tuning it is set to 1.0, noting that due to its self-adaptation and the fact that the novelty score depends strongly on the current population, the exact initial value does not seem highly critical.

Two niching methods are included as benchmarks, fixed radius and self-adaptive niche radius [17], running on top of an $(20, 140)$-ES, except for population size equal to the $(5, 35)$-ES used for the Pareto setups[1]. Niching works by parallelly converging to a pre-indicated number of q optima in the ES population guided by *niche radii*, with a single *niche* being used to zoom in on each optimum. In fixed radius niching the radius is calculated such that the entire search space is evenly divided over the q desired niches, while in self-adaptive radius niching the length of each radius is coupled to changes of the stepsize and it decreases as is zoomed in on an optimum [17]. Although there is interplay between the different niches (e.g., in case of plus selection the individuals can move between niches), recombination is not used; offspring from a certain niche is generated as mutated copies of the niche best individual. We use $q = 20$ and all remaining settings are as prescribed in [17].

The last benchmark is the *IPOP-ES*[2], a scheme in which an underlying $(\mu, 7\mu)$-ES is restarted upon convergence by resampling the population and re-intializing stepsizes to σ_{init}, while increasing the population size between restarts as is aimed for more global search as the optimization proceeds [1]. The process is started from a $(2, 14)$-ES, besides the population size equal to the ESs mentioned above, and μ is increased by a factor of 2 at each restart. The ES is assumed to be converged if the average stepsize of the parent population falls below 10^{-3} times the initialization interval, similar to the convergence criterion used in [3].

For reference we include a comparison with standard ES behavior and the result of covering the search space with a Latin Hypercube (LH) set of sample points [11]. We use LH sampling as this offers multiple sets with mostly similar spread properties. For enforcing the population of the ES to remain within the initialization interval, we employ constraint handling that sets the interestingness or novelty score of an individual to a penalty value upon boundary violation; for the standard ES, niching, and IPOP-ES the fitness is deteriorated by instead adding a penalty value. For all tested methods we use a budget of 3500 function evaluations, and all methods have been implemented using the Shark Machine Learning Library v2.3.3[3].

3.1 Results

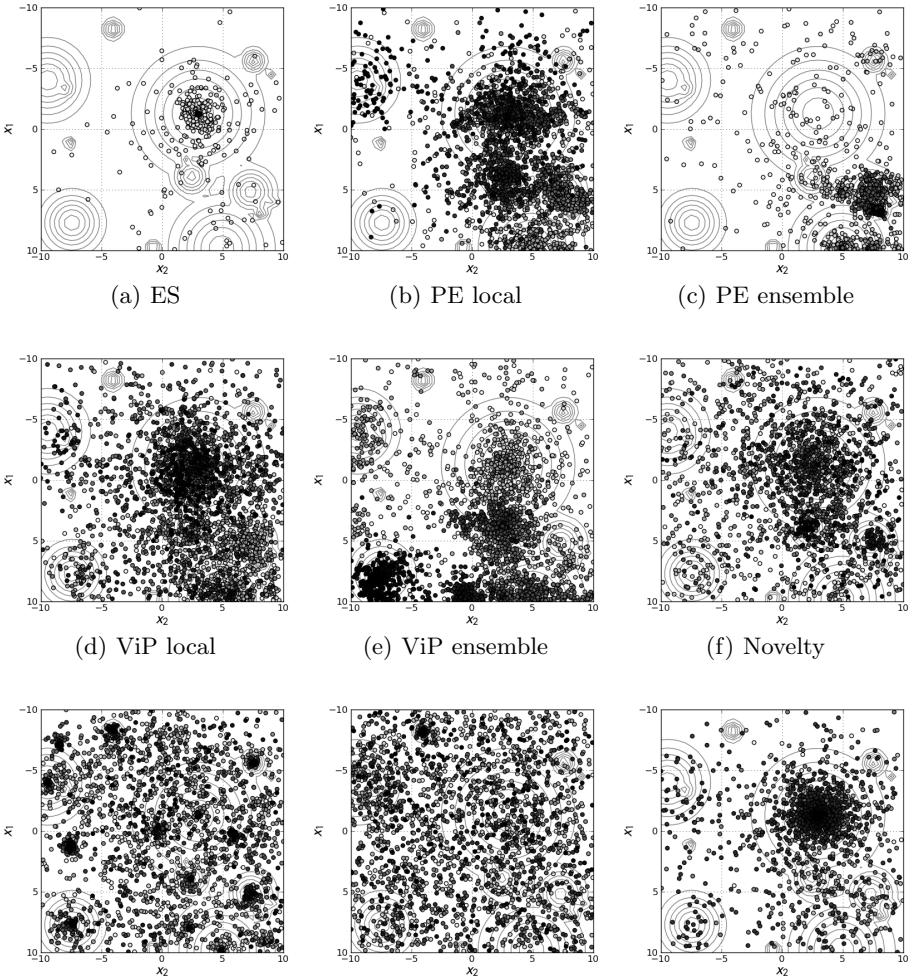
In examining the outcome of the experiments, we start by visually analyzing the sampling behavior for GaussiansVary 2D based on a single run per method, plot-

[1] Running niching on a canonical ES allows for a fair comparison, but note that self-adaptive niching was proposed in combination with the cumulative adaptation mechanism of the stepsize in the CMA-ES [17].

[2] The IPOP-ES, derived from the IPOP-CMA-ES [1], is run on a canonical ES to allow for comparison.

[3] http://shark-project.sourceforge.net

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



(a) ES (b) PE local (c) PE ensemble

(d) ViP local (e) ViP ensemble (f) Novelty

(g) Niching fixed (h) Niching self-adaptive (i) IPOP-ES

Fig. 4. Sampling Behavior on GaussiansVary 2D. Plots of all solutions generated in the search space for a single run per method (dark: recent sol., bright: early sol.).

ted in Fig. 4. The gray tones of the sample points reveal the different dynamics: The standard ES converges quickly to the optimum with the largest attractor basin, while the model-based methods visit optima in a sequential pattern. Novelty multiobjectivization keeps sampling widely across the search space with extra focus on optimal regions, while the niching methods manage to zoom in on most of the optima in parallel. The IPOP-ES shows behavior similar to that of the standard ES, although the search space is sampled more widely because of the restart phases and iteratively increased population size. Of the model-based methods the local variants can be seen revisiting the largest attractor, while ViP

ensemble shows the clearest temporal separation between the visited optima. PE ensemble however gets stuck as the ensemble model is able to approximate most of the fitness landscape, nullifying the effect of the interestingness indicator as the area directly around an optimum must have higher interestingness than the optimum itself to be able to move away from it.

Fig. 5 plots the MPR scores over 30 runs per method. First of all the model-based approaches do not manage to outperform niching, with self-adaptive radius niching showing best performance across the test problems. Of the model-based variants PE ensemble clearly has the worst scores, while ViP local performs best. It benefits from the higher variation in its local model as compared to the ensemble of models. Novelty multiobjectivization has largely comparable results to those of ViP local, while the IPOP-ES shows slightly less performance. Furthermore, uninformed sampling across the entire search space works in low dimensions but as would be expected, evaluating an LH set in 10D is not useful.

4 Discussion

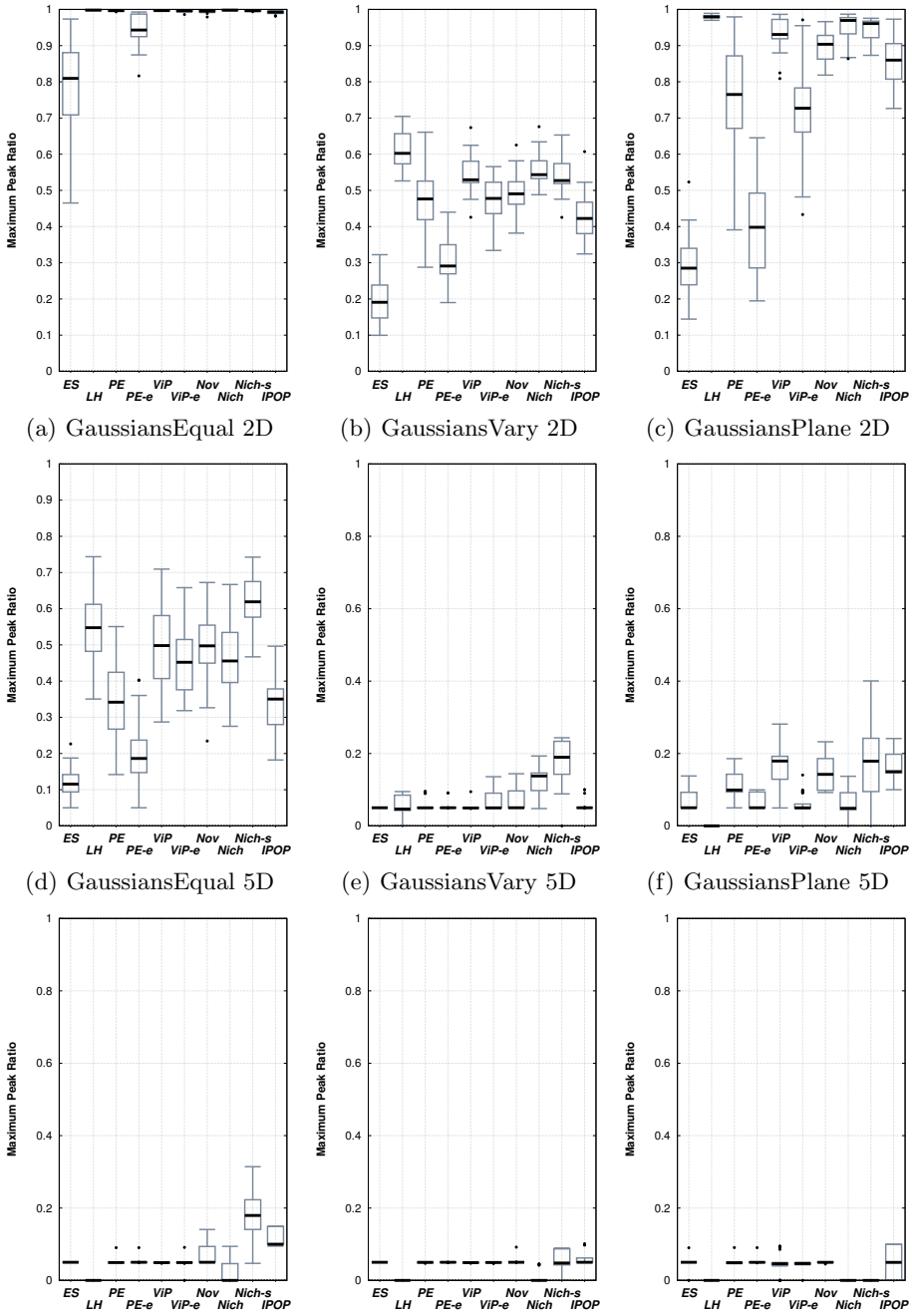
This paper presented a model-guided framework that alternates between states of exploration and exploitation to find multiple optima in a multimodal landscape. The tested instantiation comprised a canonical ES with isotropic mutation, feedforward neural networks, and Pareto-based multiobjective selection on fitness and an interestingness indicator, applying either the model prediction error (PE) or the variation in prediction (ViP) between submodels. A comparison was made to archive-based novelty multiobjectivization, niching methods, restarting with increased population, and evaluating a Latin Hypercube (LH) set of sample points.

ViP local performed best of the proposed model-based approaches with respect to the Maximum Peak Ratio; the novelty scheme showed comparable performance to it while the restart setup performed slightly less. Self-adaptive niching however showed the best overall performance. On 10-dimensional problem instances all tested methods performed weakly, while still better than LH.

While elegant in the sense of omitting weight parameters on the fitness and interestingness values, the applied Pareto-based selection in the framework induces a dispersing effect, slowing down the ES in zooming in on optima. Conversely, it hinders the ES in escaping from sufficiently exploited optima, together with the small stepsize resulting after an exploitation phase. An alternative would be dynamically combining fitness and interestingness into a single objective, although this presumably suffers from the same problem while do requiring weighting.

Cuccu et al. [3] dismiss the dynamic aggregation in favor of restarting the ES after convergence, using an archive of visited points to position it in the least explored region of the search space. Instead, a global model could be trained between restarts on *all* sampled points and searched for the least *understood* region, e.g., applying ViP. Moreover, using restarts omits the need to perform a clustering step to isolate optima, required when applying the framework to black-box problems.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



(a) GaussiansEqual 2D (b) GaussiansVary 2D (c) GaussiansPlane 2D

(d) GaussiansEqual 5D (e) GaussiansVary 5D (f) GaussiansPlane 5D

(g) GaussiansEqual 10D (h) GaussiansVary 10D (i) GaussiansPlane 10D

Fig. 5. Results Overview. The results of 30 runs per method on each test function instance are plotted, all runs involving 3500 function evaluations. The Maximum Peak Ratio indicates the quality and the quantity of the approximated optima in the set of all generated solutions per run and is to be maximized.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

References

1. Auger, A., Hansen, N.: A Restart CMA Evolution Strategy with Increasing Population Size. In: 2005 IEEE Congress on Evolutionary Computation, CEC 2005. pp. 1769–1776. IEEE Computer Society (2005)
2. Bäck, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Oxford, UK (1996)
3. Cuccu, G., Gomez, F., Glasmachers, T.: Novelty-Based Restarts for Evolution Strategies. In: 2011 IEEE Congress on Evolutionary Computation, CEC 2011. pp. 158–163. IEEE Computer Society (2011)
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Trans. Ev. Comp. 6(2), 182–197 (2002)
5. Graening, L., Aulig, N., Olhofer, M.: Towards Directed Open-ended Search by a Novelty Guided Evolution Strategy. In: Schaefer, R., et al. (ed.) PPSN XI, LNCS, vol. 6239. pp. 71–80. Springer (2010)
6. Igel, C., Hüsken, M.: Empirical evaluation of the improved Rprop learning algorithms. Neur. 50, 105–123 (2003)
7. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Comp. 9(1), 3–12 (2005)
8. Jones, D., Schonlau, M., Welch, W.: Efficient Global Optimization of Expensive Black-Box Functions. J. Glob. Opt. 13(4), 455–492 (1998)
9. Lehman, J., Stanley, K.: Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. In: Artificial Life XI. pp. 329–336. MIT Press (2008)
10. Mashinchi, M., Orgun, M., Pedrycz, W.: Hybrid optimization with improved tabu search. Appl. Soft Comp. 11(2), 1993–2006 (2011)
11. McKay, M., Beckman, R., Conover, W.: A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. Techn. 21(2), 239–245 (1979)
12. Miller, B., Shaw, M.: Genetic Algorithms with Dynamic Niche Sharing for Multimodal Function Optimization. In: 1996 IEEE Congress on Evolutionary Computation, CEC 1996. pp. 786–791. IEEE Computer Society (1996)
13. Mouret, J.B.: Novelty-Based Multiobjectivization. In: Doncieux, S., et al. (ed.) 2009 EvoDeRob Workshop, SCI, vol. 341. pp. 139–154. Springer (2011)
14. Preuss, M.: Niching the CMA-ES via Nearest-Better Clustering. In: Branke, J., et al. (ed.) 12th Annual Conference on Genetic and Evolutionary Computation, GECCO'10. pp. 1711–1718. ACM Press (2010)
15. Saha, A., Deb, K.: A Bi-criterion Approach to Multimodal Optimization: Self-adaptive Approach. In: Deb, K., et al. (ed.) SEAL 2010, LNCS, vol. 6457. pp. 95–104. Springer (2010)
16. Schwefel, H.P.: Evolution and Optimum Seeking: The Sixth Generation. John Wiley & Sons, Inc., New York, NY, USA (1993)
17. Shir, O., Emmerich, M., Bäck, T.: Adaptive Niche Radii and Niche Shapes Approaches for Niching with the CMA-ES. Ev. Comp. 18(1), 97–126 (2010)
18. Zechman, E., Ranjithan, S., Liu, L.: Niched Co-Evolution Strategies to Address Non-uniqueness in Engineering Design. In: 8th Annual Conference on Genetic and Evolutionary Computation, GECCO'06. Late breaking paper. ACM Press (2006)

Black-Box Complexity: Breaking the $O(n \log n)$ Barrier of LeadingOnes

Benjamin Doerr and Carola Winzen

Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. We show that the unrestricted black-box complexity of the n-dimensional XOR- and permutation-invariant LeadingOnes function class is $O(n \log(n)/ \log \log n)$. This shows that the recent natural looking $O(n \log n)$ bound is not tight.
The black-box optimization algorithm leading to this bound can be implemented in a way that only 3-ary unbiased variation operators are used. Hence our bound is also valid for the unbiased black-box complexity recently introduced by Lehre and Witt. The bound also remains valid if we impose the additional restriction that the black-box algorithm does not have access to the objective values but only to their relative order (ranking-based black-box complexity).

Keywords: Algorithms; black-box complexity; query complexity; runtime analysis; theory.

1 Introduction

The black-box complexity of a set $\mathcal{F}$ of functions $\mathcal{S} \to \mathbb{R}$, roughly speaking, is the number of function evaluations necessary to find the maximum of any member of $\mathcal{F}$ which—apart from the points evaluated so far—is unknown. This and related notions are used to describe how difficult a problem is to be solved via general-purpose (randomized) search heuristics. Consequently, black-box complexities are very general lower bounds which are valid for a wide range of evolutionary algorithms. A number of different black-box notions exist, each capturing different classes of randomized search heuristics, cf. [1], [2], and [3].

In this paper we consider the unrestricted black-box model by Droste, Jansen, and Wegener [1] and the unbiased black-box model by Lehre and Witt [2], and we shortly remark on the ranking-based models which we propose in [3]. A formal definition of the first two models is given in Section 2. For now, let us just mention that algorithms in the unrestricted black-box model are allowed to query any bit string, whereas in the k-ary unbiased model, an algorithm may only query search points sampled from a k-ary unbiased distribution. That is, in each iteration, the algorithm may either query a random search point or, based upon at most k previously queried search points, it may generate a new one. The new search point can be generated only by using so-called unbiased variation operators. These are operators that are symmetric both in the bit values (0 or 1) and the

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

bit positions. More formally, the variation operation must be invariant under all automorphisms of the hypercube.

In this work, we are concerned with the black-box complexity of the LeadingOnes function, which is one of the classical test functions for analyzing the optimization behavior of different search heuristics. The function itself is defined via $\text{Lo} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\}$. It was introduced in [4] to disprove a previous conjecture by Mühlenbein [5] that any unimodal function can be optimized by the well-known $(1+1)$ evolutionary algorithm (EA) in $O(n \log n)$ iterations. Rudolph [4] proves an upper bound of $O(n^2)$ for the expected optimization time of the $(1+1)$ EA on Lo and concludes from experimental studies a lower bound of $\Omega(n^2)$—a bound which was rigorously proven in 2002 by Droste, Jansen, and Wegener [6]. This $\Theta(n^2)$ expected optimization time of the simple $(1+1)$ EA seems optimal among the commonly studied evolutionary algorithms.

Note that the unrestricted black-box complexity of the Lo function is 1: The algorithm querying the all-ones vector $(1, \ldots, 1)$ in the first query is optimal. Motivated by this and by the fact that the unbiased black-box model only allows variation operators which are invariant with respect to the bit values and the bit positions, we shall study here a generalized version of the Lo function. More precisely, we consider the closure of Lo under all permutations $\sigma \in S_n$ and under all exchanges of the bit values 0 and 1. It is immediate, that each of these functions has a fitness landscape that is isomorphic to the one induced by Lo. To be more precise, we define for any bit string $z \in \{0,1\}^n$ and any permutation σ of $[n]$ the function

$$\text{Lo}_{z,\sigma} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : z_{\sigma(j)} = x_{\sigma(j)}\}\,.$$

We let LeadingOnes_n be the set $\{\text{Lo}_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\}$ of all such functions.

Note that this definition differs from the one in [1], where only the subclass $\text{LeadingOnes}_n^0 := \{\text{Lo}_{z,\text{id}} \mid z \in \{0,1\}^n\}$ not invariant under permutations of $[n]$ is studied. Here, id denotes the identity mapping on $[n]$. For this restricted subclass, Droste, Jansen, and Wegener [1] prove an unrestricted black-box complexity of $\Theta(n)$. Of course, their lower bound $\Omega(n)$ is a lower bound for the unrestricted black-box complexity of the general LeadingOnes_n function class, and consequently, a lower bound for the unbiased black-box complexity of LeadingOnes_n.

The function class LeadingOnes_n has implicitly been studied for the first time in [2], where Lehre and Witt show that indeed the (1+1) EA is provably (asymptotically) optimal among all unbiased black-box algorithms of arity at most one. This establishes a natural $\Theta(n^2)$ bound for LeadingOnes_n.

Surprisingly, it turns out that this bound does not hold for the unrestricted black-box model and that it does not even hold in the 2-ary unbiased black-box model. In [7] it is shown that, assuming knowledge on $\sigma(1), \ldots, \sigma(\ell)$, one can perform a binary search to determine $\sigma(\ell+1)$ and its corresponding bit value. Since this has to be done n times, an upper bound of $O(n \log n)$ for the

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

unrestricted black-box complexity of LeadingOnes$_n$ follows. Furthermore, this $O(n \log n)$ bound can already be achieved in the binary unbiased model. Up to now, this is the best known upper bound for the unrestricted and the 2-ary unbiased black-box complexity of LeadingOnes$_n$.

In this work we show that both in the unrestricted model (Section 3) and for arities at least three (Section 4), one can do better, namely that $O(n \log(n)/\log\log n)$ queries suffice to optimize any function in LeadingOnes$_n$. This breaks the previous $O(n \log n)$ barrier. This result also shows why previous attempts to prove an $\Omega(n \log n)$ lower bound must fail.

Unfortunately, also the ranking-based model does not help to overcome this unnatural low black-box complexity. We shall comment in Section 5 that the 3-ary unbiased ranking-based black-box complexity of LeadingOnes$_n$, too, is $O(n \log(n)/\log\log n)$.

As for the memory-restricted model we note without proof that a memory of size $O(\sqrt{\log n})$ suffices to achieve the same bound.

2 Preliminaries

In this section we briefly introduce the two black-box models considered in this work, the *unrestricted black-box model* by Droste, Jansen, and Wegener [1] and the *unbiased black-box model* by Lehre and Witt [2]. Due to space limitations, we keep the presentation as concise as possible. For a more detailed discussion of the two different black-box models and for the definition of the ranking-based versions considered in Section 5, we refer the reader to [3].

Before we introduce the two black-box models, let us fix some notation. For all positive integers $k \in \mathbb{N}$ we abbreviate $[k] := \{1, \dots, k\}$ and $[0..k] := [k] \cup \{0\}$. By e_k^n we denote the k-th unit vector $(0, \dots, 0, 1, 0, \dots, 0)$ of length n. For a set $I \subseteq [n]$ we abbreviate $e_I^n := \sum_{i \in I} e_i^n = \oplus_{i \in I} e_i^n$, where $\oplus$ denotes the bitwise exclusive-or. By S_n we denote the set of all permutations of $[n]$ and for $x = (x_1, \dots, x_n) \in \{0,1\}^n$ and $\sigma \in S_n$ we abbreviate $\sigma(x) := (x_{\sigma(1)}, \dots, x_{\sigma(n)})$. For any two strings $x, y \in \{0,1\}^n$ let $\mathcal{B}(x,y) := \{i \in [n] \mid x_i = y_i\}$, the set of positions in which x and y coincide.

For $r \in \mathbb{R}_{\geq 0}$, let $\lceil r \rceil := \min\{n \in \mathbb{N}_0 \mid n \geq r\}$ and $\lfloor r \rfloor := \max\{n \in \mathbb{N}_0 \mid n \leq r\}$. For the purpose of readability we sometime omit the $\lceil \cdot \rceil$ signs, that is, whenever we write r where an integer is required, we implicitly mean $\lceil r \rceil$. All logarithms log in this work are base two logarithms. By ln we denote the logarithm to the base $e := \exp(1)$.

Black-Box Complexity. Let $\mathcal{A}$ be a class of algorithms and $\mathcal{F}$ be a class of functions. For every $A \in \mathcal{A}$ and $f \in \mathcal{F}$ let $T(A,f) \in \mathbb{R} \cup \{\infty\}$ be the expected number of fitness evaluations until A queries for the first time some $x \in \arg\max f$. We call $T(A,f)$ the *expected optimization time of A for f*. The *A-black-box complexity of $\mathcal{F}$* is $T(A,\mathcal{F}) := \sup_{f \in \mathcal{F}} T(A,f)$, the worst-case expected optimization time of A on $\mathcal{F}$. The *$\mathcal{A}$-black-box complexity of $\mathcal{F}$* is $\inf_{A \in \mathcal{A}} T(A,\mathcal{F})$, the best worst-case expected optimization time an algorithm of $\mathcal{A}$ can exhibit on $\mathcal{F}$.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Initialization:
  Sample $x^{(0)}$ according to some probability distribution $p^{(0)}$ on $\mathcal{S}$;
  Query $f(x^{(0)})$;
Optimization:
  for $t = 1, 2, 3, \ldots$ **do**
    Depending on $\big((x^{(0)}, f(x^{(0)})), \ldots, (x^{(t-1)}, f(x^{(t-1)}))\big)$ choose a probability distribution $p^{(t)}$ on $\mathcal{S}$;
    Sample $x^{(t)}$ according to $p^{(t)}$;
    Query $f(x^{(t)})$;

Algorithm 1: Scheme of an unrestricted black-box algorithm for optimizing $f : \mathcal{S} \to \mathbb{R}$

The Unrestricted Black-Box Model. The black-box complexity of a class of functions depends crucially on the class of algorithms under consideration. If the class $\mathcal{A}$ contains all (deterministic and randomized) algorithms, we refer to the respective complexity as the *unrestricted black-box complexity.* This is the model by Droste, Jansen, and Wegener [1]. The scheme of an unrestricted algorithm is presented in Algorithm 1.

Note that this algorithm runs forever. Since our performance measure is the expected number of iterations needed until for the first time an optimal search point is queried, we do not specify a termination criterion for black-box algorithms here.

The Unbiased Black-Box Model. As observed already by Droste, Jansen, and Wegener [1], the unrestricted black-box complexity can be surprisingly low for different function classes. For example, it is shown in [1] that the unrestricted black-box complexity of the NP-hard optimization problem MaxClique is polynomial.

This motivated Lehre and Witt [2] to define a more restrictive class of algorithms, the so-called *unbiased black-box model,* where algorithms may generate new solution candidates only from random or previously generated search points and only by using *unbiased* variation operators, cf. Definition 1. Still the model captures most of the commonly studied search heuristics, such as many $(\mu + \lambda)$ and (μ, λ) evolutionary algorithms, simulated annealing algorithms, the Metropolis algorithm, and the Randomized Local Search algorithm (confer the book [8] for the definitions of these algorithms).

Definition 1 (Unbiased Variation Operator). *For all $k \in \mathbb{N}$, a k-*ary unbiased distribution $\big(D(\cdot \mid y^{(1)}, \ldots, y^{(k)})\big)_{y^{(1)}, \ldots, y^{(k)} \in \{0,1\}^n}$ *is a family of probability distributions over $\{0,1\}^n$ such that for all* inputs $y^{(1)}, \ldots, y^{(k)} \in \{0,1\}^n$ *the following two conditions hold.*

$$(i)\, \forall x, z \in \{0,1\}^n : D(x \mid y^{(1)}, \ldots, y^{(k)}) = D(x \oplus z \mid y^{(1)} \oplus z, \ldots, y^{(k)} \oplus z)\,;$$
$$(ii)\, \forall x \in \{0,1\}^n\, \forall \sigma \in S_n : D(x \mid y^{(1)}, \ldots, y^{(k)}) = D(\sigma(x) \mid \sigma(y^{(1)}), \ldots, \sigma(y^{(k)}))\,.$$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Initialization:
  Sample $x^{(0)} \in \{0,1\}^n$ uniformly at random;
  Query $f(x^{(0)})$;
Optimization:
  for $t = 1, 2, 3, \ldots$ **do**
    Depending on $\left(f(x^{(0)}), \ldots, f(x^{(t-1)})\right)$ choose up to k indices $i_1, \ldots, i_k \in [0..t-1]$ and a k-ary unbiased distribution $D(\cdot \mid x^{(i_1)}, \ldots, x^{(i_k)})$;
    Sample $x^{(t)}$ according to $D(\cdot \mid x^{(i_1)}, \ldots, x^{(i_k)})$;
    Query $f(x^{(t)})$;

Algorithm 2: Scheme of a k-ary unbiased black-box algorithm

We refer to the first condition as $\oplus$-invariance *and to the second as* permutation invariance. *An operator sampling from a k-ary unbiased distribution is called a* k-ary unbiased variation operator.

1-ary—also called *unary*—operators are sometimes referred to as mutation operators, in particular in the field of evolutionary computation. 2-ary—also called *binary*—operators are often referred to as crossover operators. If we allow arbitrary arity, we call the corresponding model the $*$-ary unbiased black-box model.

A k-ary unbiased black-box algorithm can now be described via the scheme of Algorithm 2. The *k-ary unbiased black-box complexity* of some class of functions $\mathcal{F}$ is the complexity of $\mathcal{F}$ with respect to all k-ary unbiased black-box algorithms.

3 On LeadingOnes$_n$ in the Unrestricted Model

This section is devoted to the main contribution of this work, Theorem 1. Recall from the introduction that we have defined

$$\mathrm{Lo}_{z,\sigma} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : z_{\sigma(j)} = x_{\sigma(j)}\}$$

and $\textsc{LeadingOnes}_n := \{\mathrm{Lo}_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\}$.

Theorem 1. *The unrestricted black-box complexity of* $\textsc{LeadingOnes}_n$ *is* $O(n \log(n)/\log\log n)$.

The proof of Theorem 1 is technical. For this reason, we split it into several lemmata. The main proof can be found at the end of this section. We remark already here that the algorithm certifying Theorem 1 will make use of unbiased variation operators only. Hence, it also proves that the $*$-ary unbiased black-box complexity of $\textsc{LeadingOnes}_n$ is $O(n \log(n)/\log\log n)$. This will be improved to the 3-ary model in Section 4.

The main idea of both the $*$-ary and the 3-ary algorithm is the following. Given a bit string x of fitness $\mathrm{Lo}_{z,\sigma}(x) = \ell$, we iteratively first learn $k := \sqrt{\log n}$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

bit positions $\sigma(\ell+1),\ldots,\sigma(\ell+k)$ and their corresponding bit values which we fix for all further iterations of the algorithm. Learning such a block of size k will require $O(k^3/\log k^2)$ queries. Since we have to optimize n/k such blocks, the overall expected optimization is $O(nk^2/\log k^2) = O(n\log(n)/\log\log n)$. In what follows, we shall formalize this idea.

Convention: For all following statements let us fix a positive integer n, a bit string $z \in \{0,1\}^n$ and a permutation $\sigma \in S_n$.

Definition 2 (Encoding Pairs). *Let $\ell \in [0..n]$ and let $y \in \{0,1\}^n$ with $\mathrm{Lo}_{z,\sigma}(y) = \ell$. If $x \in \{0,1\}^n$ satisfies $\mathrm{Lo}_{z,\sigma}(x) \geq \mathrm{Lo}_{z,\sigma}(y)$ and $\ell = |\{i \in [n] \mid x_i = y_i\}|$, we call (x,y) an* ℓ-encoding pair for $\mathrm{Lo}_{z,\sigma}$.

If (x,y) is an ℓ-encoding pair for $\mathrm{Lo}_{z,\sigma}$, the bit positions $\mathcal{B}(x,y)$ are called the ℓ-encoding bit positions of $\mathrm{Lo}_{z,\sigma}$ *and the bit positions $j \in [n]\backslash\mathcal{B}(x,y)$ are called* non-encoding.

If (x,y) is an ℓ-encoding pair for $\mathrm{Lo}_{z,\sigma}$ we clearly either have have $\mathrm{Lo}_{z,\sigma}(x) > \ell$ or $\mathrm{Lo}_{z,\sigma}(x) = \mathrm{Lo}_{z,\sigma}(y) = n$. For each non-optimal $y \in \{0,1\}^n$ we call the unique bit position which needs to be flipped in y in order to increase the objective value of y the *ℓ-critical bit position of* $\mathrm{Lo}_{z,\sigma}$. Clearly, the ℓ-critical bit position of $\mathrm{Lo}_{z,\sigma}$ equals $\sigma(\ell+1)$, but since σ is unknown to the algorithm we shall make use of this knowledge only in the analysis, and not in the definition of our algorithms. In the same spirit, we call the k bit positions $\sigma(\ell+1),\ldots,\sigma(\ell+k)$ the *k ℓ-critical bit positions of* $\mathrm{Lo}_{z,\sigma}$.

Lemma 1. *Let $\ell \in [0..n-1]$ and let $(x,y) \in \{0,1\}^n \times \{0,1\}^n$ be an ℓ-encoding pair for $\mathrm{Lo}_{z,\sigma}$. Furthermore, let $k \in [n - \mathrm{Lo}_{z,\sigma}(y)]$ and let $y' \in \{0,1\}^n$ with $\ell \leq \mathrm{Lo}_{z,\sigma}(y') < \ell + k$.*

If we create y'' from y' by flipping each non-encoding bit position $j \in [n]\backslash\mathcal{B}(x,y)$ with probability $1/k$, then

$$\Pr[\mathrm{Lo}_{z,\sigma}(y'') > \mathrm{Lo}_{z,\sigma}(y')] \geq (ek)^{-1}.$$

Lemma 1 motivates us to formulate Algorithm 3 which can be seen as a variant of the standard $(1+1)$ EA, in which we fix some bit positions and where we apply a non-standard mutation probability. The variation operator $\texttt{random}(y', x, y, 1/k)$ samples a bit string y'' from y' by flipping each non-encoding bit position $j \in [n]\backslash\mathcal{B}(x,y)$ with probability $1/k$. This is easily seen to be an unbiased variation operator of arity 3.

The following statement follows easily from Lemma 1 and the linearity of expectation.

Corollary 1. *Let (x,y), ℓ, and k be as in Lemma 1. Then the (x,y)-encoded $(1+1)$ EA with mutation probability $1/k$, after an expected number of $O(k^2)$ queries, outputs a bit string $y' \in \{0,1\}^n$ with $\mathrm{Lo}_{z,\sigma}(y') \geq \ell + k$.*

A second key argument in the proof of Theorem 1 is the following. Given an ℓ-encoding pair (x,y) and a bit string y' with $\mathrm{Lo}_{z,\sigma}(y') \geq \ell + \sqrt{\log n}$, we are able to learn the $\sqrt{\log n}$ ℓ-critical bit positions $\sigma(\ell+1),\ldots,\sigma(\ell+\sqrt{\log n})$ in an

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Input: ℓ-encoding pair $(x, y) \in \{0,1\}^n \times \{0,1\}^n$.
$y' \leftarrow y$;
while $\mathrm{Lo}_{z,\sigma}(y') < \mathrm{Lo}_{z,\sigma}(y) + k$ **do**
  $y'' \leftarrow \texttt{random}(y', x, y, 1/k)$;
  Query $\mathrm{Lo}_{z,\sigma}(y'')$;
  if $\mathrm{Lo}_{z,\sigma}(y'') > \mathrm{Lo}_{z,\sigma}(y')$ **then** $y' \leftarrow y''$;
Output y';

Algorithm 3: The (x, y)-encoded (1+1) evolutionary algorithm with mutation probability $1/k$.

expected number of $O(\log^{3/2}(n)/\log\log n)$ queries. This will be formalized in the following statements.

Lemma 2. *Let $\ell \in [0..n - \lceil\sqrt{\log n}\rceil]$ and let (x, y) be an ℓ-encoding pair for $\mathrm{Lo}_{z,\sigma}$. Furthermore, let y' be a bit string with $\mathrm{Lo}_{z,\sigma}(y') \geq \ell + \sqrt{\log n}$.*

For each $i \in [8e\log^{3/2}(n)/\log\log n]$ let y^i be sampled from y' by independently flipping each non-encoding bit position $j \in [n]\backslash\mathcal{B}(x, y)$ with probability $1/\sqrt{\log n}$.

For each $c \in [\sqrt{\log n}]$ let

$$X_c := \{y^i \mid i \in [8e\log^{3/2}(n)/\log\log n] \text{ and } \mathrm{Lo}_{z,\sigma}(y^i) = \ell + c - 1\},$$

the set of all samples y^i with $\mathrm{Lo}_{z,\sigma}(y^i) = \ell + c - 1$.

Then

$$\Pr\left[\forall c \in [\sqrt{\log n}] : |X_c| \geq 4\log(n)/\log\log n\right] \geq 1 - o(1).$$

These sets X_c are large enough to identify $\sigma(\ell + c)$.

Lemma 3. *Let ℓ, (x, y), and y' be as in Lemma 2 and let $t := 4\log(n)/\log\log n$.*

For any $c \in [\sqrt{\log n}]$ let X_c be a set of at least t bit strings $y^1(c), \ldots, y^{|X_c|}(c)$ with fitness $\mathrm{Lo}_{z,\sigma}(y^i(c)) = \ell + c - 1$, which are sampled from y' by independently flipping each non-encoding bit position $j \in [n]\backslash\mathcal{B}(x, y)$ with probability $1/\sqrt{\log n}$.

Then we have, with probability at least $1 - o(1)$, that for all $c \in [\sqrt{\log n}]$ there exists only one non-encoding $j := j_{\ell+c} \in [n]\backslash\mathcal{B}(x, y)$ with $y'_j = 1 - y^i_j(c)$ for all $i \in [|X_c|]$. Clearly, $j = \sigma(\ell + c)$.

Combining Lemma 2 with Lemma 3 we immediately gain the following.

Corollary 2. *Let $\ell, (x, y), y'$, and $y^i, i = 1, \ldots, 8e\log^{3/2}(n)/\log\log n$, be as in Lemma 2.*

With probability at least $1 - o(1)$ we have that for all $c \in [\sqrt{\log n}]$ there exists only one non-encoding $j := j_{\ell+c} \in [n]\backslash\mathcal{B}(x, y)$ with $y'_j = 1 - y^i_j$ for all $i \in [8e\log^{3/2}(n)/\log\log n]$ with $\mathrm{Lo}_{z,\sigma}(y^i) = \ell + c - 1$. Clearly, $j = \sigma(\ell + c)$.

We are now ready to prove Theorem 1. As mentioned above, the proof also shows that the statement remains correct if we consider the unbiased black-box model with arbitrary arity.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Proof (of Theorem 1). We need to show that there exists an algorithm which maximizes any (a priori unknown) function $\text{LO}_{z,\sigma} \in \text{LEADINGONES}_n$ using, on average, $O(n \log(n)/ \log \log n)$ queries.

For readability purposes, let us fix some function $f = \text{LO}_{z,\sigma} \in \text{LEADINGONES}_n$ to be maximized by the algorithm.

First, let us give a rough idea of our algorithm, Algorithm 4. A detailed analysis can be found below.

The main idea is the following. We maximize f block-wise, where each block has a length of $\sqrt{\log n}$ bits. Due to the influence of the permutation σ on f, these bit positions are a priori unknown. Assume for the moment that we have an ℓ-encoding pair (x, y), where $\ell \in [0..n - \lceil\sqrt{\log n}\,\rceil]$. In the beginning we have $\ell = 0$ and $y = x \oplus (1, \dots, 1)$, the bitwise complement of x. To find an $(\ell+\sqrt{\log n})$-encoding pair, we first create a string y' with objective value $f(y') \geq \ell + \sqrt{\log n}$. By Corollary 1, this requires on average $O(\log n)$ queries. Next, we need to identify the $\sqrt{\log n}$ $f(y)$-critical bit positions $\sigma(\ell+1), \dots, \sigma(\ell+\sqrt{\log n})$. To this end, we sample enough bit strings such that we can unambiguously identify these bit positions. As we shall see, this requires on average $O(\log^{3/2}(n)/ \log \log n)$ queries. After identifying the critical bits, we update (x, y) to a $(\ell + \sqrt{\log n})$-encoding pair. Since we need to optimize $n/\sqrt{\log n}$ such blocks of size $\sqrt{\log n}$, the overall expected optimization time is $O(n \log(n)/ \log \log n)$.

Let us now present a more detailed analysis. We start by querying two complementary bit strings x, y. By swapping x with y in case $f(y) \geq f(x)$, we ensure that $f(x) > f(y) = 0$. This gives us a 0-encoding pair.

Let an ℓ-encoding pair (x, y), for some fixed value $\ell \in [0..n - \lceil\sqrt{\log n}\,\rceil]$, be given. We show how from this we find an $(\ell + \sqrt{\log n})$-encoding pair in an expected number of $O(\log^{3/2}(n)/ \log \log n)$ queries.

As mentioned above, we first find a bit string y' with objective value $f(y') \geq \ell + \sqrt{\log n}$. We do this by running Algorithm 3, the (x, y)-encoded $(1 + 1)$ EA with mutation probability $1/\sqrt{\log n}$ until we obtain such a bit string y'. By Corollary 1 this takes, on average, $O(\log n)$ queries.

Next we want to identify the $\sqrt{\log n}$ ℓ-critical bit positions $\sigma(\ell+1), \dots, \sigma(\ell+\sqrt{\log n})$. To this end, we query in the i-th iteration of the second phase, a bit string y^i which has been created from y' by flipping each non-encoding bit $y'_j, j \in [n]\backslash\mathcal{B}(x, y)$ independently with probability $1/\sqrt{\log n}$. If $f(y^i) = \ell+c-1$ for some $c \in [\sqrt{\log n}]$, we update $X_{\ell+c} \leftarrow X_{\ell+c} \cup \{y^i\}$, the set of all queries with objective value $\ell + c - 1$, and we compute $\mathcal{J}_{\ell+c} := \{j \in [n]\backslash\mathcal{B}(x, y) \mid \forall w \in X_{\ell+c} : w_j = 1-y'_j\}$, the set of candidates for $\sigma(\ell+c)$. We do so until we find $|\mathcal{J}_{\ell+c}| = 1$ for all $c \in [\sqrt{\log n}]$. By Corollary 2 this takes, on average, at most $8e \log^{3/2}(n)/ \log \log n$ queries.

Thus, all we need to do in the third step is to update (x, y) to an $(\ell+\sqrt{\log n})$-encoding pair by exploiting the information gathered in the second phase. For any $c \in [\sqrt{\log n}]$ let us denote the element in $\mathcal{J}_{\ell+c}$ by $j_{\ell+c}$. We go through the positions $\sigma(\ell + 1), \dots, \sigma(\ell + \sqrt{\log n})$ one after the other and either we update $y \leftarrow y \oplus e^n_{j_{\ell+c}}$ (if $f(y) < f(x)$), and we update $x \leftarrow x \oplus e^n_{j_{\ell+c}}$ otherwise. It is easy to verify that after $\sqrt{\log n}$ such steps we have $f(x) \geq \ell + \sqrt{\log n}$ and

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Initialization:
  for $i = 1, \dots, n$ **do** $X_i \leftarrow \emptyset$;
  Sample $x \in \{0,1\}^n$ uniformly at random;
  Query $f(x)$;
  Set $y \leftarrow x \oplus (1, \dots, 1)$;
  Query $f(y)$;
  if $f(y) \geq f(x)$ **then** $(x, y) \leftarrow (y, x)$;
Optimization:
  while $|\mathcal{B}(x,y)| \leq \lfloor \frac{n}{\lceil \sqrt{\log n} \rceil} \rfloor \lceil \sqrt{\log n} \rceil$ **do**
    $\ell \leftarrow |\mathcal{B}(x,y)|$;
    Apply Algorithm 3 with input (x, y) and mutation probability $1/\sqrt{\log n}$ until it outputs a bit string y' with $f(y') \geq \ell + \sqrt{\log n}$;
    Initialize $i \leftarrow 1$;
    while $\exists c \in [\sqrt{\log n}] : |\mathcal{J}_{\ell+c}| > 1$ **do**
      $y^i \leftarrow \texttt{random}(y', x, y, 1/\sqrt{\log n})$;
      Query $f(y^i)$;
      if $f(y^i) \in [\ell, \dots, \ell + \sqrt{\log n} - 1]$ **then**
        Update $X_{f(y^i)+1} \leftarrow X_{f(y^i)+1} \cup \{y^i\}$;
        Update $\mathcal{J}_{f(y^i)}$;
      $i \leftarrow i + 1$;
    for $c = 1, \dots, \sqrt{\log n}$ **do** $\texttt{update}(x, y, y', X_{\ell+c})$;
    if $f(y) > f(x)$ **then** $(x, y) \leftarrow (y, x)$;
  Apply Algorithm 3 with input (x, y) and mutation probability $1/\sqrt{\log n}$ until it queries for the first time a string y' with $f(y') = n$;

Algorithm 4: A $*$-ary unbiased black-box algorithm for maximizing $f \in \textsc{LeadingOnes}_n$. Recall that we have defined $\mathcal{J}_{\ell+c} := \{j \in [n] \backslash \mathcal{B}(x,y) \mid \forall w \in X_{\ell+c} : w_j = 1 - y'_j\}$.

$f(y) \geq \ell + \sqrt{\log n}$. It remains to swap $(x, y) \leftarrow (y, x)$ in case $f(y) > f(x)$ in order to obtain an $(\ell + \sqrt{\log n})$-encoding pair (x, y).

This shows how, given a ℓ-encoding pair (x, y), we find an $(\ell + \sqrt{\log n})$-encoding pair in $O(\log n) + O(\log^{3/2}(n)/\log\log n) + O(\sqrt{\log n}) = O(\log^{3/2}(n)/\log\log n)$ queries.

By definition of Algorithm 4, all bit positions in $\mathcal{B}(x, y)$ remain untouched in all further iterations of the algorithm. Thus, in total, we need to optimize $\lfloor \frac{n}{\lceil \sqrt{\log n} \rceil} \rfloor$ blocks of size $\lceil \sqrt{\log n} \rceil$ until we have a $(\lfloor \frac{n}{\lceil \sqrt{\log n} \rceil} \rfloor \lceil \sqrt{\log n} \rceil)$-encoding pair (x, y). For each block, the expected number of queries needed to fix the corresponding bit positions is $O(\log^{3/2}(n)/\log\log n)$. By linearity of expectation this yields a total expected optimization time of $O(n/\sqrt{\log n})O(\log^{3/2}(n)/\log\log n) = O(n\log(n)/\log\log n)$ for optimizing the first $k := \lfloor \frac{n}{\lceil \sqrt{\log n} \rceil} \rfloor \lceil \sqrt{\log n} \rceil$ bit positions $\sigma(1), \dots, \sigma(k)$.

The remaining $n - k \leq \lfloor \sqrt{\log n} \rfloor$ bit positions can be found by Algorithm 3 in an expected number of $O(\log n)$ queries (Corollary 1). This does not change the asymptotic number of queries needed to identify z.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Input: An ℓ-encoding pair (x, y), a bit string y' with $f(y') \geq \ell + \sqrt{\log n}$, and, a set $X_{\ell+c}$ of samples w with $f(w) = \ell + c - 1$ such that $|\mathcal{J}_{\ell+c}| = |\{j \in [n] \backslash \mathcal{B}(x, y) \mid \forall w \in X_c : w_j = 1 - y'_j\}| = 1$;
if $f(y) \leq f(x)$ **then**
  $y \leftarrow y \oplus e^n_{\mathcal{J}(\ell+c)}$;
  Query $f(y)$;
else
  $x \leftarrow x \oplus e^n_{\mathcal{J}(\ell+c)}$;
  Query $f(x)$;

Algorithm 5: Subroutine $\texttt{update}(x, y, y', X_{\ell+c})$

Putting everything together, we have shown that Algorithm 4 optimizes any function $\textsc{Lo}_{z,\sigma} \in \textsc{LeadingOnes}_n$ in an expected number of $O(n \log(n)/\log\log n)$ queries. It is not difficult to verify that all variation operators are unbiased. We omit the details. □

4 The Unbiased Black-Box Complexity of LeadingOnes$_n$

Next we show how a slight modification of Algorithm 4 yields a 3-ary unbiased black-box algorithm with the same asymptotic expected optimization time.

Theorem 2. *The 3-ary unbiased black-box complexity of* $\textsc{LeadingOnes}_n$ *is* $O(n \log(n)/\log\log n)$.

Proof. Key for this result is the fact that, instead of storing for any $c \in [\sqrt{\log n}]$ the whole query history $X_{\ell+c}$, we need to store only one additional bit string $x^{\ell+c}$ to keep all the information needed to determine $\sigma(\ell + c)$.

Algorithm 6 gives the full algorithm. Here, the bit string $\texttt{update2}(w, y', x^{\ell+c})$ is defined via $\big(\texttt{update2}(w, y', x^{\ell+c})\big)_i = w_i$ if $i \in [n] \backslash \mathcal{B}(y', x^{\ell+c})$ and $\big(\texttt{update2}(w, y', x^{\ell+c})\big)_i = 1 - w_i$ for $i \in \mathcal{B}(y', x^{\ell+c})$.

Note that, throughout the run of the algorithm, the pair $(y', x^{\ell+c})$, or more precisely, the set $\mathcal{B}(y', x^{\ell+c})$ encodes which bit positions j are still possible to equal $\sigma(\ell + c)$. Expressing the latter in the notation used in the proof of Theorem 1, we have in any iteration of the first **while**-loop that for all $i \in [n]$ it holds $y'_i = x^{\ell+c}_i$ if and only if $i \in \mathcal{J}_{\ell+c}$. This can be seen as follows. In the beginning, we only know that $\sigma(\ell + c) \neq \mathcal{B}(x, y)$. Thus, we initialize $x^{\ell+c}_i \leftarrow 1 - y'_i$ if $i \in \mathcal{B}(x, y)$ and $x^{\ell+c}_i \leftarrow y'_i$ for $i \in [n] \backslash \mathcal{B}(x, y)$. In each iteration of the second **while**-loop, we update $x^{\ell+c}_i \leftarrow 1 - y'_i$ if $\sigma(\ell + c) = i$ can no longer hold, i.e., if we have sampled a bit string w with $f(w) = \ell + c - 1$ and $w_i = y'_i$.

It is easily verified that Algorithm 6 certifies Theorem 2. We omit a full proof in this extended abstract. □

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Initialization:
  Sample $x \in \{0,1\}^n$ uniformly at random;
  Query $f(x)$;
  Set $y \leftarrow x \oplus (1,\dots,1)$;
  Query $f(y)$;
  if $f(y) \geq f(x)$ **then** $(x,y) \leftarrow (y,x)$;
Optimization:
  while $|\mathcal{B}(x,y)| \leq \lfloor \frac{n}{\lceil \sqrt{\log n} \rceil} \rfloor \lceil \sqrt{\log n} \rceil$ **do**
    $\ell \leftarrow |\mathcal{B}(x,y)|$;
    Apply Algorithm 3 with input (x,y) and mutation probability $1/\sqrt{\log n}$ until it outputs a bit string y' with $f(y') \geq \ell + \sqrt{\log n}$;
    for $c = 1,\dots,\sqrt{\log n}$ **do**
      for $i = 1,\dots,n$ **do**
        if $i \in \mathcal{B}(x,y)$ **then** $x_i^{\ell+c} \leftarrow 1 - y'_i$ **else** $x_i^{\ell+c} \leftarrow y'_i$;
    while $\exists c \in [\sqrt{\log n}] : |\mathcal{B}(x^{\ell+c}, y')| > 1$ **do**
      $w \leftarrow \texttt{random}(y', x, y, 1/\sqrt{\log n})$;
      Query $f(w)$;
      if $\exists c \in [\sqrt{\log n}] : f(w) = \ell + c - 1$ **then**
        for $i = 1,\dots,n$ **do if** $x_i^{\ell+c} = y'_i = w_i$ **then** $x_i^{\ell+c} \leftarrow 1 - y'_i$;
    for $c = 1,\dots,\sqrt{\log n}$ **do**
      if $f(y) \leq f(x)$ **then** $\texttt{update2}(y, y', x^{\ell+c})$ **else** $\texttt{update2}(x, y', x^{\ell+c})$;
    if $f(y) > f(x)$ **then** $(x,y) \leftarrow (y,x)$;
  Apply Algorithm 3 with input (x,y) and mutation probability $1/\sqrt{\log n}$ until it queries for the first time a string y' with $f(y') = n$;

Algorithm 6: A 3-ary unbiased black-box algorithm for maximizing $f \in \textsc{LeadingOnes}_n$.

5 LeadingOnes$_n$ in the Ranking-Based Models

As discussed above, we introduced two ranking-based versions of the black-box complexity notion in [3]: the *unbiased ranking-based* and the *unrestricted ranking-based black-box complexity*. Instead of querying the absolute fitness values $f(x)$, in the ranking-based model, the algorithms may only query the ranking of y among all previously queried search points, cf. [3] for motivation and formal definitions. We briefly remark the following.

Theorem 3. *The 3-ary unbiased ranking-based black-box complexity of* $\textsc{LeadingOnes}_n$ *is* $O(n \log(n)/\log\log n)$.

This theorem immediately implies that the unrestricted ranking-based black-box complexity of $\textsc{LeadingOnes}_n$ is $O(n \log(n)/\log\log n)$ as well.

Theorem 3 can be proven by combining the Algorithm 6 presented in the proof of Theorem 2 with a sampling strategy as used in Lemma 2. Although the latter is not optimal, it suffices to show that after sampling $O(\log^{3/2}(n)/\log\log n)$ such samples, we can identify the rankings of $f(\ell +$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

$1), \ldots, f(\ell + \sqrt{\log n})$, with probability at least $1 - o(1)$. We do the sampling right after Line 10 of Algorithm 6. After having identified the rankings of $f(\ell+1), \ldots, f(\ell + \sqrt{\log n})$, we can continue as in Algorithm 6.

6 Conclusions

We have shown that there exists a 3-ary unbiased black-box algorithm which optimizes any function $\textsc{Lo}_{z,\sigma} \in \textsc{LeadingOnes}_n$ in an expected number of $O(n \log(n)/\log\log n)$ queries. This establishes a new upper bound on the unrestricted and the 3-ary unbiased black-box complexity of $\textsc{LeadingOnes}_n$.

Our result raises several questions for future research. The obvious one is to close the gap between the currently best lower bound of $\Omega(n)$ (cf. [1]) and our upper bound of $O(n \log(n)/\log\log n)$. Currently, we cannot even prove an $\omega(n)$ lower bound. Secondly, it would also be interesting to know whether the gap between the 2-ary and the 3-ary unbiased black-box model is an artifact of our analysis or whether 3- and higher arity operators are truly more powerful than binary ones.

Acknowledgments. Carola Winzen is a recipient of the Google Europe Fellowship in Randomized Algorithms. This research is supported in part by this Google Fellowship.

References

1. Droste, S., Jansen, T., Wegener, I.: Upper and lower bounds for randomized search heuristics in black-box optimization. Theory of Computing Systems **39** (2006) 525–544
2. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'10), ACM (2010) 1441–1448
3. Doerr, B., Winzen, C.: Towards a Complexity Theory of Randomized Search Heuristics: Ranking-Based Black-Box Complexity. In: Proc. of Computer Science Symposium in Russia (CSR'11), Springer (2011) 15–28
4. Rudolph, G.: Convergence Properties of Evolutionary Algorithms. Kovac (1997)
5. Mühlenbein, H.: How genetic algorithms really work: Mutation and hillclimbing. In: Proc. of Parallel Problem Solving from Nature (PPSN II), Elsevier (1992) 15–26
6. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science **276** (2002) 51–81
7. Doerr, B., Johannsen, D., Kötzing, T., Lehre, P.K., Wagner, M., Winzen, C.: Faster black-box algorithms through higher arity operators. In: Proc. of Foundations of Genetic Algorithms (FOGA'11), ACM (2011) 163–172
8. Auger, A., Doerr, B.: Theory of Randomized Search Heuristics. World Scientific (2011)

Evolutionary Search for Cellular Automata Simulating NAND Gate

Emmanuel Sapin

Laboratory MTG UMR CNRS IDEES 6063,
FRANCE
emmanuelsapin@hotmail.com

Abstract. We aim to construct an automatic system for the discovery of Turing-universal cellular automata. A new step toward this automatic system, presented in this paper, is an automatic method of detection of cellular automata simulating NAND gates. The heuristics of the search for these cellular automata is to pick a existing generator of mobile self-localized patterns of non-resting states and to search for a simulation of AND gate, then to search for an element able to redirect a stream of such patterns.
Results show how simulations of NAND gates can be discovered.

1 Introduction

The theories of complexity are the understanding of how independent agents are interacting in a system to influence each other and the whole system [23]. Surprising computational tasks could result from interactions of independent agents in complex systems as emergence of computation is a hot topic in the science of complexity [26]. A promising environment to study emergent computation is cellular automata [11] which are the simplest mathematical representation of complex systems [5] and an important modelling paradigm in natural sciences and an extremely useful approach in the study of complex systems [15]. They are uniform frameworks in which the simple agents are cells evolving through time on the basis of a local function, called the transition rules [24].

Emerging computation in cellular automata has different forms. Some have studied specific computation like density and synchronization tasks [8, 9, 4, 13, 22] and pattern recognition [28]. While others have considered *Turing-universal automata* [2, 7, 6, 10, 1, 14] i.e. automata encompassing the whole computational power of the class of Turing machines [12]. Some have wondered the question of the frequency of universal cellular automata as Wolfram [25].

In order to find universal automata, we aim to construct an automatic system for the discovery of Turing-universal cellular automata.

Some of the possible first steps towards an automatic system for the discovery of Turing-universal cellular automata have already been made.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

The first 2D 2-state automaton proved Turing-universal was the Game of Life of Conway *et al.* [3]. Its demonstration of universality uses the presence of mobile self-localized patterns of non-resting states [14], called *gliders* and their generators called *glider guns* which, when evolving alone, periodically recover their original shape after emitting a number of gliders. In [18], a second 2D 2-state automaton, called R, has been proven Turing-universal using gliders and glider guns. Considering the ability of glider guns to lead to universality, one of the possible first steps towards an automatic system for the discovery of Turing-universal automata is the search for glider guns.

The search for glider guns was notably explored by Sapin *et al.* who have found thousands glider guns using evolutionary algorithms [20, 21, 17]. In [19, 21], simulations of `AND` gates were searched for a specific glider and the glider guns found in [16] with an even period that emit this glider.

In this paper, we focus on `NAND` gates. The new step presented here is an automatic search for cellular automata able to simulate `NAND` gates thanks to a metaheuristic based on associated evolutionary algorithms.

The rational is to search for cellular automata that are able to simulate an `AND` gate then to search for an automaton that simulates this `AND` gate and an element, called *90-degree reflector* or *reflector*, that allows a stream to be redirected without changing its value whereas glider guns can redirect a stream but complement also its value. So two evolutionary algorithms are successively used. If one of the evolutionary algorithms does not succeed to find what it is searching for then it backtracks to the previous algorithm.

The paper is arranged as follows: Section 2 presents some formalisations and notations while the automatic search for cellular automata simulating `AND` gates. The search for reflectors is presented in Section 3 then The results of the search for `NAND` gate is shown Section 4. To finish, the last section summarizes the presented results and discusses directions for future research.

2 Cellular Automata

In [27], Wolfram studies the space $\mathcal{I}$ of 2-dimensional isotropic cellular automata, with rectangular 8-cell neighbourhoods: if two cells have the same neighbourhood states by rotations and symmetries, then these two cells take the same state at the next generation. There are 512 different rectangular 8-cell neighbourhood states. An automaton of I can be described as shown figure 1 by telling what will become of a cell in the next generation, depending on its subset of isotropic neighbourhood states.

There are 102 subsets of isotropic neighbourhood states, meaning that there are 2^{102} different automata in $\mathcal{I}$.

3 Search for `AND` Gates

This search for `AND` gates is inspired by a previous search described in [15]. A glider gun G is ramdomly chosen among the 20383 guns that were found

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

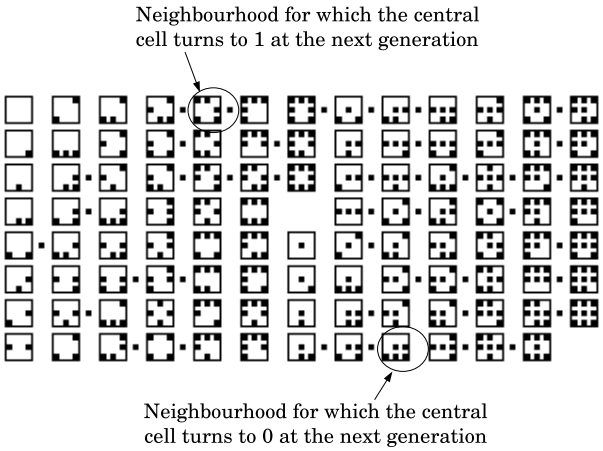


Fig. 1. The squares are the 102 neighbourhood states describing an automaton of $\mathcal{I}$. A black cell on the right of the neighbourhood state indicates a future central cell.

in [16] with a period p that is even. An evolutionary algorithm is used to find an automaton that simulates G and that is able to simulate an AND gate. The following subsections describe the characteristics of this algorithm.

3.1 Individual Structure

An individual is an isotropic automaton that can be described by a cell-state transition table and coded as a bit string of 102 Booleans representing the values of a cell at the next generation for each neighbourhood state.

3.2 Initialization

The 102 bits of an automaton are divided into two subsets. The first subset is the neighbourhood states used by the chosen glider gun and their values are determined by its evolution. The process that determines these neighbourhood states is detailed in [16].

The second subset is initialised at random. The population size is 100 individuals.

3.3 Fitness Function

The choice of the fitness function is one of the main problems in the elaboration of this algorithm.

In order to search for an automaton accepting the glider gun G of period p and able to simulate an AND gate, a pattern is built containing:

– the glider gun G,
– a stream of gliders g emitted by the glider gun G and
– a stream of gliders g that will collide the emitted stream.

The shape of the gun with a stream of two emitted gliders is stored. In order to have the other stream, a symmetry of the second gliders along a diagonal axis is computed and added to the pattern as shown in figure 2 to obtain a pattern, called *start*.

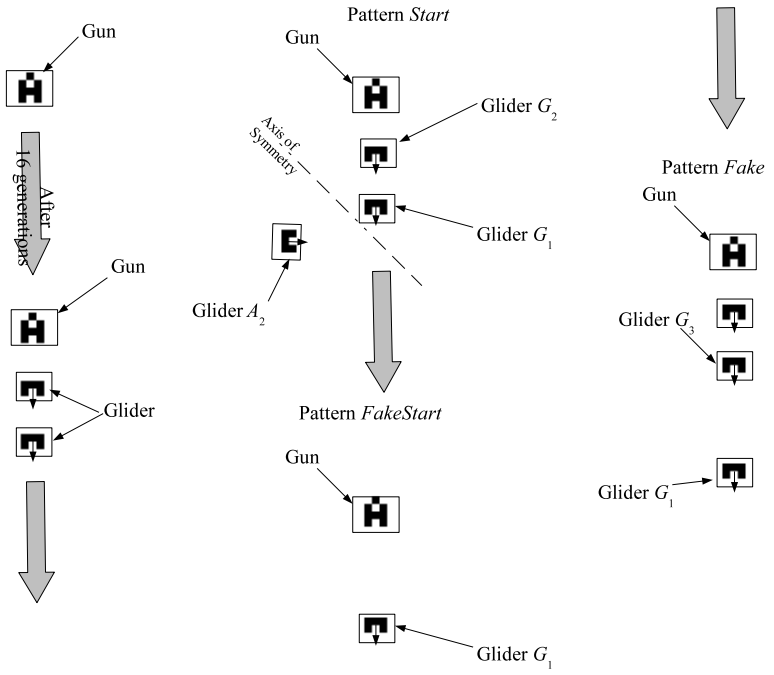


Fig. 2. Construction of the pattern called *start*.

After $2p$ generations, the first glider emitted by the gun, named G_2 and the glider of the stream, named A_2 collide. This collision must not disturb the glider A_1 and must have as result the destruction of these two gliders before the third glider G_3 emitted by the gun arrives. So, after $2p$ generations, the gliders G_2 and A_2 must have disappeared.

To know if this collision has this result, the glider G_2 and A_2 are removed from the pattern *Start* in order to obtain a pattern called $FakeStart$. This pattern evolves during $2p$ to obtain the pattern $Fake$. The pattern *Start* evolves during $2p$ generations and is compared to the pattern $Fake$. If these two patterns are

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

the same then this collision has as a result the destruction of these two gliders before the third glider G_3 emitted by the gun arrives. Then, an AND gate can be simulated.

The value of the fitness function is the number of cells with a state that is different in the patterns $FakeStart$ and $Start$ after $2p$ generations.

3.4 Genetic Operators

The mutation function simply consists of mutating one bit among the second subset of the 102 bits with a single point crossover with a locus situated exactly on the middle of the genotype. A binary tournament selection of size 2 is used.

3.5 Evolution Engine

A non-elitist strategy in which the new population is made of only children is used.

3.6 Stopping Criterion

The stopping criterion is when an AND gate can be simulated.

The value of the fitness function and the generation of the best automaton are stored. If after ten new generations the algorithm has not found a better automaton the algorithm stops.

3.7 Results

For 15 percent of the glider guns, a simulation of an AND gate can be realized with this method. For the other 85 percent, a simulation is not found with this method. Three possibilities exist:

- A simulation does not exist,
- A simulation does exist, but the algorithm did not find it.
- A simulation does exist, but it is not possible to find it with this method.

The figure 3 shows a discovered simulation of an AND gate.

4 Search for 90-degree reflectors

A possible way to simulate a NAND gate with a cellular automaton is to use a reflector. An evolutionary algorithm is used to search for automata with a reflector. The following subsections describe the characteristics of this algorithm.

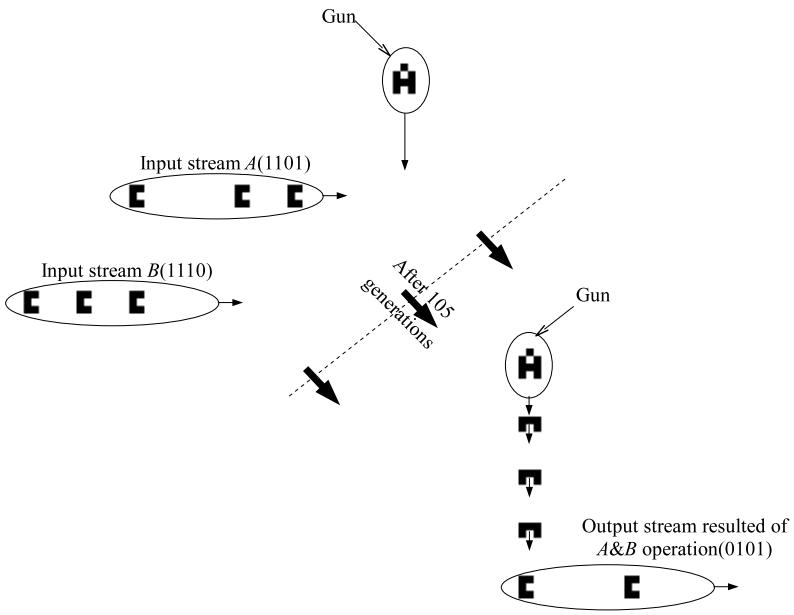


Fig. 3. A discovered simulation of an `AND` gate by the cellular automaton described in figure 1.

4.1 Individual Structure

An individual is defined by a triplet:

- Pattern: random configuration of cells in a 3×3 square,
- position: coordinates of the pattern compared with the glider stream,
- transition rule: As for the previous algorithm, an automaton can be described by a cell-state transition table and then is coded as a bit string of 102 Booleans representing the values of a cell at the next generation for each neighbourhood state. The 102 bits of an automaton are divided into two subsets. The first subset is the neighbourhood states used by the chosen glider gun and the `AND` gate and their values are determined by their evolutions. The process that determines these neighbourhood states is detailed in [16]. The second subset is composed of the other neighbourhood states.

4.2 Initialization

The algorithm manages 50 individuals. For each individual, the pattern and the position are randomly chosen with the same weight for each of the possible choices. The transition rule of each individual is initialised as the one for which the simulation of an `AND` gate has been found.

4.3 Fitness Function

The fitness function's goal is to evaluate the capacity of the pattern of an individual to survive and to redirect a glider stream. The fitness value of an individual is determined by the collision of a stream of gliders and the candidate pattern. It is equal to the number of cell births at the right of the axis of the stream minus the number of cell births below the candidate pattern after $2p$ generations where p is the period of the gun G. So the fitness function could be negative. The cells that are born in a 5×5 square centered on the candidate pattern are not taken into account.

4.4 Genetic Operators

The effects of the mutator can be one of the following possibilities:

- a cell of the pattern is modified,
- the position of the pattern is moved by ±1 for x and y.
- one bit among the second subset of the 102 bits is modified.

A crossover consists in choosing the pattern and the position of the pattern of the first parent and the transition rule of the second parent.

A binary tournament selection of size 2 is used.

4.5 Evolution Engine

It is very close to a $(\mu+\lambda)$ Evolution Strategy, although on a bitstring individual, and therefore without adaptive mutation : the population is made of 20 parents that are selected randomly to create 20 children by mutation only and 10 others by recombination. As in a straight ES+, the 20 best individuals among the 20 parents + 30 children are selected to create the next generation.

4.6 Stopping Criterion

The stopping criterion is when a reflector is found. The value of the fitness function and the generation of the best automaton are stored. If after ten new generations the algorithm has not found a better automaton the algorithm stops.

4.7 Result

One only finds reflectors for 2.05 percent of the automata that simulate an AND gate. From a simple study of the structure of these automata, it appears that their transition rule starts with four 0s and there are always three 0s in the positions 52, 53 and 54 meaning a cell never becomes 1 for the corresponding neighbourhoods. Also, all the automata that have been discovered seem to be in class IV of Wolfram described in [24].

Figure 4 shows one of the reflectors that were discovered.

The automaton of figure 5 simulates this reflector.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

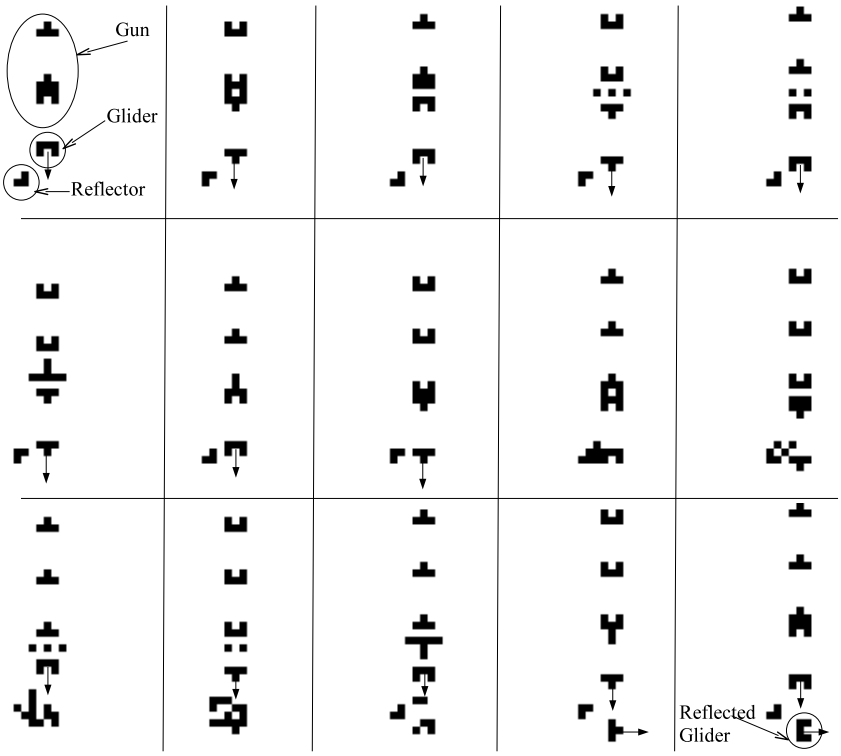


Fig. 4. A reflector generation after generation.

5 NAND Gate

With a simulation of an `AND` gate and a reflector, a `NAND` gate can be built as shown in figure 6 with two input streams A and B.

The automaton of figure 7 simulates this `NAND` gate and the glider gun used by this simulation emits a new glider every 12 generations. This gun creates a glider stream that "crashes" into stream A. If a glider is present in stream A, the two gliders are destroyed by this collision, otherwise the glider emitted by the gun continues its run. The stream resulting from this first collision, at right angles to stream A, is $\overline{A}$. This stream crashes into stream B producing a stream S_1 aligned with the stream B, which is the result of the operation A *and* B. The stream S_1 crashes into the reflector and this collision redirects this stream towards the south without changing its value and this stream crashes into a stream emitted by a last gun and the stream resulting from this last collision is to the same direction of the input streams and is equal to $\overline{AandB}$.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

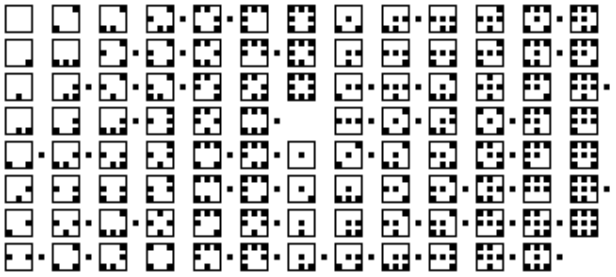


Fig. 5. An automaton of $\mathcal{I}$.

6 Synthesis and perspectives

This paper deals with the emergence of computation in complex systems with local interactions. A contribution to the well established problem of universality in cellular automata is presented. A part of this problem is the simulation of NAND gate. An automatic search for automata able to simulate NAND gates has been conducted. An evolutionary algorithm is tried to find simulation of AND gate then another one to find a 90-degree reflector. A few simulations of NAND gates are discovered.

In order to find more simulations of NAND gates the algorithm that is presented could be improved upon, notable areas of improvement include the fitness function. Another way to find more NAND gate could be to simulate them using another method.

This research is a step of an automatic system for the demonstration of universal automata. A further step will be to simulate duplication and combinaison of logic gates and memories.

Future work could also be to study how gliders appear from a random configuration of cells or a gun and which part of the transition rule makes them move throughout generations. This could lead to a better understanding of the link between the transition rule and the emergence of computation in cellular automata and therefore the emergence of computation in complex systems with simple components.

References

1. A. Adamatzky. Universal dymical computation in multi-dimensional excitable lattices. *International Journal of Theoretical Physics*, 37:3069–3108, 1998.
2. E. R. Banks. *Information and transmission in cellular automata*. PhD thesis, MIT, 1971.
3. E. Berlekamp, J. H. Conway, and R. Guy. Winning ways for your mathematical plays. *Academic Press, New York*, 1982.
4. W. Hordijk, J. P. Crutchfield, and M. Mitchell. Mechanisms of emergent computation in cellular automata. *Parallel Problem Solving from Nature-V, A. E. Eiben, T.*

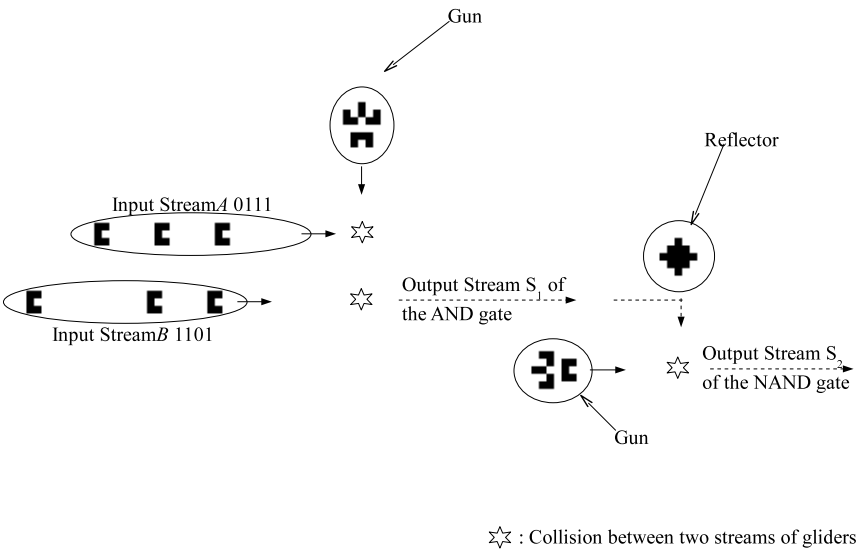


Fig. 6. A `NAND` gate.

Baeck, M. Schoenauer, and H.-P. Schwefel (eds.), Springer-Verlag, 866:344–353, 1998.

5. A. Ilachinski. *Cellular Automata*. World Scientific, 1992.
6. K. Lindgren and M. Nordahl. Universal computation in simple one dimensional cellular automata. *Complex Systems*, 4:299–318, 1990.
7. N. Margolus. Physics-like models of computation. *Physica D*, 10:81–95, 1984.
8. M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.
9. M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos : Evolving cellular automate to perform computations. *Complex systems*, 7:89–130, 1993.
10. K. Morita, Y. Tojima, I. Katsunobo, and T. Ogiro. Universal computing in reversible and number-conserving two-dimensional cellular spaces. *In A. Adamatzky (ed.), Collision-Based Computing, Springer Verlag.*, pages 161–199, 2002.
11. J. Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, Ill., 1966.
12. N. Ollinger. Universalities in cellular automata a (short) survey. *In B. Durand, editor, Symposium on Cellular Automata Journees Automates Cellulaires (JAC08)*, pages pp. 102–118, 2008.
13. N. H. Packard. Adaptation toward the edge of chaos. *In J. A. S. Kelso, A. J. Mandell, M. F. Shlesinger, eds., Dynamic Patterns in Complex Systems*, pages 293–301, 1988.
14. P. Rendell. Turing universality in the game of life. *In Adamatzky, Andrew (ed.), Collision-Based Computing, Springer*, pages pp. 513–539, 2002.

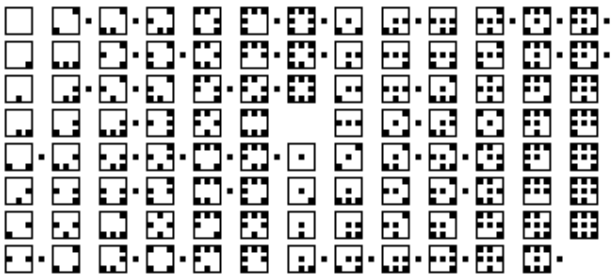


Fig. 7. An automaton of $\mathcal{I}$.

15. E. Sapin, A. Adamatzky, and L. Bull. Genetic approaches to search for computing patterns in cellular automata. *IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE*, 4.
16. E. Sapin, A. Adamatzky, P. Collet, and L. Bull. Stochastic automated search methods in cellular automata: The discovery of tens of thousands glider guns. *Natural Computing.*, 9(3):513–543, 2010.
17. E. Sapin, O. Bailleux, and J. Chabrier. Research of complexity in cellular automata through evolutionary algorightms. *Complex systems*, Volume 17, Issue 3:pp. 231–241, 2007.
18. E. Sapin, O. Bailleux, J.J. Chabrier, and P. Collet. Demonstration of the universality of a new cellular automaton. *IJUC*, 2(3), 2006.
19. E. Sapin and L. Bull. Evolutionary search for cellular automata logic gates with collision based computing. *Complex systems*, Volume 17, Issue 4:pp. 688–693, 2008.
20. E. Sapin, L. Bull, and A. Adamatzky. Searching for glider guns in cellular automata: Exploring evolutionary and other techniques. *EA07. Lecture Notes in Computer Science*, 4926:255–265, 2007.
21. E. Sapin, L. Bull, and A. Adamatzky. A genetic approach to searching for glider guns in cellular automata. *IEEE.Congress on Evolutionary Computation.*, pages 2456 – 2462, 2010.
22. M. Sipper. Evolution of parallel cellular machines. *D. Stauffer, editor, Annual Reviews of Computational Physics*, V. World Scientific:243–285, 1997.
23. M.M. Waldrop. *Complexity: The Emerging Science at the Edge of Chaos. (New York: Simon and Schuster.* Simon and Schuster, New York, NY, 1992.
24. S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.
25. S. Wolfram. Twenty problems in the theory of cellular automata. *Physica Scripta*, pages 170–183, 1985.
26. S. Wolfram. *A New Kind of Science*. Wolfram Media, Inc., Illinois, USA, 2002.
27. S. Wolfram and N.H. Packard. Two-dimensional cellular automata. *Journal of Statistical Physics*, 38:901–946, 1985.
28. D. Wolz and P.B. de Oliveira. Very effective evolutionary techniques for searching cellular automata rule spaces. *Journal of Cellular Automata*, Vol 3, Issue 4:pp. 289–312, 2008.

Propositionalisation as complex aggregates thanks to an evolutionary algorithm

Ogier Maitre[1], Pierre Collet[1], Nicolas Lachiche[1]

FDBT LSIIT - UMR 7005
Pôle API Bd Sébastien Brant
BP 10413 67412 Illkirch CEDEX FRANCE
{ogier.maitre,pierre.collet,nicolas.lachiche}@unistra.fr

Abstract. Relational data mining deals with data in several tables. Propositionalisation is a tranformation of the relational data that brings properties of subsidiary tables to the main table in order to apply usual attribute-value learning algorithms. Existing propositionalisation techniques consider separately each property of subsidiary tables. Indeed considering several properties simultaneously leads to a combinatorial explosion. Evolutionary algorithms offer a solution to search in that combinatorial space. We use an evolutionary algorithm to select attributes and thresholds on those attributes in order to build complex aggregates that are used as input to the standard decision tree learner C4.5. Thus it becomes possible to generate features that existing propositionalisation techniques could not.

1 Introduction

A relational database contains data distributed in several tables. Two typical examples are molecules and customers. A relational database about molecules might consists of a main table describing molecules and subsidiary tables for atoms and bonds between atoms. A relational database about customers might have a main table describing customers and subsidiary tables for products and purchases, to start with. Relational data mining looks for features of the individual, for example of molecules or of customers. Standard attribute-value learning can only make use of columns from the main table. The specific difficulty of relational data mining is to use columns from subsidiary tables having a one-to-many relationship with the main table [18]. Those features are aggregates such as the average amount of the customer's purchases, the number of nitrogen atoms, or the existence of a single bond to a fluorine atom.

Relational data mining techniques come from inductive logic programming mainly. Due to their nature, they generate features as existential agregates such as "the customer bought such article at least once" more frequently than cardinalities such as "the number of times the customer purchased such article". Generally, approaches that generate cardinalities take a single condition into account, eg. "the number of times the customer purchased wine. We want to learn complex aggregates, *ie.* the cardinality of a conjunction of conditions, eg.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

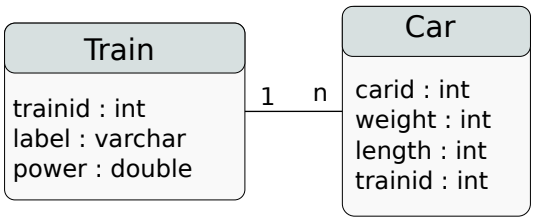


Fig. 1. Example of a relational model

"the number of times a customer bought wine more expensive than 50 euros". Existing works on the generation of complex aggregates propose an ordering of aggregation functions [23], but the conjunctions of conditions are enumerated exhaustively. The combinatorial number of such conjunctions strongly limits that approach.

We propose to use an evolutionary algorithm to explore the space of conjunctions of conditions. The fitness of a set of aggregates is defined as the accuracy of a model learned by a fast attribute-value learner, such as C4.5.

In this article we consider a relational database reduced to a main table and one subsidiary table having a one-to-many relationship. This configuration can easily be generalised to several subsidiary tables directly related to the main table. Such configurations are frequent. It is the case of dimensions around the fact table of a star schema in a datawarehouse. Several relational data mining benchmarks have a similar configuration, for instance the PKDD 1999 financial dataset where a loan is associated to transactions on the corresponding account, or the PKDD 2004 atherosclerosis dataset where a patient has several examinations. It is also the configuration of the real data considered in this article. Configurations involving sequences of one-to-many relationships are less frequent, and they are not considered in this article.

Figure 1 shows an example. We are interested in learning concepts such as the SQL example in Figure 2, where a train belongs to the concept if its power is lower than 800 and there are at least two cars whose weights are greater than 400 and whose length is lower than 1600. Currently, no method is able to handle such learning concepts.

```
select trainid from train where power < 800 and trainid in
   (select trainid from car where weight>=400 and length<=1600
    group by trainid having count(*)>=2)
```

Fig. 2. Example of concept based on complex aggregates

A simple aggregate build a feature based on a single column of a subsidiary table, eg. "the number of times the customer purchased wine", or "the minimal

price of a bottle of wine purchased by the customer". A complex aggregate involves several columns of a subsidiary table.

The next section presents related works in propositionalisation and evolutionary algorithms applied to relational data mining. Then our evolutionary algorithm will be described. Finally, our approach will be evaluated on both artificial and real data.

2 Related works

There are two main approaches in relational data mining. The first consists in using an algorithm that build a model straight from relational data. The only existing algorithm implementing complex aggregates is Tilde, but as we previously explained, those complex aggregates can seldom be generated [23]. The second approach consists in propositionalising, transforming each relational example in an attribute-value example, in order to use an attribute-value learner. Existing propositionalisation techniques generate either simple aggregates, for instance Polka [9], Relaggs [11], or existential aggregates, for instance HiFi [12].

We choose the second approach. Indeed the generation of features is dissociated from the construction of the model. We will use a decision tree learner in order to get a model that the domain expert can understand and analyse. Thus the evolutionary algorithm will focus on the generation of the feature set. The use of a stochastic search avoid the combinatorial explosion of Tilde's exhaustive search. The decision tree learner is efficient enough to be used in the fitness function of the feature set.

There are several works on the use of evolutionary algorithms in relational data mining, in particular in inductive logic programming. [24] proposes as early as 1995 an approach named Genetic Logic Programming, that uses logical trees to learn concepts, but that does not take symbolic constant into account. Most approaches, for instance [7,2,15,19,8,17], use genetic algorithms to speed up the construction of a rule set in first-order predicate logic. We have the same motivation : to speed up the search of a good solution in a combinatorial space. However the solution we look for is a combination of conditions on numeric attributes, that existing approaches cannot represent. Evolutionary algorithms are common in attribute-value learning, to build rule sets or to select attributes. We propose to use them, in a propositionalisation approach, to build features combining conditions on numeric attributes.

Evolutionary algorithms have been used to learn association rules on numeric attributes [16,21,1], but finding frequent itemset is a different learning task than classification.

Braud and Vrain [4] considered using a genetic algorithm for propositionalisation, but the genetic algorithm was used to learn partitions with constraint, but it was not applied to numeric variables. Our approach is a wrapper approach for first-order feature construction, similar to the wrappers for attribute selection [10]. Its specificities are to use an evolutionary algorithm to guide the search, to

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

build first-order features, and to use the average accuracy of an attribute-value learner as the fitness function.

Cardinalisation [13,3] is a related propositionalisation technique that deals with numeric atributes by generating more features than Relaggs, but it is limited to considering simple aggregates, *ie.* a condition on a single numeric attribute. This paper focuses on learning complex aggregates, *ie.* a conjunction of conditions on numeric attributes.

Among all these approaches, only aggregation and cardinalisation are able to handle numerical relational data.

3 Evolutionary algorithm

Evolutionary algorithms are known for their ability to find a not necessarily optimal but satisfying solution to a combinatorial problem in a reasonable time, since they do not explore the whole search space. These algorithms use crossover and mutation operations to optimise a population of candidate solutions to a problem evaluated by a fitness function.

Using a population allow to explore several path in parallel, thus reducing the risk to be stuck in a local minimum. An evolutionary algorithm is a trade-off between the exploitation of solutions and the exploration of the search space. Different parameters allow one to tune those criteria, favoring one behavior rather than the other one, according to the problem being solved and to the behaviour of the algorithm (execution speed, convergence speed, presence of very attractive local minima).

The maximum run time of an evolutionary algorithm can be set, then the algorithm outputs the best solution found so far.

Therefore an evolutionary algorithm is used in this article to find the thresholds and the cardinality, in order to find conjunctions of conditions allowing the classification algorithm to build an accurate model.

Our evolutionary algorithm has been implemented on top of ECJ, a JAVA evolutionary algorithm library. Our algorithm follows the schema of figure 3.

The main steps to deal with our problem, as well as the parameters, are presented in the following sections.

3.1 Representation

A solution of the target problem is represented as a vector of $2nm$ real numbers, where n is the number of attributes in the subsidiary table, and m is the number of generated features. Actually it is a sequence of m features describing conjunctions by the lower and upper bonds over the n attributes. We call *meta-gene* the $2n$ genes generating a feature. A genome is thus made of m meta-gene.

The thresholds are selected in the range of values occuring for the corresponding attribute in the dataset.

A meta-gene describes an interval on every attribute of the subsidiary table. But every target concept does not necessarily imply thresholds on every

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

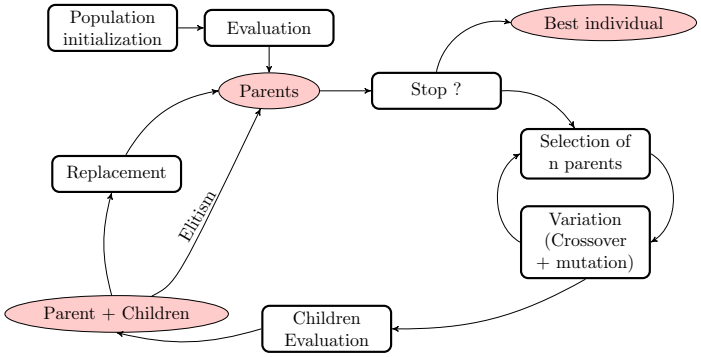


Fig. 3. Our evolutionary algorithm.

attributes. Setting the upper (resp. lower) bond to the maximal (resp. minimal) value occuring in the dataset makes the condition be true for all examples. It is one way to disable a threshold during the learning phase. However selecting the maximal (or minimal) value has a low probability, depending on the number of values of the attribute since we use an uniform distribution. Thus we chose to explicitely represent the able/disable status of a lower/upper bond by two genes (E_{Li}/E_{ui}) allowing the evolutionary algorithm to disable a threshold, or even both bonds on an attribute, in a given feature.

The generated feature is an integer corresponding to the number of objects from the subsidiary table satisfying the encoded conjunction. thus the classification algorithm will take care of the selection of the best cardinality to produce an accurate model. Our feature generalise simple aggregates built by cardinalisation [13].

While the example of figure 2 requires a single meta-gene to represent the target concept, the number of features required to learn the model is not known. So the number m of meta-gene will be chosen arbitrary. This parameter allows to increase the information brought by the genome to the learner, but it increases the run time at the same time.

Figure 4 shows the structure of a genome of m meta-genes of $2n$ genes each.

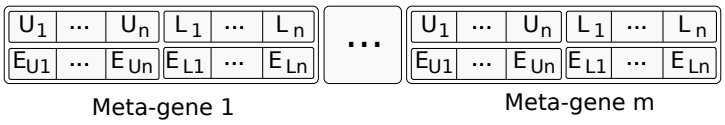


Fig. 4. Example of a genome of m meta-genes.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

3.2 Evaluation

The evaluation of an individual requires to turn its genetic representation into a phenotypic representation. It is done by generating a temporary table containing a copy of all attributes of the main table, plus the features corresponding to the genome. There are m features added to the columns of the main table.

Then the C4.5 decision tree learning algorithm is run with a k-fold cross-validation on the phenotypic representation. The fitness value of the individual is defined as the ratio $i/(i+r)$ where i is the number of examples and r is the number of misclassifications. The cross-validation and a new k-fold partitionning is chosen after each generation in order to avoid overfitting.

One meta-gene might be used or not, by one or several decision trees built during a cross-validation. We define the fitness function as the accuracy resulting from the whole genome, and do not distinguish which parts of the genome contribute or not. The optimisation depends on the learning algorithm and is only evaluated with C4.5 in this algorithm.

3.3 Initialisation

The genes are initialised by drawing randomly, with a uniform distribution, values from the corresponding columns in the dataset. Those values are more likely to make sense than an arbitrary value drawn between the minimal and the maximal values, and automatically reflect the distribution of values of each attribute in the dataset.

The selected thresholds must satisfy the constraint $L_i < U_i$ in a given number of tries, otherwise one of the threshold is disabled to make the interval valid.

3.4 Mutation and Crossover

Two mutation functions have been tried: a uniform mutation and a gaussian mutation. The gaussian mutation is centered on the current value of a gene. The gaussian mutation was selected finally because it leads to a smoother progression of the fitness values of the population. Indeed this mutation deplaces slightly the individual in the seach space compared to a uniform random mutation that move individual far away along some dimensions. Similarly to the initialisation, mutated thresholds must satisfy $L_i < U_i$ or one threshold is disabled.

Several crossover operators have been tried. All of them use two parents to breed one child. The uniform crossover chooses randomly genes from the parents as illustrated on figure 5(a), without paying particular attention to the structure of the genome. Another crossover has been tried, that take the structure of the genome into account as illustrated on figure 5(b). This specific crossover randomly choose entire meta-genes from one parent or the other one. No significant difference has been observed, so the uniform crossover was selected finally because of its genericity and simplicity.

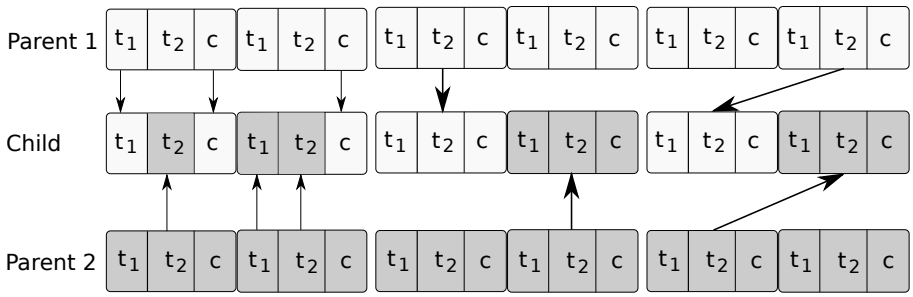
The genome being made of several muta-genes having a similar semantics, we thought of a crossover selecting meta-genes from different positions in the

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Table 1. General parameters of the evolutionary algorithm

Probability of copying a whole meta-gene	0.2
Probability of copying a gene	0.2
Probability of mutating a gene	0.5
Probability of disabling a threshold	0.1
Number of meta-genes per genome (m)	5
Number of folds for cross-validation (k)	5
Kind of elitism	weak
Elitism	1 individual
Replacement stategy	*ES+*
Replacement and selection tournament operator	Tournaments

genomes of the parents as illustrated in figure 5(c). This crossover was not implemented because the fitness function does not depend on the position of the meta-genes in the genome.



(a) Uniform crossover (b) Structured uniform crossover (c) Location independent crossover

Fig. 5. Different crossover operators considered in our study

3.5 General Parameters

Evolutionary algorithms have several parameters [5]. Table 1 summarises the values used in all our tests. The replacement stategy is *ES+* and the weak elitism is set to one individual per generation, in order to keep systematically the best solution found so far. The *ES+* replacement considers children and parents populations to produce the next generation, thus the new population may contain individuals created several generations ago or newly created. This strategy differs from *ES,* where the parents population i smade of new created children only.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

4 Experimental Validation

The evaluation relies on a cross-validation: one fold of the examples is set aside as a test set, and the examples from the remaining folds form the training set used by the evolutionary algorithm to find a set of features such that the decision tree built by C4.5 gets a good accuracy on that training set, using an internal cross-validation.

We consider an artificial dataset first, then a real dataset. We compare our approach to two propositionalisation techniques dealing with numeric attributes. Relaggs [11] uses the usual aggregate functions: the number of values, the sum of values, the minimal and maximal values, the average and the standard deviation. Cardinalisation [13,3] generates more features than Relaggs, each feature is a threshold on a single numeric attribute such that there is a minimal number of such objects. Its expressivity is similar to aggregates introduced in Tilde [23] and it is more efficient on large datasets.

To our knowledge, only two algorithms are able to handle numerical and relational data. Even if these algorithms cannot learn the concepts of the artificial database, we use them in our experiments.

4.1 Artificial Dataset

A set of artificial data is generated in order to check the efficiency of our approach. The schema of this dataset is the same as in figure 1. It contains 1000 examples of trains and approximately 5000 cars. The target correspond to the SQL query of figure 2. The number of positive examples is ≈ 300. All attribute values are integer, between 0 et 2000, drawn randomly with a uniform distribution. For this dataset, the evolutionary algorithm had the following specific parameters: 200 individuals, 20 generations, and a selection pressure of 3.

First the dataset is partitioned in 5 folds to apply cross-validation. For each fold being used as the test set and the four remaining folds being used as the training set, the evolutionary algorithm is run 10 times. The average over the 10 runs of the fitness of the best individual is plotted with respect to the generation on figure 6.

We observe a convergence between the 6th and 12th generation, whatever the initial population is. The average fitness of the population is close to the best individual, denoting that the population probably contains several individuals close to the best one, at least from the fitness value perspective. The optimal solution is found after 9 generations, while the population still presents some phenotypic diversity.

The algorithm produces solutions such as the one presented in figure 7.

Including the average size of the trees produced by C4.5 in the evaluation function leads to simpler solutions, hopefully increasing the generality of the solutions. However both objectives(reducing the size of the tree and reducing the number of errors) are contradictory in many situations, and defining a weighted sum of those objectives is difficult. It has been possible on this artificial dataset to tune the weighting and to significantly increase the number of times the target

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

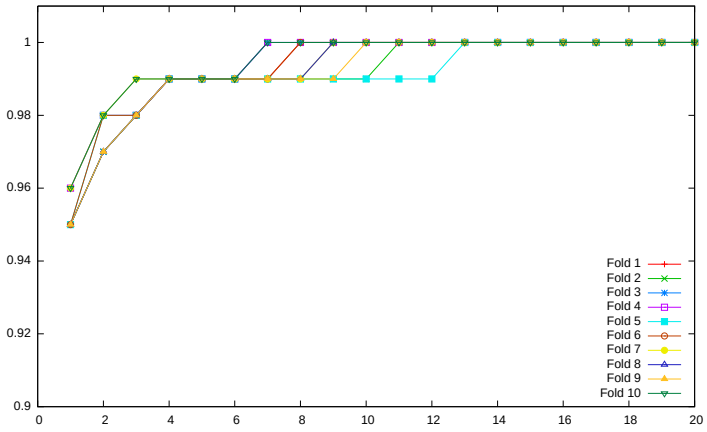


Fig. 6. Convergence of the evolutionary algorithm on the artificial dataset

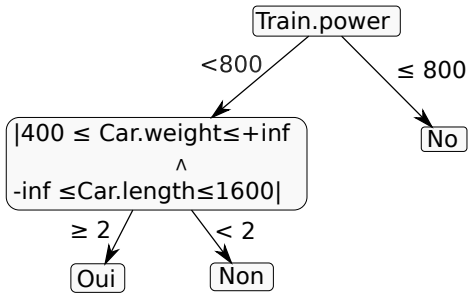


Fig. 7. Example of decision tree learned as a solution of the concept of figure 2

concept was learned, because the size of the target concept is known. But in the case of real problem, the size and structure of the target concept are not known, it makes difficult to tune the weighting. Therefore results presented in this article do not take the size of trees into account, for the real dataset as well as for the articial dataset.

Table 2. Results on the artificial dataset

	Relaggs	Cardinalisation	Cplx. Aggr.
Accuracy	96.8%	92.2%	**99.6%**
Number of nodes in the decision tree	31	7	**5**

As expected, complex aggregates allows to learn a better model than previous approaches, cf. table 2. Moreover, those aggregates allow to model exactly the original target. The expressivity of the other approaches is less fitted to this kind

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

of hypotheses, thus they are forced to combine more conditions to get accurate but larger models.

4.2 Geographical datasets

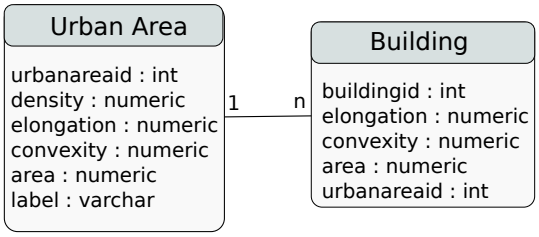


Fig. 8. Relational schema of our geographical datasets

We consider several geographical datasets. These datasets concern urban areas described by their own attributes (density of buildings, elongation, convexity, and area), plus their classes among 6 possibilities (individual houses, housing blocks, mixed housing, high density specialised area, low density specialised area, and mixed area). Those urban areas contain buildings also described by their geometrical properties (elongation, convexity and area). Figure 8 shows their relational schema. Each dataset describes a distinct geographical zone.

Table 3. Accuracies on the geographical datasets

	Relaggs	Cardinalisation	Cplx. Aggr.
Zone1	**93.2%**	91.9%	91.5%
Zone2	93.1%	93.1%	**94.5%**
Zone3	91.0%	**94.3%**	91.7%

Here a 10-fold cross-validation is used. The accuracies are reported in table 3. Each of the propositionalisation technique gets the best accuracy on one dataset. It shows that no language bias is the best for all problems, and conversely that each propositionalisation techniques offers an expressivity that is useful, *ie.* better than the others, on some problem. Therefore our complex aggregates provide an additional and complementary propositionalisation technique.

5 Conclusion

We presented a propositionalisation technique that can generate complex aggregates, despite the combinatorial number of those complex aggregates, thanks to

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

an evolutionary algorithm optimising the accuracy of an attribute-value learner. Its efficiency has been checked on an artificial dataset, showing that our approach allow to learn more expressive, therefore smaller and more accurate models than other existing techniques, still using a reasonable amount of time and memory. The comparison on real datasets shows that our approach indeed offers a different expressivity than existing techniques, and gets a better accuracy on some problem. Thus it complements the diversity of the available propositionalisation techniques.

Two contradictory objectives (reducing the size and the number of errors) were identified. The use of a multi-objective optimisation algorithm such as [22,6] is a perspective for future work in order to balance both objectives in order to get more accurate and more comprehensible models. Another perspective is to replace the sequence of m meta-genes by a parisian approach [14]. Some meta-genes would be gathered during the evaluation to be used as the input of C4.5. Then each meta-gene would get its own mark, proportional to its selection in the decision trees built by C4.5 during the cross-validation. Moreover other metrics such as F-measure could be considered to evaluate an individual, and other classification algorithms than C4.5, eg. a naive Bayesian learner could be used.

References

1. Alcalá-Fdez, J., Papè, N.F., Bonarini, A., Herrera, F.: Analysis of the effectiveness of the genetic algorithms based on extraction of association rules. Fundam. Inform. 98(1), 1–14 (2010)
2. Anglano, C., Giordana, A., Bello, G.L., Saitta, L.: A network genetic algorithm for concept learning. In: Bäck, T. (ed.) ICGA. pp. 434–441. Morgan Kaufmann (1997)
3. Braud, A., Lachiche, N.: Propositionaliser des attributs numériques sans les discrétiser, ni les agréger. In: Khenchaf, A., Poncelet, P. (eds.) EGC. Revue des Nouvelles Technologies de l'Information, vol. RNTI-E-20, pp. 437–442. Hermann-Éditions (2011)
4. Braud, A., Vrain, C.: A genetic algorithm for propositionalization. In: Rouveirol and Sebag [20], pp. 27–40
5. Collet, P.: Soft Computing Applications for Database Technologies: Techniques and Issues, chap. A Quick Presentation of Evolutionary Computation, pp. 22–38. IGI Publishing, Hershey, PA, USA (2010), http://lsiit-cnrs.unistra.fr/Publications/2010/Col10, iSBN 9781605668147
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (April 2002), http://dx.doi.org/10.1109/4235.996017
7. Giordana, A., Neri, F.: Search-intensive concept induction. Evolutionary Computation 3, 375–416 (1996)
8. Ishida, C.Y., Pozo, A.: Grammatically based genetic programming for mining relational databases. In: SCCC. pp. 86–. IEEE Computer Society (2003)
9. Knobbe, A.J., de Haas, M., Siebes, A.: Propositionalisation and aggregates. In: Raedt, L.D., Siebes, A. (eds.) PKDD. Lecture Notes in Computer Science, vol. 2168, pp. 277–288. Springer (2001)

10. Kohavi, R., John, G.: Wrappers for feature selection. Artificial Intelligence 97(1-2), 273–324 (1997)
11. Krogel, M.A., Wrobel, S.: Transformation-based learning using multirelational aggregation. In: Rouveirol and Sebag [20], pp. 142–155
12. Kuzelka, O., Zelezný, F.: Block-wise construction of acyclic relational features with monotone irreducibility and relevancy properties. In: Danyluk, A.P., Bottou, L., Littman, M.L. (eds.) ICML. ACM, vol. 382, p. 72 (2009)
13. Lesbegueries, J., Braud, A., Lachiche, N.: A propositionalisation that preserves more continuous attribute domains. In: Poster session ILP'09 (2009)
14. Louchet, J.: Using an individual evolution strategy for stereovision. Genetic Programming and Evolvable Machines 2, 101–109 (2001), http://dx.doi.org/10.1023/A:1011544128842, 10.1023/A:1011544128842
15. Martin, L., Moal, F., Vrain, C.: A relational data mining tool based on genetic programming. In: Zytkow, J.M., Quafafou, M. (eds.) PKDD. Lecture Notes in Computer Science, vol. 1510, pp. 130–138. Springer (1998)
16. Mata, J., Alvarez, J.L., C., R.J.: An evolutionary algorithm to discover numeric association rules. In: Proceedings of the 2002 ACM symposium on Applied computing. pp. 590–594. SAC '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/508791.508905
17. Muggleton, S., Tamaddoni-Nezhad, A.: Qg/ga: a stochastic search for progol. Machine Learning 70(2-3), 121–133 (2008)
18. Raedt, L.D.: Attribute-value learning versus inductive logic programming: The missing links (extended abstract). In: Page, D. (ed.) ILP. Lecture Notes in Computer Science, vol. 1446, pp. 1–8. Springer (1998)
19. Reiser, P.G.K., Riddle, P.J.: Scaling up inductive logic programming: An evolutionary wrapper approach. Appl. Intell. 15(3), 181–197 (2001)
20. Rouveirol, C., Sebag, M. (eds.): Inductive Logic Programming, 11th International Conference, ILP 2001, Strasbourg, France, September 9-11, 2001, Proceedings, Lecture Notes in Computer Science, vol. 2157. Springer (2001)
21. Salleb-Aouissi, A., Vrain, C., Nortet, C.: Quantminer: A genetic algorithm for mining quantitative association rules. In: Veloso, M.M. (ed.) IJCAI. pp. 1035–1040 (2007)
22. Sharma, D., Collet, P.: An archived-based stochastic ranking evolutionary algorithm (asrea) for multi-objective optimization. In: GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation. pp. 479–486. ACM, New York, NY, USA (2010)
23. Vens, C., Ramon, J., Blockeel, H.: Refining aggregate conditions in relational learning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD. Lecture Notes in Computer Science, vol. 4213, pp. 383–394. Springer (2006)
24. Wong, M.L., Leung, K.S.: Inducing logic programs with genetic algorithms: The genetic logic programming system. IEEE Expert 10(5), 68–76 (1995)

Brain MRI segmentation based on shortest path and biased survival exponential entropy

Salma Hajjem[2], Amir Nakib[1], Hamouche Oulhadj[1] and Patrick Siarry[1] *

[1] Laboratoire Images, Signaux et Systèmes Intelligents, LiSSi (EA 3956), 61 avenue du Général de Gaulle, 94010 Créteil, France.
[2] Ecole Nationale des Sciences de l'Informatique de La Manouba, Tunisia
nakib@u-pec.fr

Abstract. In this paper, we propose a new formulation of the image thresholding problem as a shortest path problem. Then, we propose a new ant colony optimization based algorithm to solve it. Here, to measure the cost of every path, a new criterion is proposed, named the biased survival exponential entropy, that can be seen as a thresholding criterion. This criterion considers the cumulative distribution of the gray level information and takes into account spatial quality of the segmentation result. It is well known that the ant colony optimization algorithms are slow; in order to solve this problem we enhanced the classical ant colony system by hybridizing it with a local search algorithm. The experiments on segmentation of MRI images proved that the proposed method can achieve a satisfactory segmentation with a low computation cost.

Keywords: segmentation, ant colony optimization, survival function, entropy, shortest path.

1 Introduction

The thresholding is a segmentation technique based on the assumption that the object can be distinguished by its level of gray. The optimal threshold allows to separate the objects from each other or various objects from the background. Segmenting an image in N classes consists in looking for N-1 thresholds. The optimal segmentation by a thresholding technique requires the localization of segmentation thresholds on the image histogram. Several methods of image segmentation using this technique of thresholding were proposed in the literature [7]. These methods can be classified in two classes: first, the parametric methods, which are based on the assumption that the densities of gray levels probabilities of various classes are Gaussian, the optimal thresholds take place then in their intersection. The non-parametric methods are based on the optimization of one or more criteria. In this paper, we consider only this last class of methods. Several non parametric methods were proposed as: Otsu method [8], that is based on the maximization of the interclass variance, the Kapur method [3], that is based on

* Thanks to Pr. P. Decq, from CHU Mondor Créteil (France) for his help.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

the maximization of the Shannon entropy. It is well known that the entropy of Shannon (SE) is not defined for the null probabilities. One solution consists in defining an entropy based on exponential profit on information [9]. Recently two other entropies were proposed in the literature: the entropy of Tsallis (TE) [11], and the Renyi's entropy [12], the exponential entropy [7]. In order to take into account the spatial distribution of the pixels, a two dimensional version of the Renyi's entropy was proposed [10].
Several metaheuristics were applied in the problem of image segmentation by using the technique of thresholding. Many authors used genetic algorithms [4], particle swarm optimization [5], ant colony optimization [13], simulating annealing [6] and Tabu search [2] to solve multilevel thresholding problems. However, their comparison (presented in [2]) does not provide a good answer to the choice of the best metaheuristic, because the quality of the segmentation depends also on the segmentation criterion. The paper is outlined as follows. In Section 2, the proposed segmentation criterion BSEE is described. Section 3 introduces the concept of ant colony system. Section 4 presents the proposed segmentation algorithm including the enhanced ant colony system [1]. Section 5 provides the experimental results and discussion. Finally, Section 6 concludes this paper.

2 Biased Survival exponential entropy

In this section, the information measure of random variables, called Survival Exponential Entropy (SEE) [14], is presented. Here, we show how using a biased version of this measure can allow to segment images at hand.

Let $X = (X_1 \dots X_m)$ be a random vector in $\Re^m$. We denoted by $|X|$, the random vector with components $|X_1|, ..., |X_m|$ and use the notation $|X| > x$ to mean that $|X_i| > x_i$ for $x_i \geq 0$, $i = 1, ..., m$.

The multivariate survival function $\overline{F}_{|X|}(x)$ of the random vector $|X|$ with an absolutely continuous distribution with probability density function $f(x)$ is defined by:

$$\overline{F}_{|X|}(x) = P\left(|X_1| > x_1, ..., \ |X_m| > x_m\right) \tag{1}$$

where $x \in R_+^m$. For a discrete distribution, the survival cumulative distribution function can be expressed as:

$$\bar{F}(x) = 1 - \sum_{i=0}^{x} p(i) \tag{2}$$

For the random vector X in $\Re_+^m$, the survival exponential entropy of order α is defined by:

$$M_\alpha(X) = \left(\int_{R_+^m} \overline{F}_{|X|}^{\alpha}(x) \, dx \right)^{1/1-\alpha} \tag{3}$$

for $\alpha > 0$ and $\alpha \neq 0$, where m denotes the number of dimensions for X. The SEE uses the density function with the cumulative distribution that is more

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

regular than the density function. The SEE has several advantages over the Shannon entropy and differential entropy (extension of Shannon entropy to the continuous case): it is consistently defined in both the continuous and discrete domains, it is always nonnegative and it is easy to compute from sample data. Whereas the Shannon entropy is based on the density of the random variable, that may not exist in some cases and, when it exists, must be estimated. For the application in image segmentation, we use the well known property of measures of entropy, which states that the entropy of the joint distribution is equal to the sum of entropies of the marginal distributions under the assumption of independence.Thus the proposed biased survival exponential entropies (BSEE) associated with different image classes' distributions are defined below:

– the BSEE of the class m-1 can be computed through:

$$BM_{\alpha}^{(m-1)} = \omega_{(m-1)} \cdot \left(\sum_{j=t_{n-1}}^{t_n-1} \left(\overline{F}(i,j)\right)^{\alpha} \right)^{1/(1-\alpha)} \tag{4}$$

– the BSEE of the class m can be computed through:

$$BM_{\alpha}^{(m)} = \omega_{(m)} \cdot \left(\sum_{j=t_n}^{t_{n+1}-1} \left(\overline{F}(i,j)\right)^{\alpha} \right)^{1/(1-\alpha)} \tag{5}$$

where $\omega_{(m)} = log(N_i)$, so that the first term $\left(\omega_{(m)} \cdot M_{\alpha}^{(m)}\right)$ is high for non-homogeneous regions (typically, the large ones), while N_i denotes the number of pixels in the class i, and $\omega_{(1)} = 1$. For the convenience of illustration, two threshold values $t_0 = 1$ and $t_N = 255$ were added, where $t_0 < ... < t_N$. Then the total BSEE is:

$$BM_{\alpha}^{(T)} = \sum_{i=0}^{N-1} BM_{\alpha}^{(i+1)} \tag{6}$$

According to the minimum survival exponential entropy principle that corresponds to the maximum Shannon entropy principle, the optimal vector $(t_0^* < ... < t_N^*)$ should meet:

$$BM_{\alpha}^{(T)} = \text{ArgMin}\left\{BM_{\alpha}^{(T)}\right\} \tag{7}$$

where $1 < t_1 < ... < 255$. In the case of one threshold ($N = 2$), the computational complexity for determining the optimal vector t^* is $O(L^2)$ where L is the total number of gray level. However, it is too time-consuming in the case of multilevel thresholding. For the n-thresholding problem, it requires $O(L^{n+2})$. In this paper, we use an enhanced version of ACS algorithm for solving the problem formulated in (7) efficiently.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

3 Enhanced ACS algorithm (EACS)

The original Ant Colony System (ACS) [1] was applied in different real world applications. Since the formulation of the problem as a graph optimization problem, ACS can be applied by associating two measures to each arc: the closeness $\tau(i,j)$, and the pheromone trail $\eta(i,j)$. ACS uses a mechanism based on three main operations: (1) the state transition rule provides a direct way to balance between exploration of new edges and exploitation of *a priori* and accumulated knowledge about the problem. (2) The global updating rule is applied only to edges that belong to the best ant tour. (3) While ants construct a solution, a local pheromone updating rule (local updating rule, for short) is applied.

– *ACS state transition rule*

In ACS the state transition rule is as follows: an ant positioned at the node r chooses the city v to move to by applying the rule given below.

$$v = \begin{cases} ArgMax_{u \in J_k(r)} \left\{ [\tau(r,u)] \cdot [\eta(r,u)]^b \right\} & q \leq q_0 \\ s & Otherwise \end{cases} \tag{8}$$

where q is a random number uniformly distributed in $[0,1]$, q_0 is a parameter $0 \leq q_0 \leq 1$. It determines the relative importance of exploitation versus exploration: when an ant in node r has to choose a node s to move to, it samples a random number $0 \leq q \leq 1$. b is a parameter that determines the relative importance of pheromones versus distance. s is a random variable selected according to the probability distribution. It is given by:

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)][\eta(r,s)]^b}{\sum_{u \in J_k(r)} [\tau(r,s)][\eta(r,s)]^b} & s \in J_k(r) \\ 0 & Otherwise \end{cases} \tag{9}$$

$J_k(r)$ is the set of the neighborhood solutions to the current ant r. The state transition rule resulting from equations (8) and (9) is called *pseudo-random proportional* rule. This state transition rule, as with the previous *random-proportional* rule, favors transitions towards nodes connected by short edges and with a large amount of pheromone. If $q \leq q0$, then the best edge according to (8) is chosen (exploitation), otherwise an edge is chosen according to (9) (biased exploration).

– *ACS global updating rule*

In ACS, only the globally best ant is allowed to deposit pheromone. The pheromone level is updated by applying the following global updating rule:

$$\tau(r,s) = (1-\mu) \cdot \tau(r,s) + \mu \cdot \Delta\tau(r,s) \tag{10}$$

where

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1} & if(r,s) \in gb \\ 0 & Otherwise \end{cases} \tag{11}$$

$0 < \mu < 1$ is a pheromone decay parameter, and L_{gb} is the length of the globally best tour (gb) from the beginning of the trial.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

– *ACS local updating rule*

After having crossed an edge (i, j) during the tour construction, the following local update rule is applied:

$$\tau(i,j) = (1-\xi) + \xi \cdot \tau_0 \tag{12}$$

where $0 < \xi < 1$, and τ_0 are two parameters. The value for τ_0 is suited to be the same as the initial value for the pheromone trails. In order to apply the EACS to solve the segmentation problem, it must be reformulated as a shortest path problem. Then, we define a stochastic rule of local choice of transition to carry out the good path research in this graph. It is also needed to fix the strategy of the deposit and the use of the different traces of pheromone. In our case, the graph is related and balanced. The nodes represent the various possible thresholds (255 thresholds). The weights will be placed on the arcs. The weight on an edge represents the value of the BSEE for the thresholds T bound by this edge (see Figure 1). Then, we define:

– A set of components $C = T$.
– The whole $L = T \times T$, either a total, simple and non controlled interconnection between the objects.
– The function of transition cost $J(i, j) = p_{ij}$.
– The set of the solutions, that correspond to all possible thresholds that allow to segment the image, without violation of the sort constraint.

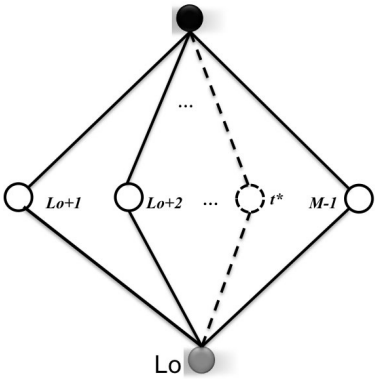


Fig. 1. Illustration of the problem formulation where M is the maximum gray-level in the image, Lo is the lowest gray-level and t^* is an example of an optimal threshold. The dashed lines correspond to the shortest path.

In order to solve the image segmentation problem, we specify the behavior of the set of the colony guided by the ACS to minimize the BSEE. Initially, N

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

ants are placed randomly on N nodes of the construction graph. Thus, each ant adds, in an incremental way, the threshold that minimizes the total BSEE. Then, every ant moves to its neighborhood using a stochastic transition rule. This rule depends on the quantity of pheromone and heuristic information locally valid. In other terms, an ant having a constructed solution s chooses to move toward a node j according to the rule (8). As we consider that the timeliness of a solution depends on all the solutions (segmentation thresholds) already found, the heuristic information and the pheromone represent a total relation between the current solution and all the found solutions from the start S (all visited nodes).

$$\tau_S(j) = \sum\nolimits_{i \in S} \tau(i,j) \tag{13}$$

The heuristic information corresponds to the value of the BSEE. Then, the candidate is more desirable with the decrease of its BSEE. The heuristic information is defined by the following equation:

$$\mu_S(j) = BSEE_S + BSEE(k,j) \tag{14}$$

where $BSEE_S$ is the value of the total biased survival exponential entropy of the solution already built, and $BSEE(k,j)$ is the value of the entropy between the threshold candidate j and the last threshold k taken by S.

4 Proposed segmentation method

The whole proposed segmentation algorithm (Algorithm 1) consists in applying the proposed EACS algorithm hybridized with Tabu Search (TS), where the size of the tabu list (TL) is $3 \times N$, and the neighbor size is equal to 10, for each node selected by an ant. The different steps of the proposed segmentation algorithm are presented in Algorithm 1. Indeed, the EACS principle consists in applying a local search based on TS for 30% of the ant colony. The aim of this local search is to accelerate the convergence of the algorithm and to avoid the local optima. Indeed, the tabu search is used for the intensification procedure as a local search and ACS is used with a specific fitting that allow a diversification.

5 Results and discussion

In this section, we first discuss the choice of the BSEE order (α), based on synthetic images. Then, the obtained results on some MR images are presented. Here, are presented only the results in the case of four and five classes' segmentation. The results were compared to those provided by other competing methods recently published. In order to analyze the variation of the performance of the proposed method with the variations of the BSEE order, we segment a synthetic image with different degrees of noise measured by PSNR (figure 2). Then, to measure the performance of all the methods, we use the misclassification error (ME) criterion. ME is defined in terms of correlation of the images with human

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Algorithm 1 The EACS algorithm

Require: image histogram, number of classes (N)
for each ant **do**
  Choose the first threshold randomly
  for $i = 2$ to $N - 1$ **do**
    Build a list of the candidates thresholds
    Choose a threshold that minimizes the BSEE
    $r = rand$
    if $r < 0.3$ **then**
      Apply a Tabu search in the neighborhood of the current threshold.
    end if
    Local update of the pheromone
  end for
  Global update of the pheromone
end for
return The best ant

observation. It corresponds to the ratio of background pixels wrongly assigned to the foreground, and vice versa. ME can be simply expressed as:

$$ME = \left(1 - \frac{|B_O \cap B_T| + |F_O \cap F_T|}{(|B_O| + |F_O|)}\right) \cdot 100 \tag{15}$$

where background and foreground are denoted by B_O and F_O for the original image, and by B_T and F_T for the thresholded image, respectively. In the best case of ideal thresholding ME is equal to 0% and in the worst case ME value is 100%. Figure 2(e) illustrates the variation of ME (percent) when the value of BSEE order goes from 0.1 to 30. As it can be seen, the quality of segmentation results depends on the value of α in the case of an order in the interval $]0, 1[$. However, one can see that beyond 1, the segmentation quality is stable. Consequently, we can conclude that for an order upper than 10 the quality of the segmentation is good enough. It was predictable because of the exponential shape of the BSEE. Then, in the case of a BSEE order less than 1, in order to find the optimal order for the segmentation process, the well known uniformity criterion is used:

$$U = 1 - 2p \sum_{j=0}^{p} \sum_{j \in C_j} (f_i - \mu_i)^2 \Big/ N \cdot (f_{\max} - f_{\min})^2 \tag{16}$$

where p is the number of thresholds, C_j the j^{th} class, N the image size, f_i the gray level of pixel i, μ_i the mean gray level of pixels in the j^{th} class, f_{max} and f_{min} the maximum and the minimum gray levels of pixels in the image, respectively. When U is close to 1, the uniformity is very good and vice versa.
In our experiments, we fixed the value of α equal to 100, in order to have stable performances. The different values of the parameters of EACS algorithm were fixed empirically and are presented in Table 1.

The different images were acquired using Siemens Avento 1.5T, the resolution was 256×256 and the field of view equal to 250. Two different pathologic MRI

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

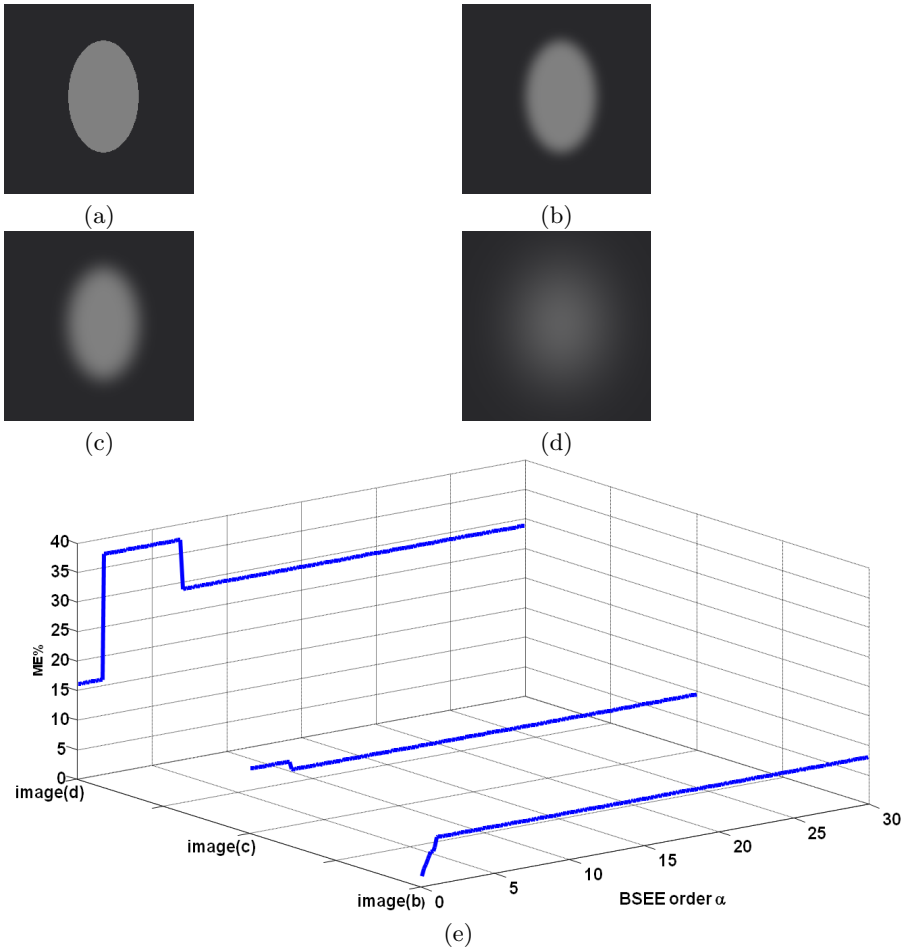


Fig. 2. Synthetic images. (a) Original image, (b) Noised image (PSNR = 20.90dB), (c) Noised image (PSNR=27.20dB), (d) Noised image (PSNR = 29.01dB), (e) Variation of ME with BSEE order α.

Parameters	Value
Population size	100
Number of Cycles	$2 \times N$
μ	0.5
b	2
T_L	$3 \times N$
S_n	10

Table 1. Suited values of EACS parameters.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

cases are presented in figure 3, with their segmentation results. Indeed, in this paper the pathology studied the hydrocephalus, that consists in atrophy in the ventricular system. Then, our goal is to calculate the volume of the cerebrospinal fluid (CSF) inside only the ventricle. The obtained result through the application of segmentation algorithm when $N = 2$, shows that the use of the BSEE allows to segment satisfactorily the brain MR images.

In order to compare the performance of the proposed method to that of the other competing methods, we present in figures 3(b), (c) and (d) the segmentation results provided by SE, and RE, respectively. Looking to these results, one can remark that the provided results by the proposed method are more accurate than those of the SE and the RE methods. To illustrate the efficiency of EACS, we present another example of the segmentation results in the case of $N = 4$ and $N = 5$ classes in figures 4(a) to (c), respectively. One can remark that the obtained segmentations are good, visually, because the different classes are homogeneous.

6 Conclusion

In this paper, an enhanced ACS algorithm namely EACS, is used for MR image segmentation. We also proposed a new segmentation criterion based on survival exponential entropy that allows to avoid the classical problems related to the use of entropy based thresholding methods. Experimental results show that the proposed optimization algorithm EACS produces satisfactory results, indicating that it can be used for brain MR image segmentation in multi-thresholding, due to its computational efficiency. In works under progress, we intend to use other criteria in order to take into account more information to segment the images.

References

1. Dorigo, M., M., G.L.: Ant colony system : A cooperative learning approach to the traveling salesman problem. IEEE Trans. on Evo. Comput. 6(4), 317–365 (1997)
2. Hammouche, K., Diaf, M., Siarry, P.: A comparative study of various meta-heuristic techniques applied to the multilevel thresholding problem. Engineering Applications of Artificial Intelligence 23, 676–688 (2010)
3. Kapur, J., Sahoo, P., Wong, A.: A new method for gray-level picture thresholding using the entropy of the histogram. Computer Vision, Graphics and Image Processing 29, 273–285 (1985)
4. Li, C., Bao, P., Shi, Z.: The strongest schema learning ga and its application to multilevel thresholding. Image and Vision Computing 26, 716–724 (2008)
5. Nakib, A., Oulhadj, H., Siarry, P.: Fast MRI segmentation based on Two dimensional survival exponential entropy and particle swarm optimization. In: In proc. of the 29th Annual Int. Conf. of the IEEE EMBS. pp. 5563–5565. Lyon (France) (August 22-26 2007)
6. Nakib, A., Oulhadj, H., Siarry, P.: Image histogram thresholding based on multiobjective optimization. Signal processing 87, 2516–2534 (2007)

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

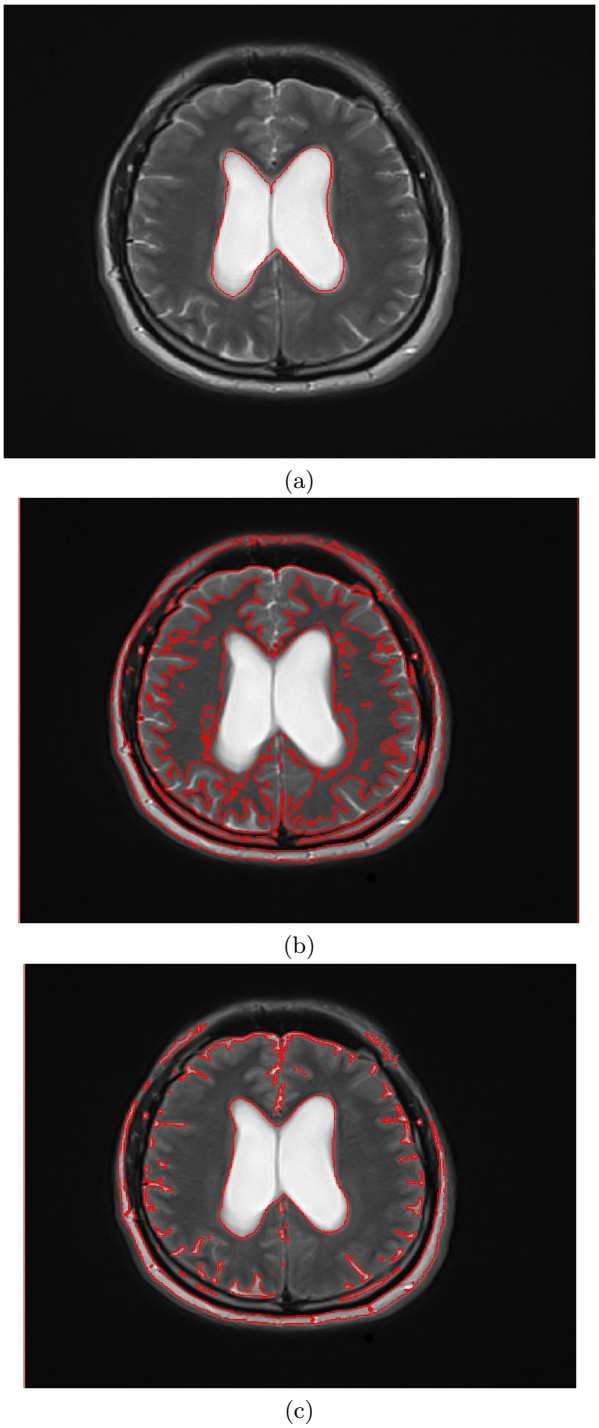
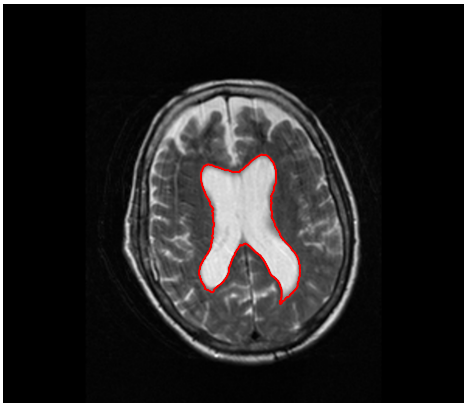
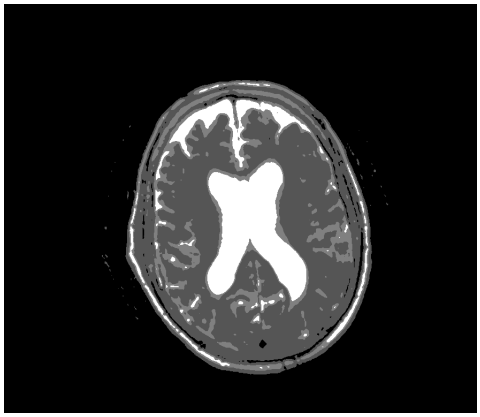


Fig. 3. Illustration of MRI segmentation results using :(a) BSEE, (b) TE, (c) SE.

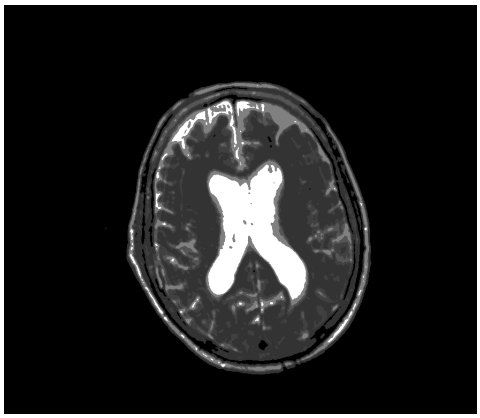
Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



(a)



(b)



(c)

Fig. 4. Illustration of a multilevel segmentation using BSEE in the case of (a) 2 classes, (b) 4 classes, (c) 5 classes.

7. Nakib, A., Oulhadj, H., Siarry, P.: A thresholding method based on Two dimensional fractional differentiation. Image Vision and Computing 27(9), 1343–1357 (2009)
8. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Trans. Sys., Man., Cyber. 9, 62–66 (1979)
9. Pal, N.K., Pal, S.K.: Entropy: A new definition and its applications. IEEE Trans. Syst. Man. Cybern. 21, 1260–1270 (1991)
10. Prasanna, P.K., Sahoo, K.P., Arora, G.: A thresholding method based on Two-dimensional Renyi's entropy. Pattern recognition 37, 1149–1161 (2004)
11. Sahoo, K.P., Arora, G.: Image thresholding using Two-dimensional Tsallis-Havrda Charvat entropy. Pattern recognition letters 37, 1149–1161 (2006)
12. Shitong, W., Chung, F.L.: Note on the equivalence relationship between Renyi entropy based and Tsallis entropy based image thresholding. Pattern recognition letters 26, 2309–2312 (2005)
13. Wenbing, T., H., J., Liman, L.: Object segmentation using ant colony optimization algorithm and fuzzy entropy. Pattern recognition letters 28(7), 788–796 (2007)
14. Zografos, K., Nadarajah, S.: Survival exponential entropies. IEEE Trans. on Inf. Th. 51(3), 420–423 (2005)

GA Optimization of Networks Against Malicious Attacks

Nuri Yazdani[1], Hans Herrmann[1,2], Fabio Daolio[3], and Marco Tomassini[3]

[1] Institut fur Baustoffe, ETH-Honggerberg, Schafmattstrasse 6, 8093 Zurich, Switzerland
[2] Departamento de Fisica, Universidade Federal do Ceara, 60451-970 Fortaleza, Ceara, Brazil
[3] Information Systems Department, Faculty of Business and Economics, University of Lausanne, 1015 Lausanne, Switzerland

Abstract. The importance of considering the network structure in studying the behavior of many modern infrastructures and naturally occurring complex systems, has become increasingly clear. The topology of many real world networks makes them vulnerable with respect to targeted attacks on their components, and there has been a recent surge of interest in determining how one can optimize the structure of these networks to minimize this vulnerability. Given a measure of robustness as objective function and an edge-swap procedure as operator, one can design an iterative algorithm to this purpose. Similar work exists with trajectory-based meta-heuristics and has shown that this method can largely improve the robustness of a given network. Here we present the first attempt at optimizing networks against malicious attacks using an evolutionary algorithm, and show that one can achieve significantly improved results with modest additional investment of computational resources.

1 Introduction

Network infrastructures such as the transportation and power systems are extremely important. They can be abstractly defined as networks for which many results on the structure and the dynamics of phenomena have been investigated and described [10]. One extremely important feature of a network is its capability to withstand failures and errors in the functionality of its nodes and links. This is obvious for transportation, communication, computer, and power networks, but it is also essential for social, economical, and biological ones. For example, the failure of a gene activation node in a genetic regulatory network may lead to a cascade of failures and, ultimately, to illness or death. Failures may occur in many different ways and to a different degree, depending on the complexity of the system under examination. Therefore, there are potentially many ways in which the functionality of a network can be impaired partially or totally. However, it is certainly useful to consider simple models of failures first. A simple but powerful model of network robustness under failures was put forward by R. Albert et al. [2].

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Albert et al. abstracted away all other possible complexities and details and only studied the influence of the network topology on the behavior of the network being attacked considering the fragmentation of the initially connected graph as a growing number of nodes is suppressed. They considered two types of attacks to nodes, but attacks to links may be studied analogously. Attacks can be either random, i.e. any node may be shut with the same uniform probability, or nodes can be attacked as a function of their connectivity, i.e. their degree. As model networks they used Erdös-Rény random graphs [10] and scale-free random graphs of the Barabási-Albert type [1]. While in the former all nodes have a degree close to the mean, as the degree is Poisson-distributed, in the latter the distribution is right-skewed: most nodes have a small degree but there is also a significant number of nodes of high degree called hubs.

It turns out that under random attacks degree-inhomogeneous scale-free graphs are much more robust than random ones and one has to remove a significant fraction of nodes before the graph falls apart and fragments itself into separated components. However, this tolerance against random attacks or failures comes at a price as scale-free graphs are much more vulnerable to the removal of nodes according to node's degree. That is, in a scale-free graph if the nodes are removed in decreasing order of degree, starting with the most connected ones, then the network falls apart very quickly because those highly connected nodes are the ones that held the network together. On the other hand, in Erdös-Rény random graphs degree fluctuation is very limited and thus targeted attacks are similar to random ones. The striking conclusions of Albert's et al. sparkled a number of other studies on the vulnerability of networks, including real-life ones such as portions of the Internet and biological nets [5, 8, 15, 7].

Being able to understand the reasons that make some networks more robust is very important in practice since this would allow to design networks to be particularly tolerant with respect to some type of perturbation or intentional attack. In a similar way, we could also be given a network and be asked how to make it more robust without altering its its degree distribution. This constraint often arises in real-life networks due to the connection capacity of the nodes. Schneider et al. [13, 12] already studied this problem using a simple hill-climbing procedure for the optimization in various network classes. This has been followed by a study on Barabási-Albert networks by using the more powerful simulating annealing search heuristic [4]. In the present investigation we extend those works using evolutionary algorithms and more general scale-free configuration graphs.

The paper is organized as follows. In the next section we summarize the attack types, the network topologies used, and the robustness measure. This is followed by a presentation of the evolutionary algorithm used and by a discussion of the results obtained. Finally, we give our conclusions.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

2 Robustness Measure and Scale-Free Network Construction

In a recent article, Schneider et al. [13] proposed a novel measure for the robustness of a network inspired by percolation theory:

$$R = \frac{1}{N+1} \sum_{Q=0}^{N} s(Q), \tag{1}$$

where N is the number of nodes in the network, and $s(Q)$ is the size of the largest connected component of the graph after removing Q nodes. The range of R is $[0, 0.5]$, where the limits represent the value of R for a set of isolated nodes and a fully connected graph respectively. While the network structure of many modern infrastructures possesses a strong resilience to random failures, they can be rather unstable against targeted attacks [2, 5], as noted in the introduction. One such attack strategy is a high degree (HD) attack, where nodes in the network are removed sequentially based on their degree, starting with the highest connected node. Subject to such an attack, removal of a small fraction of nodes can lead to complete fragmentation of the network [5].

For a network with a given degree sequence, $\boldsymbol{n_k} = \{n_1, n_2, \ldots, n_{k_{max}}\}$, one can search for a way to connect the nodes which maximizes the robustness measure given above when the network is subject to a targeted attack. Schneider et al. [13] investigated the first such optimizations. In their work, a simple hill climbing procedure was used. Initially starting with a graph with a given degree sequence, they would attempt to swap the end points of two randomly selected edges, a move that is schematically depicted in Fig. 1. Such a swap would only be accepted if the new configuration had a higher value of the robustness measure R. With this procedure, detailed in algorithm 1, the authors were able to achieve significantly increased robustness for a variety of networks, and they found that the optimized networks had an *onion-like* topology, where many paths connecting nodes of the same degree, do not contain nodes of higher degree, and nodes could be layered on rings of decreasing degree [13]. For a given degree sequence, though, the number of ways to connect the network is enormous. The search space of the optimization procedure is thus vast, and the problem appears to be fertile ground for the application of an evolutionary algorithm (EA). In this work we show that application of a EA can produce networks with a significantly improved robustness.

The most prevalent types of networks found in nature and society are scale-free (SF) networks [3]. The degree distribution of the nodes in SF networks follows a power law distribution

$$P(k) \backsim k^{-\lambda}, \tag{2}$$

where k is the degree of a node and λ is the exponent characterizing the power law. For finite sized networks, there also exists an upper cut-off for the degree, k_{max}, and often one chooses a non-zero lower bound, k_{min}. In this work $\lambda = 3$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

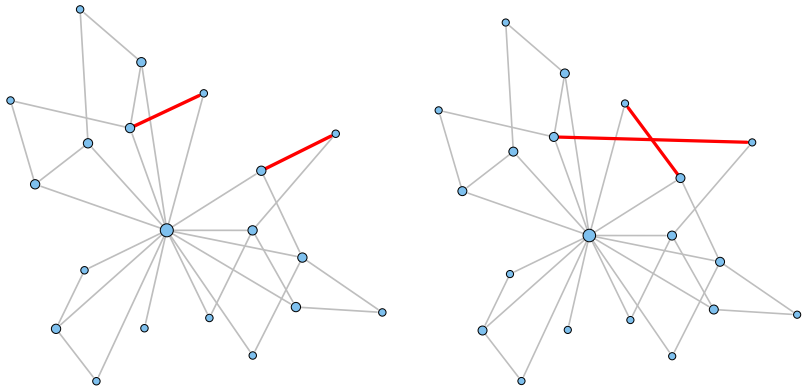


Fig. 1. The edge swap operation. Two edges are chosen at random in the original graph (thick lines, left image) and they are swapped as indicated in the right image, giving rise to a new graph with the same degree distribution.

Algorithm 1: First-Improvement Hill-Climber

```
Build graph g* ∈ S ;
repeat
    Compute R(g*);
    g ← g*;
    Choose edges e_ij, e_kl ∈ g uniformly at random and swap them;
    Compute R(g);
    if R(g) > R(g*) then
        g* ← g;
until g* is a local optimum ;
```

was chosen, with $[k_{min}, k_{max}] = [2, N_n]$. To construct the networks used in this work, the more general configuration model [9] was used. One must first generate a degree sequence, $\boldsymbol{n_k}$, from the degree distribution (Eq. 2). The links are then assigned between randomly selected pairs of nodes, making sure to preserve the selected degree sequence, and that no more than one link connects any pair of nodes.

Then, to calculate the robustness measure as defined in Eq. 1, the order of node deletion must first be determined. For a HD attack, one removes the nodes sequentially according to their degree starting from the highest. As nodes are removed, the degrees of the remaining nodes must be re-evaluated, as the deleted nodes links are also removed from the network. If at some point several nodes have the same largest degree in the network, one of the nodes is randomly chosen for deletion. Because of this stochastic aspect, there are a very large number of possible attack orders, and one must average over many of them when calculating R.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

3 Evolutionary Algorithm Setup

For the optimization procedure, a tournament selection algorithm with mutation and elitism was chosen. Recombination operators were not included as their use has been shown to be detrimental in artificial neural networks evolution for several reasons including symmetry and redundance [6, 16]. A population of N_p networks is initially constructed. The whole population of networks have the same degree sequence, but the random assignment of links is performed independently for every individual. With the exception of the most robust individual, each member of the population is mutated at each evolutionary step. Mutation of an individual simply involves swapping the end nodes of two randomly selected links. This is equivalent to the operation performed in the inner loop of algorithm 1, except for the fact that in the present case the swap is always accepted. One can transform any edge configuration to any other with a finite number of such mutations. After mutation the robustness of all of the individuals is recalculated, and the new individuals for the next evolutionary step are determined by constructing N_p unbiased tournaments [14], each with N_t individuals participating. The evolutionary cycle is sketched in algorithm 2.

Algorithm 2: Tournament Selection Evolutionary Algorithm

```
Initialize and evaluate N_p individuals;
repeat
    Mutate and re-evaluate all individuals except the most robust;
    Build N_p tournaments of N_t individuals each;
    Select N_p winners to advance into the next generation;
until termination criterion not met ;
```

The optimization procedure performs indeed like a parallel local searcher in the configuration space induced by the edge swap operator; it is the selection operator that controls the intensification-diversification trade-off and drives the population evolution. On that note, the unbiased tournament selection has been chosen to address the loss of diversity due to individuals not being sampled. In practice, when composing the N_p tournaments of size N_t, the whole population is lined up against $N_t - 1$ independent permutation of itself, and N_p tuple-wise comparisons are performed, from each one of which a winner emerges. In this way each individual gets to participate at least once in a tournament, and any bias due to random uniform sampling of tournament candidates is avoided.

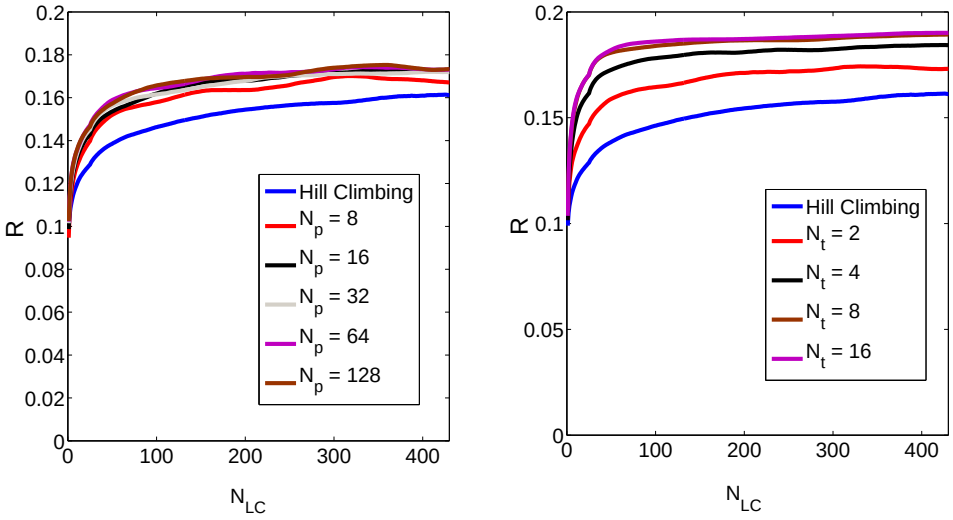
Calculation of the robustness measure (eq. 1) is by far the most computationally intensive aspect of the optimization procedure. One must average over many realizations of the attack order, and for each realization one must calculate the size of the largest connected cluster of nodes at each stage of the attack. Fortunately the optimization procedure is easily parallelized. While one node takes care of the EA implementation, calculation of the robustness of the N_p individuals for each evolutionary step is uniformly split between the available

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

processors. This parallelization allowed us to run the optimization procedure on relatively large networks.

4 Numerical Results

In order to tune the EA parameters, we performed initial optimizations on small systems varying N_n and N_t; the results are shown in figure 2. To allow for direct comparison, all optimizations were performed for networks with identical degree sequences. The results indicate that we do not need a very large population, as the EA using $N_p = 32$ performs as well as the larger population sizes, and that a relatively large N_t provides the best results, with the EA with $N_t = 8$ and 64 outperforming smaller tournament sizes.



(a) (b)

Fig. 2. Plot of the robustness vs. number of evolutionary steps (N_{LC}), for a network with $\lambda = 3$ and $N_n = 512$, varying the population size with $N_t = 2$ (a) and the tournament size for $N_p = 64$ (b).

With these preliminary results, we decided to fix the parameters of the EA to $N_p = 32$ and $N_t = 8$. To compare the performance of the EA to the hill climbing procedure, we ran the EA independently on 10 different degree sequences for a network with $N_n = 1024$ and $\alpha = 3$. Figure 3 shows the results for the performance of the EA averaged over all 10 different degree sequences. From the figure we see that the EA performs significantly better that the hill climbing procedure.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

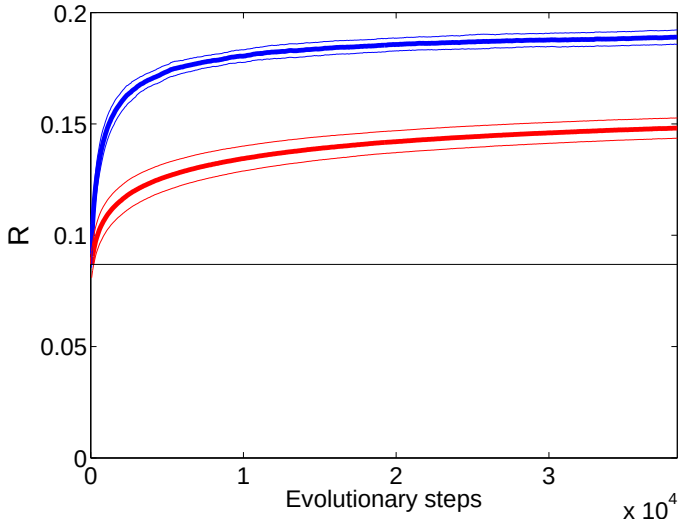


Fig. 3. Plot of the robustness vs. number of evolutionary steps (N_{LC}), for a network with $\lambda = 3$ and $N_n = 1024$ averaged over 10 different degree sequences. The thick blue line represents results using a EA with $N_p = 32$ and $N_t = 8$. The thick red line shows the results for the robustness using hill climbing. The black line shows the average robustness of randomly connected networks. The thin red and blue lines represent the 85% confidence level error-bars on the results.

We are not directly comparing here with the results published in Ref. [13], as those were obtained on Barabási-Albert scale-free network, whereas the configuration model is used here. However, the final R values are comparable and the optimized networks have the same *onion-like* topology firstly found by Schneider et. al [13]. The defining feature of *onion-like* networks is the appearance of several k-cores, where a k-core is the maximal connected sub-graph in which all nodes have degree greater less than or equal to k. Whereas in a scale-free network generated by the configuration model there is only one shell with coreness equal to the network's minimum degree, several shells of raising k values appear in the optimized network as an effect of the rewiring procedure. The tendency was observed in the optimized networks and can be seen in figure 4, which shows the fraction of nodes with degree k that are connected through nodes with a degree smaller or equal to k in the optimized networks. Figure 5 shows a visualization of one optimized network along with a visualization of the same network connected randomly.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

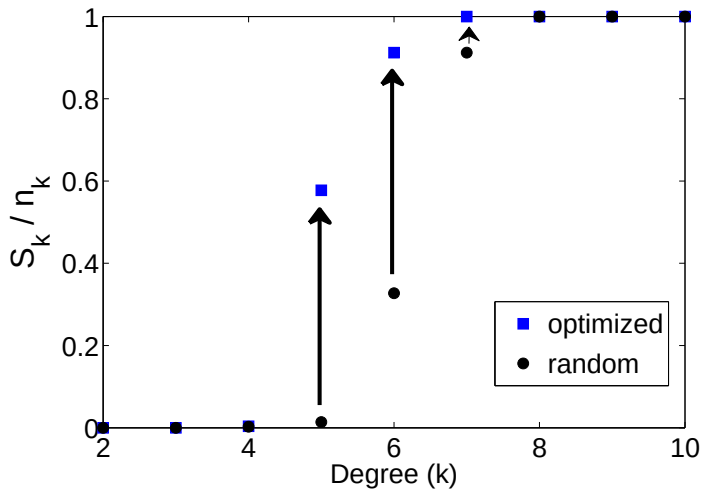
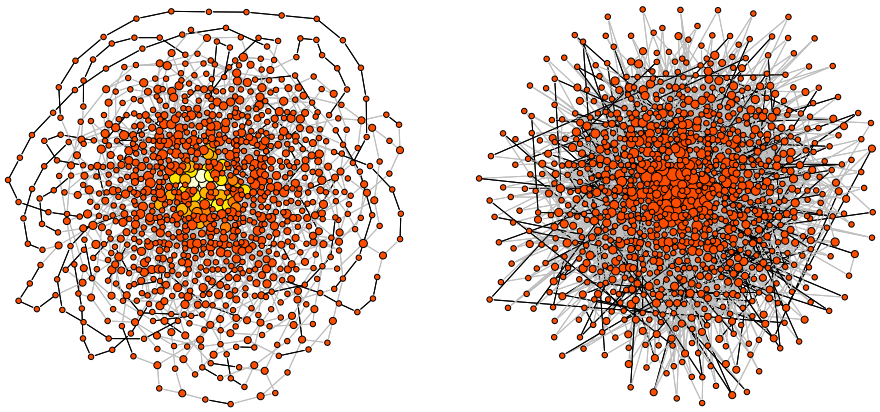


Fig. 4. Fraction of nodes, S_k,with degree k that are connected through nodes with a degree smaller or equal to k averaged over the optimized networks from Fig. 3.



(a) (b)

Fig. 5. Visualization of an optimized (a) and unoptimized (b) network with 1024 nodes. The vertex size of the nodes is proportional to the log of node degree, and vertex color scales with node coreness (the darker the lower); edges connecting nodes with equal degree are highlighted in black.

5 Summary and Conclusions

We showed in this work that the application of an EA to optimize networks against malicious attacks can produce networks with significantly enhanced robustness compared with those obtained with a standard first-improvement hill-climber. Furthermore it was shown that one does not need to use a very large population size nor maintain a large diversity in the population to achieve significantly improved results. As a population-based meta-heuristic, the proposed EA also leads naturally to a parallel implementation.

There are many avenues in which the research can proceed. While in this work the networks considered can be described by the degrees of the nodes and the manner in which they are connected, real networks often need additional parameters such as weights on the links and nodes to accurately describe their functionality. For example in air traffic networks the amount of flights operating between two connected airports, or in trade networks the amount of goods traded between two companies, is not uniform. Another problem that is just beginning to be addressed is the optimization of several coupled networks, where dependency links join nodes between two of the networks [11]. Our results imply that with only a modest additional investment of computational resources one can significantly enhance the optimization procedure, and this should therefore be utilized in further work in this field.

Acknowlegments. Nuri Yazdani, Hans Herrmann, and Marco Tomassini, gratefully acknowledge financial support by the Swiss National Science Foundation under contract 200021_126853.

References

1. Albert, R., Barabasi, A.L.: Statistical mechanics of complex networks. Reviews of Modern Physics 74, 47–97 (2002)
2. Albert, R., Jeong, H., Barabasi, A.L.: Error and attack tolerance of complex networks. Nature 406, 378–382 (2000)
3. Barabasi, A.L., Bonabeau, E.: Scale-free networks. Scientific American 288, 50–59 (2003)
4. Buesser, P., Daolio, F., Tomassini, M.: Optimizing the robustness of scale-free networks with simulated annealing. In: Dobnikar, A., Lotric, U., Ster, B. (eds.) Adaptive and Natural Computing Algorithms. LCNS, vol. 6594, pp. 167–176. Springer, Heidelberg (2011)
5. Cohen, R., Erez, K., Avraham, D.B., Havlin, S.: Breakdown of the Internet under intentional attack. Phys. Rev. Lett. 86, 3682–3685 (2001)
6. Fogel, D.B., Fogel, L.J., Porto, V.W.: Evolving neural networks. Biological Cybernetics 63, 487–493 (1990)
7. Holme, P., Kin, B.J., Yoon, C.N., Han, S.K.: Attack vulnerability of complex networks. Phys. Rev. E 65, 056109 (2002)
8. Latora, V., Marchiori, M.: Efficient behavior of small-world networks. Phys. Rev. Lett. 87, 198701 (2001)

9. Molloy, M., Reed, B.: Acritical point for random graphs with a given degree sequence. Random Structures & Algorithms 6, 161–179 (1995)
10. Newman, M.E.J.: Networks: An Introduction. Oxford University Press, Oxford, UK (2010)
11. Parshani, R., Buldyrev, S.V., Havlin, S.: Interdependent networks: Reducing the coupling strength leads to a change from first to second order percolation transition. Phys. Rev. Lett. 105, 048701 (2010)
12. Schneider, C.M., Andrade, J.S., Shinbrot, T., Herrmann, H.J.: Protein interaction networks are fragile against random attacks and robust against malicious attacks. Tech. rep. (2010)
13. Schneider, C.M., Moreira, A., Andrade, J.S., Havlin, S., Herrmann, H.J.: Onion-like network topology enhances robustness against malicious attacks. J. Stat. Mech. (2010), to appear
14. Sokolov, A., Whitley, D.: Unbiased tournament selection. GECCO pp. 1131–1138 (2005)
15. Valente, A.X.C.N., Sarkar, A., Stone, H.: 2-peak and 3-peak optimal complex networks. Phys. Rev. Lett. 92, 118702 (2004)
16. Yao, X., Liu, Y.: A new evolutionary system for evolving neural networks. IEEE Transactions on Neural Networks 8, 694–713 (1997)

Imperialist Competitive Algorithm for Dynamic Optimization of Economic Dispatch in Power Systems

Robin Roche[1], Lhassane Idoumghar[2], Benjamin Blunier[1], and Abdellatif Miraoui[1]

[1] Université de Technologie de Belfort-Montbéliard,
Laboratoire Systèmes et Transports, 90010 Belfort, France,
robin.roche@utbm.fr
[2] Université de Haute-Alsace,
LMIA / INRIA Grand Est, 68093 Mulhouse, France,
lhassane.idoumghar@uha.fr

Abstract. As energy costs are expected to keep rising in the coming years, mostly due to a growing worldwide demand, optimizing power generation is of crucial importance for utilities. Economic power dispatch is a tool commonly used by electric power plant operators to optimize the use of generation units. Optimization algorithms are at the center of such techniques and several different types of algorithms, such as genetic or particle swarm algorithms, have been proposed in the literature. This paper proposes the use of a new metaheuristic called imperialist competitive algorithm (ICA) for solving the economic dispatch problem. The algorithm performance is compared with the ones of other common algorithms. The accuracy and speed of the algorithm are especially studied. Results are obtained through several simulations on power plants and microgrids in which variable numbers of generators, storage units, loads and grid import/export lines are connected.

Keywords: metaheuristic, imperialist competitive algorithm, dynamic optimization, economic dispatch, microgrid

1 Introduction

With fossil resources becoming harder and harder to extract and worldwide demand continuously increasing, energy costs are expected to rise significantly in the coming years. Therefore, optimizing the use of these resources is of crucial importance in order to minimize costs. Such considerations are essential to power plant and transmission grid operators, for which optimizing operating costs can result in significant savings and profit. Economic power dispatch enables such an optimization and aims at determining the most cost-efficient and reliable operation of power systems, such as power plants. This objective is achieved by optimally dispatching available generation resources to supply the load connected to the system.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Optimization algorithms play an important role in economic dispatch, as they are the central tool used to obtain the optimal dispatch. Over the years, numerous algorithms have been used but imperialist competitive algorithms, a new type of metaheuristic based on imperialistic competition, are still largely unexplored for this problem. Moreover, very few were utilized in a dynamic optimization context.

The following studies whether this algorithm could and should be more widely used for this application. It describes the economic dispatch optimization problem, the imperialist competitive algorithm we used, how we tested it on mathematical functions and compared it with other common algorithms and finally how it was tested on two power systems for solving the dynamic economic dispatch problem.

2 Economic Power Dispatch Problem

2.1 Economic Dispatch Concept

The US Energy Policy Act [10] defines economic dispatch (ED) as "the operation of generation facilities to produce energy at the lowest cost to reliably serve consumers, recognizing any operational limits of generation and transmission facilities." In short, ED aims at optimally dispatching power generation between available generation units to meet demand. Its objectives are usually to minimize fuel utilization and sometimes also greenhouse gases emissions, compared to what less efficient generation sources would result in. Finding the best trade-off between costs, environmental impact and reliability is thus the main challenge of ED. In order to reduce operation costs, ED has been used in various forms for years, especially by transmission system and power plant operators.

Two types of ED can be considered: economic dispatch for the current day, and for future days.

- The first one, sometimes referred to as load following, consists in dispatching power for the current day, by monitoring load, generation and power imports/exports, while ensuring a balance between load and supply. The stability of the frequency of the grid (50 or 60 Hz) is the consequence of this balance, and is required by most loads which use it as a reference.
- The second one corresponds to dispatch for the following day, or several days after. Performed by the generation group or an independent market operator, it mainly consists in scheduling generators for each hour of the next day's dispatch, based on load forecasts. The units to use are selected based on their characteristics, costs and maintenance requirements.

The following will focus on the first type of dispatch, for the current day. Generators, such as gas turbines or fuel cells, will not be scheduled but will be sent set points their own control system should take into account immediately, while ensuring the reliability of the system. However, algorithms similar to the ones described in the following can be used for scheduling and committing generators. It should also be mentioned that determining if an algorithm behaves well

for ED is a necessary task, as the consequences of a bad ED can have a significant impact on costs for the operator and the consumer and cause instabilities or even blackouts.

2.2 Objective, Constraints and Hypotheses

To achieve an optimal power dispatch from an economic point of view, an objective function needs to be defined. This function (1) corresponds to the total cost of generation, storage, and power imports and exports. In order to maintain the grid frequency stable and the system reliability, constraint (2), requiring a balance between generation and supply, must be met.

$$\text{Minimize} \quad c_{\text{tot}}(t) = \sum_{i=0}^{n_{\text{gen}}} c_{\text{gen}}(P_{\text{gen},i}(t)) + \sum_{i=0}^{n_{\text{s}}} c_{\text{s}}(P_{\text{s},i}(t)) + \sum_{i=0}^{n_{\text{g}}} c_{\text{g}}(P_{\text{g},i}(t)) \quad (1)$$

$$\text{Subject to} \quad P_{\text{imb}}(t) = \sum_{i=0}^{n_{\text{gen}}} P_{\text{gen},i}(t) + \sum_{i=0}^{n_{\text{s}}} P_{\text{s},i}(t) - \sum_{i=0}^{n_{\text{l}}} P_{\text{l},i}(t) - \sum_{i=0}^{n_{\text{g}}} P_{\text{g},i}(t) = 0 \quad (2)$$

where $c_{\text{tot}}(t)$ is the total generation cost, n_{gen}, n_{s}, n_{g} and n_{l} are respectively the number of generating units, of storage units, of grid import/export lines and of loads. Their respective costs c_X and power outputs P_X use the same indexes.

In the following, constraint (2) is taken into account by the algorithm through its objective function. The actual fitness or objective function $f(t)$ (3) is a combination of the total cost defined in (1) and of the imbalance between load and supply. A coefficient α is set according to the ratio between the magnitude of the power system (W, kW, MW, etc.) and the estimated costs to give more importance to keeping the imbalance as low as possible while minimizing costs.

$$\text{Minimize} \quad f(t) = c_{\text{tot}}(t) + \alpha \cdot P_{\text{imb}}(t) \quad (3)$$

Additional constraints are also to be respected, notably for the optimization bounds. These bounds reflect the characteristics and dynamics of the controlled units. For example, a generating unit i such as a gas turbine can only operate within its operating range (4), and its power output variation is limited by ramp-up and ramp-down limits R_{u} and R_{d} (5). In other words, the bounds at time $t+1$ depend on the ones at t because of the dynamic of the units and are thus updated at every call of the algorithm by an expert system.

$$P_{i,\min} \leq P_{i,t} \leq P_{i,\max} \quad (4)$$

$$P_{i,t} - R_{i,\text{d}} \leq P_{i,t+1} \leq P_{i,t} + R_{i,\text{u}} \quad (5)$$

In addition to these objectives and constraints, the following simplifying assumptions are made, as the primary focus of this paper is the performance of the algorithm:

– Valve-point effects, reactive power, line losses and emissions (e.g., CO_2 or NO_x) are not considered.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

– All voltage magnitudes are assumed to be nominal. Bus and node capacities are assumed to be sufficient.
– Scheduling, starting and stopping generators is not achieved by the algorithm.

2.3 Problem Characteristics

The ED problem, as we will treat it, can be considered as a dynamic, non-linear, constrained, optimal control problem. It is at first dynamic because the cost function and the constraints change or can change at every call of the algorithm. The controlled power system is indeed itself a dynamic system, which evolves over time. A gas or wind turbine can for example be stopped for maintenance. Due to this dynamic nature, the algorithm is run at a fixed frequency, which can be as low as a few seconds. To a certain extent, it can also be considered as an online optimization problem, as the optimal control is performed in almost real-time. This is enabled by the small scale of the considered systems.

The variables to optimize are control set points corresponding to powers that are defined in continuous spaces, themselves defined by the energy management policy of the operator and by an expert system based on models. Algorithms able to operate in real-valued search spaces are thus required, as well as algorithms which do not require precise information on the search space or on the objective function. The reason for this is that the algorithm should be able to adapt to as many systems as possible without requiring much effort. Stochastic or randomized search methods are ideal for such constraints.

2.4 Optimization Algorithms for this Problem

The ED problem is a classical optimization problem in power systems and has been solved using numerous optimization algorithms:

– Linear programming [6, 5], for which the objective function and the constraints must be linear. As we do not want to have any hypothesis on the shape of the functions, those algorithms are not selected. However, they are commonly used in the industry due to their speed and relative simplicity, but with constraint relaxation methods (such as Lagrange multipliers) and cost functions approximated by quadratic functions.
– Combinatorial algorithms, such as dynamic programming [11], can include integer variables and are based on the idea of splitting the problem into subproblems. Although these algorithms can be used for scheduling sources, their characteristics do not fit the needs of the current problem.
– Metaheuristics, such as genetic algorithms [12], particle swarm optimization [3] or simulated annealing. This last category of algorithms makes few or no assumption about the problem (e.g., the objective function does not need to be differentiable) but do not guarantee an optimal solution is ever found as they rely on random variables. As we do not know how the objective function looks like and evolves over time, possibly with discontinuities, these algorithms are selected.

Although the variety of these algorithms is quite large, the literature contains few examples of imperialist competitive algorithms applied to the solving of problems in power systems [2], and none for the ED problem specifically.

3 Imperialist Competitive Algorithm

The imperialist competitive algorithm (ICA) is a new evolutionary optimization approach introduced in 2007 by E. Atashpaz-Gargari [1]. It is inspired by the imperialistic competition processes of human societies. The algorithm can be seen as a social counterpart of genetic algorithms. Several of its steps are indeed similar: countries can undergo revolutions as chromosomes can mutate, for example.

This algorithm uses a precise terminology, in which a solution is called a country. There are two types of countries: imperialist countries, and colonies, which depend on these imperialists. An imperialist and its countries form a group of countries called empire.

Algorithm 1 Imperialist Competitive Algorithm

```
Initialize the countries and form the empires
while the stop condition is not satisfied do
    for all empires do
        Move the colonies toward the imperialist (assimilation)
        Make some colonies undergo a revolution
        if a colony is more powerful than the imperialist then
            The colony becomes the imperialist and vice versa (overthrow)
        end if
    end for
    if two empires are too close then
        Merge them (unification)
    end if
    Make imperialistic competition occur
    if there is an empire with no colonies then
        Eliminate this empire
    end if
end while
```

ICA works as illustrated in Algorithm 1, where the following processes are used:

– Initialization and empire formation: Like other evolutionary algorithms, ICA starts with an initial population of solutions called countries, of size N_{pop}. Among them, the N_{imp} best countries (the most powerful) are selected to be imperialists. The remaining N_{col} countries form the colonies of these imperialists. The n initial empires are formed by dividing the colonies among impe-

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

rialists according to their normalized power P derived from their cost c (6).

$$P_n = \left| \frac{c_n - \max_i c_i}{\sum_{i=1}^{N_{\text{imp}}} (c_n - \max_i c_i)} \right| \tag{6}$$

The number of colonies $N_{\text{col},n}$ attached to empire n is computed according to (7).

$$N_{\text{col},n} = \text{round}\,(P_n \cdot N_{\text{col}}) \tag{7}$$

– Assimilation: Imperialist countries attract colonies to them using the assimilating policy illustrated in Fig. 1. To update its position x, each colony moves toward its imperialist by updating its position using (8).

$$x_{t+1} = x_t + \beta \cdot \gamma \cdot r \cdot d \tag{8}$$

where $\beta > 1$ causes the colonies to get closer to the imperialist, $\gamma < 1$ corresponds to an assimilation coefficient, r is a random number chosen from the uniform distribution $\mathcal{U}(-\theta, \theta)$, θ adjusts the deviation from the original direction and enables searching around the imperialist and d is the distance between the colony and the imperialist.

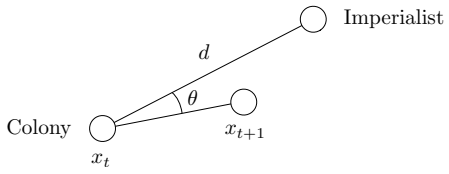


Fig. 1. Movement of a colony toward its imperialist

– Revolution: The revolution process introduces sudden random changes in the position of some countries. It plays the same role as the mutation operator in a genetic algorithm.
– Overthrow: After assimilation and revolution, a colony might reach a better position than the imperialist of the empire. In this case, the colony can become the imperialist and vice versa.
– Unification: If two empires are too close to each other, they can unite and become a single empire, with the sum of the colonies of the two initial empires.
– Imperialistic competition: Each empire tries to take possession of colonies of other empires and control them. This imperialistic competition is modeled by selecting the weakest colonies of the weakest empire and giving them to the empire that has the highest likelihood to possess them.
The total power $P_{\text{tot,n}}$ of each empire n is defined by the power of its imperialist plus its average colonies' power, as defined in (9) where $\zeta \ll 1$, I refers to the empire's imperialist and C to its colonies.

$$P_{\text{tot,n}} = P(I_n) + \zeta \cdot \text{mean}(P(C_n)) \tag{9}$$

The likelihood p_n, called possession probability, is then derived from each empire's power (10).

$$p_n = \left| \frac{c_{\text{tot,norm},n}}{\sum_{i=1}^{N_{\text{imp}}} c_{\text{tot,norm},i}} \right| \tag{10}$$

where $c_{\text{tot,norm},n} = c_{\text{tot},n} - \max_i c_{\text{tot},i}$ is the total normalized cost of empire n and $c_{\text{tot},n}$ its total cost.
In order to divide the colonies among empires based on their possession probability, a vector A is built (11), where P_n is the power of empire n and r_n a random value between 0 and 1. The selected colonies are then assigned to the empire whose relevant index in A is maximum.

$$A = [P_1 - r_1, P_2 - r_2, ..., P_{N_{\text{imp}}} - r_{N_{\text{imp}}}] \tag{11}$$

4 Performance on Mathematical Problems

Before testing the algorithm on the ED problem, its performance is compared to other evolutionary algorithms. These algorithms are:

- Metropolis Particle Swarm Optimization Algorithm with Mutation Operator (MPSOM) [4]: This hybrid PSO variant uses the Metropolis rule and a mutation operator to avoid local minima.
- Differential Evolution (DE) [8]: This algorithm combines the positions of solutions, called agents, to move them in the search space. Only the moves that lead to an improvement are accepted, others are discarded.
- Imperialist Competitive Algorithm (ICA), which was just presented.

The parameters of the ICA were empirically determined by running iterative trials using the mathematical functions described in Table 2, and starting with the parameters given by the authors in [1]. The tuned parameters are summarized in Table 1.

Several benchmark functions [9] described in Table 2 were used to test the algorithms performance. As the focus in primarily on the ED problem, the number of functions is limited to four. These functions provide a good start for testing the credibility of an optimization algorithm. Each of these functions has many local optima in its solution space. The amount of local optima increases with their dimension, which was set to 20 as in [7]. For each algorithm, the maximum number of function evaluations is $300,000$. A total of 30 runs for each algorithm was conducted and the average fitnesses of the best solutions were recorded.

The mean solutions and the corresponding standard deviations obtained for the algorithms are listed in Table 3. MPSOM obtains the best results for the first two functions, and DE for the third, Rosenbrock. ICA performs best on Ackley, with slightly better results than DE. A finer tuning of ICA parameters

Table 1. Parameter settings for the ICA approach

Parameter	Variable	Value
Number of initial solutions	N_{pop}	60
Number of initial imperialists	N_{imp}	6
Maximum number of iterations	N_d	5,000
Revolution rate	R_r	0.1
Assimilation coefficient	β	2
Assimilation angle coefficient	θ	$\frac{\pi}{6}$
ζ coefficient	ζ	0.02
Uniting threshold	U_t	0.02

Table 2. Standard benchmark functions adopted in this work

Function	Problem	Range
Sphere	$\sum_{i=1}^{n} x_i^2$	[-100;100]
Rastrigin	$\sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	[-5.12;5.12]
Rosenbrock	$\sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	[-2.048;2.048]
Ackley	$20 + e - 20\ e^{-0.2\ (\frac{1}{n}\sum_{i=1}^{n} x_i^2)^{\frac{1}{2}}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$	$[-30.0; 30.0]$

Table 3. Comparison of the solutions obtained by the four selected metaheuritics. No lower threshold limit is set for the results.

Function	MPSOM	DE	ICA
Sphere	1.17×10^{-114}	7.145×10^{-40}	8.458×10^{-12}
	$\pm 1.17 \times 10^{-114}$	$\pm 1.390 \times 10^{-39}$	$\pm 2.532 \times 10^{-11}$
Rastrigin	0	4.551	2.251×10^{-06}
	± 0	± 0.919	$\pm 3.957 \times 10^{-06}$
Rosenbrock	1.44×10^{-02}	1.880×10^{-11}	0.3209
	$\pm 1.63 \times 10^{-02}$	$\pm 1.211 \times 10^{-11}$	± 0.4014
Ackley	1.16×10^{-10}	3.946×10^{-15}	8.025×10^{-16}
	$\pm 2.42 \times 10^{-11}$	$\pm 1.119 \times 10^{-16}$	$\pm 8.365 \times 10^{-15}$

and hybridizing it with another algorithm would probably strongly improve its results on such functions. The following examines how the algorithm performs on the ED problem.

5 Simulations for the Economic Dispatch Problem

Two different grid configurations are used to test the performance of the algorithm, one is a microgrid where the stability of the algorithm will be tested and the other a fuel cell power plant where the optimization will focus on the total cost of the simulation. Both tests run over a period of four to five days and are based on real load profiles[3]. The optimization process is run every 60 s. Coefficient α from (3) is set to 10^5. The tests were run several times and returned very similar results.

5.1 Microgrid Test

The first test corresponds to a microgrid with two identical 83 kW fuel cells, photovoltaic panels with a rated peak power of 600 kW, two 500 kW wind turbines, a 1 MW import/export power line and a group of loads corresponding to the consumption of a residential area. The proportion of renewable energy sources in the microgrid corresponds to a very high penetration rate, which often implies control difficulties due to their intermittency. Simple models are used to extract the sources' power output from input values such as solar irradiation and wind speed[4], as well as to extract the hydrogen consumption of the fuel cells.

The expert system, which reflects the merit order chosen by the operator, determines the optimization bounds for each element. Renewable and non-controllable energy sources are used in priority and controllable generation units such as fuel cells are only allowed to provide the missing power to meet demand. Importing or exporting energy to the main grid is only allowed when the other sources are not sufficient (a similar rule could be used for storage).

The cost functions of the fuel cells reflect their fuel consumption, and generation costs from renewable energy sources are considered as equal to zero. The cost for importing a given amount of energy from the grid is lower than the price at which the same amount is bought by the grid, according to the principles of feed-in tariffs.

The test is run with three algorithms: ICA, DE and MPSOM. Results are shown in Fig. 2 for ICA and in Table 4 for all algorithms. They show that the algorithms perform well regarding the accuracy of constraint verification, i.e., the balance between load and supply (not displayed in Fig. 2 for the sake of clarity) is maintained. DE is the fastest but also the least efficient for minimizing total costs. MPSOM performs well, as well as ICA, except for its duration.

5.2 Fuel Cell Power Plant Test

The second configuration is a power plant based on a fuel cell array. A group of six 83 kW fuel cells generate power and supply a local load assimilable to an industrial and commercial area.

[3] Adapted from Southern California Edison's load profiles at: http://www.sce.com/AboutSCE/Regulatory/loadprofiles

[4] Extracted from: http://www.unige.ch/cuepe/html/meteo/donnees-csv.php

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

10

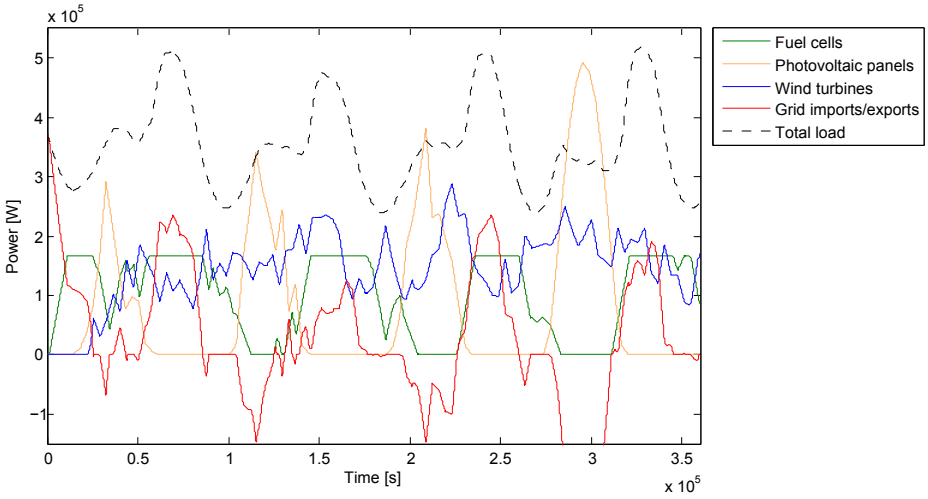


Fig. 2. Results of the microgrid simulation with ICA. Although intermittent sources induce large variations in generation, the use of fuel cells and of grid imports/exports, and the corresponding costs, are minimized by the algorithm while verifying the power balance constraint.

Table 4. Microgrid test results comparison

Algorithm	Unit	MPSOM	DE	ICA
Total cost	€	1,075	1,155	1,088
Mean imbalance	W	-0.0175	0.1344	0.1314
Mean duration	ms	8.126	4.075	257.0

The cost functions of the fuel cells differ from one fuel cell to the other. The aging and wearing of the units is taken into account by adding a multiplier coefficient η (12), which degrades or improves the cost of the fuel cells according to their age and how they are used and maintained. Coefficients between 0.9 and 1.2 are selected (see Fig. 3's legend – a high coefficient corresponds to an old and more expensive fuel cell), and are used to determine which algorithm behaves best for this kind of optimization.

$$c_{\mathrm{fc}}(P_{\mathrm{fc}}) = \eta \cdot m((P_{\mathrm{fc}})) \tag{12}$$

where c_{fc} is the cost of the fuel cell for set point P_{fc}, and m the fuel cost function.

Table 5 summarizes the results of the tests shown in Fig. 3 for ICA. The same test profile is run with the three algorithms used in the previous test and a reference algorithm called equal dispatch (EqD), which simply consists in dividing the total load by the number of operating fuel cells without any optimization process.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

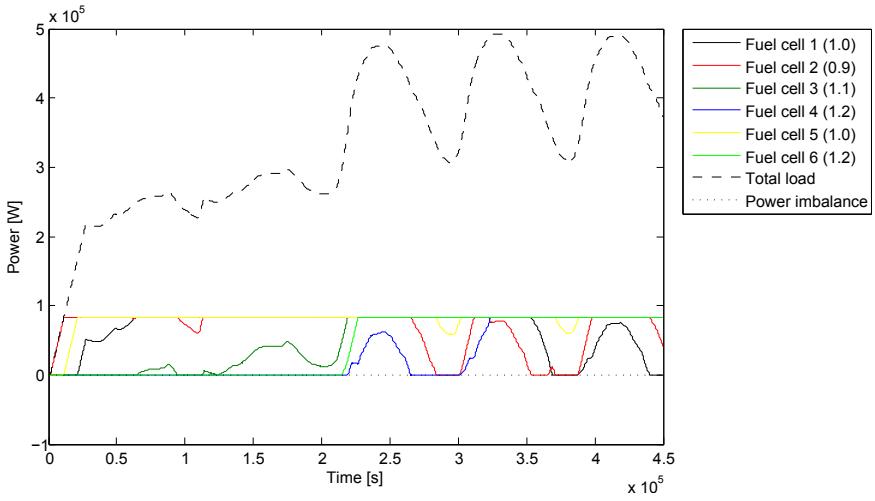


Fig. 3. Results of the power plant simulation with ICA. By properly selecting how much to use each fuel cell, the algorithm manages to achieve the best total cost. Older and less efficient fuel cells indeed tend to be less used by the algorithm to meet demand.

Table 5. Fuel cell power plant test results comparison

Algorithm	Unit	EqD	MPSOM	DE	ICA
Total cost	€	4,097	2,165	2,160	2,119
Mean imbalance	W	$\simeq 0$	-0.099	-2.286	-2.888
Mean duration	ms	< 1	11.82	8.391	454.8

Results show that ICA gives the best results in terms of total cost, as opposed to EqD. However, as with the previous test, it is also slower than the other algorithms. DE is particularly fast but is also rather imprecise, although this imbalance only represents 10^{-4} % of the maximum total load. It should also be noted that the imbalance for ICA is almost always close to zero except for a few minutes around $4{\times}10^5$ s, where the imbalance reaches 500 W.

6 Conclusion and Future Work

An application to dynamic ED of a new metaheuristic based on imperialist competition, the ICA, was described in this article. It showed that ICA provides good results for the ED problem, which was the aim of this work. It helps minimize costs even more than the other tested algorithms, while maintaining a good accuracy regarding the constraint. However, it is slower and performs less well on mathematical functions, illustrating the famous "no free lunch theorem". It therefore needs to be improved by further tuning its parameters and by hybridiz-

ing it with another optimization algorithm, similarly to what was achieved with MPSOM. Future work will also focus on speeding up the algorithm.

As the objective of this article was to focus on the performance of the algorithm for the dynamic ED problem, several simplifying assumptions were made. However, in the future, emissions of classical energy sources will be taken into account to reflect current environmental concerns. Deciding whether generation sources should be started or stopped, as well as demand-side management, will also be integrated in the algorithm, which will enable next-day scheduling.

References

1. Atashpaz-Gargari, E., Lucas, C.: Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In: IEEE Congress on Evolutionary Computation. pp. 4661–4667 (2007)
2. Duki, E.A., Mansoorkhani, H.R.A., Soroudi, A., Ehsan, M.: A discrete imperialist competition algorithm for transmission expansion planning. In: 25th International Power System Conference (2010)
3. El-Gallad, A., El-Hawary, M., Sallam, A., Kalas, A.: Particle swarm optimizer for constrained economic dispatch with prohibited operating zones. In: Canadian Conference on Electrical and Computer Engineering. vol. 1, pp. 78–81 (2002)
4. Idoumghar, L., Idrissi-Aouad, M., Melkemi, M., Schott, R.: Metropolis particle swarm optimization algorithm with mutation operator for global optimization problems. In: 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI). vol. 1, pp. 35–42 (oct 2010)
5. Irving, M., Sterling, M.: Economic dispatch of active power with constraint relaxation. IEE Proceedings C Generation, Transmission and Distribution 130(4), 172–177 (1983)
6. Nabona, N., Freris, L.: Optimisation of economic dispatch through quadratic and linear programming. Proceedings of the Institution of Electrical Engineers 120(5), 574–580 (1973)
7. Pant, M., Thangaraj, R., Abraham, A.: Particle swarm based meta-heuristics for function optimization and engineering applications. In: 7^{th} Conf. Computer Information Systems and Industrial Management Applications. vol. 7, pp. 84–90. IEEE Computer Society (2008)
8. Storn, R., Price, K.: Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11(4), 341–359 (1997)
9. Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y.P., Auger, A., Tiwari, S.: Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Tech. Rep. 2005005, Nanyang Technological University, Singapore and IIT Kanpur, India (2005)
10. United States Department of Energy: Economic dispatch of electric generation capacity (2007), a report to Congress and the States pursuant to sections 1234 and 1832 of the energy policy act of 2005
11. Waight, J., Albuyeh, F., Bose, A.: Scheduling of generation and reserve margin using dynamic and linear programming. IEEE Transactions on Power Apparatus and Systems PAS-100(5), 2226–2230 (1981)
12. Walters, D., Sheble, G.: Genetic algorithm solution of economic dispatch with valve point loading. IEEE Transactions on Power Systems 8(3), 1325–1332 (1993)

Simulated annealing based metaheuristics and binary cellular automata to generate 2D shapes

Fazia Aïboud[1], Nathalie Grangeon[1] and Sylvie Norre[1]

[1] LIMOS CNRS UMR 6158, 24 avenue des Landais. BP10125.
63173 AUBIERE Cedex, France
{aïboud, grangeon, norre}@moniut.univ-bpclermont

Abstract. In this paper, we consider the generation of 2D binary preset shapes. We propose a combination of metaheuristic (simulated annealing or iterated local search) and cellular automaton. The proposed method determines the transition function and the number of generations that allow to the cellular automaton to evolve and to give the nearest shape to the preset shape. Mutual information is used as similarity measure between two shapes. Different neighborhood systems for the metaheuristic are proposed.

Keywords: Binary cellular automaton, simulated annealing, iterated local search, neighboring system, mutual information.

1. Introduction

The generation of shapes is shown as an interested problem in the literature used to understand the different natural processes allowing the development of forms of species known as morphogenesis. Two categories of approaches are proposed by Kenneth et al [1]: a grammatical approach and chemistry approach. In the grammatical approach development is guided by sets of grammatical rewrite rules (De Garis [2], Chavoya [3] and Shin [4]). The cell chemistry approach is based on the chemical signals exchanged between cells (Turing [5], Bai et al [6]).

Several grammatical models use cellular automata. They are discrete dynamic systems in space and time with simple local interactions but complex global behavior [7]. The cellular automaton consists in a regular cell lattice. Each cell can take a state chosen among a finite set of states. The state of a cell evolves during a number of step time (generations) according to the state of their neighboring cells through a local evolutionary rule. The set of these local rules forms a transition function of the cellular automaton. The number of transition functions increases when the number of considered neighboring cells and the number of states increase.

In this work, we are interested in the generation of 2D binary preset shape with cellular automaton from a single cell located in the middle of the lattice. The objective is to select a transition function allowing to the cellular automaton to evolve toward the preset shape and to determine the number of generations. This problem is known as inverse problem and is illustrated by Figure 1. This work comes within the scope of

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

preliminary studies of simulation of morphogenesis with the objective to simulate, by cellular automaton, the generation of a full organ from a single cell.

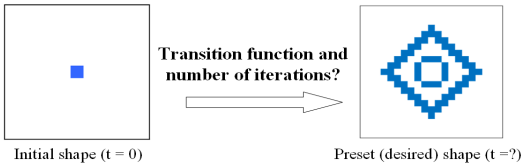


Fig. 1. Inverse problem

Several works in the literature propose genetic algorithms to solve these problems. Mitchell et al [8] and Back et al [9] have applied a genetic algorithm to evolve behaviour of 1D cellular automaton and solve the density classification problem. Garis [2] and Chavoya and Duthen [3] have generated preset shapes with cellular automata guided by genetic algorithm.

The method of Chavoya and Duthen is based on a combination of a cellular automaton and a genetic algorithm. The genetic algorithm determines the transition function and the number of generations of the cellular automaton. This method is easy to implement but is restricted to the generation of full shapes.

An extension of the model proposed by Chavoya and Duthen is proposed in [10] in order to generate any shapes: full shape, hollow shape (shape with holes). This extension concerns the modification of the interaction system and the updating system for the cellular automaton, the modification of the chromosome encoding and the proposition of a new fitness function for the genetic algorithm. In these models, the chromosome is coded in binary and contains two fields: an action field and a control field. The action field gives the transition function. The control field gives the number of generations *ng* that the cellular automaton applies this transition function.

In this paper we present a combination of cellular automaton and simulated annealing based metaheuristic to generate 2D binary preset shape. The metaheuristic proposes a transition function of the cellular automaton. The cellular automaton is used to compute the performance criterion and the number of generations for each proposed transition function.

This paper is organized as follow: in part 2, we present a formal definition of cellular automata in order to define the notations. We present our proposed method in the third part. The results obtained with different full and hollow shapes are presented in the fourth section. Finally, we finish with conclusion and further works.

2. Cellular automata

We consider a 2D cellular automaton defined by the quadruplet $A=(L, \sigma, N, F)$ where:

- L is a regular lattice that consists in a regular paving of the domain Ω of IR^2 . A cell is identified by its coordinates in the lattice.

- $\sigma=\{\sigma_1, \ldots,\sigma_k\}$ is the set of possible states of a cell. At a given time t, a cell c has a state $S_t(c)$.
  Let $l=(c_1, \ldots,c_n)$ a list of cells, by misnomer, we denote $S_t(l)$ the list of states of each cell in the list l i.e. $S_t(l) =(S_t (c_1), \ldots, S_t (c_n))$.
- N is the interaction system of a cell c defined by:

$$\begin{array}{rcl} N: L & \rightarrow & L^n \\ c & \rightarrow & N(c)=(c_1,...,c_n) \end{array} . \quad (1)$$

  Where c_i for $i = 1\ldots n$ is a neighbor cell relied to the observed cell c by proximity or influence relation. n is the size of interaction system N(c).
- F is the transition function that defines the evolution of the cellular automaton over time. It defines for each combination of states of the interaction system its local evolutionary rule f.
  When the transition function is applied simultaneously to all cells, the updating system is said synchronous. f is defined by:

$$\begin{array}{c} f:\sigma^n \rightarrow \sigma \\ S_t(N(c)) \rightarrow S_{t+1}(c) = f\left(S_t(N(c))\right) \end{array} . \quad (2)$$

  f determines the state of the observed cell c at time $t+1$ according to the state of its neighboring cells at time t.
  When the transition function is not applied simultaneously for all cells, the updating system is said asynchronous and the local evolutionary rule is given by:

$$\begin{array}{c} f:\sigma^n \rightarrow \sigma \\ S_t(N(c)) \rightarrow S_{t+1}(c) = f\left(S_t(N_1(c)), S_{t+1}(N_2(c))\right) \end{array} . \quad (3)$$

  Where: $N_1(c) \subset N(c)$ and $N_2(c) \subset N(c)$ are two complementary subsets in the considered interaction system $N(c)$ with $N_1(c) \cup N_2(c)=N(c)$ and $N_1(c) \cap N_2(c)= \varnothing$.
  In this case, f determines the state of the observed cell c at time $t+1$ according to the state of cells of the interaction system $N_1(c)$ at time t and the neighboring cells of the interaction system $N_2(c)$ at time $t+1$.

The number of possible transition functions increases when the number of states of a cell and the size of the neighbourhood system increase, it is given by: $TF = k^{Nc}$ where $Nc = k^n$ is the number of combination of states in the neighbourhood system where k is the number of states.

3. Proposed method

Our approach consists in an iterative principle depicted in figure 2. The metaheuristic starts with an initial transition function randomly generated. Then, at each iteration, the metaheuristic sends to the cellular automaton a transition function X. the cellular automaton builds different shapes by applying ng_{max} (maximum number of

generations) times this transition function to an initial shape composed of a single cell in the middle of the lattice.

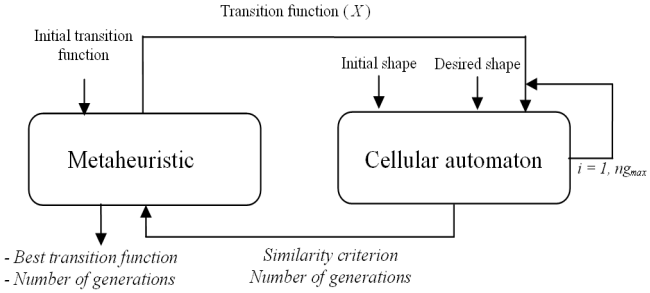


Fig. 2. Proposed approach

For instance, figure 6 gives six different shapes obtained with the same transition function and six values for the number of generations.

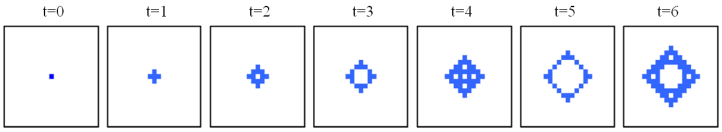


Fig. 3. Evolution of cellular automata

At each number of generation *t*, the obtained shape $Sh_{X,t}$ is compared to the desired shape Sh^*. The shape with the best similarity criterion $C(Sh_{X,ng},Sh^*)$ is recorded:

$$ng = t \quad \text{such as} \quad C(Sh_{X,ng}, Sh^*) = \max_{t=1,ng_{\max}} \left(C\left(Sh_{X,t}, Sh^*\right)\right) \tag{4}$$

Then, the cellular automaton returns to the metaheuristic $C(Sh_{X,ng},Sh^*)$ and the corresponding number of generations *ng*.

3.1. Studied cellular automaton

The characteristics of our cellular automaton are:

- *L* is a 2D regular lattice with the size 21*21.
- The set of states is $\sigma = \{0,1\}$ where "0" is an empty cell and "1" presents an occupied cell.
- In order to allow an occupied cell to become empty, we propose to consider the state of the observed cell in the neighbourhood systems. Moreover, the results obtained by Chavoya and Duthen showed that Moore and Von Neumann

neighbourhood systems perform better than Margolus and 2D-Radial one. So we consider the two following systems:

- Von Neumann: $N(c_{i,j})=(c_{i,j}, c_{i-1,j}, c_{i,j-1}, c_{i+1,j}, c_{i,j+1})$.
- Moore: $N(c_{i,j})=(c_{i,j}, c_{i-1,j-1}, c_{i,j-1}, c_{i+1,j-1}, c_{i-1,j}, c_{i+1,j}, c_{i-1,j+1}, c_{i,j+1}, c_{i+1,j+1})$.

i and *j* are the coordinates of the observed cell in the lattice.

- *F* is determined by the genetic algorithm.

In order to simplify the evolution of the cellular automaton, we propose to consider a synchronous updating system. In this way, the shape corresponding to *F* is the same for each run of the cellular automaton.

3.2. Metaheuristic

To use a simulated annealing based metaheuristic, we need to propose:
- a solution encoding for a transition function,
- a interaction system that modifies the transition function at each iteration,
- an observed function to compare two shapes.

Finally, we present the principle algorithm of the two used metaheuristics: iterated local search and inhomogeneous simulated annealing.

3.2.1. Solution encoding

A solution *X* is a vector coded in binary. It defines the transition function of the cellular automaton. Each element of this solution $S_{t+1}^{(i)}(c)$ represents the state of the observed cell *c* at time *t+1* according to the configuration *i* of its neighbouring cells. The length of solution is given by the number of local evolutionary rules *Nc*.

$$X = \left(S_{t+1}^{(0)}(c),...,S_{t+1}^{(i)}(c),...,S_{t+1}^{(Nc-1)}(c)\right). \quad (5)$$

Example

We consider a Von Neumann interaction system. The possible combinations of states of interaction system are 2^5. They are numbered from (0) to (*Nc*-1=32) in base-10 according to the lexicographical order of $N(c_{i,j})=(c_{i,j}, c_{i-1,j}, c_{i,j-1}, c_{i+1,j}, c_{i,j+1})$ in base-2, where the first combination presents an empty observed cell *c* surrounded by empty cells and the last combination is an occupied observed cell surrounded by occupied cells.

We present in Figure 4, an example of a transition function with Von Neumann interaction system. We have defined for each combination a new state by a local evolutionary rule. For example: in the combination (0) the next state of the observed cell *c* is "0" and the next state of the observed cell in the combination (32) is "0". We have 2^{32} possibilities to define this transition function. The sixteen first combinations describe the local evolutionary rules when the observed cell $c_{i,j}$ is empty and the sixteen next combinations when the observed cell is occupied.

Combination of neighbor cells	$c_{i,i}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	$c_{i-1,i}$	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1
	$c_{i,i-1}$	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	$c_{i+1,i}$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	$c_{i,i+1}$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
New state of	$c_{i,i}$	0	1	1	1	1	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	0	1	0	0	0	1	0	0	0	0	0	0

Fig. 4. Example of a transition function

The size of a solution does not depend on the size of the lattice; it depends only on the size of the neighbourhood system and the number of states of cell. When we use Moore neighbourhood system with 8 neighbouring cells and binary cells, the size of the solution is 512 and the number of possible transition functions is 2^{512}.

3.2.2. Neighborhood systems

We propose three neighbourhood systems; each system depends on the number of elements which are modified to get neighbour solution. The modification of an element of solution consists in replacing it by its complement to 1: $\overline{S_{t+1}^{(i)}(c)} = 1 - S_{t+1}^{(i)}(c)$.

Γ1-neighborhood system

One local evolutionary rule is chosen randomly and uniformly and is modified. In this neighbourhood system, all elements of a solution have the same probability (*1/Nc*) to be selected. The size of the neighbourhood system (number of neighbour solution) is given by the solution length: $Card(\Gamma_1) = Nc$.

Γ2-neighborhood system

Two local evolutionary rules are selected randomly and uniformly. Each element is chosen once. The probability to be selected is (1/ Nc) for the first element and (1/(Nc-1)) for the second one. The size of this neighbouring system is $Card(\Gamma_2) = Nc\ (Nc-1)$.

Γ3-neighborhood system

Unlike the two previous systems, in this one, the number of element to be modified is variable from one solution to another. We fixe M the maximal number of elements which could be modified, hence we choose $m \in [1, M]$ the number of local evolutionary rules to modify. The size of the neighbourhood system is the number of arrangements of *m* elements selected through *Nc*, it is given: $Card(\Gamma_m) = Nc!/(Nc-m)!$.

3.2.3. Objective function

We propose to compare two shapes *Sh1* and *Sh2* by using the mutual information [11]. The mutual information is a quantity to measure a correlation; it is used specially in the image registration domain. The advantage of this measure is the consideration of all possible transitions from one state to another. These transitions are given by the joint probabilities. When two shapes are identical, their mutual information is maximal.

$$MI(Sh1, Sh2) = \sum_{a\in\sigma}\sum_{b\in\sigma} p(a,b,Sh1,Sh2)\ln\left(\frac{p(a,b,Sh1,Sh2)}{p(a,Sh1)\,p(b,Sh2)}\right). \quad (\mathbf{6})$$

With:

- $p(a,b,Sh1,Sh2)=Card((i,j)\in L \text{ such as } (Sh1_{i,j}=a)\wedge(Sh2_{i,j}=b)/Card((i,j)\in L)$ is the joint probabilities to have a state a in the shape $Sh1$ and the state b in the shape $Sh2$.

- $p(a,Sh)=Card((i,j)\in L \text{ such as } Sh_{i,j}=a)/Card((i,j)\in L)$ is the marginal probability to have a state a in the shape Sh.

A comparison study between the mutual information and the similarity measure proposed by Garis, Duthen and Chavoya is given in [10]. This study shows that the mutual information is more sensible to detect the difference between two shapes.

We present in the Figure 5 an example of two binary shapes *Sh1* and *Sh2* to compare. The joint probabilities *p(a, b, Sh1, Sh2)* are presented in the Table 1. Joint probabilities

From these probabilities, we deduce the marginal probabilities *p(a, Sh1)* and *p(b, Sh2)*, they are respectively the sum of the joint probabilities by column and line.

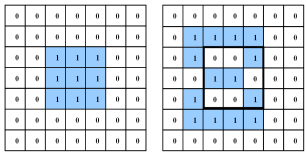


Fig. 5. Two shapes to compare *Sh1* in the left and *Sh2* in the right.

Table 1. Joint probabilities

P(a,b,Sh1,Sh2)		*B*	
		0	1
A	0	30/49	10/49
	1	5/49	4/49

p(0, Sh1)=40/49 ; p(1, Sh1)=9/49
p(0, Sh2)=35/49 ; p(1, Sh2)=14/49
MI(Sh1, Sh2)=0.013

For each solution X, the mutual information $C(Sh_{X,ng}, Sh^*)$ is computed where Sh^* is the desired shape and $Sh_{X,ng}$ the shape obtained by the ng applications of the transition function X.

3.2.4. Simulated annealing

Simulated annealing is an analogy to physical annealing of solid used in metallurgy. To improve the quality of a solid, from a liquid state of matter at high temperature which is gradually decreased, this leads the material to regain its solid shape. This process is used to have a stable structure of the metal which corresponds to a minimum free energy. It is an evolutionary process which therefore allows to avoid to stay in local optima of the objective function, occasionally allowing a decrease of this function.

In this work, we use an inhomogeneous annealing. The efficiency of simulated annealing depends on its parameters: initial temperature T_0, final temperature T_f and the factor of decrease of temperature T_d given by: $T_d=(T_0/T_f)^{1/NIter}$ where: *NIter* is the number of iterations of the simulated annealing.

The different steps of the simulated annealing are described by the algorithm 1.

```
Generate an initial solution X
Cobj := C(Sh*, Sh*), T := T0
While (T > Tf) ∧ (C(Sh_{X,ng} ,Sh*) ≠ Cobj ) do
    Choose X'∈ Γ(X)
    If C(Sh_{X',ng'} ,Sh*) ≥ C(Sh_{X,ng},Sh*) then
          X := X', ng:=ng'
    Else
          α ∈ [0,1]
          If α ≤ exp(-(C(Sh_{X,ng},Sh*) - C(Sh_{X',ng'},Sh*))/T) then
                X := X', ng := ng'
          End If
    End If
    T := T*Td
End While
Return X
```

Algorithm 1: principle algorithm of the inhomogeneous simulated annealing

3.2.5. Iterated local search

The basic simulated annealing based algorithm is the stochastic descent, which accepts the neighbour solution if its criterion is better or equal to the criterion of the current solution. This algorithm allows generally to find a local minimum. A simple way to leave a local minimum is to restart a stochastic descent from a new randomly chosen starting point. This scheme introduces the successive descents algorithm. The iterated local search (algorithm 1) follows this principle, but the new starting point is obtained by perturbing the local minimum. This algorithm allows, after a stochastic descent to accept any solution (by using another neighbourhood system) and to start again with a new stochastic descent [12].

Two neighbourhood systems are used, a first one Γ (Γ1, Γ2 or Γ3) for the stochastic descent and a second one Γ' (Γ is applied five times), generally larger than Γ, for the perturbation.

The algorithm has been to converge in probability if neighbouring system Γ' satisfies the accessibility property[13]. The proof of the converge lies in the fact that the iterated local search builds a Markov chain where any state can lead to an absorbing state and the absorbing states constitute the global optimal set.

A is the maximum of iterations without improvement, *best* corresponds to the best found solution, *k* is the number of iterations since the last improvement.

```
Generate an initial solution X
C_obj :=C(Sh*,Sh*) , i:= 1,k := 0,Best := 0
While (i ≤ NIter)∧( C(Sh_{X,ng} ,Sh*) ≠ C_obj )do
```

```
      Choose X'∈ Γ(X)
      If C(Sh_{X',ng'},Sh*) ≥ C(Sh_{X,ng},Sh*) then
            X := X',ng:=ng'
            If C(Sh_{X',ng'},Sh*) ≠ C(Sh_{X,ng},Sh*)then
                  k := 0
            End If
      End If
      k := k+1
      If k ≥ A then
            Choose X' ∈ Γ'(X)
            If C(Sh_{X',ng'}, Sh*) ≠ C(Sh_{X,ng}, Sh*) then
                  X := X',k:= 0, ng :=ng'
            End If
      End If
      If C(Sh_{X,,ng}, Sh*)> C(Best, Sh*) then
            Best := X
      End If
      i := i+1
End While
Return Best
```

Algorithm 2: principle algorithm of the iterated local search

4. Results

We have applied our approach to produce form a cell in the middle of the lattice different full and hollow shapes shown in Figure 6. The three first shapes (Square, Triangle, Diamond) represent the full shapes studied by Chavoya and Duthen [3]. The ten next shapes are hollow shapes generated by applying a given number of generations a given transition function. Shapes (M1…M5) are generated with Moore interaction system and shapes (VN1…VN5) are generated with Von Neumann interaction system.

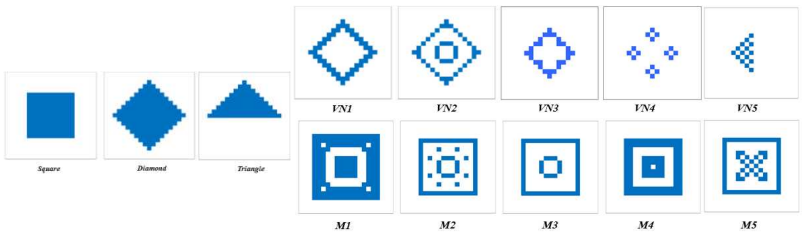


Fig. 6. The tested shapes

The obtained results for the combination of simulated annealing and a cellular automaton with Von Neumann interaction system are given in the Table 2 and with Moore interaction system in the Table 3. Parameters for the simulated annealing are: NIter =25.000.000 iterations, T_0 = 10. As it is a stochastic method, we have run it five

times in order to produce mean values and to test its robustness. For each shape, we give:

- Mean, SD, Min and Max representing respectively: the mean, the standard deviation, the minimum and the maximum of the mutual information for the five runs.
- *R* the ratio of the overlapping cells of the obtained shape and the preset shape to the total cells of the preset shape. This ratio is given by this equation:

$$R(Sh1, Sh2) = Card((i,j) \in L \text{ such as } (Sh1_{i,j} = Sh2_{i,j}))/Card((i,j) \in L). \quad (7)$$

Table 2. Results for simulated annealing and cellular automaton with Von Neumann

		Γ1					Γ2					Γ3				
Shape	**Objective value**	**Min**	**Mean**	**Max**	**SD**	**R**	**Min**	**Mean**	**Max**	**SD**	**R**	**Min**	**Mean**	**Max**	**SD**	**R**
Square	**0,587**	0,358	0,358	0,358	0,000	**86,70**	0,358	0,358	0,358	0,000	**86,70**	0,358	0,358	0,358	0,000	**86,70**
Triangle	**0,476**	0,476	0,476	0,476	0,000	**100**	0,476	0,476	0,476	0,000	**100**	0,476	0,476	0,476	0,000	**100**
Diamond	**0,633**	0,633	0,633	0,633	0,000	**100**	0,633	0,633	0,633	0,000	**100**	0,633	0,633	0,633	0,000	**100**
M1	**0,665**	0,330	0,334	0,336	0,002	**79,33**	0,330	0,334	0,336	0,002	**83,15**	0,330	0,334	0,336	0,002	**83,21**
M2	**0,429**	0,197	0,197	0,197	0,000	**87,53**	0,197	0,197	0,197	0,000	**87,53**	0,197	0,197	0,197	0,000	**87,53**
M3	**0,397**	0,251	0,251	0,251	0,000	**89,75**	0,251	0,251	0,251	0,000	**89,75**	0,251	0,251	0,251	0,000	**89,75**
M4	**0,566**	0,321	0,321	0,321	0,000	**84,21**	0,321	0,321	0,321	0,000	**84,21**	0,321	0,321	0,321	0,000	**84,21**
M5	**0,445**	0,137	0,137	0,137	0,000	**70,37**	0,183	0,184	0,186	0,001	**75,40**	0,186	0,186	0,186	0,000	**84,48**
VN1	**0,362**	0,362	0,362	0,362	0,000	**100**	0,362	0,362	0,362	0,000	**100**	0,362	0,362	0,362	0,000	**100**
VN2	**0,324**	0,324	0,324	0,324	0,000	**100**	0,324	0,324	0,324	0,000	**100**	0,324	0,324	0,324	0,000	**100**
VN3	**0,260**	0,260	0,260	0,260	0,000	**100**	0,260	0,260	0,260	0,000	**100**	0,260	0,260	0,260	0,000	**100**
VN4	**0,155**	0,155	0,155	0,155	0,000	**100**	0,155	0,155	0,155	0,000	**100**	0,155	0,155	0,155	0,000	**100**
VN5	**0,148**	0,148	0,148	0,148	0,000	**100**	0,148	0,148	0,148	0,000	**100**	0,148	0,148	0,148	0,000	**100**

Table 3. Results for simulated annealing and cellular automaton with Moore

		Γ1					Γ2					Γ3				
Shape	**Objective value**	**Min**	**Mean**	**Max**	**SD**	**R**	**Min**	**Mean**	**Max**	**SD**	**R**	**Min**	**Mean**	**Max**	**SD**	**R**
Square	**0,587**	0,587	0,587	0,587	0,000	**100**	0,587	0,587	0,587	0,000	**100**	0,587	0,587	0,587	0,000	**100**
Triangle	**0,476**	0,476	0,476	0,476	0,000	**100**	0,476	0,476	0,476	0,000	**100**	0,476	0,476	0,476	0,000	**100**
Diamond	**0,633**	0,633	0,633	0,633	0,000	**100**	0,633	0,633	0,633	0,000	**100**	0,633	0,633	0,633	0,000	**100**
M1	**0,665**	0,281	0,391	0,438	0,056	**88,03**	0,406	0,426	0,444	0,013	**91,47**	0,272	0,386	0,424	0,057	**87,64**
M2	**0,429**	0,240	0,318	0,409	0,062	**93,40**	0,173	0,238	0,290	0,044	**88,47**	0,259	0,269	0,285	0,008	**89,80**
M3	**0,397**	0,179	0,235	0,283	0,037	**93,13**	0,178	0,218	0,243	0,025	**91,13**	0,202	0,245	0,305	0,037	**94,73**
M4	**0,566**	0,519	0,536	0,553	0,013	**99,16**	0,361	0,405	0,472	0,037	**93,13**	0,368	0,390	0,422	0,019	**91,80**
M5	**0,445**	0,174	0,216	0,237	0,022	**89,69**	0,376	0,386	0,397	0,008	**97,78**	0,366	0,394	0,409	0,015	**98,17**
VN1	**0,362**	0,285	0,330	0,362	0,034	**98,67**	0,281	0,314	0,362	0,039	**98,00**	0,262	0,326	0,363	0,045	**98,33**
VN2	**0,324**	0,234	0,26	0,293	0,021	**97,56**	0,245	0,269	0,324	0,028	**98,44**	0,239	0,261	0,285	0,014	**98,11**
VN3	**0,260**	0,260	0,260	0,260	0,000	**100**	0,260	0,260	0,260	0,000	**100**	0,260	0,260	0,260	0,000	**100**
VN4	**0,155**	0,155	0,155	0,155	0,000	**100**	0,155	0,155	0,155	0,000	**100**	0,155	0,155	0,155	0,000	**100**
VN5	**0,148**	0,148	0,148	0,148	0,000	**100**	0,148	0,148	0,148	0,000	**100**	0,148	0,148	0,148	0,000	**100**

The iterated local search is used to generate the preset shapes with 25.000.000 iterations. We present in the Table 4, the obtained results with Von Neumann interaction system and with Moore interaction neighbourhood in the Table 5.

The obtained results by the iterated local search corroborate the obtained results by the simulating annealing:

- The optimal solution is obtained for the triangle and the diamond whatever the used interaction system for the cellular automaton and the neighborhood system for the metaheuristic.
- For the square, the optimal solution is obtained only with Moore interaction system, this behavior of cellular automaton is due to the structure of this shape.
- For the shapes (VN1,…,VN5), we obtained the optimal solution with Von Neumann interaction system for all runs, and results with good quality with Moore interaction system .
- The obtained results with the shapes (M1,…, M5) are not satisfying with all neighborhood system Γ but the obtained results by Moore are better results than the results with Von Neumann. Also, the iterated local search gives better ratio for these shapes than the simulated annealing. These results can be explained by the fact that these shapes are generated with Moore interaction system and the size of search space (2^{512}) in this case.
- We can see that with Γ3 neighborhood system, we obtain better ratio *R*.

Table 4. Results for iterated local search and cellular automaton with Von Neumann

		Γ1					Γ2					Γ3				
Shape	**Objective value**	**Min**	**Mean**	**Max**	**SD**	**R**	**Min**	**Mean**	**Max**	**SD**	**R**	**Min**	**Mean**	**Max**	**SD**	**R**
Square	**0,587**	0,358	0,358	0,358	0,000	**86,7**	0,358	0,424	0,457	0,036	**92,02**	0,356	0,356	0,358	0,001	**86,48**
Triangle	**0,476**	0,476	0,476	0,476	0,000	**100**	0,476	0,476	0,476	0,000	**100**	0,476	0,476	0,476	0,000	**100**
Diamond	**0,633**	0,633	0,633	0,633	0,000	**100**	0,633	0,633	0,633	0,000	**100**	0,633	0,633	0,633	0,000	**100**
M1	**0,665**	0,336	0,336	0,336	0,000	**83,37**	0,336	0,336	0,336	0,000	**83,37**	0,336	0,336	0,336	0,000	**83,37**
M2	**0,429**	0,249	0,249	0,249	0,000	**86,42**	0,249	0,249	0,249	0,000	**86,42**	0,249	0,249	0,249	0,000	**86,42**
M3	**0,397**	0,251	0,251	0,251	0,000	**89,75**	0,251	0,251	0,251	0,000	**89,75**	0,251	0,251	0,251	0,000	**89,75**
M4	**0,566**	0,321	0,321	0,321	0,000	**84,21**	0,321	0,321	0,321	0,000	**84,21**	0,321	0,321	0,321	0,000	**84,21**
M5	**0,445**	0,186	0,186	0,186	0,000	**84,48**	0,186	0,186	0,186	0,000	**77,67**	0,186	0,186	0,186	0,000	**86,75**
VN1	**0,362**	0,362	0,362	0,362	0,000	**100**	0,330	0,356	0,362	0,013	**99,77**	0,362	0,362	0,362	0,000	**100**
VN2	**0,324**	0,258	0,298	0,324	0,032	**98,67**	0,324	0,324	0,324	0,000	**100**	0,324	0,324	0,324	0,000	**100**
VN3	**0,260**	0,260	0,260	0,260	0,000	**100**	0,260	0,260	0,260	0,000	**100**	0,260	0,260	0,260	0,000	**100**
VN4	**0,155**	0,155	0,155	0,155	0,000	**100**	0,155	0,155	0,155	0,000	**100**	0,155	0,155	0,155	0,000	**100**
VN5	**0,148**	0,148	0,148	0,148	0,000	**100**	0,148	0,148	0,148	0,000	**100**	0,148	0,148	0,148	0,000	**100**

Table 5. Results for iterated local search and cellular automaton with Moore

		Γ1					Γ2					Γ3				
Shape	**Objective value**	**Min**	**Mean**	**Max**	**SD**	**R**	**Min**	**Mean**	**Max**	**SD**	**R**	**Min**	**Mean**	**Max**	**SD**	**R**
Square	**0,587**	0,493	0,547	0,587	0,036	**99,6**	0,538	0,577	0,587	0,019	**99,78**	0,587	0,587	0,587	0,000	**100**
Triangle	**0,476**	0,204	0,282	0,476	0,098	**91,9**	0,321	0,445	0,476	0,061	**86,7**	0,476	0,476	0,476	0,000	**100**
Diamond	**0,633**	0,633	0,633	0,633	0,000	**100**	0,633	0,633	0,633	0,000	**100**	0,633	0,633	0,633	0,000	**100**
M1	**0,665**	0,465	0,465	0,465	0,000	**91,2**	0,465	0,465	0,465	0,000	**91,14**	0,465	0,465	0,465	0,000	**91,14**
M2	**0,429**	0,299	0,350	0,403	0,039	**86,82**	0,247	0,358	0,418	0,063	**96,63**	0,379	0,403	0,429	0,018	**99,23**
M3	**0,397**	0,239	0,317	0,397	0,067	**94,85**	0,245	0,360	0,397	0,058	**97,62**	0,243	0,334	0,397	0,063	**96,85**
M4	**0,566**	0,464	0,530	0,566	0,037	**93,47**	0,456	0,530	0,566	0,038	**99,06**	0,512	0,553	0,566	0,020	**99,62**
M5	**0,445**	0,233	0,273	0,313	0,025	**89,92**	0,263	0,273	0,277	0,005	**88,65**	0,274	0,312	0,424	0,064	**92,97**
VN1	**0,362**	0,115	0,214	0,362	0,120	**78,4**	0,117	0,313	0,362	0,098	**92,91**	0,362	0,362	0,362	0,000	**100**
VN2	**0,324**	0,177	0,266	0,313	0,047	**97,29**	0,102	0,273	0,324	0,085	**94,85**	0,204	0,295	0,324	0,046	**99,01**

VN3	**0,260**	0,260	0,260	0,260	0,000	**99,95**	0,260	0,260	0,260	0,000	**99,95**	0,260	0,260	0,260	0,000	**100**
VN4	**0,155**	0,155	0,155	0,155	0,000	**100**	0,155	0,155	0,155	0,000	**100**	0,155	0,155	0,155	0,000	**100**
VN5	**0,148**	0,148	0,148	0,148	0,000	**100**	0,148	0,148	0,148	0,000	**100**	0,148	0,148	0,148	0,000	**100**

Conclusion

In this paper, we have proposed a metaheuristic based on the simulated annealing and binary cellular automata to generate 2D shapes from a single cell in the middle of the lattice. The first results are promising, other tests are in progress.

Our future works consist to test other neighborhood systems and other objective functions, in order to improve our results.We envisage to apply this method to generate complex shapes: from any initial shape with a number of states greater than 2, extend this model to produce 3D shapes, to consider the asynchronous updating system because it is nearest to the biological evolution and to include the different parameters considered in the morphogenesis as differentiation, division and death of cells.

References

1. Kenneth O., Stanly, K., Miikkulainen, R.: A Taxonomy for Artificial Embryogeny. Artificial Life Journal. Vol. 9(2), pp. 93--130 (2003).
2. Garis, H.D.: Artificial Embryology and Cellular Differentiation. Evolutionary Design by Computers, Morgan Kaufmann Publishers. USA (1999).
3. Chavoya, A., Duthen, Y.: Using a Genetic Algorithm to Evolve Cellular Automata for 2D/3D Computational Development. In: 8th Annual Conference on Genetic and Evolutionary Computation. USA (2006).
4. Shin, J.K.: Atlas of Patterns from One-rule Firing Cellular Automata. Proceeding of the Alife Conference. Denmark (2010).
5. Turing, A.M.: The Chemical Basis of Morphogenesis. Philosophical Transactions of the Royal Society of London. Vol. 237 (641), pp. 37--72 (1952).
6. Bai, L., Manolya, E., Breen, D.: Automated Shape Composition Based On Cell Biology and Distributed Genetic Programming. GECCO'08. pp. 1179--1186. USA (2008).
7. Wolfram, S.:Universality and Complexity in Cellular Automata. Physica D10. North Holland (1984).
8. Mitchell, M., Crutchfield, J.P., Das, R.: Evolving Cellular Automata with Genetic Algorithms: A review of Recent Work. In: 1st International Conference on Evolutionary Computation and its Applications. Russia (1996).
9. Back, T., BreuKelaar, R.: Using Genetic Algorithms to Evolve Behavior in Cellular Automata. In: 4th International Conference on Unconventional Computation. Spain (2005).
10. Aïboud, F., Grangeon, N., Norre, S.: Generation of 2D Shapes with Cellular Automaton and Genetic Algorithm: Extension of Chavoya and Duthen Work. In: 9th Metaheuristics International Conference MIC. Italy (2011).
11. Gray, R.: Entropy and Information Theory. Springer-Verlag. USA (1990).
12. Lima, C.F., Lobo, F.G.: Parameter-Less Optimization with the Extended Compact Genetic Algorithm and Iterated Local Search. GECCO 2004, LNCS 3102, pp. 1328–1339, (2004).
13. Fleury, G.: Méthodes stochastiques et déterministes pour les problèmes NP-difficiles. Phd Thesis, Blaise Pascal University, France (1993).

Posters

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

GA-based System for Achieving High Recall and Precision in Information Retrieval

Ammar Al-Dallal[1] and and Rasha Shaker Abdulwahab[2]

[1]School of Information Systems Computing and Mathematics, Brunel University, U.K
[2]College of Information Technology, Ahlia University, Bahrain
{Ammar.AlDallal@brunel.ac.uk,rasha_sh_abdul-wahab@ahliauniversity.edu.bh}

Abstract. The main purpose of this paper is to highlight the importance of retrieving relevant documents by developing new system capable of managing and organizing the retrieved documents. In particular, attention will be paid to Genetic Algorithm for developing such system. Genetic Algorithm influences the process of Information retrieval by directly contributing to retrieve relevant documents through its mechanism which was inspired by nature. In this paper we propose GA system that has promising results in terms of recall and precision. These results are achieved via developing a new hybrid crossover operator and a new fitness evaluation function. Results are compared with two well known techniques applied in IR domain which are Okapi-BM25 and Bayesian interface network model. The comparative study shows that precision and recall of the retrieved documents by the proposed method outperforms these two techniques.

Keywords: Genetic Algorithm, information retrieval (IR), Terms Proximity, Term Distance, Crossover, Evaluation function

1 Introduction

Rapid growth to the number of Web pages needs continuous challenges for helping Web users to find relevant information from the Internet. Information Retrieval (IR) is an essential and useful technique for Web users so studying of such system have increased since the coming of the World Wide Web.

In recent years, emphasis in the applicability of Artificial Intelligence (AI) has been increased with IR. One of the AI areas is Evolutionary Computation (EC) which is based on models of natural selection. A classical and important technique in EC is Genetic Algorithm (GA). The GA is biologically inspired and has many mechanisms inspired by natural evolution. Because of its parallel mechanism with high-dimensional space, GA has been used to solve many of scientific and engineering problems. This in turn led to encourage researchers for using this algorithm in IR. Besides, GA plays an important approach to provide suitable information for the user's needs.

IR and GA integrated to avoid web users suffering from specific problems when trying to retrieve useful information such that:

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

– Many of the retrieved documents are not related to the user query.
– Some of relevant documents have not been retrieved yet[16].

However, retrieving relevant information is not a simple process. However, the complexity of this process is further increased by the fact that more and more of this information appears in natural language and not in structured formats [10].

The rest of this paper is organized as follows: in section 2, we describe problem statement. Section 3 describes the main objectives of this research. Section 4 discusses related works. Document representation is introduced in section 5. Section 6 introduces the proposed approach. Section 7 represents the experimental results of the proposed method. Section 8 gives the conclusions of this study.

2 Problem Statement

Given a user query, there is a need to have an IR system that is able to retrieve all and only relevant documents to the user query. The performance of the developed IR is evaluated using two well known measures which are precision and recall.

3 Research Objectives

The objective of this paper is to explain how the new crossover operator can affect the efficiency of the proposed GA system. Another objective presented in this study is to develop a new fitness function. This new function will be used to evaluate documents and retrieve the related documents at high rank.

4 Related Work

The long history of GAs with IR is presented by integrating GA in IR with the aim of solving IR problems. One of these approaches is the one developed by Kim and Zhang[8][9]. They proposed a GA-based retrieval method which is used to learn the importance of HTML tags. This method shows an improvement of average precision when using tagged information over non-tagged information.

Picarougne et al[16] reported their experience in designing different fitness function for web search using Genetic Programming (GP). They tested this set of fitness functions with other existing order-based fitness functions on the task of ranking function discovery. Their results show the design of such fitness functions lead to an increase in performance.

An integrated framework for discovering and optimizing ranking functions proposed by Weiguo et al[23]. These novel ranking functions are discovered by using GP technique. The structural and statistical information available in HTML documents collected intelligently in their work. However, combining the scores of various well-known ranking functions is achieved by using GA. Their framework is tested on the well-known TREC collection of web documents. They viewed a

significant increase in retrieval performance and their results outperformed the other well-known ranking functions.

Finally, Fan and Fox [7] introduced another technique in which vector space model was used as a base to represent the documents. In their study, they proposed different fitness functions include Okapi-BM25, average precision and CHK fitness functions [11]. They decided after their comparison the suggested fitness functions are work well in the web search context.

Based on the techniques developed by different researchers, the following comments earned:

- Most of the existing IR systems used vector space model to present the document collection.
- Non-structured documents used in most of the mentioned studies as a way to represent their documents.
- Although some of presented studies are applied on structured documents like HTML but they focused to obtain the proper HTML weight.
- Finally, some studies are focused on analyzing the links within the document rather than the terms within the document.

After analyzing the features of most existing IR systems and based on some successful experiments mentioned above as a benchmark, a new challenge will be made. This challenge tries to find out if the proposed IR could retrieve more relevant documents and display them at high rank.

5 Document Representation

Effective adoption way to represent documents influenced the scientists' thought in IR arena. Most of existing documents used by IR system can be either plain text, semi-structured (i.e., HTML (HyperText Markup Language) documents) or structured. Since most of web-documents are written in HTML[8], that is why this format adopted in this study to perform the proposed system.

Selecting proper index terms[15] and indexing model can be considered one of the main concern in documents representation. Many indexing models were developed in IR such as Boolean indexing model[21], vector spacing model [2][19] and latent Symantec indexing model[2][25]. However, several limits are raised with these models which are:

1. These models require large space to store the index,
2. Require long time to retrieve the required terms, and
3. The stored information in the index is limited.

To overcome these drawbacks, a well-known indexing schema is developed in our system. This indexing schema is called inverted index [10]. It is perhaps the most important index method used in search engines as stated by Liu. [10].

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

6 Document Indexing

GA is run on set of documents which are represented in special template. This representation is done through process called indexing. One of the important reasons for indexing documents is the difficulties involved in implementing document-term weighting [15]. This indexing schema not only allows different retrieval of documents, but also very fast to build.

The power of the proposed system comes from the indexing technique used. However, it is created by improving the known inverted index. In the traditional inverted index[26], the schema is created by adding the document identifier and the position of word in the collection. Word position data is a list of offsets at which the words occur in the document. Such occurrence information (i.e. document ID and word position data) for each word is expressed as a list, called "inverted list". This schema in our proposed technique is improved by encapsulating extra information related to the indexed term. This information includes term weight, term frequency and term position or offset. The term weight is evaluated based on the HTML tag weight that this term falls in. Term frequency represents the number of times the term appears within the document. However, the position or offset of the term within the document will be used in calculating the term distance.

During index creation, the offset or position of each term within the document is stored in the index. This position is used to find out the minimum term distance (MTD) between all query keywords within the document and the first appearance (FA) of the minimum distance. FA is obtained depending on the calculated value of MTD.

An Oracle database is used to store and access the improved inverted index with the aim of promoting document retrieval in the face of relevant queries.

The essence of our approach in the proposed technique is simplifying the process of querying the information from the database. In addition, the applicability of the proposed system for dealing with large amounts of documents increases. Because of the nature of the Oracle engine this will help to achieve what we seek.

7 GA in IR

GA's form in our information retrieval system is presented in this section.

7.1 Chromosomes and Population

The simple method to generate initial population is to create individuals randomly. The vector of document descriptors is encoded as a chromosome. That is, a chromosome is represented as a document descriptors vector, i.e., $ind = (d_1, d_2, ..., d_n)$ where $d_i \in [1..\texttt{max document size}]$.

Despite the rapid implementation of this method and require less processing but it delays the process of finding optimal solution. In order to avoid this, d_i is

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

initially generated with some restrictions. So that, this will enhance the quality of individuals and in the same time this will not affect the performance of the system in terms of time. The adopted restrictions in this approach include:

1. The document must include at least one keyword from the user query, i.e., given a set of terms d_i where $D = d_1, d_2, d_n$, the document D is selected such that $D \bigcap Q \neq \varnothing$ where query $Q = q_1, q_2, \ldots,$ and q_i is the query term within the user query.
2. The query must contain more than one keyword, i.e., $Q = q_1, q_2, , q_n$ where $n > 1$.

7.2 Fitness Function

For our ranking technique, the decision about whether to take or reject a document depends only on the calculated value by the proposed fitness function. That is some definitions about local and global factor with term weight's concept will be described in the following subsections to explain the suggested fitness function.

Local Factors Verses Global Factors Index Adoption of new fitness function is the core of the proposed IR system in this study. This function is a performance measure or reward function that measures the relativity of documents to the user query. To define the proposed fitness function for information retrieval, we need to define (1) local factors; and (2) global factors. Local factors are those earned from the document under consideration such as document size, number of unique terms within the document and total number of a specific term within the document. On the other hand, global factors earned from the search space such as total number of documents in the space, total indexed terms and total number of a specific term within the search space [12].

For large documents set, local factors are used. This is practical when the number of documents in the set is unlimited like the web. On the other hand, global factors need to be obtained if the number of document set is limited. This paper has avoided the needed extra time for extracting global factors. Therefore, only local factors are computed in the proposed fitness function.

Term Weight Term weight is the second idea that needs to be clarified in this section. In HTML documents, tags play an important role in highlighting the importance of some terms within a document. Each HTML tag has a specific weight depending on it's importance[1]. Terms will have weights ranging from 1 if it is normal text within the body tag to 6 if it is in the title tag. While tag weights is summed if the term appears in nested tags. This weight will be one of the factors used in the new fitness function.

The Proposed Fitness Function Indeed, two new ideas are used in the proposed evaluation function with HTML term weight which includes:

* keyword proximity (i.e., term distance) and
* Number of unique query keywords that exist in the document.

Keyword proximity is computed by finding out minimum term distance (MTD) between all queries keywords within the document and first appearance (FA) of the minimum distance found from previous term.

Definition 1: **Minimum Term Distance (MTD)** is the shortest distance between all query's keywords. A large value is initially assigned to MTD whereas this value should be greater than document size. While, if the possibility of distance between query keywords is found, a shorter distance value is assigned to MTD. However, the large value of MTD points out there is some keywords are missing from this document.

Definition 2: **First Appearance(FA)** of the minimum distance is calculated by relying on MTD value. This factor is computed by dividing the document size over the average distance of MTD. Thus, the resulting value is divided again on value of the document size.

By using the above notations (factors), the proposed evaluation function is represented as in 1. Table 1 illustrates the terminologies of equation 1.

$$f(document) = a\frac{\sum_{i=1}^{k} k_{ui}}{K} + b\frac{\sum_{i=1}^{k} k_{ui} - 1}{\sum_{i=1}^{k-1} min(d_i, i+1)} + c\frac{\overline{\frac{F}{avg(\sum i=1^{k-1} min(d_i, i+1))}}}{F}) + d\log\frac{\sum_{i=1}^{k} w_i}{\sum_{i=1}^{k} k_i} \quad (1)$$

Table 1. Description of Terminals of Equation 1.

Terminology	Description
k_{ui}	Unique query keyword exists in the document
K	Total number of unique keywords in the query
$d_i, i+1$	Distance between term i and term i+1of the query terms
F	Document size (total number of terms in the document)
w_i	Weight of term i in the document
a, b, c and d	Weighting factors for each component

This function has four components. First one is the unique number of the keywords of the query that exist in the document. This factor reflects how many of the query terms are exist in the document. This component has maximum value of one in case of all query keywords exist in the document.

Second component employ the minimum distance between query keywords in the document. It is evaluated by subtracting one from the number of unique keywords in the document. The resulting value is divided on the shortest distance between query terms in the document.

Example 1: to explain the manner of calculating component 1 and 2 , assume

the document's content is the following string of length 12:
ABCDEFGHCIJD
and assume the query value is CDF, while the offset of these words within the document are 3,4, and 6 respectively.
The MTD and the average position of minimum distance between query terms are calculated as follows:
MTD= min(C,D) + min (DF) = 1+2 = 3
$avg(\sum_{i=1}^{K-1} min(d_i^{i+1}))$ = 12/ [(3+4+6)/3] /12 = 0.231

Third component depends on the position of first occurrence of MTD. It is evaluated by dividing the document size F on the average position of minimum distance between query terms (i.e., $avg(\sum_{i=1}^{K-1} min(d_i^{i+1}))$). Thus, the resulting value is divided again on the document size. In the case where the keywords are close to the beginning of the document, the value of this component will be high. Also, this factor will return a value close to one if all query keywords are exist and the positions of these keywords are close to the beginning of the document.

$\log \frac{\sum_{i=1}^{k} w_i}{\sum_{i=1}^{k} k_i}$ is a symbol of the fourth component in the proposed evaluation function to represent the average weight of query terms in the document. The maximum value of this component is one if the summation of terms' weight is 10 times greater than the frequency of these terms. Log function is used to control the upper limit of this component.

7.3 Genetic Recombination Operators

In our proposed approach genetic operators are applied after creating the initial population. The same process is repeated until a problem solution is found.

Crossover: The most common form of crossover operator is applied with a predefined probability on two selected individuals of a population to generate new offsprings. Indeed, several inherited features from the selected parents will be given to the resulting offspring[20][22].

The proposed crossover operator is combination of one-point crossover, reordering crossover [22]and fusion crossover [26]. In one-point crossover, a randomly point is selected to perform exchange of genes. While, descending order of chromosomes' genes based on their fitness is adopted in reordering crossover operator. The rationale behind using ordered crossover method over other methods is the need to inherit the good genes and pass the good building blocks to the resulting offspring. On fusion crossover[20] only one offspring is generated from the two selected parents. In this operator, the offspring inherits the genes from one of the parents with a probability according to its performance. The advantage of this technique is the good genes of both parents are inherited to the offsprings.

The proposed crossover operates on the two parents x and y. These two members are randomly selected using tournament selection from current population P to produce one offspring O. Descending order of chromosomes' genes based

on their fitness is applied firstly. The generated chromosomes $\overline{x}$ and $\overline{y}$ after the process of ordering illustrated in Fig1. Then one-point crossover between $\overline{x}$ and $\overline{y}$ of length L is adopted. $\overline{x}$ and $\overline{y}$ are aligned with each other and one crosspoint cp is chosen at random over the range $[1..L]$.

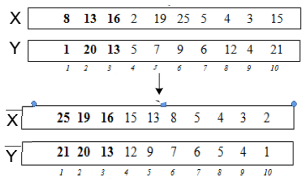


Fig. 1. X and Y after the process of ordering.

The first O's genes $[O_0, \ldots, O_{cp}]$ are copied from the candidate parents having greatest genes value. The remaining genes of O are copied from the second parent starting from the most left position. One must consider the uniqueness of the copied gene through the process of copying the remaining genes from the parents (each gene can occur only once). This is implemented by excluding the genes that are exist in O. If O is not up to the specified length, the fitness values of other genes in both parents are compared starting from $cp + 1$. The gene that has a higher fitness value contributes to O. This is done to generate offspring with appropriate genes from each parent.

Fig 2 gives an example of the proposed crossover. The two candidates $\overline{x}$ and $\overline{y}$ which are shown in Fig 1 are considered the contributors. The value of cp to show the mechanism of this operator is 3. $\overline{X}$'s genes along with the fitness values are considered the first three genes of O. That is because of the gene of $\overline{x}$ has greater fitness value than the first gene of $\overline{y}$. The three other genes of O are copied starting from the most left position of $\overline{y}$. Then a competition between genes in both $\overline{x}$ and $\overline{y}$ is done to complete the creation of O.

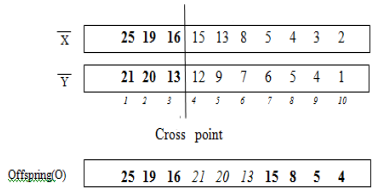


Fig. 2. The processes of the proposed crossover.

Mutation: Mutation is applied to the chromosomes which are forwarded by the crossover operator[8][9]. Mutation is done by replacing a new random value by a randomly selected g_i. This process is done after checking the uniqueness of the selected g_i.

7.4 Selection Operator

In this study, binary tournament selection [9] is adopted. However, two individuals are randomly chosen from the population. The one with highest fitness value is candidate to be the winner. A mating pool is generated by following the same process of this operator.

7.5 Termination Criteria and Solution

The iterative processes of our approach will be stopped by one of the following termination criteria:

- There is no improvement have been achieved from previous generation. For each generation, the generation performance is measured as summation of fitness value of all individuals within the generation. However, the upper limit is reached if the difference between two consecutive generations is less than the specified threshold, i.e., $E(G_i) - E(G_{i-1}) < threshold$. In order to have better solution, this threshold is set to very small value, Or
- An upper limit on the number of generations is reached.

In both cases, the solution is one individual of the last generation that has maximum fitness value.

8 Experiments and Results

This section describes the experimental setup used. Actually, the proposed system is examined on a document set of 8350 using 75 different queries; the length of the queries ranges from 2 to 5.

8.1 Data Set Description

The data set used in this system is Carnegie Mellon University data set. The collected data is an HTML documents gathered from computer science departments of various universities in January 1997 by the World Wide Knowledge Base project of the CMU text learning group[24]. This set is grouped into seven categories, named: course, department, faculty, project, staff, student and others.

8.2 Coefficients and GA' Parameters Setting

In most of our experiments, we gave high importance to the components that reflect the term distance and number of referenced keywords of the query. Actually, 0.3 sets for coefficients a, b and c to give high importance to the components that use term distance. While 0.1 is set to d so that lower importance is given to the weight component.

All the control parameters used in our genetic algorithm determined experimentally. The crossover probability is 80%. The mutation probability is 0.3. Moreover, the maximum number of generations is 50 with population size of 100 and chromosome length of 50.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

8.3 System's Evaluation

The results of the proposed system are evaluated by using precision and recall measures. Recall is defined as the percentage of relevant retrieved documents to the total number of relevant documents. While, the precision is defined as the percentage of relevant retrieved documents to the total number of retrieved documents. A combined method of recall and precision is used to evaluate the validity of the proposed system.

Precision versus recall is also used by adopting two different methods to measure system's result. The first method[8] is Precision at Rank N (P@RankN) and Recall and Rank N (R@RankN) where N is multiple of 10. In this method, the retrieved documents are ranked in descending order based on the fitness value to calculate the average of precision and recall. While, the second method [17] obtains the precision when recall is multiple of 10%.

8.4 System's Performance

An experimental study is conducted to evaluate the performing of the proposed fitness function. Two evaluation functions using 75 different queries are used here. One of them is OKAPI- BM25[18] while the second is Bayesian interface network model[8]. Eq. 2 and Eq. 3 pictures the form of these two functions. The results of these two functions are compared with the proposed evaluation functions' results.

The aim of these experimental studies is to examine whether the new evaluation function achieves a better performance than either the OKAPI- BM25 or Bayesian interface network model.

$$w_i = (d_t * H + (1 - d_t) * \frac{log(tf_i + 0.5)}{log(maxtf_i + 1.0)} * \frac{log\frac{N}{n}}{logN} \tag{2}$$

$$f(d) = \sum_{T=Q} \frac{(k_1 + 1) * tf}{k_1 * ((1 - b) + b\frac{length}{length_{avg}} + tf * log\frac{N - df_{0.5}}{df + 0.5}} \tag{3}$$

8.5 Results Analysis Using P@N and R@N Measure

It is obvious from the results shown in Fig 3 the proposed system has highest average in the precision values which is 75% in first top 10 ranked documents. Besides, the two other models which are Bayesian network interface model and Okapi-BM25 retrieved only 56% and 57% respectively. The improvement in precision values (32%) of the proposed evaluation function than that OKAPI-BM25 and Bayesian interface network model shows that the performance of the former is more efficient than the latter.

Another point highlighted in Fig 4 is the proposed system managed to retrieve 87% of related documents at level 45. In contrast, Bayesian network interface and Okapi-BM25 retrieved only 75% and 71% of related documents.

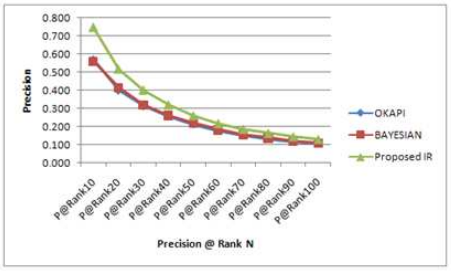


Fig. 3. Comparison of Precision at rank N for the three evaluation functions.

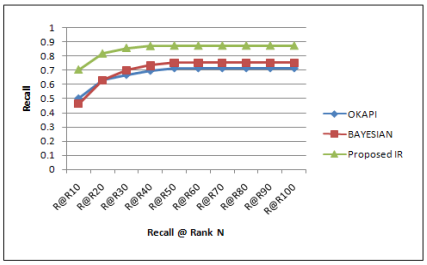


Fig. 4. Comparison of Recall for the three evaluation functions.

8.6 Results Analysis Using Recall-Precision Measure

The average of recall- precision is examined in this section. The proposed system's performance starts with precision of 0.77 when the system retrieves 10% of relevant document. Fig 5 pictures the results of this experiment. Besides, the precision of the proposed system varies between 0.91 and 0.95 until reaching recall of 100%. So, only 5-9% of retrieved documents are not relevant to user query and they appear in low rank until the system will retrieve all relevant documents.

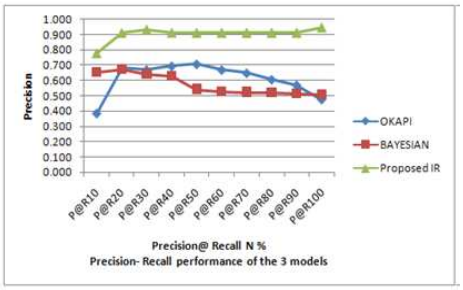


Fig. 5. Average recall-precision relationship for the three evaluation functions.

On the other hand, the precision of Bayesian network interface model starts from 0.65 at recall of 10% . However, 0.51 of precision value is achieved at the 100% recall. This means that almost half of the retrieved documents are not relevant. While OKAPI-BM25 shows unstable performance. It starts with precision of 0.39 at recall 10. The increase of the retrieved documents in OKAPI-BM25 begins at level 40 to achieve 0.71 and then begin to decrease until it reaches the value of 0.48.

Therefore, the efficiency of the achieved outcomes is the result of the proposed system has adopted a set of factors. These factors include the following:

- Terms' frequency within the document ,
- Terms' importance based on HTML tag and
- Terms' position within the document.

9 Conclusion

In this study, a new evaluation function is used to retrieve relevant documents in IR by applying GA. Such function is useful for evaluating the documents independently of other documents in the collection. As well as, the probability of query keywords within the document and the terms proximity concept are used. Another objective introduced in this study is the hybrid crossover operator. The proposed crossover operator helps to produce very high quality offspring that will speed up the process of finding the optimal solution.

The proposed system has been implemented and tested on dataset from Carnegie Mellon University. A comparison is done with two well known evaluation functions which are Okapi-BM25 and Bayesian network interface model. The obtained results presented in this paper shows the quality of retrieved documents over others. Indeed, the proposed system reaches precision of 100% for many queries at first top 10 retrieved documents.

References

1. Al-Dallal, A., Abdul-Wahab, R.S., Genetic Algorithm Based to Improve HTML Document Retrieval, Second International Conference on Developments in eSystems Engineering (DESE), 2009, Page(s): 343 - 348.
2. Aly, A, Applying Genetic Algorithm In Query Improvement Problem, ln: Information Technologies and Knowledge Vol.1, 2007, p 309 - 316.
3. Billhardt, H., Borrajo, D., and Maojo, V. Using genetic algorithms to find suboptimal re-trieval expert combinations. In Proceedings of SAC. 2002, 657-662.
4. Cutler, M., Deng, H., Maniccam, S., Meng, W., A new study on using HTML structures to improve retrieval, Tools with Artificial Intelligence, Proceedings. 11th IEEE International Conference on Tools with AI, Volume , Issue , 1999 Page(s):406 - 409. –31
5. Desjardins, G., Godin, R., Proulx, R., A Genetic Algorithm for Text Mining. Proceedings of the 6th International Conference on Data Mining, Text Mining and their Business Ap-plications, Vol. 35, pp. 133-142, 2005.

6. Deng-Yiv Chiu *, Ya-Chen Pan, Shih-Yi Yu: An automated knowledge structure construction approach: Applying information retrieval and genetic algorithm to journal of Expert Systems with Applications. 36, 94389447 (2009)
7. Fan, W., Fox, E., Pathak, P., Wu, H., The effects of fitness functions on genetic program-ming-based ranking discovery for Web search, Research Articles, Journal of the American Society for Information Science and Technology, v.55 n.7, May 2004, p.628-636.
8. Kim, S., Zhang, B., Genetic Mining of HTML Structures for Effective Web-Document Re-trieval, Applied Intelligence, v.18 n.3, p.243-256, 2003.
9. Kim, S., Zhang, B., Web-Document Retrieval by Genetic Learning of Importance Factors for HTML Tags. In Proceedings of PRICAI Workshop on Text and Web Mining2000. pp.13-23.
10. Liu, B., Web Data Mining, Springer-Verlag New York, LLC, Dec 2006, p204- 208.
11. Lopez-Pujalte, C., Guerrero-Bote, V.P., de Moya-Anegon, F., Genetic algorithms in relev-ance feedback: a second test and new contributions, Information Processing and Management 39 pp (2003) 669-687.
12. Marghny, M. H., Ali, A. F., Web Mining Based On Genetic Algorithm, AIML 05 Confe-rence, 19-21 Dec 2005, CICC, Cairo, Egypt.
13. Milutinovic, V., Cvetkovic, D. Mirkovic, J., Genetic Search Based on Multiple Mutations, IEEE Computer, pp. 118-119, November 2000.
14. Minaei-Bidgoli, B., Punch, W., Using genetic algorithms for data mining optimization in an educational web-based system. In Genetic and Evolutionary Computation Conference, Chicago, USA, 2003, pp. 2252-2263.
15. Pathak, P., Gordon, M. Fan, W., Effective information retrieval using genetic algorithms based matching functions adaption, In: Proceedings of: 33rd Hawaii International Conference on Science (HICS), Hawaii, USA, 2000.
16. Picarougne, F., Monmarche, N., Oliver, A., Venturini, G. 2002, Geniminer: Web mining with a genetic based algorithm, In: Proceedings of the IADIS International Conference WWW/Internet, Lisbon, Portugal p. 263-270.
17. Radwan, A., Abdel Latef, B., Ali, A., Sadek, O., Using Genetic Algorithm to Improve In-formation Retrieval Systems, Proceedings Of World Academy Of Science, Engineering And Technology, Vol 17, Dec 2006, p 6-12Y.
18. Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., Gatford, M.. Okapi at TREC-4. In Harman, D. K., editor, Proceedings of the Fourth Text Retrieval Conference, pages 73-97. NIST Special Publication 500-236, 1996.
19. Tian, C., Tezuka, T., Oyama, S., Tajima, K., Tanaka, K., Improving web retrieval precision based on semantic relationships and proximity of query keywords. In: Bressan, S., Kung, J., Wagner, R. (eds.) DEXA 2006. LNCS, vol. 4080, pp. 54-63. Springer, Heidelberg (2006).
20. Vrajitoru, D., Crossover Improvement For The Genetic Algorithm In Information Retrieval. Information Processing and Management ,1998, 34(4) , 405-415.
21. Vrajitoru, D., Genetic Algorithms in Information Retrieval. AIDRI97: Learning; From Natural Principles to Artificial Methods. , Genve, June. 1997.
22. Vrajitoru, D., Large Population or Many Generations for Genetic Algorithms ? Implications in Information Retrieval. Soft Computing in Information Retrieval. Techniques and Applications, Physica-Verlag, Heidelberg, 199-222.
23. Weiguo Fan a, Praveen Pathak b,., Mi Zhou: Genetic-based approaches in ranking function discovery and optimization in information retrieval A framework. 47, 398407 (2009)
24. Carnegie Mellon University Data Set [Online]. Available: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/

25. Description of LSI, [Online]. Available: http://en.wikipedia.org/wiki/Latent semantic indexing
26. LEE, W., Hierarchical Web Structure Mining, [Online]. Available: http://www.ieice.org/ de/DEWS/DEWS2006/doc/2A-v1.pdf

Tuning GPGPU Based Hydrogeological Simulations by Means of Evolutionary Optimization

Jan Quadflieg[1], Ulf Zimmermann[2], Claudius Bürger[2], Mike Preuss[1], Günter Rudolph[1], and Olaf A. Cirpka[2]

[1] Chair of Algorithm Engineering, Computational Intelligence Group, Dept. of Computer Science, Technische Universität Dortmund, Germany
Email: firstname.lastname@tu-dortmund.de
[2] Center of Applied Geosciences, Eberhard Karls Universität, Tübingen, Germany
firstname.lastname@uni-tuebingen.de

Abstract. Graphics processing units are nowadays becoming a widely used and cheap parallel computing platform that is even used to run extensive simulations. They are highly configurable, and although vendor defaults are known and optimal configurations can be determined at the instruction level, this is not possible for more complex simulation codes as often employed in hydrogeology. We show that by choosing non-default configurations, speedups of **factor 10** and more can be achieved. In addition, we provide a simple evolutionary algorithm that helps detecting the best parameters much faster than by enumeration or random search, thereby enabling rapid and fully automated configuration.

1 Introduction

Driven by the demand of the entertainment industries, e.g. the games industry, modern *graphics processing units* (GPUs) have become more and more powerful in the last years. It makes sense to use them as general purpose, massively data-parallel *single instruction multiple data* (SIMD) co-processors for scientific computations. In this role, they are also termed *general purpose GPUs* (GPGPUs). It goes without saying that an enormous number of personal computers would be potentially available for parallel computing instead of being challenged most by screen saver programs if it were easier to run existing code on their GPUs. However, this is currently not possible for several reasons. One is that GPUs are not able to handle the full instruction set of a processor but a subset, with vendor specific extension for handling data parallelism. This is dealt with in more detail in section 2.

The other reason is that many algorithms are not easily parallelized as they were designed with a sequential machine in mind. In this case, according to the famous Amdahl's law, the possible speedup by parallelization is bound by the amount of sequential computations necessary. Fortunately, many modern simulation approaches can be parallelized to almost 100% because they already break down the modeled (2D or 3D) space

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

into small finite elements. Such models are ubiquitous e.g. in hydrogeology. One example is given in section 3. This simulation is sped up considerably by transferring it from a CPU to a GPU. However, computation can still take days so that further acceleration is highly desirable.

It comes to mind that for actually running any GPU code, one has to set several parameters. The hardware vendors provide defaults and it is commonly believed that these are quite robust. However, it is also well known that misconfigurations are possible and setting parameters well may make a difference even on the instruction level. Zhang and Owens [9] propose an approach which consists of modeling the execution of code on the GPU, to determine bottlenecks in the instruction pipeline, shared memory or global memory accesses. Building the model actually involves disassembling the machine code. Thus, this approach may provide detailed insight into the performance characteristic of the code but is hardly applicable to complex real-world problems. Porting a complicated real world problem like the hydrogeological simulation discussed in this work to a GPU is hard enough, so that manual micro tuning of e.g. memory access patterns to increase performance is surely out of the scope of most programmers.

Davidson and Owens [2] work towards an automated process to tune parameters for GPU computations but their methods are closely related to the four problems they analyzed and therefore not applicable to a real world problem with unknown characteristics. It is interesting to note that the results from this work actually caused *Nvidia* to update the default parameters for their GT200 line. Unfortunately, an older generation of cards (GT80) performs worse with the new parameter set, which leads Davidson and Owens to the conclusion that an automated tuning for the hardware *actually used* (instead of using some general default parameters) is highly desirable. We strongly agree with that.

Tuning GPU parameters is however different from tuning *evolutionary algorithm* (EA) parameters as the search space is discrete, relatively small and heavily constrained. Additionally, one may experience measuring effects originating from machine load, clock and numeric imprecision which may even change the global optimum and at least provide an additional difficulty for any search process. However, measuring the effects of one parametrization is usually much faster than running the full computation, which enables trying out at least some set of possible candidates without much loss.

By first enumerating all possible configurations, we find that on the treated hydrogeological problems, **speedup factors of 10** or more are possible. In order to establish a fully automated tuning process that provides at least a near optimal solution while only using a very small set of samples, we adapt a simple EA to the GPU tuning problem, with very promising results (section 5).

Note that our approach is inverse to the one of running an EA on a GPGPU as e.g. described in [7]. Instead, we use an EA as fast tuning heuristic to set up a GPGPU for an already existing application, taking into account the concrete hardware found on a specific system. It is easy to imagine running such a process independently on a large pool of heterogeneous machines, thereby making them all available for the application at their peak GPGPU performance.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

2 The NVIDIA CUDA parallel computing model

The NVIDIA CUDA architecture is among the most widely used General Purpose computation on Graphical Processing Units (GPGPU) technologies currently available. Hereafter we shortly explain important terms and definitions of the CUDA C programming model. Concerning hardware, the **device** consists of several so-called **streaming multiprocessors** (sm) which in turn consist of several **thread processors**. In the software analogue we speak about **threads** and **kernels**, respectively, which are executed on a single thread processor. A number of threads are bundled in a **thread block** and executed concurrently on one multiprocessor. A bundle of thread blocks make up the so-called **grid**.

To bring numerical or other computational problems to the GPU, the developer has to write kernels in CUDA C, which is a C language extension providing functionality to access graphics hardware. These kernels are called from a CUDA C program which resides in the host system, i.e. typically a customary PC. The programmer is responsible for memory allocation on host and device, memory transfer between both and for the determination of block and grid sizes for each kernel. In the source code these sizes are normally set to constants by the programmer.

Block and grid sizes are bound by various constraints, the most important being the limits the hardware imposes on the block size. On the NVIDIA 9800GT we utilized for all our experiments, one block can only consist of at most 512 threads. Possible block sizes[3] are e.g. 16×16 or 16×32 but not 17×32 because $17 \times 32 = 544$ and $544 > 512$. If the computation domain exceeds the maximum block size, a number of blocks have to be arranged in a grid (a single block would simply be a 1×1 grid). At this point the programmer has to decide if for example a 256×256 computation domain should be divided into a 16×16 grid of 16×16 blocks, or a 128×32 grid of 2×8 blocks or some other of the many possible configurations. According to NVIDIA, 16×16 blocks are a common choice.

However, at the time of coding, block and grid sizes can not be set optimally for every GPGPU-hardware and/or intended software application. For example, one can imagine parallel computing clusters of differing GPGPU-hardware elements where the particular piece of graphics hardware assigned to a particular computing job is unknown at the time of submission. Hence, our approach is to obtain the optimal size of grid and block for each kernel by utilizing an evolutionary algorithm that learns and adapts these parameters while the computing job is running.

3 Diffusive wave simulation

The diffusive wave is a problem from hydrogeology and models the overland flow of water, e.g. the flow of rain water down a hill over time. This section is meant to give only a brief overview of the problem and its complexity.

[3] Although 3-dimensional block and grid sizes are possible to support calculations on volumetric data, we limit the discussion to two dimensions in this paper because both our real world problems are 2-dimensional.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Catchment-scale hydrogeology attends to the examination of flow and transport processes in coupled hydrosystems, i.e. subsurface water flow in the vadose zone and the saturated zone, as well as run off and infiltration and exfiltration processes at the earth's surface. For our test problem we consider only water flow at the surface, which is driven by gravitation and ultimately a function of topography, surface finish (e.g. grassland, pavement, unworked soil) and the amount of available water. The diffusive wave approximation has been widely used for the simulation in water flow in wetlands and for overland flow [8],[4], as well as for water flow in the surface domain of coupled hydrosystems e.g. [5], [6].

The quantitative description of time-dependent processes within nature, like the diffusive wave, is typically based on conservation laws formulated as partial differential equations (PDEs). In the most general form, solutions to partial differential equations have to be found honoring given initial conditions and boundary conditions (BC). As this is analytically only possible in very specialized cases, numerical approximation methods have gained paramount importance in various science and engineering disciplines. One solution approach is **semi-discretization**, where time dependent PDEs are spatially discretized by Finite Difference (FD), Finite Element (FE) or Finite Volume (FV) methods. The approximation is then found by solving a system of ordinary differential equations (ODEs) containing the remaining time derivative:

$$\frac{du}{dt} = f(u,t),$$

where $u(x,t)$ is an unknown vector valued function, which depends on the vector of discretized spatial coordinates $x \in \Omega \subset \mathbb{R}^n$ and time $t \in \mathbb{R}_0^+$. Several methods exist for the so-called time integration of the system of ODEs: examples are the explicit Euler method, the implicit Euler method, higher-order Runge-Kutta or multi-step methods. A central step common to all of these methods is the evaluation of the (possibly nonlinear) function $f = f(u,t)$. This evaluation represents the most time consuming part of the numerical approximation procedure. For illustration purposes we here consider only the simple Euler schemes: The equation

$$u_{n+1} = u_n + h \cdot f(u_n, t_n)$$

represents the explicit Euler scheme, where $h \in \mathbb{R}$ is the so-called time-step size. The new value u_{n+1} is obtained by simply adding f evaluated at time n multiplied by h.

In the implicit scheme, given by

$$u_{n+1} = u_n + h \cdot f(u_{n+1}, t_{n+1}),$$

the function f is evaluated at time $n+1$, which translates into a nonlinear problem with the need for Newton iterations. In both schemes, evaluating f is one of the most time consuming parts in approximating a solution.

In this study we consider a simple, but computation-time consuming numerical model of the so-called **diffusive wave** approximation of the shallow water equations. This approximation is widely applied in hydraulic engineering and geosciences for the simulation of river and overland flow. Despite the strong step-size constraint for the

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

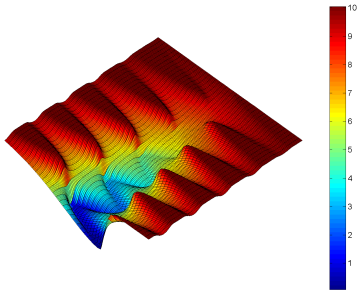


Fig. 1. 3-D Visualization of the small (512×128) valley. The colour represents the height of the terrain.

stability of the explicit Euler scheme [1], we chose it in this study due to its simplicity. As our main focus is on the acceleration of the evaluation of the multi-kernel function $f(u,t)$ on GPUs and on optimizing grid and block sizes for the CUDA implementation this choice should be reasonable.

The diffusive wave approximation for overland flow is a simplification of the shallow water equations, which constitute a set of coupled hyperbolic PDEs that are derived from the incompressible Navier-Stokes equations by integration over depth.

The discretization of a 2-D surface domain is done by a cell centered Finite Volume (FV) scheme, which yields a time-dependant ODE-System $\mathbf{u}' = \mathbf{f}(t, \mathbf{u})$ which is time integrated by an explicit euler scheme.

3.1 GPU Implementation

For the the GPU implementation, the discretized 2-D surface domains forms the computation domain. The multidimensional function $\mathbf{f}(t, \mathbf{u})$ is realized by a set of 14 different CUDA-Kernels and is the most time consuming part in the diffusive wave simulation. Two of these kernels are called to initialize the data structures used for the surface domain. The twelve remaining kernels are called in a loop, where each iteration of the loop is the integration of one time step. The twelve kernels therefore implement the explicit Euler scheme and have to be called in a sequential order.

3.2 Test Data

For numerical tests and for the optimization of the grid and the block sizes, we here use synthetic but realistic test topographies, which consist of a valley of a meandering stream with surrounding hillslopes. For the simple test case the valley has an outlet where the water accumulating along the plunging valley axis can leave the system. A stronger test of numerical stability was constructed out of the simple test case by generating a mirror image of the valley (mirror situated at the outlet perpendicular to the mean valley axis) and adding the mirrored domain at the outlet. In this kind of

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

setting no outlet exists and all surface run off collects at the former outlet and forms a small lake that migrates outward as time proceeds and water further accumulates.

For our tests we assume an initial uniform water depth throughout the domain without later addition of rainfall. This could be viewed as an overland flow situation developing right after a short, strong rainfall event. For the numerical solution of these test cases we use an explicit Euler scheme with a fixed time step size of 0.01 seconds. The time step size was chosen small enough in order to stay within the bounds of the CFL-condition [1] for a comparatively large range of initial water tables and Manning's n values.

Figure 1 illustrates the (512×128) cell test case of the meandering valley with outlet.

4 Experimental Analysis

Since NVIDIA states that a block size of 16×16 is a common choice, the **research question** we want to answer is simply: Are there block sizes different from 16×16 that perform better on the two problems presented in this paper? And if that should be the case, can we derive any relationship between the input data and the optimal block size?

The search space for the block size is discrete and all possible solutions can be enumerated, so we did full samples to find the optimal block size. The possible block sizes are bounded by hardware constraints and by the size of the problem instance. On the hardware side, there is a limit t of the maximum number of threads the GPU can handle and the number of available registers. The latter might further limit the number of possible threads, depending on how much registers the compiled kernels use[4]. In our case the GPU can handle $t = 512$ threads and has 8192 registers while the most demanding kernel needs 13 registers. This leads to a total maximum number of threads $\hat{t}$ of $\hat{t} = \min\{512, \lfloor \frac{8192}{13} \rfloor\} = 512$. The size of the problem instance might make some possible configurations obsolete, e.g. it might not be beneficial to use wider blocks than the width of the problem instance (a test valley) even if the hardware would allow that. Instead we here limit the maximum block width $\hat{w}$ by the width w_p of the problem instance, so $\hat{w} = \min\{\hat{t}, w_p\}$. The same can be applied to the maximum block height $\hat{h}$, so $\hat{h} = \min\{\hat{t}, h_p\}$. Putting all together, the total number n of possible configurations becomes

$$n = \sum_{w=1}^{\hat{w}} \left\lfloor \frac{\hat{h}}{w} \right\rfloor .$$

For inputs that are at least 512×512 in size like the big valley, this leads to 3280 possible configurations to be tested. For smaller inputs, there are less configurations, due to limiting the block size by the size of the problem instance as described earlier. As an example, the small valley with size 512×128 has only 2726 feasible configurations.

One might argue that odd block sizes do not make sense for whatever reasons or that only powers of two should be considered. But we want to keep an unbiased perspective here and so all enumerated configurations were tested.

[4] One can obtain this information by passing –ptxas-options=-v to the cuda compiler.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

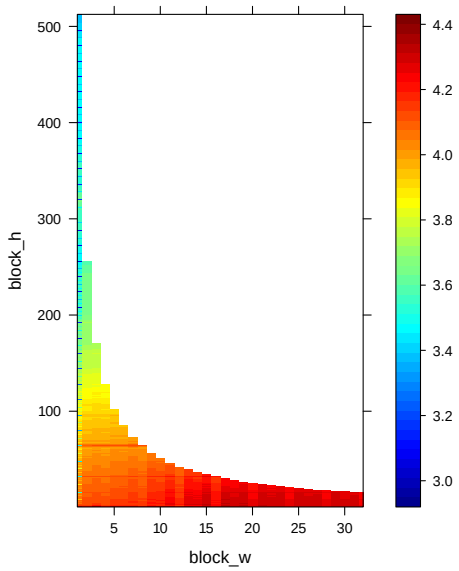


Fig. 2. Visualization of the results for the big 1024×1024 valley. One point in the plot represents one block size (block_w being the block width and block_h being the block height). For the sake of clarity, block widths larger than 32 have been omitted. Note that the scale is logarithmic, i.e. configurations depicted in dark blue ($10^{3,012}$ms $\approx$ 1s) are roughly 20 times(!) faster than the ones in red ($10^{4,34}$ms $\approx$ 22s)! The block size performing best is 1×256.

4.1 Experimental Setup

We have only two possible inputs available for the diffusive wave problem: The small and the large valley. There are 2726 possible block sizes for the former and 3280 for the latter. However, the diffusive wave problem consists of 14 different CUDA kernels and theoretically all 14 kernels could have a different optimal block size. But to avoid testing 14×2726 (or 14×3280 respectively) configurations, we enforced the same block size for all 14 kernels. To keep the tests short, we measured the time for only 100 timesteps, which equals one second of simulated waterflow. Nevertheless, it takes about 13 hours to test all 3280 configurations for the big valley. Note that this is most likely a characteristic of many spatially discretized simulation problems: The testing time for one configuration can be much smaller than a whole simulation run would be, enabling a grid configuration tuning on-the-fly.

4.2 Results

Figure 2 visualizes the results for the big valley: The diffusive wave problem is *extremely* sensitive to the block size. For both valleys, narrow block sizes are best: 1×128

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

for the small valley, 1×256 for the big one. These optimal configurations are not just a bit better than the NVIDIA default of 16×16 blocks but are actually *an order of magnitude faster*: For the small valley, the optimum is 5 times faster (85ms vs. 420ms) and for the big valley, we get a speedup of factor 16 (16.3s vs. 1s).

Besides the optimal solutions, there are a lot of other good configurations with a block width of one and a block height which is a multiple of 16. This raises the question, why a search for good configurations should be limited to powers of two. Since 16 is half of 32, the warp size of the 9800GT, there might be a connection between the warp size and good configurations. Blocks of width one and a height that is not a multiple of 16 perform worse but are still substantially better than the NVIDIA default.

When interpreting these times, one has to keep in mind that we only measured the time for 100 iterations, with a timestep of 0.01s (compare section 3.2) which equals one second of simulated waterflow. For a real simulation which covers e.g. two days of waterflow, one needs

$$2 \times 24 \times 3600 \times 100 \approx 17\text{million}$$

iterations. This computation would take 32 days with the NVIDIA configuration but only 2 days with the optimal block size! One might argue that we are overfitting the block size to the short simulation times and that the model dynamics might change over time, leading to different optimal block sizes. To test if this might be the case, we ran a test simulation for 3.6 million iterations (10 hours of simulated waterflow) with the optimal block sizes. During this simulation, the runtime for a fixed number of iterations remained constant (within the limits of timer granularity). Longer simulations will be done as soon as we have some dedicated hardware for longtime experiments.

To see if there is a difference between the 14 kernels we performed a sensivity analysis by using the optimal block size for one of the 14 kernels and the NVIDIA default for the 13 other kernels. The result is the same for both valleys: All kernels benefit from the optimal block size and we only get the great speedups when all 14 kernels use the optimal block size. This of course might not be the case for other real world problems that consist of more than one kernel.

5 Evolutionary Approach

Of course simply enumerating all possible configurations, as done in the previous section, is a very naive approach to finding a good configuration. Even though we can limit the computation time of the diffusive wave problem by only calculating a small number of iterations (100), it may get very time consuming to test all possible block sizes. Enumerating is clearly undesirable for problems with longer computation times, especially when the employed hardware is heterogeneous or unknown before actually starting the computation and the tuning has to be done on-the-fly.

In this situation, stochastic search algorithms that can come up with a near-optimal configuration fast form a viable alternative. It is not necessary to always detect the global optimum for obtaining considerable speedup. Note that for the diffusive wave simulation, many grid configurations (multiples of 16) exist that already perform very good, although not optimal. The described setting calls for application of an EA, so the

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

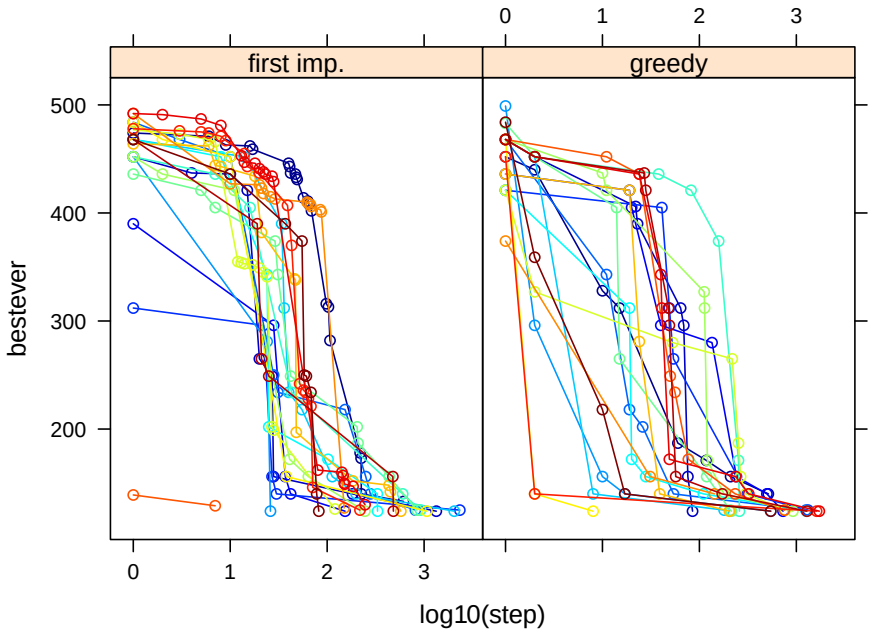


Fig. 3. Comparison of first improvement (left) and greedy EA with 20 runs each on the small valley. The points denote improvments, the lines shall only enhance the visibility. Note that the x-axis is logarithmically scaled. With first improvement, all 20 runs reach a good solution after around 100 to 200 function evaluations (of 2726 until completing enumeration). The greedy alternative shows greater variance, some runs find a good solution very fast, in one lucky case even the global optimum.

research question we investigate in the following is if a simple evolutionary algorithm can find a good configuration substantially faster.

5.1 Experimental Setup

Using a randomized search heuristic of course induces the need to repeat the experiment a number of times to get meaningful results. We therefore only use the small valley here, which allows faster computation times and hence more repetitions. We use a simple (1+1) evolution strategy [3] with some adaptations to the application scenario:

- Offspring is only generated in the immediate (Moore) neighborhood.
- We constrain the search space to only valid solutions by connecting the outer edges of the allowed configurations in x- and y-direction to the opposite extremes. This shapes the search space into an irregular torus. See figure 2 to get an idea of the set of allowed configurations.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

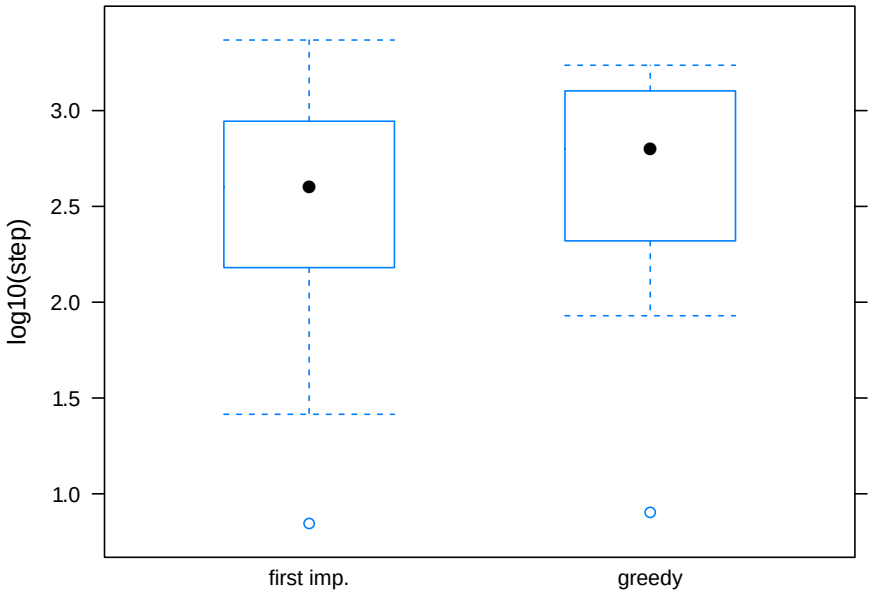


Fig. 4. Comparison of the number of steps needed to reach the global optimum for both EA variants, logarithmically scaled. The slight advantage of the first improvement variant is not statistically significant (Wilcoxon rank-sum test with α-level of 5 %).

– To prevent (unnecessarily) using more function evaluations than the enumeration would take, we employ a taboo list. If the neighborhood of the current parent individual gets empty, a restart is performed by randomly placing it to a position that is not yet on the taboo list.

The start individual is randomly chosen from the possible configurations with uniform distribution. To evaluate a block size, we measure the time for 100 iterations, as done in section 4. The task of the EA is to minimize the number of steps until obtaining the global optimum (or at least a comparably good solution). We assume that by employing an EA here, the number of steps to reach the global optimum can be dropped to around one tenth compared to the enumeration on average.

We compare two alternatives of the EA: With *first improvement*, the EA generates one offspring and performs the usual $+$-selection. The *greedy* alternative evaluates all neighbors not yet in the taboo list and choses the best one as the offspring. The EA terminates when all possible points are in the taboo list, i.e. all configurations have been evaluated. For both alternatives, we performed 20 runs.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

5.2 Results

Figure 3 shows the fitness of the best ever sampled individual for all runs of both alternatives. We can make the general observation that runs find a good solution after 100 to 200 fitness evaluations. Figure 4 depicts the number of steps needed to reach the global optimum for both EA variants. There is no clear winner, as the observed slight difference is not statistically significant.

This leads to the conclusion that applying the EA (any of the two variants) makes sense if configuration tuning is time critical. For the investigated application, it indeed needs less than one tenth of the number of steps compared to enumeration. Of course, for different problems, this relation cannot be guaranteed. However, since the EA will ultimately check all configurations just like the brute force approach, the global optimum will never be missed but with very high probability it will be found much faster, so that the search may be stopped at any time.

6 Conclusions

In this work, we analyzed the influence of the block sizes for a real world problem computated on the GPU. This complex problem from hydrogeology is very sensitive to the right block size and enormous speedups are possible with configurations which are substantially different from the common NVIDIA settings.

We showed that a simple (1+1) EA can find these good configurations substantially faster than the brute force approach of testing all configurations. Following this avenue to an automatic parameterization of a GPGPU should lead to a piece of code framing the actual application code to provide the functionality we have exemplified here. In this case the application programmer can concentrate on programming a solution for the problem at hand and needs not care about good parameterizations since this is done by the online tuning framework possibly based on an evolutionary algorithm.

Finally, no knowledge about the internals of the hardware is needed (which is probably outdated anyway when the next generation of GPUs arrives) and no code changes or hand tuned algorithms are needed. Instead, a programmer can think about the problem to solve, implement a solution without thinking about memory access patterns, etc and let the EA do the tuning automatically in a reasonable amount of time.

References

1. Courant, R., Friedrichs, K., Lewy, H.: Über die partiellen Differenzengleichungen der mathematischen Physik. Mathematische Annalen 100, 32–74 (1928), http://dx.doi.org/10.1007/BF01448839, 10.1007/BF01448839
2. Davidson, A., Owens, J.D.: Toward techniques for auto-tuning GPU algorithms. In: Para 2010: State of the Art in Scientific and Parallel Computing (Jun 2010), http://www.idav.ucdavis.edu/publications/print_pub?pub_id=1035
3. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer (2003)
4. Feng, K., Molz, F.: A 2-d, diffusion-based, wetland flow model. Journal of Hydrology 196(1-4), 230 – 250 (1997), http://www.sciencedirect.com/science/article/B6V6C-3T7JKM3-C/2/8eef02683e25eb8aa4a2358f96c41ed4

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

5. Jones, J., Sudicky, E.: An assessment of the tracer-based approach to quantifying groundwater contributions to streamflow. Water resources research 42, W02407 (2006)
6. Jones, J., Sudicky, E.: Application of a fully-integrated surface-subsurface flow model at the watershed-scale: A case study. Water Resources Research 44, W03407 (2008)
7. Maitre, O., Krüger, F., Querry, S., Lachiche, N., Collet, P.: Easea: Specification and execution of evolutionary algorithms on gpgpu. Soft Computing (2011), accepted
8. Zhang, W., Cundy, T.W.: Modeling of two-dimensional overland flow. Water Resources Research 25(9), 2019–2035 (1989), http://dx.doi.org/10.1029/WR025i009p02019
9. Zhang, Y., Owens, J.D.: A quantitative performance analysis model for GPU architectures. In: Proceedings of the 17th IEEE International Symposium on High-Performance Computer Architecture (HPCA 17) (Feb 2011 (in print)), \url{http://www.idav.ucdavis.edu/publications/print_pub?pub_id=1050}

A Study of the Hybridization to Improve the Efficiency of an Adaptive Particle Swarm Optimizer

Nadia Smairi[1] , Sadok Bouamama[1], Khaled Ghedira[1], Patrick Siarry[2]
(1)National School of Computer Sciences, University of Manouba, Manouba 2010,Tunisia
(2)LISSI Laboratory, University of Paris-Est Val de Marne, 94010 Créteil, France
{nadia.smairi@gmail.com, Sadok.Bouamama@ensi.rnu.tn, khaled.ghedira@isg.rnu.tn, siarry@univ-paris12.fr}

Abstract. Tribes is an adaptive Particle Swarm Optimization (PSO) technique. This paper presents an adaptation to the multi-objective optimization (MO) case. In fact, it introduces novel models of improving its performance through its hybridization with a local search (LS) technique. These models are based on three different placements to apply the local search: the archive, the best particle among each tribe and each particle of the swarm. As a result of our study, we present three versions of our hybridized algorithm. In order to generalize our conclusions, we hybridize Tribes with Tabu Search (TS). The proposed mechanisms are experimented and validated using well known functions from specialized literature.

Keywords: Particle Swarm Optimization; Tribes; Tabu search; Multi-objective Optimization.

1 Introduction

Many real world problems are involving multiple contradictory objectives to be optimized simultaneously. In those problems, the solution is different from that of a mono-objective optimization one. Therefore, it is necessary to develop a new definition for the optimality. In fact, MO problems have not one, but a set of solutions, that are equally good.
In the last years, many techniques were proposed. Among them, the multi-objective evolutionary algorithms have been considered as successful. But one of many drawbacks of them is that each one has many parameters to be tuned each time we want to solve a different problem. Tribes, an adaptive PSO technique, has the advantage to be designed as a black box, the user defining only the search space, the function to minimize, the required accuracy and a maximum number of evaluations. The aim of this work is to design a competitive multi-objective algorithm free from parameter fitting based on Tribes. However, the cost of this adaptation is the premature convergence and the early stagnation of the swarm. It has then become evident that the concentration on Tribes only is restrictive. A skilled combination of Tribes with local search techniques can provide a more efficient behavior and higher flexibility when dealing with the real-world problems. In addition, we study the

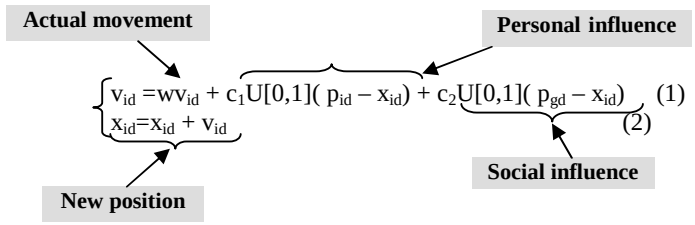
Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

impact of the place where we apply local search on the performance of the proposed techniques.

2 PSO and MO

2.1 PSO core

PSO is a bio-inspired optimization algorithm that was proposed by James Kennedy and Russell Eberhart in 1995, and which is inspired on the choreography of a bird flock. It is a population based algorithm which deals with swarm intelligence. Each particle in this swarm has a n dimensional vector used as a position in the parameter space. At each iteration, particles are moving using some core equations to compute their velocity and decide their movements. The main advantage of PSO is its simple implementation as it can be reduced to the two following equations [10]:



Where c_1 and c_2 are constants that indicate the attraction from the p_i or p_g position, respectively; w refers to the inertia of the previous movement; $x_i = (x_{i1}, x_{i2}, ..., x_{in})$ represents the i−th particle; d = 1, 2, . . . , n and n represents the dimension; U[0,1] denotes a uniformly random number generated within the range (0,1); $v_i = (v_{i1}, v_{i2}, ..., v_{in})$ represents the rate change (velocity) of particle i. Equation (1) describes the velocity that is constantly updated by each particle and equation (2) updates the position of the particle in each decision variable.

2.2 State of art of Multi-objective Particle Swarm Optimization

In the last few years, several PSO algorithms have been proposed to tackle the multi-objective optimization problem. Here we briefly review the most relevant of them.
Parsopoulos and Vrahatis propose three different types of aggregation: a classic linear aggregation, for which the weights are fixed, a dynamic aggregation where the weights are gradually modified during the treatment and an aggregation the weights of which are brutally modified [15].
Hu, Eberhart and Shi propose an algorithm optimizing each time one single objective using a lexicographical order [8].
The VEPSO strategy was introduced by Parsopoulos, Tasoulis and Vrahatis. It presents an adaptation of VEGA to the particle swarm optimization. It uses M

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

different swarms to explore the space of search. Every swarm is optimized according to one objective. But, its movement is influenced by the information coming from the other swarms [14].
Moore and Chapman propose an algorithm based on the Pareto dominance and a PSO algorithm with a circular topology of the neighborhood. In this approach, the choice of the personal guide, for every particle, is arbitrarily made from a list containing the not dominated positions that are found [13].
Ray and Liew propose a PSO algorithm using the Pareto dominance. They combine evolutionary techniques with those of the PSO. They also use an operator of density on the neighborhood to promote the density in the swarm [18].
This approach, proposed by Coello and Lechuga, is based on having an external archive to store the not dominated positions. Furthermore, the updates of the archive are executed considering a geographical system which decomposes the space of the objectives to a set of hypercubes. The archive is also used to identify a leader which will drive the search [6].
The authors propose a multi-objective PSO algorithm, called DOPS, in which several techniques are integrated for the selection of the leaders and the update of the archive [1].
Quintero, Santiago and Coello suggest a hybridization of a PSO algorithm with local search techniques, such as scatter search and a rough sets based technique [17].
The proposed algorithm is based on the dominance of Pareto: every not dominated position presents a potential candidate to be selected as a leader. A crowd function is also used to filter all the leaders. This approach also integrates the concept of the ε-dominance to fix the size of the archive [19].
The author has developed a multi-objective version of Tribes. In fact, MO-Tribes uses an approach based on the Pareto dominance. The not dominated particles are stored in an external archive of which size and updates are automatically defined. Furthermore, the variety is maintained thanks to a criterion of maximization of the crowd distance and also thanks to the multiple restarts of the swarm [7].

3 Tribes

3.1 Presentation

Tribes is a PSO algorithm that works in an autonomous way. Indeed, it is enough to describe the problem to be resolved at the beginning of the execution. Then, it is the role of the program to choose the strategies to be involved [5].

3.2 Tribes components

- The particle informer: A particle A is an informer of a particle B, if B is capable of reading the best position visited by A.

- The tribe: In the general case, a given particle A can not inform the rest of the other particles of the swarm. That's why we define the tribe, which is a subset of the swarm, where every particle is capable of communicating directly with the rest of the tribe's particles.
- The swarm: it is formed by a set of tribes; each one is looking to finding a local optimum. A group of decisions is therefore necessary to find the global optimum. For this reason, tribes have to communicate between themselves.
- Quality of a particle: The notion of a good or bad particle is relative to its tribe. We organize particles, belonging to the same tribe according to their fitness evaluations. As a result, we can define the best particle and the worst particle following every tribe.
- Quality of a tribe: to decide if a tribe T is good or bad, we generate a random number t between 0 and n (n being the size of T). If the number of good particles is bigger than t, then the tribe is considered as a good one. Otherwise, the tribe is bad.

3.3 Tribe evolution

The evolution of a tribe involves the removal or the generation of a particle. The removal of a particle consists in eliminating a particle without risking to miss the optimal solution. For that purpose, only the good tribes are capable of eliminating their worst elements. The creation of a particle is made for bad tribes, as they need new information to improve their situation. However, we note that it is neither necessary nor desirable to perform these structural adaptations at each iteration because some time must be allowed for the information to propagate among the particles. Several possible rules can be formulated to ensure this. The rule used is: if the total number of information links is L after one structural adaptation, then the next structural adaptation will occur after L/2 iterations [5].

3.4 Swarm move

The only remarkable difference between the movements of the classic PSO algorithm and those of Tribes is situated at the level of the probability distribution of the next position which is modified; it is D-spherical in the case of Tribes and D-rectangular in the case of the classic PSO. In fact, as in classical PSO, a particle is guided by its best performance and a global guide called informer: if the particle is not the best of its tribe, its informer is then the best particle of the tribe (BT), otherwise a random selected informer from the other best tribe's particles.

3.5 Swarm evolution

At the beginning, we start with a single particle forming a tribe. After the first iteration, the first adaptation takes place and we generate a new particle which is going to form a new tribe, while keeping in touch with the generative tribe. In the

following iteration, if the situation of both particles does not improve, then every tribe creates two new particles: we form a new tribe containing four particles. In this way, if the situation deteriorates, then the size of the swarm grows (creation of new particles). However, if we are close to an optimal solution, the process is reversed and we begin to eliminate particles, or even tribes [5].

4 Our approaches

4.1 Presentation

In the multi-objective case, we have essentially to obtain a set of solutions close to the true Pareto front and to maintain the diversity within the found set. For that purpose, several problems are detected. In fact, we need to develop new ways to define the informer and the best performance of every particle. Moreover, we need to remedy to the premature convergence. In our approach, we propose to use the Pareto dominance to respect the completeness of every objective and to add an external archive to save the found not dominated solutions. Furthermore, the hybridization between Tribes and a local search (LS) algorithm can be considered as a solution to handle the premature stagnation of the swarm by improving the capacity of exploitation of Tribes. We apply a local search technique: TS. In fact, the local search is not going to be inevitably applied in a canonical way, that is on all the particles of the swarm: we also propose two other techniques, the first one consists in applying the local search only to the best particle of every tribe. The second one consists in applying it to the particles of the archive. We shall have then three versions of the algorithm.
The first one consists in applying the LS only to the particles of the archive which are situated in the least crowded zones. Let us note that, in this case, the local search is not applied, unless the archive is full, so that some time is allowed to the information to propagate in the swarm (see Figure 1).
The second version of the algorithm consists in applying the LS only to the best particles of the tribes. In fact, we consider that those particles are situated in promising zones and probably they need further intensification to find out other solutions. This process is repeated at each iteration of the algorithm (see Figure 2).

The third version consists in applying the LS to all the particles of the swarm. It is made at the moment of the swarm adaptation, in order to let the propagation of the information through the swarm (see Figure 3).

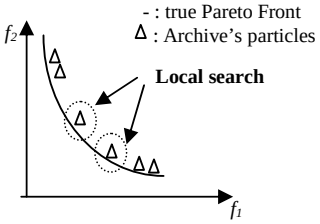


Fig.1. TS-TribesV1

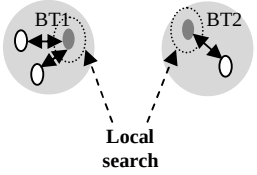


Fig.2. TS-TribesV2

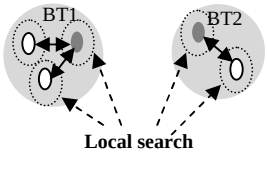


Fig.3. TS-TribesV3

The update of the archive consists in adding all the not dominated particles to the archive and deleting the already present dominated ones. If the number of particles in the archive exceeds a fixed number, we apply a crowd function to reduce the size of the archive and to maintain its variety. Indeed, that process divides the objective space into a set of hypercubes.

The role of this function is to give, for every particle, the number of the archive's particles that occupy the same hypercube. In that case, when the addition of a particle to the archive creates an overflow, we eliminate the one which has the highest crowd value [11].

The choice of the particle informer is similar to the case of mono-objective Tribes. Indeed, if we take a particle which is not the best of its tribe, its guide is then the best particle of the tribe. If we consider, on the other hand, the best particle of a given tribe, the informer is then some random particle from the archive.

4.2 Hybridizing Tribes with TS

The combination between the high convergence rate of Tribes with TS can be considered as a competitive approach that is able to spread the nondominated solutions found, so that a good distribution along the Pareto front is achieved. Moreover, it can handle the premature stagnation of the swarm by improving the capacity of exploitation of Tribes.

The TS was introduced by Glover. It consists in the examination of a neighbourhood of a current solution x and it retains the best neighbour x_0, even if x_0 is worse than x. However, this strategy can pull cycles. To prevent this kind of situation from appearing, we store the k last visited configurations in a short-term memory and we forbid to hold any other configuration which is already a member of it. This memory, called the tabu list, is one of the main constituents of the method. In fact, it prevents all the cycles of lower length than k, k being determined according to the problem.

TS is essentially intended for the resolution of the combinatorial problems. Few works considered its adaptation to the continuous optimization case. In our work, we are inspired from the approach of Chelouah and Siarry [4]. In that case, the obtained method is similar to the classic TS. The difference lies essentially in the generation of the neighbourhood. It is necessary to define, first of all, a way to discretize the search space. In fact, the neighbourhood is defined by using the concept of "ball". A ball B(x, r) is centred on x (current solution) with radius r. To obtain a homogeneous exploration of the space, we consider a set of balls centred on the current solution x, with radius r_0, r_1, r_2,…r_n. Hence the space is partitioned into concentric crowns (see Figure 4).

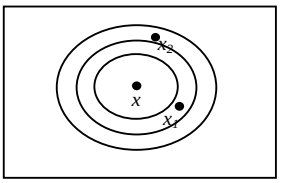


Fig.4. Generating the neighbourhood

The Figure 5 shows the pseudo-code of the TS algorithm:

```
Begin
 x    search space, L : iterations number
 i=0
 while i<L
    choose the best authorized movement m  (using the Pareto dominance)
     update x
     update the Tabu List
     update the archive
 EndWhile
End
```

Fig.5. Pseudo-code of TS algorithm

5 Experimentations and results

5.1 Test functions

The experiments in this work adhere to the demands of [9] where a test set consisting of 10 test functions is described. This set is composed of well-known classical multi-objective functions (Oka2, Sympart, WFG1, WFG8 and WFG9) and extended and shifted versions of the popular ZDT test suite. Extending, shifting and rotating of the ZDT functions were done to overcome some shortcomings like a global optimum in the center or on the bounds of the search space or to make the problem inseparable [9]. These functions present different difficulties, such as convexity, concavity, multimodality, etc. The detailed description of these functions was omitted, due to space restrictions. However, all of them are unconstrained functions to minimize and have between 3 and 30 decision variables (see Table 1).

25 independent runs with a maximum number of Max FES = 5e+4 function evaluations are conducted for every test problem. The non-dominated solutions are stored in an external archive that is updated after generating new trial vectors. The size of the archive is given in [9] for every test function: 100 for the two-objective functions and 150 for the three-objective ones.

In order to validate our approaches and to justify the use of the local search techniques, we are going to compare our proposed algorithms against the multi-objective Tribes without local search (Tribes-V4), NSGA-II, which is a multi-objective evolutionary algorithm representative of the state of art in the area and MO-Tribes.
The TS search in our approaches uses three parameters which are r_n, L and the neighbourhood size. In fact, the neighbourhood size is fixed to 10. A detailed study of its impact is presented in our previous work [20]. The most important conclusion is that the neighbourhood size has no significant effect on the performances of the considered algorithms. In fact, they keep the same tendency regardless of the

neighbourhood size variation. That's why we choose the size 10 as it is not very small, which means a non significant exploration of the neighbourhood, and not very large, which leads us to a prohibitive calculation cost. Moreover, r_n and L are fixed respectively to 0.1 and 20 for the three versions. The NSGA-II used the following parameters: crossover rate=0.8, mutation rate= 0.1 and population size= 100.

5.2 Metrics of comparison

For assessing the performance of the algorithms, there exist many unary and binary indicators measuring quality, diversity and convergence. In the literature, there are many proposed combinations in order to perform a convenient study and comparison. We choose the combination of two binary indicators, that was proposed in [12]: R indicator and hypervolume indicator.

Table 1. Test functions

Test functions	Objective	Modality	Geometry	Dimension
Oka2	f1 f2	Uni-modal Multi-modal	Concave	3
Sympart	f1:2	Multi-modal	Concave	30
S_ZDT1	f1:2	Uni-modal	Convex	30
S_ZDT2	f1:2	Uni-modal	Concave	30
S_ZDT4	f1 f2	Uni-modal Multi-modal	Convex	30
R_ZDT4	f1:2	Multi-modal	Convex	30
S_ZDT6	f1:2	Multi-modal	Concave	10
WFG1	f1:3	Uni-modal	Convex	24
WFG8	f1:3	Uni-modal	Concave	24
WFG9	f1:3	Multi-modal	Concave	24

5.3 Results

The binary indicators used to make the comparison measure both convergence and diversity. The results regarding the R indicator are given in Table 2 (R can take values between -1 and 1, where smaller values correspond to better results). The hypervolume difference is given for all test functions in Table 3. Again, smaller values mean better quality of the results, because the difference to a reference set is measured.

For both indicators, we present the summary of the results obtained. In each case, median solutions, mean values and standard deviations are displayed.

According to these tables, we notice that the hybridization with TS gives generally better results than Tribes-V4. In fact, the first version of the hybridization (TS-TribesV1) outperforms generally the other versions, except for test functions S_ZDT4 and R_ZDT4, where the third version gives the best results (TS-TribesV3) (see Table 2 column 3 and Table 3 column 3). In fact, in this case, a bad convergence behavior is detected for S_ZDT4 and R_ZDT4 for all the versions, except for TS-TribesV3 (see Table 2 column 5, lines 6 and 7 and Table 3 column 5, lines 6 and 7). This can be

explained by the fact that Tribes begins with one particle and needs further time to explore the search space. The third version has a better convergence behavior in that case, thanks to the strong exploration of the search space provided by the local search, which is applied to all particles of the swarm. We note that a bad convergence behavior is detected also with another PSO algorithm for ZDT4 in [8].
The results of MO-Tribes are very close to those of TS-TribesV1. This can be explained by the fact that MO-Tribes uses also a local search technique applied only on the archive's particles.
Our results indicate that our approaches, especially TS-TribesV1, are able to compete with NSGA-II, a highly competitive multi-objective evolutionary algorithm.
Finally, we conclude that the hybridization is very competitive, as it supports both intensification and diversification. In fact, the choice of the particle's informer is done arbitrarily from the archive's particles, in order to accelerate the swarm's convergence towards the search space zones where the archive's particles are situated. This can be considered as an intensification process. Moreover, the archive updating is done thanks to the crowd function, that maintains the archive diversity. This can be considered as a diversification process. Indeed, TS supports both intensification and diversification. In fact, the good neighbourhood exploration intensifies the search towards specific zones in the search space. Besides, TS mechanisms, such as tabu list, allow avoiding the risk of trapping in non Pareto solutions.

6 Conclusion

We have proposed new hybrid multi-objective evolutionary algorithms based on Tribes and TS. Those hybrids aim at combining the high convergence rate of Tribes with the good neighbourhood exploration performed by the local search algorithms. Therefore, we have studied the impact of the placement of the local search technique on the performance of the algorithms. Two widely used metrics have been used to evaluate the results. The performance of our approaches over all the benchmark functions studied turned out to be satisfactory, in the sense that the aim was to determine if the hybridization is justified. In fact, the results showed that the hybridization is a very promising approach for multi-objective optimization. However, for some complex problems, such as S-ZDT4 and R-ZDT4, TS-Tribes still need to improve their performances. As part of our ongoing work, we plan to study some other strategies of diversification and adaptation in order to remedy those problems.

7 References

1. Bartz-Beielstein, T., Limbourg, P., Parsopoulos, K.E., Vrahatis, M.N., Mehnen, J., and Schmitt, K., Particle Swarm Optimizers for Pareto Optimization with Enhanced Archiving Techniques. *In congress on Evolutionary Computation Canberra, Australia*, IEEE Press, Vol. 3, 1780-1787. (2003, December)

2. Bergh, F., *An Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa. (2002)
3. Carlos, A. and Coello, C.A.C., An Updated Survey of GA-Based Multiobjective Optimization Techniques. *ACM Computing Surveys*, Vol. 32, No. 2. (2000, June)
4. Chelouah, R. and Siarry, P., Tabu Search applied to global optimization. *European Journal of Operational Research 123*, 256-270. (2000)
5. Clerc, M., *Particle Swarm Optimization*. International Scientific and Technical Encyclopaedia, John Wiley & sons. (2006)
6. Coello, C.A.C and Lechuga, M.S., MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization. *Congress on Evolutionary Computation (CEC'2002)*, IEEE Service Center, Piscataway, New Jersey, Vol. 2, 1051-1056. (2002, May)
7. Cooren, Y., *Perfectionnement d'un algorithme adaptatif d'optimisation par essaim particulaire. Applications en génie médical et en électronique*. PhD thesis, Université Paris 12. (2008)
8. Hu, X., Eberhart, R. and Shi, Y., Particle Swarm with Extended Memory for Multi-objective Optimization. *In IEEE Swarm Intelligence Symposium*. (2003)
9. Huang, V., Qin, A., Deb, K., Suganthan, P., Liang, J., Preuss, M., and Huband, S., Problem Definitions for Performance Assessment & Competition on Multi-objective Optimization Algorithms. Nanyang Technological University, Singapore, Tech. Rep., (2007, January)
10. Kennedy, J. and Eberhart, R., Particle Swarm Optimization. In international Conference on Neural Networks. Piscataway, NJ., IEEE Service Center, 1942-1948, (1995)
11. Knowles, J. and Corne, D., M-paes: A memetic algorithm for multiobjective optimization. In Congress on Evolutionary Computation, Piscataway,NJ, IEEE Service Center, 325-332, (2000)
12. Knowles, J., Thiele, L. and Zitzler, E., A tutorial on the Performance Assessment of Stochastic Multi-objective Optimizers. *Tik-Report No-214*, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland. (2006, February)
13. Moore, J. and Chapman, R., *Application of particle swarm to multiobjective optimization*. Department of Computer Science and Software Engineering, Auburn University. (1999)
14. Parsopoulos, K.E., Tasoulis, D.K. and Vrahatis, M.N., Multiobjective optimization using parallel vector evaluated particle swarm optimization. *In Proceedings of the IASTED International Conference on Artificial Intelligence and Applications (AIA 2004)*, Innsbruck, Austria, ACTA Press, Vol. 2, 823-828. (2004, February)
15. Parsopoulos, K.E. and Vrahatis, M.N., Particle Swarm Optimization Method in Multi-objective Problems. *Proceedings of the ACM 2002 Symposium on Applied Computing (SAC'2002)*, 603-607. (2002)
16. Pulido, G.T. and Coello, C.A.C., Using clustering techniques to improve the performance of a particle swarm optimizer. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2004)*, Seattle, Washington, USA, Springer-Verlag, Lecture Notes in Computer Science Vol. 3102, 225-237. (2004, June)

17. Quintero, L.V.S., Santiago, N.R. and Coello, C.A.C., Towards a More efficient Multi-objective Particle Swarm Optimizer. *Multi-objective Optimization in computational intelligence: Theory and practice*, Information Science Reference, Hershey, USA, In Lam Thu Bui and Sameer Alam (editors), 76-105. (2008)
18. Ray, T. and Liew, K.M., A swarm metaphor for multiobjective design optimization. *Engineering Optimization*, 34(2), 142-153. (2002, March)
19. Sierra, M.R. and Coello, C.A.C., A study of techniques to improve the efficiency of a multi-objective particle swarm optimizer. *Evolutionary Computation in Dynamic and Uncertain Environments*, Springer, 269-296. (2007)
20. Smairi, N., Bouamama, S., Ghedira, K. and Siarry, P., New Proposal For a Multi-objective Technique Using Tribes and Tabu Search. Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics, Vol. 1, 86-91. (2010)

Table 2. Results for R indicator (best results are in bold)

Test functions		TS-TribesV1	TS-TribesV2	TS-TribesV3	Tribes-V4	MO-Tribes	NSGA-II
Oka2	Mean	**-1,13e-3**	-1,09e-3	-1,08e-3	8,51e-5	-1,10e-3	-1,05e-3
	Median	-1,13 e-3	-1,11e-3	-1,09e-3	7,22e-5	-1,11e-3	-1.06e-3
	Sd	1,02e-5	2,19e-4	1,01e-4	1,36e-3	1,05e-4	3.12e-5
Sympart	Mean	**3,44e-5**	4,65e-5	6,15e-5	2,39e-4	5,18e-5	1,44e-4
	Median	3,42e-5	4,40e-5	5,68e-5	3,22e-4	5,17e-5	1.38e-4
	Sd	3,08e-5	1,66e-4	3,14e-4	1,05e-4	2,83e-4	3.01e-5
S_ZDT1	Mean	4,73e-4	1,16e-3	8,63e-4	2,79e-3	5,12e-4	**1,14e-4**
	Median	3,12e-4	1,19e-3	8,24e-4	3,15e-3	5,11e-4	1.09e-4
	Sd	5,71e-5	2,34e-4	1,92e-5	2,29e-3	2,04e-4	2.31e-5
S_ZDT2	Mean	**3,07e-5**	1,28e-3	6,15e-5	2,80e-4	5,01e-5	4,37e-5
	Median	4,32e-5	1,23e-3	5,43e-5	2,73e-4	4,96e-5	3.54e-5
	Sd	1,02e-5	1,08e-4	3,16e-5	1,75e-4	1,47e-5	4.20e-5
S_ZDT4	Mean	2,65e-3	7,76e-3	**9,72e-6**	2,07e-3	4,96e-3	8,77e-4
	Median	2,47e-3	5,81e-3	8,97e-6	2,08e-3	4,93e-3	8.40e-4
	Sd	1,52e-3	2,16e-3	1,02e-6	1,22e-3	7,32e-4	3.11e-4
R_ZDT4	Mean	6,91e-3	2,79e-3	**1,73e-4**	6,98e-3	5,23e-3	2,94e-3
	Median	6,74e-3	2,77e-3	1,86e-4	6,82e-3	5,31e-3	3.06e-3
	Sd	1,32e-3	2,34e-4	1,54e-4	2,14e-3	1,12e-3	1.45e-3
S_ZDT6	Mean	**2,75e-3**	7,53e-3	2,97e-3	3,05e-3	3,51e-3	2,52e-2
	Median	2,61e-3	7,82e-3	2,41e-3	2,64e-3	3,48e-3	2.56e-2
	Sd	3,46e-4	1,32e-3	1,04e-3	2,16e-3	2,31e-3	1.92e-3
WFG1	Mean	2,60e-2	4,55e-2	4,21e-2	**1,22e-2**	1,53e-2	5,36e-2
	Median	2,51e-2	4,59e-2	4,23e-2	1,18e-2	1,62e-2	5.43e-2
	Sd	1,06e-4	2,17e-5	1,72e-5	1,02e-4	4,16e-4	2.86e-3
WFG8	Mean	-1,88e-2	-1,24e-2	-4,39e-3	-4,59e-4	**-2,26e-2**	-7.64e-3
	Median	-1,86e-2	-1,21e-2	-4,68e-3	-5,34e-4	-2,28e-2	-7.80e-3
	Sd	4,29e-6	1,14e-5	2,21e-4	1,06e-3	1,12e-5	7.58e-4
WFG9	Mean	-9,24e-3	-6,20e-3	-3,93e-3	-5,06e-3	-9,10e-3	**-9.79e-3**
	Median	-1,98e-2	-6,64e-3	-4,11e-3	-4,85e-3	-8,44e-2	-9.20e-3
	Sd	3,23e-3	4,02e-3	8,29e-4	2,78e-3	2,46e-3	1.26e-3

Table 3. Results for Hypervolume indicator (best results are in bold)

Test functions		TS-TribesV1	TS-TribesV2	TS-TribesV3	Tribes-V4	MO-Tribes	NSGA-II
Oka2	Mean	**-1,21e-3**	-1,08e-3	-1,09e-3	-1,10e-4	-1,12e-3	-1,04e-3
	Median	-1,22e-3	-1,11e-3	-1,11e-3	-1,08e-4	-1,13e-3	-1.04e-3
	Sd	1,02e-4	2,12e-4	2,01e-4	1,03e-4	1,08e-4	2.10e-5
Sympart	Mean	1,36e-4	**1,08e-4**	1,83e-4	1,28e-4	1,52e-4	4,29e-4
	Median	1,23e-4	1,07e-4	1,89e-4	1,26e-4	1,33e-4	4.08e-4
	Sd	2,13e-4	3,15e-5	1,44e-4	4,02e-5	2,47e-4	8.84e-5
S_ZDT1	Mean	1,51e-3	3,99e-3	4,98e-3	2,05e-3	2,25e-3	**7,81e-4**
	Median	1,39e-3	3,72e-3	4,97e-3	2,10e-3	2,27e-3	7.70e-4
	Sd	2,14-3	2,16e-3	3,12e-4	1,52e-3	3,28e-4	8.36e-5
S_ZDT2	Mean	3,28e-4	2,49e-3	4,50e-4	2,87e-4	3,38e-4	**-1,15e-2**
	Median	3,25e-4	2,32e-3	4,19e-4	2,86e-4	3,34e-4	-1.15e-2
	Sd	1,09e-4	1,56e-3	3,41e-4	1,04e-4	1,12e-4	8.54e-5
S_ZDT4	Mean	7,41e-3	2,13e-2	**1,55e-3**	2,16e-2	2,12e-2	2,39e-3
	Median	7,44e-3	2,10e-2	1,62e-3	2,17e-2	2,11e-2	2.28e-3
	Sd	2,82e-4	3,72e-3	6,15e-4	1,05e-3	1,04e-3	9.10e-4
R_ZDT4	Mean	1,81e-2	7,37e-3	**1,47e-3**	2,06e-2	1,55e-2	9,26e-3
	Median	1,77e-2	7,26e-3	1,41e-3	2,05e-2	1,47e-2	9.37e-3
	Sd	2,16e-3	3,11e-4	4,28e-4	1,16e-3	2,19e-3	4.09e-3
S_ZDT6	Mean	7,49e-3	1,92e-2	1,30e-2	6,54e-2	**7,41e-3**	5,67e-2
	Median	7,31e-3	1,95e-2	1,27e-2	6,43e-2	7,31e-3	5.77e-2
	Sd	2,48e-4	1,08e-3	1,02e-3	2,28e-3	2,41e-4	4.49e-3
WFG1	Mean	**1,67e-1**	2,27e-1	2,27e-1	3,44e-1	3,44e-1	5.36e-1
	Median	1,64e-1	2,26e-1	2,21e-1	3,21e-1	3,24e-1	5.37e-1
	Sd	1,13e-2	1,06e-2	2,18e-2	4,76e-2	4,57e-2	6.05e-3
WFG8	Mean	**-1,20e-1**	-7,73e-2	-2,82e-2	-2,95e-2	-1,43e-2	-1,04e-1
	Median	-1,22e-1	-7,91e-2	-2,73e-2	-2,93e-2	-1,69e-2	-1.05e-1
	Sd	1,08e-2	3,01e-3	2,41e-3	1,12e-3	4,23e-3	1.18e-2
WFG9	Mean	-4,52e-2	-2,91e-2	-6,76e-3	-3,28e-2	**-5,72e-2**	-2,69e-2
	Median	-4,51e-2	-2,96e-2	-7,02e-3	-3,25e-2	-5,83e-2	-2.60e-2
	Sd	3,06e-4	2,13e-3	3,27e-4	1,52e-3	2,48e-3	7.02e-3

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

The *Pachycondyla apicalis* ants search strategy for data clustering problems

Djibrilla Amadou Kountché, Nicolas Monmarché, and Mohamed Slimane

University François Rabelais, Tours, Laboratoire d'Informatique,
64 Av. Jean Portalis, 37200 Tours, France

Abstract. This paper presents a work inspired by *Pachycondyla apicalis* ants behavior. These particular ants have a simple but efficient prey search strategy: when they find food, i.e. a prey, they return straight to their nest to drop back the prey and later they return back to the last successfull position. This behavior has already been applied to optimization, as the API meta-heuristic, combined with the ability of ants to sort and cluster gives us for the clustering problem very encouraging results on the UCI wine and iris datasets compared to Ant Clustering Algorithm (ACA) and K-means algorithm. In addition to its simplicity, API can be used as semi-supervised or unsurpervised clustering algorithm.

1 Introduction

The API algorithm, based on the foraging behavior of *Pachycondyla apicalis* ants, has been initially proposed for continuous optimization [16]. However, as a meta-heuristic, API can be applied to various search spaces, such as discrete or mixed ones [10]. But, it is also one of its advantages to be based on a particular ant species since the foraging behavior of *Pachycondyla apicalis* ants is interesting because it is simple but quite efficient according to these ants' environment. We suggest here that this simplicity can however lead to fertile ideas, not only in optimization within various search spaces, but also in data clustering field.

Several attempts have been provided to transpose various behaviors of ants to tackle clustering problems (see recent reviews in [9, 7, 15]), but, to the best of our knowledge, API algorithm for clustering has been studied only once [1] in a different way that we do in the present paper.

The remainder of this paper is organized as follows: the next section describes the clustering problem we consider but also a quick presentation of the basic ideas behind ant-based clustering methods since we have used the same kind of ideas to provide clustering skills to API . Then in section 3, we explain the API algorithm and how we can adapt it to clustering. In section 4, we present the experiments we have done to evaluate the API algorithm as a clustering tool and the last section gives conclusions and perspectives about this work.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

2 Problem description and related works about artificial ants

2.1 The clustering problem

Clustering is an unsupervised classification task which builds clusters from a set of objects or patterns based on similarity or dissimilarity measures between objects. In unsupervised classification the number of resulting classes that the algorithm should automatically determine is not known [11, 12]. The unknown number of clusters, the difficulty of defining a cluster for the given data (thereby choosing the appropriate dissimilarity measure) and the challenge of high-dimensionality of the data results in the important number of clustering algorithms that is continually published [12]. These algorithms can be categorized into partitioning or hierarchical. The clustering problem considered here is partitioning clustering in the case of exploratory data analysis. A comprehensive study of clustering can be found in [11, 12].

Given a clustering problem, many challenges have to be faced in order to choose the appropriate algorithm. Among them is how to assess the output of a given algorithm. Statistical indexes (Ward, inter/intra class, F-Measure, Rand, Dunn) are the mostly used criteria to answer this question [11].

In the ever increasing endeavor to propose solutions to the problems related to clustering, bio-inspired paradigm have been proposed which we will describe in the next section.

2.2 Bio-inspired and ant based clustering algorithm

Bio-inspired methods are based on the behavior of self-organized, decentralized systems. "The hope behind these methods is for them to inherit the simple design, scalability and robustness of their naturally occurring counterparts" [9]. Examples of methods applied to the clustering problem are Swarm Intelligence (SI), Evolutionary Computation (EC) or Artificial Neural Network [4]. The mostly used methods in the SI group are ant based models, on which we will focus in the following, and Particle Swarm Optimization (PSO) [13].

In Nature, many ant species have the ability to sort and cluster their dead ants or their brood (eggs, larvae,...) into several piles and have foraging mechanisms [2]. Stigmergy is the well known ant foraging mechanism that has inspired the stochastic Ant Colony Optimization meta-heuristic (ACO) [6]. Thus for the clustering problem, ant-based models have been applied as: i) direct mimics of the natural sorting and clustering behavior of ants; ii) casting the clustering problem to an optimization problem and applying ACO; iii) hybridization of the two ant behaviors with other clustering algorithms like K-means [9]. Our work belongs to the first category.

A simple model (first applied in robotics) introduced by Deneubourg et al. [5] aimed at simulating this behavior: in the Deneubourg model (DM), a randomly moving ant which does not carry an item, will pick up the most dissimilar item in a set of items with the probability P_{pickup}. On the opposite, when the ant

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

is carrying an object, it may drop it with the probability P_{drop}. These two probabilities are computed as follows:

$$P_{\text{pickup}} = \left(\frac{k_1}{k_1 + f}\right)^2 \text{ and } P_{\text{drop}} = \left(\frac{f}{k_2 + f}\right)^2 \quad (1)$$

f is the perceived fraction of items in the neighborhood of the ant. k_1 and k_2 are two constant parameters.

Lumer and Faieta [14] have proposed an adaptation of Deneubourg's work for data clustering, we call it LF, where ants and objects are initially randomly placed on a two dimensional grid. They define a density function f based on an Euclidean distance d in the space of object attributes (i.e. not based on the position of objects in the grid). Therefore, items are picked up and dropped based on this density function. For an ant which finds an object o_i, the density $f(o_i)$ measures of the average similarity of object o_i with the other objects o_j present in the neighborhood of o_i, and is given by:

$$f(o_i) = \max\left\{0; \frac{1}{s^2} \sum_{j \in \mathcal{W}(i)} \left(1 - \frac{d(o_i, o_j)}{\alpha}\right)\right\} \quad (2)$$

α is a factor that defines the scale for dissimilarity, s^2 is the number of elements in a small surrounding area (a square of $s \times s$ in the original LF model) and $\mathcal{W}(i)$ the set of objects in cells of the neighborhood. The picking up and dropping probabilities are now:

$$P_{\text{pickup}} = \left(\frac{k_1}{k_1 + f(o_i)}\right)^2 \quad (3)$$

$$P_{\text{drop}} = \begin{cases} 2f(o_i) \text{ when } f(o_i) < k_2 \\ 1, \quad \text{otherwise} \end{cases} \quad (4)$$

k_1, k_2 are constant parameters of the model. This algorithm has firstly been experimented on a synthetic low-dimensional datasets with $k_1 = 0.1, k_2 = 0.15, \alpha = 0.5$ and $s^2 = 9$. The drawback of LF algorithm is the large number of clusters found [14]. Then, three modifications have been proposed: (i) ants are provided with a speed v so that fast moving ants form coarse clusters and slowly moving ones group them more accurately; (ii) a short memory (a vector of recently seen items) allows each ant to move towards the most similar data item in its memory instead of moving randomly; (iii) ants can destroy clusters if no action has taken place for a given number of steps.

The DM and LF algorithms are the basis of many algorithms that have been proposed. The complete description of these ideas is beyond the scope of this paper (since there are several complementary and recent published states of the art about clustering with artificial ants). However, a constant difficulty over the proposed methods is the way we need to choose the parameter values. A first idea is to divide parameters into two groups: those which are linked to the data (size, type, internal properties...) and those which are independent. For details see the ACA [3] algorithm to which we will compare our results.

Before introducing the API algorithm and its clustering version, it is interesting to quickly describe the work presented in [1] as it is an adaptation of API algorithm to the clustering problem. At the beginning of the algorithm, called AntPart, all the objects are in the nest, and ants are getting out of the nest with one object and tries to find a valuable class for this object. The search for a good class to introduce the object is done according to ant's own memory and, if necessary, to a common memory of the whole colony. Thus, clusters are being built in the opposite way ants usually work: usually ants are looking for food on hunting sites and are bringing some when they are coming back to their nest. In AntPart, ants are spreading the "food" outside of their nest, using hunting sites as clusters they feed with objects. There is also a mechanism of repairing with specialized ants which are capable to move objects from one hunting site/cluster towards another one. As we will explain in the next section, our adaptation of API does not operate in this direction.

3 API for data clustering

3.1 API principles

As we have previously said, API algorithm is based on the foraging behavior of *Pachycondyla apicalis* ants. These ants have the ability to search for and exploit some distributed food sources without any complicated and centralized behavior such as mass recruitment often encountered in several more populous ant species. Instead of that, *P. apicalis* ants display a simple behavior to bring back enough prey for the survival of their colony. We can describe this behavior with a few rules that can be easily transformed in algorithms:

- *P. apicalis* are looking for food around their nest (notion of central point and random exploration),
- A *P. apicalis* ant memorizes the place where it catches a prey (mainly another insect, dead or alive) and goes back straight to its nest (there is no pheromone trails),
- At its next nest exit, a *P. apicalis* ant systematically goes back to its last hunting successful site (what we call a hunting site) and starts to look for a new prey from this position (notion of local exploration),
- *P. apicalis* ants can not build their own nest and are consequently obliged to move their colony when living conditions are not perennial (restart operator),
- When the colony decides to move, workers are using a simple form of recruitment, called tandem running: one ant leads another one to the new nest location and again until all the colony is aware of the new nest.

The model we can deduce from these rules embeds ant agents in a search space in which points correspond to solutions of the corresponding problem. For instance, for a continuous optimization problem, the search space can be a subspace S of $\mathbf{R}^n$ in which we are looking for a minimum value reached by an unknown function f defined on this subspace. The API algorithm in this case, consists in

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

starting at a random position of S for the nest. Then ants are generating random points around the nest which correspond to their initial hunting sites. At each iteration, every ant goes back to its hunting site, operates a local random search. If this tentative is a success, then the ant is considered to have found a prey. If not, several unsuccessful visits at the same hunting site can discourage the ant which consequently starts with a new random point generated around the nest. The general outline of API algorithm shows that it is a population based stochastic optimization algorithm: hunting sites memorized by ants correspond to the population and the nest (and other details not explained here) is playing a role of information exchange between elements of the population.

In the following, we are focusing on API for clustering data.

3.2 Notations and initial conditions

We define four sets:

- $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_O\}$ is the set of O objects (i.e. the input dataset);
- $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_A\}$ is the set of A ants;
- $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_N\}$ is the set of N nests;
- $\mathcal{G}$ is a two dimensions grid (which can be toroidal or not, often squared) composed of cells. Each cell has 4 neighbor cells (in the toroidal case).

All elements of sets $\mathcal{O}$, $\mathcal{A}$ and $\mathcal{N}$ are located on the grid $\mathcal{G}$.

	$\mathcal{O}_7$					
		•	•	$\mathcal{O}_5$	$\mathcal{O}_{10}$	
	•	$\mathcal{O}_2$	$\mathcal{O}_3$	•		
		•	$\mathcal{O}_4$	•		$\mathcal{O}_1$
		•	$\mathcal{O}_6$	•		
$\mathcal{O}_8$			•	$\mathcal{O}_9$		

Fig. 1. In a 7×7 grid, the set of neighboring empty cells of object $\mathcal{O}_3$, denoted by $\mathcal{E}(\mathcal{O}_3)$, is represented by the set of • in the figure. The set $v(\mathcal{O}_3) = \{\mathcal{O}_2, \mathcal{O}_4\}$ denotes the set of objects that are direct neighbors of $\mathcal{O}_3$.

We define the following notations:

- $|\mathcal{N}_i|$ denotes the number of objects that belong to the nest $\mathcal{N}_i$;
- $d(\mathcal{O}_i, \mathcal{O}_j)$ denotes the distance between objects $\mathcal{O}_i$ and $\mathcal{O}_j$ (we use an euclidean distance in the parameters space[1]);

[1] If each object $\mathcal{O}_i$ is described by M numerical parameters, the euclidean distance is given by: $d(\mathcal{O}_i, \mathcal{O}_j) = \sqrt{\sum_{k=1}^{M} (\mathcal{O}_i^k - \mathcal{O}_j^k)^2}$, where $\mathcal{O}_i^k$ (resp. $\mathcal{O}_j^k$) denotes the value of the $k-$th parameter value of object $\mathcal{O}_i$ (resp. $\mathcal{O}_j$).

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

- $\mathcal{V}(X)$ corresponds to the Von Neumann neighborhood of X, where $X \in \mathcal{O} \cup \mathcal{A} \cup \mathcal{N} \cup \mathcal{G}$;
- $\mathcal{E}(X)$ corresponds to the extended empty neighboring cells of object X ($X \in \mathcal{O} \cup \mathcal{A} \cup \mathcal{N} \cup \mathcal{G}$). $\mathcal{E}(X) = \{cells = \phi \in \mathcal{V}(Y), Y \in \mathcal{V}(X)\}$. See figure 1;
- $v(X)$ denotes the set of objects that are direct neighbors (i.e the Von Neumann neighborhood) of X ($X \in \mathcal{O} \cup \mathcal{A} \cup \mathcal{N} \cup \mathcal{G}$). See figure 1.

First, the toroidal grid is built with at least $4 \times O$ cells (side length of the grid: $\lceil \sqrt{4 \times O} \rceil$) and nests are regularly scattered on the grid so that there are at least several empty cells between each couple of nests (in order to have an initial separation of nests). Then objects of $\mathcal{O}$ are randomly scattered on the grid $\mathcal{G}$ (with a maximum of one object per cell). Objects can not be initially positioned on one nest.

A ants are equally assigned to the N nests (hence, about A/N ants are available for each nest) and are given a random location on the grid. One can note that in API for clustering, several nests are introduced, but it was not the case for continuous optimization.

3.3 Description of ant-agent behavior

Once the objects, nests and ants are initialized, the algorithm simulates the ants behavior: at each time step, ants can move and perform an action (pick-up and deposit an object). An ant can move from its cell to one of the four neighboring cells and its decisions are governed by probabilities that we describe precisely in the following.

Ant behavior for picking-up objects. If one ant finds an object (i.e. the object is on the same cell as the ant), then its behavior depends on the object:

- if the object is free (i.e. it does not belong to any nest) or its similarity with its nest is worse than with the nest of the ant, then the ant picks-up the object $\mathcal{O}_j$ with the probability:

$$P_{\text{pickup}}(\mathcal{O}_i, \mathcal{N}_j) = \left(\frac{f^{\text{pickup}}(\mathcal{O}_i, \mathcal{N}_j)}{k_p + f^{\text{pickup}}(\mathcal{O}_i, \mathcal{N}_j)} \right)^2 \tag{5}$$

  where $\mathcal{N}_j$ is the nest from which the ant belongs to, k_p is a constant parameter of the algorithm, and f^{pickup} is a density function calculated according to distances between object $\mathcal{O}_i$ and objects already in nest $\mathcal{N}_j$:

$$f^{\text{pickup}}(\mathcal{O}_i, \mathcal{N}_j) = \max \left\{ \frac{1}{|\mathcal{N}_j|} \sum_{\mathcal{O}_k \in \mathcal{N}_j} 1 - \frac{d(\mathcal{O}_i, \mathcal{O}_k)}{\alpha}; 0 \right\} \tag{6}$$

  where $|\mathcal{N}_j|$ denotes the number of objects belonging to $\mathcal{N}_j$, α is a constant and a given parameter to the algorithm.Note that if the nest $\mathcal{N}_j$ is empty (i.e. $|\mathcal{N}_j| = 0$) the probability $P_{\text{pickup}}(\mathcal{O}_i, \mathcal{N}_j)$ is set to 1. That means that the first free object found by an ant of an empty nest is systematically picked up.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

– if the object $\mathcal{O}_i$ already belongs to a nest $\mathcal{N}_j$, the similarity between $\mathcal{O}_i$ and a nest $\mathcal{N}_k$ is calculated by:

$$g(\mathcal{O}_i, \mathcal{N}_k) = \frac{f^{\text{pickup}}(\mathcal{O}_i, \mathcal{N}_j)}{f^{\text{pickup}}(\mathcal{O}_i, \mathcal{N}_k) + \varepsilon} \tag{7}$$

where ε is an arbitrary small value. If $g(\mathcal{O}_i, \mathcal{N}_k) < 1$ then object $\mathcal{O}_i$ is considered to be more similar to nest $\mathcal{N}_k$ than to its own nest ($\mathcal{N}_j$) then the ant picks up the object $\mathcal{O}_i$.

Ant behavior for dropping objects. When an ant has just picked up an object $\mathcal{O}_i$, it goes straight back to its nest $\mathcal{N}_j$ and lays down the object in a free cell of the neighborhood $\mathcal{E}(\mathcal{N}_j)$ of its nest. For each empty cell c of $\mathcal{E}(\mathcal{N}_j)$ the probability of dropping the object $\mathcal{O}_i$ is given by:

$$P_{\text{drop}}(\mathcal{O}_i, c) = \left(\frac{f^{\text{drop}}(\mathcal{O}_i, c)}{k_d + f^{\text{drop}}(\mathcal{O}_i, c)} \right)^2 \tag{8}$$

Similarly to equation 5, k_d is a constant value given as a parameter of the algorithm. Function f^{drop} is a kind of density function (also similar to the function given in formula 6) calculated as follows:

$$f^{\text{drop}}(\mathcal{O}_i, c) = \max \left\{ \frac{1}{|v(c)|} \sum_{\mathcal{O}_k \in v(c)} 1 - \frac{d(\mathcal{O}_i, \mathcal{O}_k)}{\alpha}; 0 \right\} \tag{9}$$

Recall that α is the same parameter than in formula 6 and $v(c)$ is the set of objects that are in the direct neighborhood of cell c. Moreover, if $v(c) = \emptyset$ then $f^{\text{drop}}(\mathcal{O}_i, c) = 1$ (the probability to drop an object to a cell without any object in its neighborhood is 1).

To summarize the dropping and picking-up behavior of one ant we can say that each time an ant takes an object, it brings it back straight to its nest, drops it in the neighborhood of the nest and goes back to the cell where the object has been found. This is exactly what an ant of *Pachycondyla apicalis* species does when it captures a prey: the prey is brought back straight to the nest and at its next nest exit, the ant goes straight to the position, the so called hunting site, of its last prey catch.

Recruitment behavior In addition to the foraging behavior of ants, it is possible for one ant to recruit another ant. The main goal of this behavior is to self-adapt the ant population size of each nest to the size of the corresponding cluster. Consequently, with this mechanism, it is possible that one nest gets empty of its ants if they have been attracted by other nests and then disappears from the grid.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Building new nests As previously said, the recruitment behavior would allow some nests to disappear, so it would be interesting to introduce a counterbalancing behavior to produce new nests according to some disappointed ants looking for a better place.

The two last behaviors are interesting when we focus on the ability of API algorithm to find the number of classes within the data. But, in this first study, we are more concerned with the ability of the algorithm to converge towards a solution. So these two behaviors are not included yet but may be useful in further studies. Algorithm 1 displays the API algorithm for clustering as it will be evaluated in the next experiment section.

Algorithm 1 API for data clustering

Initialization: objects and nests are scattered randomly on the grid $\mathcal{G}$, ants are on the same cell than their respective nests.
for $T = 1$ to $T_{\max}$ **do**
  for each nest $\mathcal{N}_i \in \mathcal{N}$ **do**
    for each ant $\mathcal{A}_j$ of nest $\mathcal{N}_i$ **do**
      if $\mathcal{A}_j$ is not carrying an object **then**
        if object $\mathcal{O}_k$ is on the same cell than $\mathcal{A}_j$ **then**
          if $\mathcal{O}_k$ does not belong to any nest **then**
            Ant $\mathcal{A}_j$ picks up object $\mathcal{O}_k$
          else
            if object $\mathcal{O}_k \in \mathcal{N}_\ell$ with $\ell \neq i$ **then**
              if $g(\mathcal{O}_k, \mathcal{N}_i) < 1$ **then**
                $\mathcal{O}_k$ is picked up by ant $\mathcal{A}_j$ with a probability $P_{\text{pickup}}(\mathcal{O}_k, \mathcal{N}_i)$ (see formula 6)
              end if
            end if
          end if
        else
          {there is no object on current ant's cell}
          the ant is doing a random walk in its Von Neuman neighborhood
        end if
      else
        $\mathcal{A}_j$ memorizes its current position, goes back to its nest $\mathcal{N}_i$ and deposits the carried object $\mathcal{O}_k$ in $\mathcal{E}(\mathcal{N}_i)$ according to the probability $P_{\text{drop}}(\mathcal{O}_k, c)$ with $c \in \mathcal{E}(\mathcal{N}_i)$ (see formula 8)
        $\mathcal{A}_j$ comes back to the memorized cell
      end if
    end for
  end for
end for
return Position of each object on the grid and its corresponding nest

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

4 Experiments

In this section we describe our first experiments with API algorithm for the clustering problem. In a first step we need to evaluate if API is able to produce valid results on classical datasets and secondly we provide a comparison of our results to K-means and ACA. We will consider the two well known Iris and Wine datasets (taken from the Machine Learning Repository). Iris dataset is made of 150 objects, each object is described with 4 numerical values and there are 3 initial clusters. Wine dataset is made of 178 objects, each object is described with 13 numerical values and there are also 3 initial clusters. The data attributes and distances are normalized before using ants.

To study the clustering ability of API, we first focus on two classical indexes: F-Measure, the Classification Error (see [8] for a precise description of these indexes) which will measure the behavior of API during the clustering process. However, clustering quality evaluation is a difficult question and using several points of view can be very informative when we study a new clustering process such as API. In the second step, we provide two more indexes, the Intra cluster variance and the Dunn Index.

Our experiments are conducted with the following values for parameters: $\alpha = 0.5, k_p = 0.1, k_d = 0.15, \epsilon = 0.03$, the number of ants is chosen to be 10. These values are the commonly used values in ant-based clustering methods based on LF model. In the present version of the paper, API is semi-supervised since the number of clusters cannot vary during one run. Consequently, we will focus our experiments to study the classification process without considering the problem of finding the best number of clusters. Of course it should not be the case for an unsupervised clustering method and next step in API will be to introduce an adaptive variation of nests/clusters number.

The Figure 2 gives the evolution of 3 runs of API (10,000 iterations) in order to show the global trend of the performance measures over the time. The F-Measure, computed as the weighted average of the precision and recall, shows that in both datasets, API is reaching good values of F-Measure (i.e. 1 which represents a good equilibrium between precision and recall). Also along the clustering process, the Classification Error is continually decreasing.

Table 1 (*resp.* Table 2) presents the mean and standard deviation values (obtained from 10 independant runs) of the evaluation functions for API(10,000 iterations) and K-Means when the number of clusters is set to $k = 2$ (*resp.* $k = 4$).

Table 3 and Table 4 present the comparison of the results of API, K-means and ACA. API and ACA are run (10 independent runs) during 1,000,000 iterations with 10 ants. The table gives the mean values and standard deviations obtained with Iris and Wine datasets for the 4 indexes. According to the results obtained with these 3 measures, API results are close to K-means, and they slightly improve ACA results. While K-means performs well than API and ACA on wine dataset.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

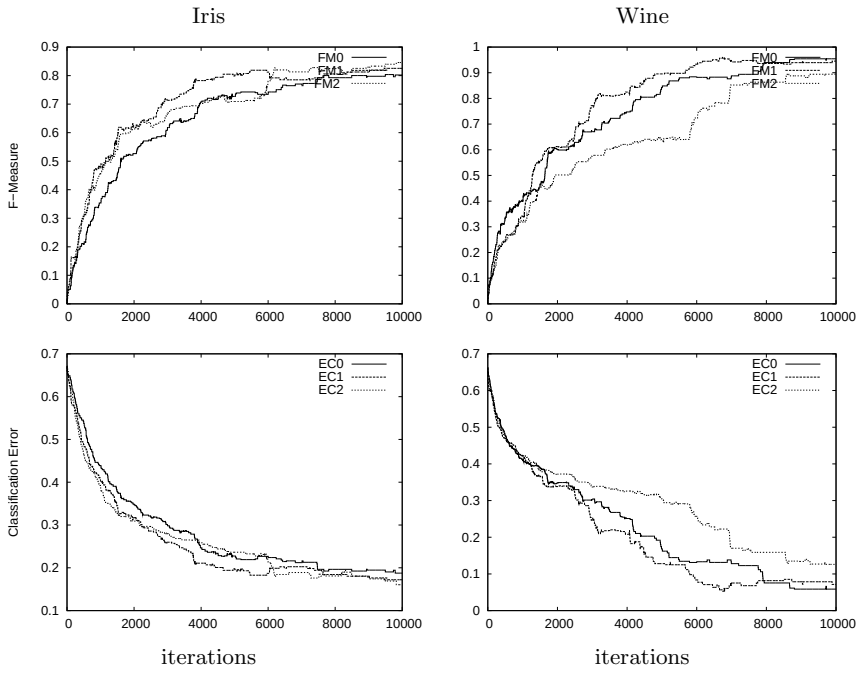


Fig. 2. Evolution of F-Measure (FM) and Classification Error (EC) along the first 10,000 iterations of 3 independent runs for Iris (left) and Wine (right) datasets.

We can conclude this experimental section by saying that the first results obtained by API are very promising, both in terms of quality and time, even if several points would need deeper experiments.

5 Conclusions

In this article, we have presented an adaptation of API meta-heuristic to the clustering problem. Using the Lumer and Faieta based approach we have applied successfully API for clustering by mimicking the natural sorting and clustering behavior of these ants. Therefore, API is not based ACO or hybridization with another clustering paradigms. Our results are very encouraging compared to ACA and K-means algorithms. Then, our further work is to implement the recruitment behavior: a mechanism that replaces stigmergy and the determination of nests called construction of nests is to be automatic. An important part of this further work will be to give a rigorous study of the parameters choice as well as a comparative study of ant-based clustering algorithms by using a non-parametric test. Finally, we will also investigate real application of our method on images and clustering in wireless sensor network.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

	Iris dataset		Wine dataset	
	API	**K-Means**	**API**	**K-Means**
Classification Error	0.223(0.000)	0.223 (0.000)	0.31 (0.004)	0.315 (0.009)
F-Measure	0.777 (0.000)	0.777 (0.000)	0.652 (0.012)	0.677 (0.016)
Dunn-Index	11.473 (0.040)	0.575 (0.000)	2.851 (0.303)	0.309 (0.004)
Intra Cluster Variance	2.76 (0.007)	0.0181 (0.000)	0.666 (0.008)	0.106 (0.0004)

Table 1. Results obtained with API and K-means algorithms for Iris and Wine datasets when $k = 2$

	Iris dataset		Wine dataset	
	API	**K-Means**	**API**	**K-Means**
Classification Error	0.166 (0.008)	0.211 (0.019)	0.108 (0.006)	0.114 (0.019)
F-Measure	0.800 (0.008)	0.753 (0.029)	0.889 (0.008)	0.870 (0.037)
Dunn-Index	2.342 (0.517)	0.396 (0.089)	1.328 (0.162)	0.194 (0.023)
Intra Cluster Variance	1.915 (0.374)	0.019 (0.001)	0.345 (0.007)	0.162 (0.008)

Table 2. Results obtained with API and K-means algorithms for Iris and Wine datasets when $k = 4$

Iris dataset	**API**	**K-Means**	**ACA**
Classification Error	0.123 (0.004)	0.165 (0.009)	0.230 (0.053)
F-Measure	0.89 (0.006)	0.838 (0.015)	0.773 (0.022)
Dunn-Index	5.04 (0.106)	0.411 (0.068)	2.120 (0.628)
Intra Cluster Variance	2.37 (0.089)	0.018 (0.0004)	4.213 (1.609)

Table 3. Results obtained with API, K-means and ACA for the Iris dataset

Wine dataset	**API**	**K-Means**	**ACA**
Classification Error	0.072 (0.007)	0.053 (0.006)	0.142 (0.030)
F-Measure	0.943 (0.006)	0.96 (0.004)	0.855 (0.023)
Dunn-Index	2.233 (0.177)	0.293 (0.005)	1.384 (0.101)
Intra Cluster Variance	0.460 (0.133)	0.123 (0.035)	8.521 (0.991)

Table 4. Results obtained with API, K-means and ACA for the Wine dataset

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

References

1. L. Admane, K. Benatchba, M. Koudil, L. Siad, and S. Maziz. Antpart: an algorithm for the unsupervised classification problem using ants. *Applied Mathematics and Computation*, 180(1):16 – 28, 2006.
2. E. Bonabeau, M. Dorigo, and G. Theraulaz. *From Natural to Artificial Swarm Intelligence*. Oxford University Press, 1999.
3. Urszula Boryczka. Ant Clustering Algorithm. In *Intelligent Information Systems*, number 1998, pages 455–458. Ieee, December 2008.
4. C. Sammut and G. I. Webb, editors. *Encyclopedia of Machine Learning*.
5. J L Deneubourg, S Goss, N Franks, A Sendova-Franks, C Detrain, and L Chrétien. The Dynamics of Collective Sorting: Robot-Like Ants and Ant-Like Robots. In J A Meyer and S W Wilson, editors, *proc. 1st Int Conf on Simulation of Adaptive Behaviour*, pages 356–363. MIT Press, 1991.
6. M. Dorigo, M. A. Montes de Oca, S. Oliveira, and T. Stützle. *Ant Colony Optimization*. John Wiley & Sons, Inc., 2010.
7. A. Hamdi, V. Antoine, N. Monmarché, A. Alimi, and M. Slimane. Artificial ants for automatic classification. In N. Monmarché, F. Guinand, and P. Siarry, editors, *Artificial Ants: from collective intelligence to real life optimization and beyond*, chapter 13. ISTE - Wiley, 2010.
8. J Handl, J Knowles, and M Dorigo. Ant-based clustering and topographic mapping. *Artificial Life*, 12(1):35–61, 2006.
9. J. Handl and B. Meyer. Ant-based and swarm-based clustering. *Swarm Intelligence*, 1(2):95–113, december 2007.
10. S.L. Ho, S. Yang, G. Ni, and J.M. Machado. A modified ant colony optimization algorithm modeled on tabu-search methods. *IEEE Transactions on Magnetics*, 42(4):1195–1198, April 2006.
11. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31:264–323, September 1999.
12. Anil K Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.
13. Y. Lu, S. Wang, S. Li, and C. Zhou. Particle swarm optimizer for variable weighting in clustering high-dimensional data. *Mach. Learn.*, 82:43–70, January 2011.
14. E Lumer and B Faieta. Diversity and adaptation in populations of clustering ants. In J A Meyer and S W Wilson, editors, *Proc. of the Third Int. Conf. on Simulation of Adaptive Behavior From Animals to Animats 3*, pages 501–508, 1994.
15. D. Martens, B. Baesens, and T. Fawcett. Editorial survey: swarm intelligence for data mining. *Machine Learning*, 82(1):1–42, September 2010.
16. N. Monmarché, G. Venturini, and M. Slimane. On how *Pachycondyla apicalis* ants suggest a new search algorithm. *Future Generation Computer Systems*, 16(8):937–946, 2000.

An Evaluation of Adaptive Control for Evolutionary Algorithms

Giacomo di Tollo[1], Frédéric Lardeux[1], Jorge Maturana[2], and Frédéric Saubion[1]

[1] LERIA, University of Angers (France)
name@info.univ-angers.fr
[2] Instituto de Informática, Universidad Austral de Chile (Chile)
jorge.maturana@inf.uach.cl

Abstract. Adaptive evolutionary algorithms have been widely developed to improve the management of the balance between intensification and diversification during the search. Nevertheless, this balance may need to be dynamically adjusted over time. In this paper we investigate how an adaptive controller can be used to achieve more dynamic search scenarios and what is the real impact of possible combinations of control components. This study may be helpful for the development of more autonomous and efficient evolutionary algorithms.

1 Introduction

Recent advances in adaptive evolutionary algorithms (EAs) provide promising opportunities for designing more autonomous solvers for optimization problems [12]. The behaviour and the performances of EAs are actually well known to be subjected to –often numerous– parameters that are difficult to adjust. One may indeed identify two general classes of parameters: *behavioural*, mainly operator application rates or population size and *structural*, that define the main features of the algorithm and could eventually transform it radically, e.g., those related with the encoding and the choice of operators. Parameter setting in EAs [17] is thus nowadays a clearly identified problem and the methods to do it are usually classified according to the taxonomy proposed in [8]. On the one hand, impressive progresses have been made in automating tuning methods for providing efficient tools [3, 14, 21] that are able to outperform traditional empirical and costly tuning tasks. On the other hand, another approach consist in controlling the parameters along the run in an adaptive process [22, 11, 10, 19, 18].

From a high level point of view, Evolutionary Algorithms (EAs) [9] manage a set (population) of possible configurations of the problem solutions(individuals), which are progressively modified by variation operators, in order to converge to an optimal solution or, at least, to a sub-optimum of good quality. The management of the balance between the exploration and the exploitation of the search space (also known as *diversification* and *intensification*) is largely recognized as a key feature for the overall performance of the search process.

In this paper, we focus on the adaptive control of operators, i.e., which operator –among several ones– should be applied at a certain time during the search.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Approaches such as Adaptive Pursuit [22] or Adaptive Operator Selection (AOS) [11, 10] have shown promising results tackling this problem, however, they often focus only in the quality of the individuals or the population. This may cause premature convergence, given the fact that just fitness-improving operators would be systemically chosen. In order to avoid this problem, [19] proposed to consider both diversity and quality as intensification and diversification measures. In this way, each operator is characterized in terms of their ability to either exploit or explore the search space. This knowledge is the used to determine the next operator to apply. The work presented in [18] combines the ideas coming from previous studies [10, 6, 19] aiming to obtain a controller that be both generic and produces quality results. In [20], the adaptive management of the operators themselves is also addressed. In all cases, however, the trade-off among intensification and diversification, used to guide the search, is kept fixed. Since the search may need different emphasis on these trends in different moments of the search in order to obtain better results [16], we want to explore ways to adjust this trade-off dynamically. In this work, our purpose is to provide a fair a methodological experimental analysis of the controller behaviour. In [7] , we have exhibited preliminary results and in this paper, we test now more hypothesis on the operators management and we also focus on the real solving ability of the controller. We also want to asses the benefit of dynamic control strategies, especially from a solving performance point of view. Such a study will help algorithm designers to better understand the actual effects of such adaptive control techniques and thus to select the suitable components for achieving more autonomous algorithms.

2 Towards a Generic Controller for Variation Operators

Adaptive Operator Selection (AOS) can be formalized into 4 distinct stages, whose aim is to analyse the behaviour of the operators over the current run and to determine which operator should be applied next. These stages constitutes a chain of components that can be depicted by figure 1. Each component has specific inputs and outputs and specific parameters that will be fully described below.

The Criteria : Quality, Diversity and Time The first thing to do is to select suitable criteria in order to assess the overall performance of an evolutionary algorithm. Based our previous works [20, 18, 19], we use both the mean quality and the diversity (entropy) of the population that reflect the balance between intensification and diversification. The computation time may also be important when dealing with operators that have a different behaviour and perform different computation processes on the individuals. Therefore, after each computation step, the EA send to the impact computation module, the name of the operator that has been applied and the observed variation of the values of these criteria.

Impact Computation The main purpose of impact computation is to record the impact of the successive applications of an operator along the run. This

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

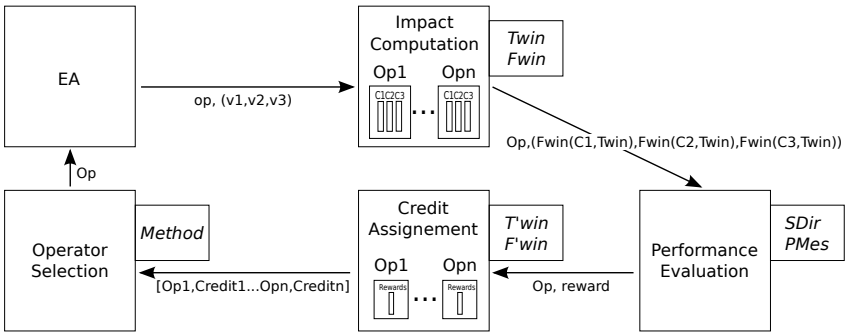


Fig. 1. AOS General Scheme

impact corresponds to the criteria previously defined. In order to avoid the noise in measures, the values are recorded over a sliding window of a given size $Twin$. Then, the impact is calculated with a function $Fwin$ applied on this window (e.g., mean value, max value). Therefore, as output, impact computation provide a tuple of values for the last applied operator, where each value of the tuple corresponds to a criteria.

Performance Evaluation The purpose of this stage is to evaluate the performance of an operator with regard to a particular search trajectory $SDir$. This search trajectory corresponds to a compromise between the chosen criteria. Here we plot the computed impact on the two-dimensional space corresponding to the variations of quality and diversity. It is important to note that intensification and diversification are generally not well defined. In our context, thanks to the two complementary criteria, diversification means that diversity is favoured while intensification means that mean quality is preferred. Then a measure $PMes$ is used to assess the impact with regards to the desired direction. We use here a simple scalar product between the search direction and the resulting direction computed for the operator. This value is normalized w.r.t. the computation time. Therefore, the performance evaluation module aggregate the impact of an operator as a single value that represents its reward according to the desired search trajectory. Notice that other $PMes$ can be used, including for instance Pareto dominance based functions (see [20]).

Credit Assignment As for impact computation, the rewards can be computed over a given period of time $T'win$ using a specific function $F'win$. The purpose of the credit assignment is thus to compute the credit assigned to the operator from its successive rewards. Therefore, at this step, all the observations concerning an operator are aggregated into a single value.

Operator Selection The operator selection component selects the operator to be applied by the EA for the next computation step, according to the credits that have been assigned to the operator. Different $Methods$ can be used in order

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

to progressively learn what operator is the most suitable. Probability Matching (PM) is probably the most common one, choosing operators with a probability proportional to their rewards. Operator selection is also related to reinforcement learning, since we are interested in discovering the optimal application policy that applies the best possible operators, without neglecting the potential offered by formerly-bad and not-recently-used ones. We thus use a multi-armed bandit based method (MAB). These methods may involve their own parameters as deeply studied in [18].

3 Evolutionary Algorithm and Controller

This section presents the details of the the algorithms used to perform the experiments.

Evolutionary Algorithm We use a steady state evolutionary algorithm to solves the satisfiability problem (SAT) [2]. The selection process is a classic tournament selection over three randomly chosen individuals and the insertion procedure is a classic replacement of the oldest individual of the population. The algorithm applies one operator at each step producing one individual from two parents. The operator to be applied is selected by the controller. Therefore, the parameters of the algorithm are just the size of the population and the number of allowed generations.

We use the SAT problem to investigate how the devised controller influences the search. We have chosen the satisfiability problem (SAT) for two reasons: first, because SAT can encode a variety of problems with different fitness landscapes; then, because we use a SAT-specialized algorithm [15] that has several crossover operators whose performance is already known due to previous studies [20].

The Operators We have used a selection of 20 crossover operators from the set of more than 300 crossover operators defined in [20]. We should insist here on the fact that these operators are specialized to the SAT problem and mainly correspond to combination of four basic features: selection of clauses that are false in both parents, action on each of the false clauses, selection of clauses that are true in both parents and action on each of the true clauses.
All variables that remain undefined in the child are valued using the uniform crossover process.

The Controller For this study, we used the controller presented in [18], called Compass. Briefly explained, the profile of each operator is build based on the effects that its application has caused in the population in recent applications. A parameter θ, corresponding to an angle in a geometrical representation of operators profile over a scatter plot, defines the "intention" of the controller when selecting the next operator to apply: values of θ range from 0, that favours diversification, to $\pi/2$ that favours intensification.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Experimental Settings Population size has been set to 30 for every different experiments. To show that the controller's behaviour is not dependent on the category of instances, we used 32 different SAT instances coming from different problems categories (3-colour-able flat graphs [13], Subgraph Isomorphism Problems [1], Hard handmade instance [4], Random 3-SAT instances sampled from the phase transition region [5]). Instances have been chosen because they are representative of different kind of instances and show different landscapes[1]. Experiments have been run on a 280-core, 792 GFlop computer cluster. As for computational time, the execution time is constant for each operator application, so it will be assessed as the number of crossovers performed during the search.

As for the strategies used,we will focus on: using a *fixed value* for the parameter θ during the whole execution; splitting the execution in several epochs and alternating the angle value between 0 and $\frac{\pi}{2}$ (*Alwaysmoving*) , either starting with $\theta = 0$ (*Alwaysmoving* 01) or $\theta = \frac{\pi}{2}$ (*Alwaysmoving* 10); splitting the the execution time in several epochs and set angle values at equally distributed levels in $[0, \frac{\pi}{2}]$ for each one. Here the angle can either increase (*Angleincrease*) or decrease (*Angledecrease*). As Method, we will use PM and MAB, described in section 2. Since several combinations of $Method-FWin$ and Operator Selection Method are possible, we aim in introducing strategies which are robust w.r.t. these parameter settings. [2]

4 A Methodological Comparison

In this section we propose a principled approach for assessing the performances of our possible controller configurations. This is an attempt of definition of clear methodology for comparing performances of adaptive solvers. The objective is to check the reliability of an adaptive techniques when faced to various benchmarks and to various settings of the controlled algorithm. It is necessary to study its performance with regards to both its ability to solve the problem and the reliability of its learning components. To do so, we propose to answer two main questions, that seem to us as essential when studying a learning based adaptive approach:

- Is the controller able to perform better than a random uniform selection?
- Is the controller reliable when faced to a noise in the operators of the controlled algorithm?

[1] For more details, we forward the interested reader to the SAT competition's website `http://www.satcompetition.org/`.

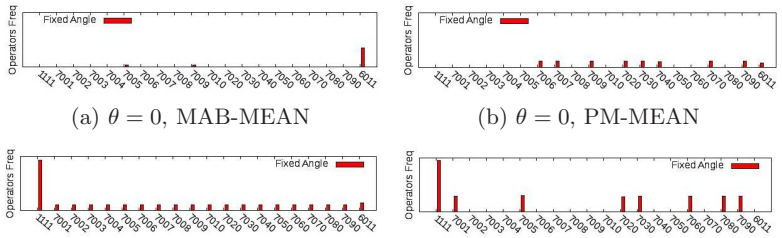
[2] In the following, we will refer to [In/De]crease-th[n] to indicate the *Angleincrease* or *Angledecrease* strategy which takes into account n values. The starting θ value is 0 in *Increase* and $\pi/2$ in *Decrease*. For each strategy, the combination of $Method-FWin$ and Operator Selection Method will be noted between squared brackets, e.g., [MAB-MAX].

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

4.1 Assessing the Benefit of the Learning Process

Management of the Operators Preliminary experiments have been made in order to show that the controller is able to identify the right operators needed for the current balance of diversification-intensification required.

We use the term "null operator" to identify operators that take two individuals as parents, and produce the very same individuals as offset, thus having no effects over the population, neither in terms of diversification nor in terms of intensification. We have performed experiments pooling a diversification-oriented operator (6011) and an intensification oriented one (1111) together with 18 null operators (identified by the tuples $70**$), in order to understand if the controller discriminates amongst the proposed operators in function of the desired level of intensification-diversification required. The desired outcome of these experiments is to get the desired search direction in term of entropy and quality, depending on the current angle value, while neglecting completely the null operators. When using a fixed angle,*MAB* is able to identify the required operators, either using *MEAN* and *MAX*. PM instead, show that it is not utterly able to detect the right operator, since if a wrong operator has been applied at the beginning of the search, its probability to being applied stays high until a more suitable operator has been found (see Fig. 2). This difference between Operator Selection Methods is even stronger when introducing a strategy to change the angle value during search (i.e., *Alwaysmoving*): the high rewards given to a non-null operator during the first phase of the search, make the other non-null difficult to be detected when the desired intensification-diversification change when using PM (see Fig. 3 (a) and (b)). The use of MAB, instead, does not show this shortcoming, and the right operator is soon detected and used (see Fig. 3 (c) and (d)).



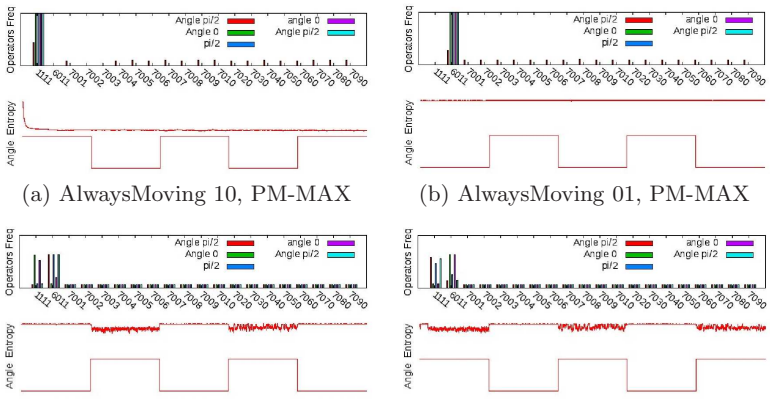
(a) $\theta = 0$, MAB-MEAN (b) $\theta = 0$, PM-MEAN

(c) $\theta = \pi/2$, MAB-MEAN (d) $\theta = \pi/2$, PM-MEAN

Fig. 2. Experiments with null operators, SAT instance 3BIT and different settings of $Method - FWin$, fixed angle θ

Adaptive Selection Process vs. Random Selection As a first step in our experimental analysis, we will compare the quality of solution found by using the controller devised in section 2, with a genetic algorithm which relies on the same

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer



(a) AlwaysMoving 10, PM-MAX (b) AlwaysMoving 01, PM-MAX

(c) AlwaysMoving 01, MAB-MEAN (d) AlwaysMoving 10, MAB-MEAN

Fig. 3. Experiments with null operators, SAT instance SIMON, different θ-variation strategies and settings of $Method - FWin$, fixed angle θ

operators introduced in section 3, but that uses a random selection procedure. As for the other parameters, they stay the same between the two approaches. As for the controller, we have used $\theta = \frac{\pi}{4}$ since, accordingly to previous research[20], it leads to the best compromise between intensification and diversification. Results are shown in Fig. 4 (a), where the quality (number of false clauses: low values indicate better solution) of the best solution found over 30 runs is reported on the y-axis. Each projection of points over the x-axis represents an instance.

Some interesting phenomena can be observed by analysing these results. As a first remark, a simple strategy such PM attains better results than a more complex one as MAB over almost all instances: best results are found using the combination [PM MEAN]. This is rather surprising , since the introduction of MAB should be useful to overcome the shortcomings of PM, such as the likelihood of sticking on applying the same operators. However, these results show that such an hypothesis is not useful when the angle value is $\theta = \frac{\pi}{4}$, and seems to suggest that the benefits of having the best static compromise between intensification and diversification is counterbalanced by loosing the possibility of exploiting better operators capabilities. This is more evident when comparing the results provided by the controller with the ones found by the random selection procedure: even though PM shows generally the best results, those are not always better than the random ones, and in some cases they lead to the same results. The difference between the two Operator Selection Methods is not significant, and performances are not satisfactory (assignments are hardly found): the need for a dynamic strategy arises.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

4.2 Assessing the Control Ability

Amongst the controller main features, we point out its ability to control the level of intensification–diversification desired by means of the parameter θ, in order to discriminate and use the operator needed to reach a certain trade-off between intensification and diversification. It is easy to see the effect of imposing a given angle value by analysing the behaviour of the population in terms of false clauses: $\theta = \frac{\pi}{2}$ leads the population to converge toward low false clauses solutions, whilst using $\theta = 0$, we observe a spread amongst the individuals' false clauses. $\theta = \frac{\pi}{4}$ instead, leads to behaviours which favour either diversification or intensification, and not necessarily an intermediate compromise between them.

It has been remarked that in most cases it seems to exist a threshold from where the behaviour changes radically, supporting the hypothesis of the "positive feedback" proposed in [18], except that this threshold is not always found at $\theta = \pi/4$ as suggested therein.

Changing the angle during the search can better help us to assess the control ability: dynamic strategies which account for progressive angle variations (*Decrease* and *Increase*) do not represent an advantage in terms of control, since the search behaviour does not react to progressive angle changes, but rather when trespassing a threshold. *AlwaysMoving*, on the contrary, produces the desired switch between intensification and diversification at the right time, reacting to sudden angle changes (see Fig. 3), no matter other algorithm's parameters.

This control mechanism affects the Operator Selection during search: as a first step, we remark that when favouring diversification ($\theta = 0$), the controller is always able to detect the operator 6011, no matter the used criteria; when imposing the maximum desired quality ($\theta = \frac{\pi}{2}$) instead, the controller is able to identify operator 1111. Note that when applying MAB, the operator selection process makes all operators to be selected at least a minimum number of times through the search: for this reason, also operators which are known to not provide any intensification are used. This is the case of the diversifying operator 6011, that has been detected and used in both *MEAN* and *MAX* as *FWin*. The same does not happen with PM: if an operator has not been randomly selected and used in the first epochs, it is not likely to be selected any more.

By letting the angle change, the operator frequency of application self-adapts to extreme angle changes, and operator 6011 is always detected in the right phase for almost every controller settings.

The behaviour of AngleIncrease and AngleDecrease is a bit less clear, since when applying Increase [PM *MAX*] identifies soon the operator 6011, applying it less and less as the angle value increases. On the other hand [PM *MEAN*] also discriminates among search epochs, but the operator 6011 is uniformly applied over the first two epochs, then showing no changes triggered by a new angle value. In *ANGLEDECREASE*, PM does not detect the operator 6011 as diversifying agent: This is given by the fact that in the first search stages, when intensification is required, high rewards are assigned to other operators, which make them to be applied also in further search stages. MAB does not show this shortcoming,

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

being able to correctly identify the operator 6011, in both *ANGLEINCREASE* and *ANGLEDECREASE*.

We can conclude that, in both terms of operators management and effect over the population quality and diversity, the AlwaysMoving strategy is the one which "controls" the best.

4.3 Assessing Solving Performances

As seen in Section 4.2, Alwaysmoving is the strategy most capable to select operators according to the desired balance of intensification-diversification required. Nevertheless, after having run the experiments we remark that its performances are far from being satisfactory (see Fig.4(b)).

We remark nevertheless that in this strategy, the operator control influences the time solution are found on. This phenomenon can be observed in Figure 4(d): We have determined, for each strategies, the time (crossover) in which the best solution[3] have been found, on every instances and on every runs. Then, we have pooled together the points, to show the aggregate behaviour. The four main strategies are grouped in the y axis: AlwaysMoving, Fixed Angle, Increase, Decrease; inside the group, every y-coordinate indicates a pair of [Instance-Run][4]. Time is reported on the x-axis. We see that fixed angles lead to find the best solutions in the very first phase of the search, and introducing Always moving leads to a diversification phase that is useful to find better solutions in the next intensification epoch (the fit between intensification epochs and best solution found is evident). *Increasing* and *Decreasing* show best solutions spread more uniformly over the whole search procedure. Thus, we can remark that all dynamic strategies offer a good exploitation of the computational time allocated for execution, whilst fixed angle ones found the best solution at the beginning of the search, then entering into an idle phase.

For this reason, *AlwaysMoving* can be useful to improve solutions found by fixed angle strategies: an extreme change on the required balance of Intensification–Diversification, leads to better solutions to be found: this is the case for all fixed strategies but $\frac{\pi}{2}$ [PM-MEAN], whose results are already good, but whose variability is too high (see Fig.4 (c)).

Best results are offered by *Increase*. An analysis of the time in which solutions are found show similar results for all combinations of [Mean, Max] and [PM, MAB], leading to stable results in term of solution quality, and more stable than AlwaysMoving in terms of solution time, no matter the Fwin and Operator Selection method used. *Decrease* performs comparably well. These two strategies

[3] Note that we use the notion of best solution to talk about the best solution found during a run for the given configuration of the controller and not for the best solution among all the runs

[4] We have chosen, respectively, the following combinations: *Alwaysmoving10*[MAB-MAX] with epochs of 20000 crossovers; $\frac{\pi}{2}$, [PM-MEAN]; Increase [PM-MEAN]; Decrease [PM-MEAN]. The overall behaviour of other combinations is similar the ones' chosen.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

perform better than *fixed angles*, except $\frac{\pi}{2}$ [PM-MEAN], which offers the best results amongst fixed angles. Nevertheless, the standard deviation of this fixed strategy is much higher, leading dynamic strategies to produce the most stable results in term of solution quality.

These results are corroborated by a Wilcoxon Pairwise test, which allow us to reject the null hypothesis that fixed angle best results (apart from $\frac{\pi}{2}$ [PM-MEAN]) are drawn from the same distribution than *Increase* and *Decrease*. It is not possible to reject this hypothesis, though, when comparing Fixed Angles with *AlwaysMoving*. Notably, most *Increase* and *Decrease* are not statistically different, no matter the number of threshold taken into account, the Fwin method and Operator Selection method, showing that these strategies are really robust w.r.t. algorithm's parameter settings. It could be surprising that going from diversification to intensification by decreasing the angle allows us to find good solution. But, as already mentioned, this is a bi-dimensional problem and an operator can be used to favour entropy while not necessarily producing only bad individual or deleting the best ones; this is the case when using the operator 6011. Moreover, the mean quality of the population also appears as a good criteria to manage operators, by favouring the other possible search direction.

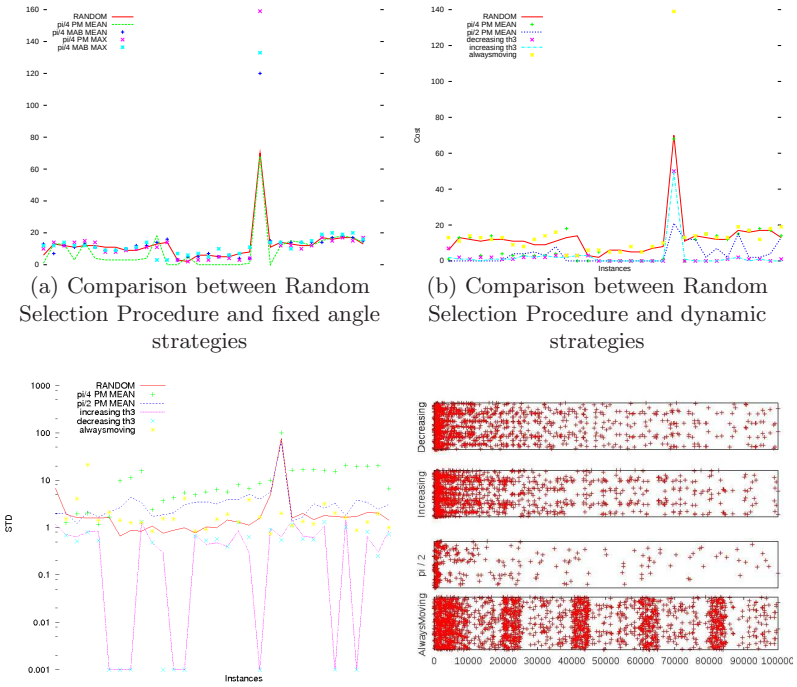
It is necessary to remark that, apart from $\frac{\pi}{2}$ and $\frac{\pi}{4}$ ([PM MEAN]), strategies with fixed angle has been hardly able to find an assignment to the problems: those strategies offer good best results (not statistically different from those obtained with increasing), but their average behaviour is indeed different (worse).

As for the difference between PM and MAB, they are not statistically significant in *Increase* and *Decrease*. When the angle is kept fixed, PM provide better results, but the variability of the output is too high.

5 Conclusion

In this work we have proposed, in the frame of Evolutionary Algorithms, a controller for operator selection, in which the operator is chosen accordingly to the user's desired search balance in term of mean quality and entropy of the population. This balance is determined by means of a parameter (θ) which is modified during the search accordingly to pre-defined strategies. Results show that dynamical strategies performs better than the controller based on keeping fixed this parameter over time, in both terms of solution quality and operators management. Furthermore, the dynamic version make the search to better exploit the computational time allocated to the execution, and it is more robust w.r.t. algorithms parameter's settings. Further research will be aimed to autonomously changing the θ value overtime, in order to better exploit the computational time and to provide better quality solutions.

This work was supported by the French Chilean ECOS-Conicyt program C10E07 and partially by project DID S-2010-50



(a) Comparison between Random Selection Procedure and fixed angle strategies

(b) Comparison between Random Selection Procedure and dynamic strategies

(c) Strategies variability in term of best solution found (standard deviation of the number of false clauses)

(d) Time in which the best solution is found

Fig. 4. Experiments Statistics

References

1. C. Anton and L. Olson. Generating satisfiable sat instances using random subgraph isomorphism. In Yong Gao and Nathalie Japkowicz, editors, *Advances in Artificial Intelligence*, volume 5549 of *Lecture Notes in Computer Science*, pages 16–26. Springer Berlin / Heidelberg, 2009.
2. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
3. M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

4. P. Chatalic and L. Simon. Multi-resolution on compressed sets of clauses. In *Twelth International Conference on Tools with Artificial Intelligence (ICTAI00)*, pages 2–10, 2000.
5. P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are ? In *Proceedings of IJCAI–91*, pages 331–337, 1991.
6. L. Da Costa, Á. Fialho, M. Schoenauer, and M. Sebag. Adaptive operator selection with dynamic multi-armed bandits. In M. Keijzer et al., editor, *Proc. GECCO'08*, pages 913–920. ACM Press, 2008.
7. G. di Tollo, F. Lardeux, J. Maturana, and F. Saubion. From adaptive to more dynamic control in evolutionary algorithms. In *Proceedings of EvoCOP*, pages 130–141, 2011.
8. A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 3(2):124–141, 1999.
9. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003.
10. A. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Extreme value based adaptive operator selection. In G. Rudolph et al., editor, *Parallel Problem Solving from Nature - PPSN X, 10th International Conference*, volume 5199 of *Lecture Notes in Computer Science*, pages 175–184. Springer, 2008.
11. W. Gong, Á. Fialho, and Z. Cai. Adaptive strategy selection in differential evolution. In *Genetic and Evolutionary Computation Conference,GECCO*, pages 409–416. ACM, 2010.
12. Y. Hamadi, E. Monfroy, and F. Saubion, editors. *Autonomous Search*. Springer, 2011. to appear.
13. T. Hogg. Refining the phase transition in combinatorial search. *Artificial Inteligence*, 81:127–154, 1996.
14. F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proc. of the Twenty-Second Conference on Artifical Intelligence (AAAI '07)*, pages 1152–1157, 2007.
15. F. Lardeux, F. Saubion, and J-K. Hao. GASAT: A genetic local search algorithm for the satisfiability problem. *Evolutionary Computation*, 14(2):223–253, 2006.
16. A. Linhares and H. Yanasse. Search intensity versus search diversity: a false trade off? *Applied Intelligence*, 32(3):279–291, 2010.
17. F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.
18. J. Maturana, A. Fialho, F. Saubion, M. Schoenauer, and M. Sebag. Compass and dynamic multi-armed bandits for adaptive operator selection. In *Proceedings of IEEE Congress on Evolutionary Computation CEC*, 2009.
19. J. Maturana and F. Saubion. A compass to guide genetic algorithms. In G. Rudolph et al., editor, *Parallel Problem Solving from Nature - PPSN X, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 256–265. Springer, 2008.
20. J. Maturana, F. Lardeux, and F. Saubion. Autonomous operator management for evolutionary algorithms. *Journal of Heuristics*, 2010.
21. V. Nannen, S. K. Smit, and A. E. Eiben. Costs and benefits of tuning parameters of evolutionary algorithms. In *Parallel Problem Solving from Nature - PPSN X, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 528–538. Springer, 2008.
22. D. Thierens. Adaptive Strategies for Operator Allocation. In F.G. Lobo, C.F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, pages 77–90. Springer Verlag, 2007.

AntBee: Clustering with Artificial ants in hexagonal grid

Amira Hamdi[1,2], Nicolas Monmarché[2], M. Adel Alimi[1] and Mohamed Slimane[2]

[1]Department of Electrical Engineering, National School of Engineers (ENIS)
[2]Université François Rabelais Tours, France, Laboratoire informatique

{amira.hamdi, adel.alimi}@ieee.org
{nicolas.monmarche, mohamed.slimane}@univ-tours.fr

Abstract. Based on the characteristics of hexagonal form of the bee's honeycomb, a swarm based clustering algorithm is presented in this paper. Inspired by the geometric shape observed in honeycomb, we use hexagonal gird to scattered data in ant's environment. Experimental results show that the proposed approach is significantly better than LF algorithm in term of both speed and quality.

1 Introduction

Bees produce more honey than they actually need and store it in honeycombs. The hexagonal structure of the honeycomb attracted attention of mathematicians: why is the shape of a honeycomb cell hexagonal, and not square, round, triangle, rectangle, or even pentagon?

In fact, the natural world contains an infinite variety of hexagonal patterns. The hexagonal shape is found in honeycombs and in many other forms. Researches have shown that only regular hexagons, squares and equilateral triangles can be tessellated together so there is no wasted space. Of the three, the hexagon has the smallest perimeter when their areas are equal.

Pappus tried to explain the hexagonal shape of the bee-cells by the fact that among the regular plane-fillers of equal area the hexagon has the least perimeter. According to a widely spread hypothesis, going back to Pappus, the bees aim at economy: If, by some reason, the volume of a cell and the width of the whole layer are given, they try to use the minimum amount of wax per cell. Although among the various effects which interact in producing the honeycomb the utilitarian human motive attributed to the bees seems to play the least part, the above hypothesis was the source of highly interesting investigations.

In this paper, a previous algorithm is thus improved. In place of the rectangular shape of grid used in LF clustering algorithm, we propose to use an hexagonal grid inspired by the shape of honeycomb. Section 2 reviews some ant based clustering algorithm. Section 3 reports experiments carried out on the proposed approach. Finally, the last section is devoted to the conclusion and further work.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

2 Related work

Self-organization is a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components. [1]

One of the most interesting examples of self-organized systems is brood sorting behavior of ants which has stimulated researches to design new algorithms for data analysis [2]: isolated items should be picked up and dropped at some other location where more items of that type are present. The model of rules formulated by Deneubourg et al. in [2] is relatively simple:

- When an ant carries no element, its probability of picking up one element, encountered along its way, is given by:

$$p_p = \left(\frac{k_p}{k_p + f} \right)^2 \quad (1)$$

Where k_p is a positive constant and f is the proportion of elements that are present in the neighborhood of the ant. When there are few objects in the neighborhood of the object coveted by the ant, $f << k_p$, which means that p_p is close to 1 and the object is very likely to be picked up. On the other hand, when the neighborhood is dense in element, $f >> k_p$, and then p_p is close to 0.

- When an ant carrying an object is moving, its probability of deposing the object is given by:

$$p_d = \left(\frac{f}{k_d + f} \right)^2 \quad (2)$$

Where k_d is a positive constant.

In [4], Lumer and Faieta have generalized Deneubourg's model to apply it to exploratory data analysis with an algorithm we will call LF. The idea is to define a dissimilarity measure between objects that are immersed in a discrete space of lower dimension (typically of 2 dimensions). This discrete space then becomes similar to a grid G, each cell of which may contain an object. Ants move on G and are aware of a region RS of $s \times s$ cells in their neighborhood.

In [5], Monmarché et al. coupled the LF algorithm with k-means algorithm to overcome the minor errors left by ants. They also introduced ants capable of carrying several objects simultaneously, and the ability to store several objects on the same cell.

In [3], Kanade and Hall combine the work of Monmarché et al. with the classical classification algorithm (fuzzy c-means). Then, in [6], Schockaert et al. implemented fuzzy rules to model the ant's behavior.

Other improvements have been made to the LF algorithm in [7] such as the variation of the parameter k_p, the variation of the vision s, and the use of pheromones.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

3 Ant-based principles of AntBee

Initially all the objects are scattered randomly in a 2D hexagonal grid. As in LF algorithm, ants are capable of moving on the grid and carrying objects.

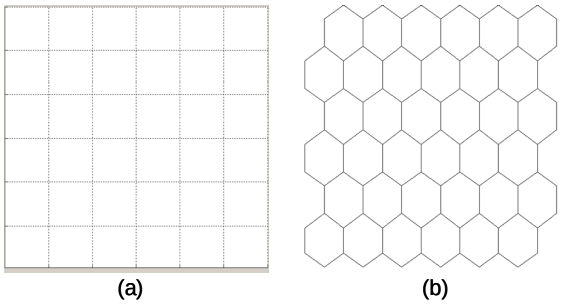


Fig. 1. Grid used in (a) LF algorithm and (b) AntBee algorithm. As can be seen in fig. 2, each lattice cell interacts with its six nearest neighbors which are situated at the distance 1 from it

The hexagon grid has several major advantages over the square grid: firstly it has the capability of spacing out each constituent hexagon more or less evenly from its neighbors. Secondly when hexagonal lattice is "holding up" an object, it is clear that the alternating angles of the hexagonal lattices would do more to distribute weight to the lower levels than the orthogonally-confined square lattice would. Since a square has more acute angles, points near its corners are further away from points elsewhere in its area than they would be in similar locations in a hexagon.

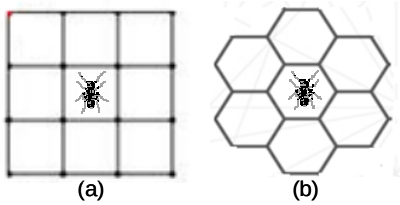


Fig. 2. Neighborhood in both LF (a) and AntBee (b) algorithm

Other improvement have been made to the LF algorithm: in AntBee the deposit of an object by an ant depends on the local information in the cells surrounding the cell occupied by the ant. This heuristic information is represented by an amount of pheromone deposit by an ant on the edges of hexagonal cell in the grid.

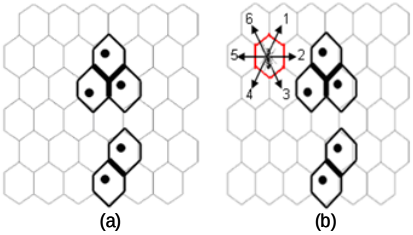


Fig. 3. Each cell containing an object is characterized by a pheromone trail on its edges (a). The ant has a high probability of moving to the cell 2 or 3 in its neighborhood, since these cells contain the same concentration of pheromone. Cells 1, 4, 5 and 6 do not contain pheromones however the probability of an ant to move away is low (b)

In LF algorithms the ant evolves arbitrarily on the grid. In AntBee, the move strategy is determined by an individual rule for all ants depending on their status (loaded or unloaded from an object). More precisely, the ant can perform two kinds of move:

- When the ant is not carrying an object, it looks for a possible cell to move by considering the 6 cells around its current position. The probability of choosing a cell increases with low-density pheromones and decreases with cell containing high amount pheromones among cell in the surrounding area. The transition probability of the *K-th* ant, moving from cell *i* to cell *j*, is given by:

$$Pmove_{ij}^{k}(t) = \frac{\theta}{\theta + C_j(t)} \tag{3}$$

Where θ is a positive constant and C_j is the amount of pheromones presents in the six edges of the cell *j*. When there is a small quantity of pheromone laid on cell *j*, $C_j(t) << \theta$, which means that *Pmove* is close to 1and the ant decide to move to cell *j*. On the other hand, when the cell *j* is dense en pheromone, $C_j(t) >> \theta$, and then *Pmove* is close to 0. This behavior is used to send empty ants towards cells containing isolated objects because high pheromone density often means that the clustering work has more or less been done in this area.

- When the ant is carrying an object, then it looks also at the 6 cells around its current location. The probability of moving to a cell increases with high densities of pheromones in its edges. The transition probability of the *K-th* ant, moving from cell *i* to cell *j*, is given by:

$$Pmove_{ij}^{k}(t) = \frac{\left(C_j(t)\right)^{\alpha}\left(1/d_{ij}\right)^{\beta}}{\sum_{l \in V_i}\left(C_l(t)\right)^{\alpha}\left(1/d_{il}\right)^{\beta}} \tag{4}$$

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Each ant k is characterized by a neighborhood V_i of cells around its current position i. d_{ij} is the heuristic value associated with the cell j. It represents the size of step according to the ant k at each move. Furthermore, α and ß are positive real parameters whose values determine the relative importance of pheromone versus heuristic information.

The intensity of pheromone in cell j at time t is calculated according to the following formula:

$$C_j(t) = \begin{cases} C_{\min} + n_{ov}(t)\dfrac{C_{\min}}{6} & \text{if } C_j \text{ is not empty} \\ n_{ov}(t)\dfrac{C_{\min}}{6} & \text{if } C_j \text{ is empty} \end{cases} \quad (5)$$

Where C_{min} is the amount of pheromone deposited on each cell coveted by an object and n_{ov} is the number of objects in the neighborhood of cell j.

To evaluate the contribution of our method, we use several artificial data. They are generated according to distinct Gaussian and uniform laws. The obtained classification can be compared to the original one of data set, this comparison can be done on the basis of percentage error.

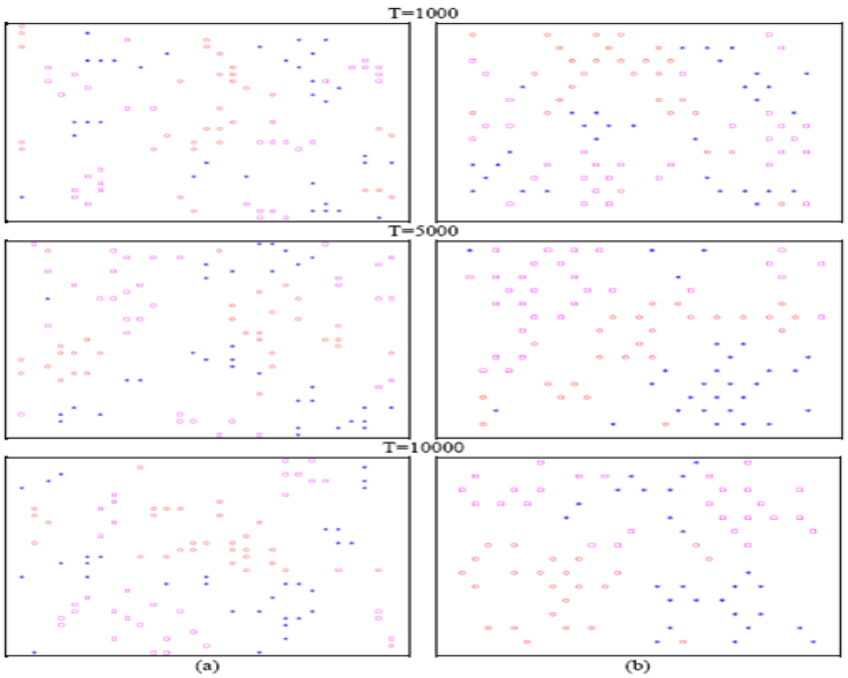


Fig. 4. Execution result of LF algorithm (a) and AntBee algorithm (b) at different times

4 Conclusion

We have presented in this paper an improvement ant based algorithm named AntBee for data clustering in a knowledge discovery context. The main features of this algorithm are the following ones. Hexagonal structure is another lattice structure alternative to traditional square grid used in the LF algorithm

We have also introduced in AntBee new heuristic for the ant colony, inspired by the stigmergy principles observed in real wasps nest building, to allow the artificial ants to move only in areas were there are objects, thus reducing the number of iterations. Future work consists in testing how this model scales with image segmentation as image segmentation can be considered as a clustering problem, which aims to partition the image into clusters.

References

1. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York (1999)
2. Deneubourg, J.L., Goss, S., Pasteels, J.M., Fresneau, D., Lachaud, J.P.: Self-organization mechanisms in ant societies (ii): learning in foraging and division of labor. In: Pasteels, J..M., Deneubourg, J.L. (eds.) From individual to collective behavior in social insects, Experientia supplementum, vol. 54, pp. 177–196. Bikhauser Verlag (1987)
3. Kanade, P.M., Hall, L.O.: Fuzzy Ants as a Clustering Concept. In: Proceedings of the 22nd international conference of the North American fuzzy information processing society, pp. 227–232, (2003)
4. Lumer, E., Faieta, B.: Diversity and Adaptation in Populations of Clustering Ants. In: Cliff, D., Husbands, P., Meyer, J., Wilson, S.W. (eds.) Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB), pp. 501–508. MIT Press, Cambridge (1994)
5. Monmarché, N., Slimane, M., Venturini, G.: On Improving Clustering in Numerical Databases with Artificial Ants. In: Floreano, D., Nicoud, J.D., Mondala, F. (eds.) ECAL 1999. LNAI, vol. 1674, pp. 626–635. Springer, Switzerland (1999)
6. Schockaert, S., De Cock, M., Cornelis, C., Kerre, E.E: Fuzzy ant based clustering. In: Proceedings of ANTS'2004. LNCS, vol. 3172, pp. 342–349. Springer, Berlin (2004)
7. Vizine, A.L., Castro, L.N., Gudwin, R.R.: Text document classification using swarm intelligence. In: Proceedings of the International Conference on the Integration of knowledge Intensive Muli-agent Systems, pp. 134–139, Waltham (2005)

Comparing Parameter Tuning and Parameter Control for SAT Solving Genetic Algorithms

S.K. Smit, A.E. Eiben

Vrije Universiteit Amsterdam

When defining an evolutionary algorithm (EA) one needs to set its parameters, for instance the mutation probability or the population size. The values of these parameters largely influence the performance of the algorithm. Choosing the right parameter values is therefore an important aspect of algorithm design. The field of evolutionary computing (EC) traditionally distinguishes two approaches to this end: 1) *parameter tuning*, where parameter values are determined before the EA run and do not change while the EA is running, 2) *parameter control*, where parameter values are given an initial value when starting the EA and undergo changes while the EA is running. Ever since the introduction of this distinction, numerous evolutionary computing scientists and practitioners posed the question:

Which approach is better, parameter tuning or parameter control?'

Quite remarkably, there are hardly any studies trying to answer this straightforward question. In theory, there are two major differences between parameter tuning and parameter control. The control approach has the advantage that it is capable to use adequate parameter values in different stages of the search. Furthermore, control is more general, because using constant parameter values can be seen as a special case, where the control policy is simply `keep-initial-value`. In practice, one should compare efforts vs. results, thus the question becomes the following:

> Spending the same computational effort on off-line learning a good parameter vector $\bar{p}$ and a good control strategy c, which EA performs better, the one using the values in $\bar{p}$ (not changing) or the one using the control strategy c that changes the values of parameters during the EA run?

Note that this question focuses on adaptive parameter control, deterministic and self-adaptive control schemes are not covered.

For an experimental assessment, we need test problems, an EA, a generic syntax for a control scheme, and an (off-line) algorithm learning good parameter vectors, respectively control strategies. As for test problems we have chosen two instances from the Järvisalo SAT Competition 2007 testsuite: instance 2158(1) –assumed do be easy– and instance 2248(2) –assumed to be hard. The EA is a basic bit-string GA using of k_1-tournament parent selection, k_2-tournament survivor selection, uniform crossover and bitflip mutation. This algorithm has 6 parameters: population-size, generation-gap, k_1, k_2, p_c and p_m. Defining the form of the parameter control strategies requires two decisions: 1) Which observables, i.e., state descriptors for the evolutionary search process, are used, 2) what is the syntax of the control strategies? Obviously, more observables imply a bigger learning space, which can slow down the off-line learning process.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

However, using fewer (or wrong) observables, the control strategy will not have enough (or inadequate) information and will fail. We have selected four out of the thousands of possible observables: Average pairwise Hamming distance, child improvement rate, average hamming distance between the parents, and percentage of individuals responsible for 80% of the total fitness. As for the syntax, we use a multiplicative update rule for each each parameter p^i, such that $p^i_{new} = p^i_{old} \cdot F^i$. The multiplier F^i is a polynomial formula with a linear and a quadratic term for each observable $x_1, \ldots, x_n$ in the form of $1 + \omega^i_0 + (\omega^i_1 \cdot x_1 + \omega^i_2 \cdot x^2_1) + \ldots (\omega^i_{2n-1} \cdot x_n + \omega^i_{2n} \cdot x^2_n)$. Hence, for each parameter we need to learn $2n + 1$ coefficients, or weights, and an initial value p^i_0. Finally, for learning/optimizing $\bar{p}$ and c we have selected a previously successfully used method: REVAC. In the learning runs, REVAC is allowed to test 500 vectors (of 6 GA parameters, or of 60 control strategy parameters) and performs 20 repetitions to measure the performance of the resulting GA.

The actual experiments entailed four series of runs, learning a good parameter vector $\bar{p}$ and a good control strategy c for the easy and the hard SAT problem. After each series of runs the best $\bar{p}$ or c was validated by 100 validation runs on the whole 3SAT Järvisalo testsuite and these outcomes are considered to show the quality of the given $\bar{p}$ or c. Analyzing the results[1], we observed that the GA using the static parameter values found on the hard problem significantly outperformed all others. Furthermore, it appeared that the GA parameters tuned for the easy problem were different from those tuned for the hard problem and the same was true for the parameters of the control strategy. These results are not surprising. However, it was unexpected that the learned control strategies hardly varied the values of the six parameters over a GA run. Thus, the learned control strategies actually represented constant parameter values. This either means that keeping the parameter values static is the best strategy, or, which is more likely, that the current set of observables does not provide the necessary information to successfully adapt to the search process.

Based on our results, the question stated above can be answered: It is better to spend off-line learning effort on tuning GA parameters, then on trying to discover a good control strategy. Inevitably, this conclusion depends on the design choices and the experimental setup. Changing either of these can (and will), lead to a different conclusion. Since the search space of control strategies consists of numerous possible observables, all possible combinations of observables, syntax and weights, we have only sampled a very small portion of this space.

Regarding the future, we expect more studies along these ideas. One specific line of work we foresee concerns the off-line learning of control strategies, deterministic, as well as adaptive. As we noted earlier, deterministic and adaptive control strategies can be seen as quite static on the meta level, because the strategy itself is not changing during an EA run. This means that parameter control strategies can be sought by off-line learning methods – pretty much like good parameter vectors can be found by tuning before the actual EA runs. To this end, the existing tuning methods can be of immediate benefit.

[1] http://www.cs.vu.nl/ sksmit

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Exploiting clusters of GPU machines with the EASEA platform

Frédéric Krüger[1], Laurent Baumes[2], and Pierre Collet[1]

[1] Univ. Strasbourg, LSIIT, FDBT, Illkrirch, France
frederic.kruger@etu.unistra.fr
[2] ITQ, UPV-CSIC, Valencia, Spain

Abstract. This paper presents an implementation of an asynchronous island model running on clusters of machines equipped with GPGPU cards, *i.e.* the current trend for super-computers and cloud computing.
Tests done on a benchmark and a real work problem using different cluster sizes show that artificial evolution is capable of exploiting hierarchical massively parallel systems in a very efficient way, with supra-linear speedup over one machine and linear speedup when clusters of different sizes are compared.
This paper shows how the EASEA platform (which was up to now automatically parallelizing evolutionary algorithms over one or several GPGPU cards) can now parallelize over several potentially heterogeneous machines using an island implementation.

1 Introduction

Many projects have shown what can be achieved when many machines work together on the same task. *Seti@Home*[3] is a perfect example: users all over the world allow their machines to connect through the internet to analyze data from radio-telescopes in order to look for evidence of extraterrestrial intelligence. Then, Cloud computing is now coming, with even companies starting to sell computing time such as Amazon Elastic Compute Cloud (EC2) web service.

Evolutionary computation is inherently parallel so it is important to start developing algorithms and software platforms that would allow EC to be used as a generic optimization tool to be run in Cloud mode.

The aim of this paper is to show how a group of machines, connected through a local network or through the internet, can be used very efficiently to solve complex problems and how efficient hierarchically massively parallel code can be produced by the EASEA platform.

The paper starts by giving a quick presentation of the EASEA platform and its major features. After a quick examination of the state of the art, the EASEA island model is described. Results are presented on a benchmark function as well as on a real-world problem.

[3] http://setiathome.berkeley.edu/

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

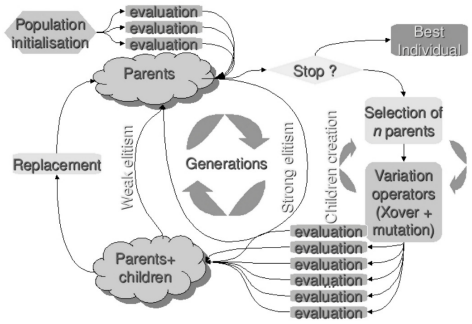


Fig. 1. Flowchart of a parallel evolutionary algorithm implemented by EASEA.

2 EASEA

2.1 Brief overview of the EASEA parallelizing platform

The EASEA (for EAsy Specification of Evolutionary Algorithms) massively parallel platform[4] started as a language more than 10 years ago [1]. Its aim was to allow non expert programmers to nevertheless implement their evolutionary algorithms by just supplying the compiler with problem-specific simple pieces of code, namely a description of the individual representation, how to initialize an individual, how to mutate it and how to recombine two individuals in order to produce a child and finally, how to evaluate an individual. All these functions are provided in a `.ez` file that also contains some default parameters for a specified evolutionary engine to evolve a population of individuals (parameters include population size, number of generations, which selection operators to use, a.s.o.).

The EASEA compiler then assembles a C++ source file that wraps a complete evolutionary engine around the different problem specific functions provided by the user by using man-made templates, which allows the output source code to be understood by humans. The EASEA language uses basic functions implemented in the EASEA library which is also human-readable and that allows users to customize their algorithms, should they not find what they need in the default EASEA release which can already implement quite many different paradigms, such as standard genetic algorithms, Evolution Strategies, CMA-ES, memetic algorithms, tree-based genetic programming, linear genetic programming and in a short future, differential evolution.

Code for these paradigms can be produced either for a standard CPU (for sequential execution on one core) or for execution over one or several NVIDIA GPGPU cards, if the `-cuda` option is invoked on the command line. EASEA parallelization over GPU cores has been extensively described in a number of recent papers [2][3].

Sharing projects and results is also very easy. Only the `.ez` specification file must be exchanged as it contains all that is needed to produce a complete

[4] http://sourcefore.net/projects/easea

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

execution code for the problem to be solved. This feature is extremely useful in particular to run a same experiment on (potentially) different machines linked with an island model.

2.2 EASEA and GPGPU cards

One of the main strengths of the EASEA language is that the massively parallel algorithms published to take advantage of GPGPU cards are directly implemented by simply specifying the *–cuda* option on the `easea` compiler command line.

By default, EASEA keeps the evolutionary engine on the CPU and parallelizes the evaluations on the GPU card (cf. fig. 1).

3 Island Model

3.1 Island Model and related works

A good technique to drive through sand dunes is to use several vehicles simultaneously. If one gets stranded in soft sand, another vehicle can pull it out using a towing cable and progression can resume.

Some multimodal problems can look like a dune sea, in which an evolutionary algorithm can get stranded if its population has prematurely converged in a local optimum. However, if several independent algorithms (let us call them islands) explore the same search space, they can act like several vehicles exploring an erg and help one another out of local optima in order to find even better areas.

Rather than using a towing cable, an EA island can send its best individual to another island. If the incoming individual (immigrant) is worse than the individuals in the stranded island, it will get discarded at the next generation. If however, it is among the good individuals of the islands, other individuals will want to mate with him to create children.

If a barycentric-type crossover is used, the children will appear on a line joining the two islands, allowing to explore unknown yet possibly better places.

If n stranded islands share the same search space and exchange individuals, children will spawn over $n(n-1)/2$ lines, therefore improving chances to find better places in the search space. However, individuals should not be exchanged among islands too often: local exploitation should take place in order so that a new spot is well explored before the island migrates to another place towards the direction of another good immigrant.

This means that infrequent communication is actually an advantage over frequent communication, which is a very important point as communication is usually what prevents parallel machines from yielding a linear speedup with the number of machines: usually, 10 machines do not go $10\times$ faster as one machine because of synchronization and communication constraints between machines. Here, migration can take place asynchronously and in a loosely coupled manner.

A lot of research has already been done on island models. Theoretical studies on the number and the size of populations have been done by Whitley et al. [4].

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Other studies show the influence of different island model parameters such as migration rate, connectivity etc... [5]. Alba and Troya studied asynchronism [6], Branke *et al.* studied the influence of heterogeneous networks on island models [7].

Some research has been done on parallelizing evolutionary computation (EC) using an island model on a GPGPU card, but this research was limited to a single machine equipped with a GPGPU card that hosted all the islands [8].

3.2 EASEA Island Model

The default island model implemented by the EASEA language is quite basic. It allows to periodically send (and receive) an individual to (from) one of several IP addresses listed in a file.

The configuration of the EASEA island model relies on two main parameters of the `.ez` file:

- the name of the file containing the IP addresses (`ip.txt` by default), and
- the migration probability.

Traditional approaches usually let several islands run on the same machine. The EASEA island model allows only a single island to run per machine and relies on establishing connexions with other machines for a multi-island system.

Individuals are sent between islands using a connexion-less UDP/IP protocol. No connexion means that the process can be asynchronous and therefore quite robust: an island can stop working (due to a hardware problem for instance) and then start again later on. It will receive good individuals from other machines that did not crash and will rapidly find good spots again without disturbing the other islands.

New islands can join the search at any moment.

Sent individuals may get lost, but then, the whole EC island paradigm is stochastic, so losing an individual once in a while can be considered as being part of the algorithm.

No time is lost in island synchronization or acknowledgement that a sent individual has effectively been received.

As EASEA can produce code for machines running with or without GPU, running under Linux, Windows or MacOsX, the EASEA island model can run on any number of heterogenous machines connected *via* the internet over the world to work on solving the same problem, provided their IP address is present in the IP file of each island and provided they exchange individuals sharing the same structure.

Algorithms may be different depending on machines: slower machines can have an exploratory algorithm while faster ones can concentrate on exploitation of good spots.

3.3 Implementation

Before the evolution begins on an island, the file containing the IP addresses of the other islands is parsed creating a list of clients. The IP address of a machine

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

can occur several times. Because one IP number will be picked at random among the number of IP entries, replicating 10 times an identical IP number will give it 10 times more chances to be selected than the other machines, therefore allowing to implement weighted edges connecting machines.

Then, communication needs not be symmetric, as a machine can have the IP number of another one, but the other may not have the IP number of the first. This allows to implement a unidirectional flow of individuals between slow exploratory machines that would find good spots and fast exploitation machines that could concentrate on finding the local minimum. However, the fast machines may not send back individuals to the slow machines so as to prevent premature convergence in the cluster of slow machines.

Then, at every generation, a migration function is called with a probability p set by the user. A destination island is chosen randomly amongst the list of clients. Once a destination is chosen, n individuals are selected amongst the population using different selection operators. The selected individuals are then serialized and sent to the destination island.

On every island, at the beginning of the run, a thread is launched that is in charge of receiving incoming individuals. Upon arrival of a newcomer, a selector decides who in the population will be replaced. The integration process of newcomers is performed at every new generation.

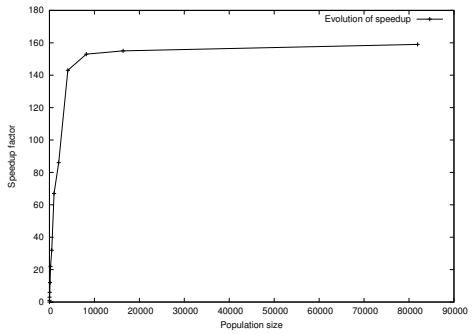


Fig. 2. GPU *vs.* CPU speedup on the Weierstrass-Mandelbrot benchmark over 1000 dimensions performing 120 iterations and using a hölder parameter value of 0.35. Greater population sizes allow to get better speedups.

The described implementation of an island model is very simple, robust and versatile, as it allows to easily design complex topologies. For instance, it would be really easy to create a ring topology by just giving each island the address of the following island. One could also creating a square or hexagonal torodial grid, or virtually any other topology. Then by changing migration probability, communication rate can easily be increased or decreased: for instance, the center island in a star shaped topology may need more frequent communications with

other islands, or one could imagine to modify migration rate depending on some strategy (increase or decrease migration with the number of generations).

4 Experiments

In this section, the EASEA GPU island model will be tested on a cluster (a classroom) of 20 PCs each containing an ev GTX275 GPU card, for an advertised computing power of around 20 TFlops. It was therefore necessary to find a benchmark and a real-world problem that would be tough enough to match this enormous computing power.

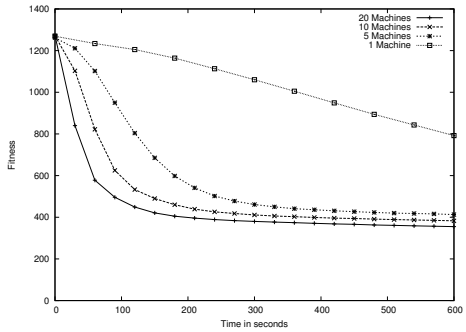


Fig. 3. Evolution of the best fitness on different clusters sizes. These results represent an average of 20 runs.

4.1 Speedup of the benchmark problem on the GPU card

As a benchmark function, we used a Weierstrass-Mandelbrot function, whose irregularity can be tuned thanks to its Hölder coefficient h. The Weierstrass-Mandelbrot benchmark function is defined as such:

$$W_{b,h}(x) = \sum_{i=1}^{\infty} b^{-ih} sin(b^i x)$$

with $b < 1$ and $0 < h < 1$.

Usually, a Hölder coefficient of 0.5 is used but it turned out that this value created a too simple function for the described cluster, so irregularity was increased by using a 0.35 Hölder coefficient (irregularity increases as h is smaller) and 120 iterations (an infinite number of iterations being impossible to compute). Then 2 dimensions only was too small a search space for 20 GPU machines, so we needed to go up 1000 dimensions to make for a tough enough problem on which conclusions could be drawn.

First of all, before the island model is tested, it must be reminded that evaluation can be parallelized on each machine's GPU card, so a first figure is

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

presented that shows the obtained speedup by comparing a sequential implementation of an EA solving the Weierstrass problem on one core of an Intel Xeon 5500 2.57Ghz Core i7 CPU with 8GB Ram *vs* a parallel implementation of the same algorithm on an NVIDIA GTX275 240 cores card.

Different population sizes were used ranging from 8 to 81920, showing that the obtained speedup reaches a plateau at around $\times 160$ for a population size larger than 8192, while a still reasonable speedup can be obtained with 4096 individuals only (cf. fig. 2).

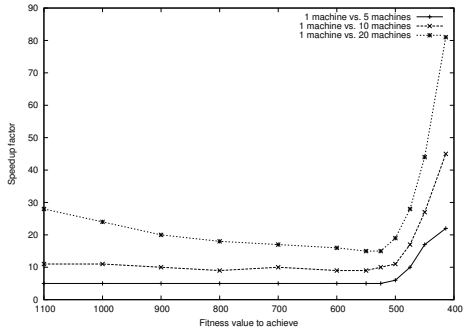


Fig. 4. Speedup factor achieved comparing 1 machine *v.s.* a 5, 10 and 20 machines cluster. These results represent an average of 20 runs.

4.2 Observed speedup on the 20 machines cluster

A classroom of 20 identical machines is used as a cluster of GPU machines (each machine having 8GB of memory, an Intel Xeon 5500 2.57 Core i7 CPU and an NVIDIA GTX275 GPU card).

In order to study the efficiency of the EASEA island model, the Weierstrass-Mandelbrot function was tested on different cluster sizes, each machine of the cluster hosting a single EASEA island.

For things to be comparable, all experiments are done at constant population size, meaning that each island of a 20-machine experiment will contain 20 times fewer individuals than the same experiment on a single machine. The population size was chosen so that the smallest population sizes should allow to obtain a reasonable enough speedup on the GPU card. This means that each of the 20 machines should have a minimum of 4096 individuals, for a total of $20 \times 4096 = 81920$ individuals for the global experiment.

On a 10 machines run, all machines will share the 81920 individuals, meaning that each island will contain 8192 individuals. On 5 machines experiments, each island will contain 16384 individuals.

The topology used for the experiments is a very straightforward fully connected network (each island uses the same IP file containing the IP numbers of all participating machines, and no machine can send an individual to itself) and all presented results were obtained over an average of 20 runs.

Figure 3 shows the evolution of the average of the best fitness on 20 runs over 10 minutes of evolution on the Weierstrass-Mandelbrot benchmark for 4

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

different cluster sizes: 1 machine with 81920 individuals, a cluster of 5 machines with 16384 individuals per island, a cluster of 10 machines with 8192 individuals per island and a cluster of 20 machines with 4096 individuals per island.

As expected, 20 machines obtain better results faster than 1 machine only, but this figure does not allow to visualize the obtained speedup well.

However, it is possible to visualize the obtained speedup by looking at how long it takes each cluster configuration to reach predefined values such as 1100, 1000, 900, ... It was not possible to go down to value 400 because some machines did not improve their results after more than 10 hours. The minimum value found on 20 single machines run was 414.

Figure 4 can be read the following way: it was 5 times faster for a 5 machines cluster to obtain a fitness of 1100, compared to one machine only, meaning that a linear speedup is obtained for 5 machines. Indeed, the $\times 5$ speedup is quite constant until value 500 after which it increases up to more than 20 for value 414 (these are all average values over 20 runs).

The speedup factor for a 10 machines cluster is quite similar: around $\times 10$ speedup from value 1100 down to value 500 after which speedup rises up to around $\times 45$.

With 20 machines, speedup starts with a supra-linear value of 28, before going down to slightly under 20 and rising up again at value 500, up to slightly above $\times 80$.

What happens here is that speedup is roughly linear with the number of machines until value 500, after which single machines with one island of 81920 individuals get stranded in the sand dunes. At this point, multi-island models show their advantage as they help each other out of local optima. The speedup of multi-island models over a single island gets very supra-linear.

It is very interesting to see that beyond value 500 (*i.e.* after single islands tend to get stuck in local optima, the speedup between multi-island configurations remains roughly the same: for value 414, the speedup of a 5 machines cluster is slightly above $\times 20$ while it is roughly $\times 45$ for 10 machines and above $\times 80$ for 20 machines, which is quite satisfying.

The reader must be reminded that these curves are for GPU-islands for which fig. 2 show that they are already $\times 160$ faster than a sequential execution of the same code on a CPU. Therefore, during the linear speedup phase, the speedup of the 20 GPU cluster *vs* one single machine is of $\sim 160 \times 20 = 3200$, reaching $\sim 160 \times 80 = 12800$ when single machines get stranded. On this problem, a one day run on this cluster would therefore be equivalent to at least 35 years on a single island but probably much more as value 414 was obtained in less than 3 minutes only and the speedup slope is quite steep.

5 Real world problem: looking for a zeolite structure

Benchmark problems are always ideal to test performance. However, it is often the case that real-world problems do not behave like benchmarks. It was therefore

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

challenging to try the island mode on a chemistry problem: finding the structure of a 20 or 21 atoms zeolite.

Zeolites are crystalline materials with regular structures consisting of molecular-sized pores and channels. These crystals are widely used in important industrial applications such as in the field of adsorption, ion-exchange, heterogeneous catalysis [10], as well as in health, sensors, solar energy conversion. They are made of tetrahedrons of oxygen atoms that contain a single silicon or aluminum atom in their center. Crystal structure prediction is a very challenging problem as solving a new zeolite structure may require a team of chemists to work for several years.

The problem posed by prediction is to obtain approximate models in a reasonable time which can then easily be subsequently refined by routine modeling techniques. Therefore, the proposed structure prediction approach is expected to provide stable candidate solutions without comparing their theoretical diffraction data to the experimental one during the evaluation of goodness. Next, the different potential solutions proposed by the algorithm are refined by a routine based on inter-atomic potential technique using the GULP code.

The entire methodology is explained elsewhere in a previous paper containing detailed results for an unknown case [9]. Only unspecified T (tetrahedral) atoms are considered, oxygen is omitted, and the required inputs are the unit cell parameters, the space group, and a density range for T atoms. Based on the X-ray diffraction indexing data, it has been decided to focus on the tetragonal space group C222 (No. 21), while the measure of density allowed us to define a number of 160 T per unit cell. An error of 2.5% is integrated in the density giving a range of 156- 164 T per unit cell. The methodology employed is based on an evolution strategy (ES) which independently manipulates fixed arrays of variables corresponding to atoms coordinates belonging to the asymmetric unit. The initial population is randomly constituted by starting solutions with 20 and 21T, *i.e.* arrays of 60 and 63 floating variables for x,y,z coordinates. Improvements are continuously integrated in new versions. Among them, after all atoms have been randomly placed in the asymmetric unit during the initialization procedure, each atom is iteratively and randomly selected and its position is translated toward the closest atom already present which is not tetracoordinated.

The final position is the optimal distance defined at 3.07A. Wickoff positions have also been integrated, allowing atoms being at a distance inferior to 0.8A to be automatically placed at the corresponding special position before assessment of the solution. Note that the new position is virtual, *i.e.* calculated, in order to assess the structure viability as (x,y,z) coordinates are not modified in the genome code.

5.1 Fitness function

The fitness function computes an approximated value of the real energy in the system. The function is based on the angle and the distance between atoms.

The upper left part of figure 5 represents the distance between atoms whereas the upper right part represents the angle between atoms. The lower left part of figure 5 shows the average angle between atoms and the lower right part

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

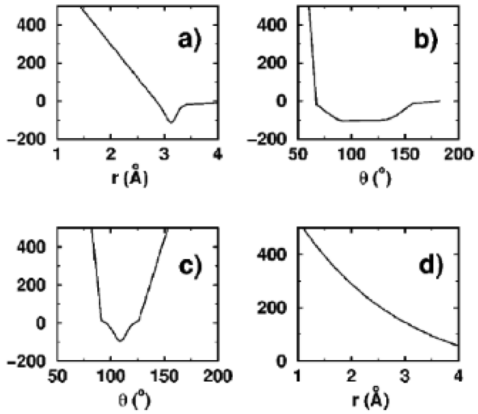


Fig. 5. Fitness function of the zeolite problem.

represents the connectivity of the atoms. The final fitness is a function of all the characteristics.

The goal of the evolutionary algorithm is to minimize this fitness.

5.2 Experiment

The real world problem presented in this paper is much more complex than the Weierstrass-Mandelbrot toy problem. A simple fully connected island model was not efficient enough to find interesting results. The 20 machines were separated in 2 different clusters: a cluster of 16 machines that would look for new interesting spots (exploration) and a cluster of 4 machines with a different algorithm to exploit the best spots.

The first cluster would periodically send their best individuals to the 4 other machines while the 4 machines would exchange individuals between themselves only. They would try to find the best local value around an individual given to them by the cluster of 16 machines. If no improvement was found for a certain time, the four machines would restart simultaneously and wait for a new suggestion coming from the 16 other machines.

Periodically, the four machines would send their best individual to an independent slow machine in our lab, that would simply run to collect the best individuals found to date. This machine was not part of the optimization process.

The same settings were used as for the Weierstrass test bench (population sizes depending on the number of machines in each tested cluster configuration).

The runs were performed on a time span of 2 hours except for the run on the 20 machines, which was performed over 40 hours.

5.3 Results

Figure 6 shows the evolution of the best fitness for the crystal structure prediction problem for runs on a cluster of 1, 5, 10 and 20 machines. During the 2 hour

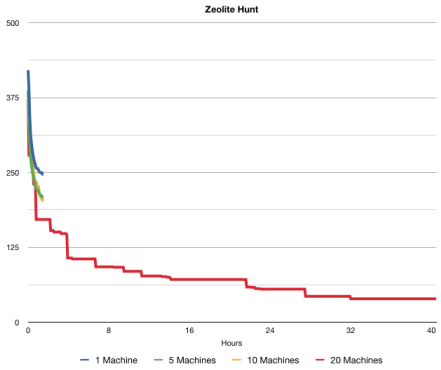


Fig. 6. Evolution of the best fitness for the real world problem.

time span, the 20 machines cluster (and its special topology) outperformed all the others, giving better results for the same computation time.

Because of the relative difficulty brought by germanium-containing zeolites in the comparison between theoretical and experimental X-ray diffraction patterns, the material was sent to Germany where a new microscopy technique was employed and finally revealed the correct structure. Based on their result, we could observe that the correct framework structure belonged to the 50 most probable previously generated candidate solutions obtained by the 42 hours run shown in fig. 6. The integration of T composition probability depending on substructure schemes in the frameworks and the corresponding calculation of theoretical diffraction patterns is currently under work in order to further reduced the final number of potential solutions while turning the methodology into a real predictive tool. This work lead to a submission to Science and a large number of real-world applications.

6 Conclusion and extension

After automatically parallelizing the evaluation stage of evolutionary algorithms, the EASEA platform now integrates a basic yet powerful island model that allows to turn any number of computers connected to the Internet into a cluster of machines.

Experiments show that the published linear and supra-linear performance of evolutionary algorithms can be attained on both a benchmark and a real-world problem that a run on 20 machines contributed to solve.

Measured speedup on the zeolite problem showed a $\times$ 120 factor between a standard CPU implementation and a massively parallel GPU implementation on an NVIDIA card. This speedup being at least multiplied by 20 thanks to EASEA island model running on a cluster of 20 machines, we are looking at a

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

global speedup factor of 2400 over 42 hours, meaning that this same run would have taken slightly more than 11 and a half years on an Intel core i7 computer.

The architecture that was used to obtain these results is close to that of Tianhe-1a, China's current fastest supercomputer that yields a theoretical peak performance of 4.7 PetaFlops, thanks to 14336 2.93GHz CPUs and 3 million 575Mhz GPU cores.

This paper shows that Evolutionary Computation can efficiently exploit such machines, that prefigure desktop computers of years to come (Intel has announced that future CPUs would include a large number of GPU cores). It is therefore very important to prepare this revolution and work on platforms such as EASEA, that will allow EC to run on such computers as a generic optimizer.

Current work include testing EASEA over several clusters of machines, implemented over the world.

References

1. Collet, P., Lutton, E., Schoenauer, M., Louchet, J.: Take It EASEA. Proceedings of the 6th International Conference on Parallel Problem Solving from Nature. **PPSN VI** (2000) 891–901
2. Maitre, O., Baumes, L., Lachiche, N., Corma, A., Collet, P.: Coarse Grain Parallelization of Evolutionary Algorithms on GPGPU Cards with EASEA. 11th Annual Conference on Genetic and Evolutionary Computation. GECCO 2009 403–1410
3. Krüger, F., Maitre, O., Jimenez, S., Baumes, L., Collet, P.: Speedups between x70 and x120 for a generic local search (memetic) algorithm on a single GPGPU chip. EvoNum 2010. **LNCS** (2010)
4. Whitley, D., Rana, S., Heckendorn, R.B.: The island model genetic algorithm: On separability, population size and convergence. Journal of Computing and Information Technology. **Vol. 7** (1999) 33–48
5. Cantu-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers. (2000)
6. Alba, E., Troya, J.: An analysis of synchronous and asynchronous parallel distributed genetic algorithms with structured and panmictic Islands. Proceedings of the 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing. (1999) 248–256
7. Branke, J., Kamper, A., Schmeck, H.: Distribution of Evolutionary Algorithms in Heterogeneous Networks. GECCO 2004 923–934
8. Luong, T.V., Melab, N., Talbi, E.: GPU-based Island Model for Evolutionary Algorithms. Genetic and Evolutionary Computation Conference. GECCO 2010 (2010)
9. Baumes, L.A., Krüger, F., Jimenez, S., Collet, P., Corma, A.: Boosting theoretical zeolitic framework generation for the determination of new materials structures using GPU programming. Phys. Chem. Chem. Phys. **Vol. 13** (2011) 4674-4678
10. Corma, A., Moliner, M., Serra, J.M., Serna, P., Díaz-Cabañas, M.J., Baumes, L.A.: A New Mapping/Exploration Approach for HT Synthesis of Zeolites. Chemistry of Materials. **Vol. 18** (2006) 3287–3296

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

An Extended Beam Search-Based Algorithm for the Strip Packing Problem

Hakim Akeb[1] and Mhand Hifi[2]

[1] ISC Paris School of Management, 22 Boulevard du Fort de Vaux, 75017 Paris, France (hakeb@iscparis.com)
[2] Université de Picardie Jules Verne, Unité de Recherche EPROAD, 5 rue du Moulin Neuf, 80039 Amiens, France (mhand.hifi@u-picardie.fr)

Abstract. This paper solves the Strip Packing Problem (SPP), an NP-hard combinatorial optimization problem of the Cutting and Packing (CP) family. SPP consists in packing a set of n circular pieces $C = \{C_1, \ldots, C_n\}$ where C_i is characterized by its radius $r_i, i = 1, .., n$, into a strip (or rectangle) of fixed width W and unlimited length L. The objective is to pack all the circular pieces so as to minimize the length of the strip. SPP is solved by using an extended beam search (a population-based technique) combining the principle of beam search, look-ahead, and a restarting strategy. The contribution of this work consists in the use of the look-ahead as well as a meticulous study on the parameters used by the restarting technique. Computational tests, on a set of benchmark instances of the literature, shows the effectiveness of the proposed method. Indeed, several new solution values are obtained.

Keywords. Beam search, population, selection, cutting and packing, strip packing, look-ahead, restarting strategy.

1 Introduction

In this paper, we investigate the use of the Beam Search (BS) approach (see Ow and Morton [11]) for solving a special packing/cutting problem; that is, the Strip Packing Problem (SPP) which belongs to the well-known family of Cutting and Packing (CP) problems (see Wäscher et al. [14]). It can be viewed as a category of combinatorial optimization problems defined as follows: a set of predetermined items must be placed in one or several larger containers so as to minimize the unused area or in some cases to maximize a utility function; furthermore, items and containers can have rectangular, circular, or irregular form.

Such problems have many industrial applications such as cutting pieces of rectangular form from a stock rectangle of wood, metal, or glass (Lewis et al. [10]), bundling wires that connect a car's sensors to the display board (see Sugihara et al. [13]), and grouping cables into pipes for the telecommunication and oil related companies. In other cases, a space may be optimized when considering the transportation of products or goods in freighters (see Baltacioglu et al. [3]), or when loading patterns in order to deliver some assembly components (see Kim and Kim[8]).

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

The studied problem (SPP) consists in packing a set of n circular pieces $C = \{C_1, \ldots, C_n\}$ where C_i is characterized by its radius $r_i, i = 1, .., n$ into a strip (or rectangle) of fixed width W and unlimited length L. The objective of the problem is to pack all the pieces so as to minimize the length of the strip. We propose herein, to solve SPP, an extended version of Beam Search (BS). The term "Beam Search" was defined by Raj Reddy (Carnegie Mellon University) in 1976. This technique was first used in artificial intelligence for designing translators and was after that adapted for other problems such as scheduling. The approach is mainly based upon a tree search in order to provide a set of internal and terminal nodes. The resulting tree is reached by heuristically applying the best-first search strategy for (i) tackling large problem instances and (ii) reaching near-optimal solutions within a reasonable computation time. According to its principle, BS explores only a selected subset of nodes, namely the subset of the most promising ones, according to an evaluation criterion.

The rest of the paper is organized as follows. Section 2 gives a literature review for the SPP and the problem formulation is discussed in Sect. 3. Section 4 exposes the principle of the minimum local-distance position strategy-based procedure. Section 5 details the proposed extended beam search-based algorithm, which combines beam search (using separate beams), a restarting strategy, and a look-ahead strategy. Section 6 describes the results obtained by the proposed algorithm on a set of known benchmark instances. Finally, Sect. 7 summarizes the contribution of this paper and gives some orientations for future works.

2 Literature Review

The problem of packing circles into a rectangular container was first studied by George et al. [5]. The authors proposed an approach in which several rules of sorting pieces have been used before packing them. Among the strategies used, it was shown that the best rules are those using a quasi-random approach and a genetic algorithm. For the problem of packing circles into a stock rectangle or a strip, Stoyan and Yaskov [12] proposed a mathematical model that searches for feasible local optima by combining a tree-search procedure with a reduced-gradient method. For the same problems, Hifi and M'Hallah [6] proposed a constructive procedure and a genetic algorithm, and Birgin et al. [4] tackled them by considering a non-linear model-based approach.

For the SPP, two versions of a solution procedure was proposed by Huang et al. [7]. The first one, called B1.0, starts by placing two circles into the stock rectangle before applying a greedy procedure for packing the $n-2$ remaining circles. The second version, called B1.5, combines B1.0 with a look-ahead strategy trying then to improve the solution quality of B1.0. Kubach et al. [9] proposed a parallel version of B1.5 called B1.6. Akeb and Hifi [1] proposed three algorithms for SPP: (i) an open-strip generation solution procedure (OSGSP) which is based on an optimization problem, (ii) a local beam-search solution procedure that combines beam search and OSGSP, and (iii) a hybridization between beam search, binary interval search, and the open-strip generation procedure. Finally,

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Akeb et al. [2] developed an augmented beam search algorithm based essentially on the use of separate searches in the tree.

This paper discusses an extended beam-search based algorithm for the SPP. This one combines separate-beams search, a restarting strategy, and look-ahead. In addition, a study of the parameters of the restarting strategy is done in order to try to stabilize the computation time while improving the solution quality. The contribution of this work consists in the introduction of the look-ahead strategy and the modification of the restarting strategy.

3 Problem Formulation and Notations

SPP can be stated as follows:

$$\min\ L \tag{1}$$

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq (r_i + r_j)^2,\ \text{for } j < i,\ (i,j) \in N^2 \tag{2}$$

$$x_i - r_i \geq 0,\ y_i - r_i \geq 0, W - y_i - r_i \geq 0,\ L - x_i - r_i \geq 0,\ \ \forall i \in N \tag{3}$$

where the bottom-left corner of the strip $S = L \times W$ coincides with the origin $(0,0)$ of the Euclidean plan and (x_i, y_i) are the coordinates of the circular piece C_i. Equation (2) corresponds to the non-overlap constraint of any pair of distinct pieces (C_i, C_j); that is, the distance between the centers of both pieces must be greater than or equal to the sum of their radii. Finally, (3) ensures that any piece $C_i,\ i \in N$, belongs to the target rectangle of dimensions (L, W). For the rest of the paper, we use the following notations.

1. $N = \{1, ..., n\}$ represents the set of the circles to pack into the strip S,
2. The four edges of S are denoted by S_{left}, S_{top}, S_{right}, and S_{bottom},
3. A circular piece C_i of radius r_i is placed with its center at coordinates (x_i, y_i),
4. I_i denotes the set of i circles already packed inside the strip,
5. $\overline{I}_i$ contains the circles which are not yet placed,
6. P_{I_i} denotes the set of distinct corner positions for the next circle to place C_{i+1} given the set I_i,
7. A corner position $p_{i+1} \in P_{I_i}$ for (C_{i+1}) is determined by using two elements e_1 and e_2. An element is either a piece already placed (set I_i) or one of the three edges of S (S_{left}, S_{top}, S_{bottom}). $T_{p_{i+1}}$ denotes the set composed of both elements e_1 and e_2.

4 The greedy constructive procedure

Herein, we introduce a greedy constructive procedure that is based upon the Minimum Local Distance Position (MLDP). Indeed, such a strategy can be used as a greedy algorithm for completing local or final solutions. MLDP selects, at each step i, the best corner position of the next piece C_{i+1} to pack without overlapping any already placed pieces but by touching two already placed pieces or by touching a piece and one of the edges of the strip.

In order to explain the mechanism of MLDP, a partial solution as illustrated in Fig. 1 where four circular pieces $(i = 4)$ are already placed inside the strip S

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

and four possible positions emerge (dotted-line circles) for the next circular piece C_5. So we have $I_4 = \{C_1, C_2, C_3, C_4\}$ and $P_{I_4} = \{p_5^{(k)}, k = 1, .., 4\}$. For example, the third position $p_5^{(3)}$ touches both C_1 and C_2 and position $p_5^{(4)}$ is computed by using C_1 and the bottom-edge of the strip. It follows that $T_{p_5^{(3)}} = \{C_1, C_2\}$ and $T_{p_5^{(4)}} = \{C_1, S_{\text{bottom}}\}$.

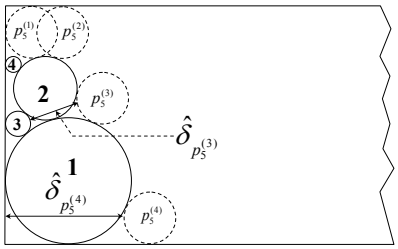


Fig. 1. Feasible distinct corner positions for circle C_5 in the strip

Let C_{i+1} be the selected piece to pack at position p_{i+1} and let $\delta_{i+1}(edge)$, $edge \in E_{\text{edge}} = \{S_{\text{left}}, S_{\text{bottom}}, S_{\text{top}}\}$, be the three distances defined as follows: $\delta_{i+1}(S_{\text{left}}) = x_i - r_i$, $\delta_{i+1}(S_{\text{bottom}}) = y_i - r_i$, and $\delta_{i+1}(S_{\text{top}}) = W - y_i - r_i$.

The distance between the edge of the next circle to place C_{i+1} (when positioned at p_{i+1}) and C_j is denoted by $\delta_{i+1}(j)$ and is calculated as follows:

$$\delta_{i+1}(j) = \sqrt{(x_{i+1} - x_j)^2 + (y_{i+1} - y_j)^2} - (r_{i+1} + r_j). \tag{4}$$

Then, the MLDP of the circular piece C_{i+1} when positioned at $p_{i+1} \in P_{I_i}$ is computed as follows:

$$\hat{\delta}_{p_{i+1}} = \min_{\alpha \in I_i \cup E_{\text{edge}} \setminus T_{p_{i+1}}} \{\delta_{i+1}(\alpha)\}. \tag{5}$$

Equation (5) indicates that the MLDP of C_{i+1} is computed by using the distances between the piece to place at position $p_{i+1}$and the elements of the set $I_i \cup \{S_{\text{left}}, S_{\text{bottom}}, S_{\text{top}}\} \setminus T_{p_{i+1}}$ composed of the pieces already placed, and the three edges of the strip. In this case, both elements of $T_{p_{i+1}}$ used for computing the coordinates of C_{i+1} are excluded, because such a distance is always equal to zero. Note also that the MLDP is equal to zero when C_{i+1} touches more than two elements. For example, the MLDP of position $p_5^{(1)}$ in Fig. 1 is equal to zero because this position touches three elements: circle C_2, the top-edge, and the left-edge of S. The figure indicates also the MLDP $\hat{\delta}_{p_5^{(3)}}$ of position $P_5^{(3)}$ and $\hat{\delta}_{p_5^{(4)}}$ which is the MLPD of position $P_5^{(4)}$.

Finally, when the greedy MLDP procedure is used, then it starts by positioning the first circular piece C_1 at the bottom-left corner, i.e., at the position (r_1, r_1), while the remaining $n-1$ pieces are successively positioned by using the MLDP rule. As illustrated by Fig. 1, C_5 will be placed at position $p_5^{(1)}$ because the corresponding MLDP has the minimum value (zero in this example).

5 An Extended Beam Search-Based Algorithm for SPP

The standard beam search is a tree search in which the root node corresponds to a partial solution. At each level of the tree, a population B (the beam) of nodes is used, this corresponds to different partial solutions. Each node $\eta \in B$ generates several descendants (the offspring nodes). The descendants of all the nodes are stored in a list denoted by B_ω. The nodes of B_ω are then evaluated and the ω best ones are chosen for branching (ω is called the "beam width"). After that, the population B is replaced by the chosen nodes in B_ω. The search stops when a final solution is reached or when no branching is possible ($B_\omega = \emptyset$).

Akeb and Hifi [1] proposed an algorithm, denoted by Beam Search Binary Interval Search (BSBIS), for approximately solving SPP. Such an algorithm searches for the best solution in an interval search which is characterized by both lower and upper bound limits, denoted by $\underline{L}$ and $\overline{L}$ respectively. First, $\underline{L}$ is setting equal to the trivial value $\frac{\pi}{W} \times \sum_{i \in N} r_i^2$ and $\overline{L}$ is generated by applying an intuitive Open Strip Generation Solution Procedure (OSGSP$_a$). Second, during the resolution phase, BSBIS tries to solve a decision problem which consists in packing all the n pieces into a rectangle of fixed length by using BS applying a width-first search (for more details, the reader can refer to [1]).

In this section, we describe a new version of BSBIS, that is, an extended look-ahead strategy-based algorithm (denoted by EBS). The proposed algorithm combines several strategies including the separate-beams search [2], the multi-start, and the look-ahead ones. In addition, a study showing the usefulness of the look-ahead is given as well as an investigation of the multi-start strategy settings in order to improve the quality of the provided solutions.

5.1 The Beam-Search Look-Ahead Algorithm (BSLA)

Generally greedy algorithms do not provide solutions of good quality for several combinatorial optimization problems. This is due to the "local evaluation" which is based upon a selection criterion. The criterion serves, at each step, to make the "best local choice" among several choices and there is no backtracking. One way to improve the quality of the selection consists to use the look-ahead strategy which explores several paths from the current step and to choose the choice leading to the best final solution. Such an exploration can be considered as a "global evaluation" even if it remains a heuristic. For the studied problem (SPP) a look-ahead strategy is implemented in procedure LASP (Fig. 2) and is explained below. Algorithm BSLA (Beam-Search Look-Ahead) and is given in Fig. 3. It combines BSBIS and the look-ahead. BSLA uses, at each level of the search tree, the Look-Ahead Selection Procedure (LASP) whose aim is to evaluate the nodes of the current level of the search tree.

The objective of LASP (Fig.2) is to evaluate the nodes of the current level ℓ of the developed tree. It receives two parameters: the first one corresponds to the set $B = \{\eta_\ell^1, ..., \eta_\ell^\omega\}$ representing the nodes of the current level of the tree and, the second parameter `feasible` is a boolean value which takes the value `true` if a feasible solution is reached and the value `false` otherwise. In the `Iterative`

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Input. Set $B = \{\eta_\ell^1, ..., \eta_\ell^\omega\}$ of ω nodes and a boolean variable `feasible = false`.
Output. A feasible solution corresponding to `feasible=true` or a set B_ω of ω nodes (those leading to the highest densities through the MLDP packing procedure).

`Initialization phase`
Let P_{ℓ_i} be the set of corner positions of node $\eta_\ell^i \in B$;

`Iterative phase`
`for each` node η_ℓ^i of B `do`
`for each` corner position $p_i \in P_{\ell_i}$ `do`

1. Pack C_{i+1} in p_i and insert the resulting node $\eta_{\ell+1}$ into B_ω;
2. Evaluate the new inserted node $\eta_{\ell+1}$ by placing the remaining pieces using the MLDP packing procedure;
3. `if` all circles are placed `then` set `feasible = true`, `exit`; `else` assign to $\eta_{\ell+1}$ the density obtained by MLDP;

`enddo`
`enddo`
`Terminal phase`

4. Reduce B_ω to the ω nodes that led to the highest densities by using MLDP;
5. Exit with B_ω.

Fig. 2. The look-ahead selection procedure (LASP)

`phase`, the procedure expends the nodes of B according to the positions of each node (Step 1). Such a branching creates the list B_ω of successors. Each node of B_ω is then evaluated according to the MLDP (Step 2). If MLDP reaches a feasible solution (Step 3) then the procedure stops with `feasible = true` which means that the n pieces were placed into the current fictive rectangle. Otherwise, the density obtained by MLDP is assigned to the node. If no node has led to a feasible solution, then the best nodes leading to the highest densities are returned (Steps 4–5) and so, BSLA (Fig. 3) considers these nodes for branching. Note that the density of a node corresponds to the ratio between the area of the pieces packed and the global area of the current fictive rectangle.

Figure 3 describes the main steps of BSLA. Its input contains the staring node η_ℓ and the bounds of the interval search $\{\underline{L}, \overline{L}\}$. A binary interval search is performed in the first `while` loop. First, the set B of nodes is setting equal to η_ℓ (Step 1), which corresponds to ℓ pieces already packed in the partial solution ($\ell < n$).The current length L^* of the fictive rectangle is setting equal to the middle of the interval (Step 2). Then, the second `while` loop takes place and its objective is to use LASP in order to evaluate the nodes. If LASP reached a feasible solution (the n pieces have been placed) then (i) `feasible=true`, (ii) the upper bound $\overline{L}$ is setting equal to the current value L^* (Step 6), (iii) the `while` loop stops (we can reduce the rectangle length) and (iv) rerun the first `while` loop. If LASP does not provide a feasible solution (`feasible=false`), then the best expanded nodes are returned (included in B_ω) and so, the search continues in the second `while` loop trying the computation of a feasible solution with the created nodes ($B = B_\omega$). Finally, if the procedure fails to pack all the n pieces, then ($\underline{L}$) is increased (Step 7) trying then another fictive rectangle. Note also that

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Input. A node η_ℓ, the beam width ω, and the bounds of the interval search $[\overline{L}, \underline{L}]$.
Output. A feasible packing and the best value for the rectangle length L_{best}.

`Initialization phase`

- Let B and B_ω be the sets of nodes to be considered and the offspring nodes of the node currently being considered;
- Let `feasible` be a boolean value;

`Iterative Phase.`
`while` $(\overline{L} - \underline{L} > \delta)$ `do`

1. Set $B = \{\eta_\ell\}$, where η_ℓ is a node of level ℓ characterized by I_ℓ, $\overline{I}_\ell$, and P_{I_ℓ};
2. Set $L^* = (\overline{L} + \underline{L})/2$;
3. Set `feasible` = `false`;

`while` ($B \neq \emptyset$ and `feasible`=`false`) `do`

4. Set $\ell = \ell + 1$;
5. Set B_ω = LASP(B, `feasible`);
6. `if feasible`=`true then` set $\overline{L} = L^*$ and $L_{\text{best}} = L^*$
   `else` set $B = B_\omega$ and $B_\omega = \emptyset$;

`enddo`

7. `if feasible`=`false then` set $\underline{L} = L^*$;

`enddo`

Fig. 3. Beam search look-ahead algorithm (BSLA)

the binary interval search is guided with a tolerance gap ($\delta = \overline{L} - \underline{L}$) which stops the procedure when such a difference is small enough (limited computational results showed that $\delta = 10^{-4}$ remains competitive, as used in [2]).

Figure 4 shows the "global evaluation" effect of the look-ahead strategy when combined with the beam search method. It displays the solution obtained by BSBIS and BSLA for the instance SY1 (with $n = 30$ pieces). It is to note that all the circles of this instance are of different radii, and since the SPP instances must be strongly heterogeneous (radii are practically all different, see Wäscher et al. [14]), SY1 is a good candidate to show the behavior of the method. So we can observe that the standard beam search (BSBIS) solution oscillates when the beam width varies. This phenomenon decreases much when adding the look-ahead strategy. This can be explained by the fact that the strategy used considers a more extended global evaluation, as discussed above. Also, the solution quality often improves when the beam width ω increases with the look-ahead strategy. It is to note also that the same phenomenon was observed for all tested instances. As a consequence, the method can be executed with only some "large values" of the beam width ω. More precisely EBS (detailed below) is executed with the following tunings: $\omega = 10 + 5 \times i, i \in \mathbb{N}$.

5.2 Extended Beam Search-Based Algorithm (EBS)

A separate-multi-start beam search algorithm, denoted by SEP-MSBS, was proposed by Akeb et al. [2]. It is based on combining both separate-beams and

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

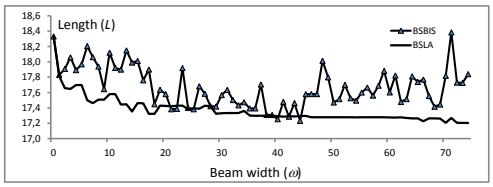


Fig. 4. Effect of the use of the look-ahead strategy on the solution of the instance SY1 (SY1 contains $n = 30$ circular pieces)

multi-start strategies. The first strategy considers distinct searches in the developed tree and the second one consists in running the algorithm by focusing on the first piece to be packed. Let $M = \{C_j\}, j = 1, \ldots, m$ be the subset of the m circular types of the problem (pieces corresponding to the different radii of N among the n pieces to pack). Hence, the multi-start strategy tries to restart the treatment at most m times, each time with a new piece of M. In this work, we propose to introduce an additional stage which is based on the look-ahead strategy whose aim is to perform a better selection mechanism used by the algorithm at each level of the developed tree.

Note that in SPP, the pieces to pack are generally strongly heterogeneous, i.e., their radii are different (c.f. [14]) and so, $m \lessapprox n$ (c.f. Columns 2–3 of Tab. 2). Here, in order to improve the efficiency of the algorithm in terms of computation time and solution quality, the multi-start is run with a subset $M' \subseteq M$ containing $m' \leq m$ pieces such that $m' = m \times \alpha,\ (0 < \alpha \leq 1)$. The objective is to detect the best value for the parameter α according to the instance size n. For example, if $\alpha = \frac{1}{2}$, then M' will contain half of the pieces of M, i.e. $M' = \{C_1, C_3, ...\}$, if $\alpha = \frac{3}{4}$ then $M' = \{C_1, C_2, C_3, C_5, ...\}$, i.e. three pieces among four ones are retained, and so on. In what follows, we consider the generalized formula for generating M' according to M: $M' = \left\{C_{\lfloor 1+\frac{k-1}{\alpha} \rfloor}\right\},\ C_{\lfloor 1+\frac{k-1}{\alpha} \rfloor} \in M,\ k = 1,\ 2,\ \ldots$

Figure 5 describes the main steps of algorithm EBS which works as follows. EBS starts by ranking the pieces in non-increasing order of their radii, the best length is initialized to the upper bound $\overline{L}$ (Step 2). In Step 3, the subset M' is constructed as explained above, and finally the index j of the first circular piece of M' is initialized (Step 4). The iterative phase is composed of two main steps: The first one is the *Generational step* and serves to generate the root node η_1 which packs the piece C_j into the current fictive rectangle. Second, the algorithm branches out of η_1 and generates the nodes corresponding to the second level $\ell = 2$. It then selects the ω^{th} node from B (Step 8), which corresponds to η_2 containing two pieces already packed. Hence, the separate beams can then be used from the aforementioned level (for more details, the reader can refer to [2]). The *Improvement step* uses BSLA as a starting point (Step 9) with η_2 as an input node. It then evaluates the feasibility of the process: if `feasible=true`, then BSLA reached a feasible solution and so, both $\overline{L}$ and L_{best} are updated (Step 10). Otherwise, the best expanded nodes B_ω are retained. The lower bound $\underline{L}$ is then reinitialized to the trivial lower bound (Step 11) and another piece is selected (Step 12) for restarting the algorithm. Finally, EBS returns L_{best} which corresponding to the best (smallest) length obtained.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Input. The beam width value ω, and parameter α $(0 < \alpha \leq 1)$.
Output. A feasible packing with the best length L_{best} for the strip.

`Initialization phase`

1. Rank the pieces of N in decreasing value of their radii;
2. Set $L_{\text{best}} = \overline{L}$, the best target length found so far;
3. Construct the set M' according to the set M;
4. Set $j = 1$, where j denotes the index of the first circular piece of M', $(1 \leq j \leq m')$;

`Iterative phase`
`Do`
Generational step

5. Generate the node η_1, characterized by I_1, $\overline{I}_1$, and P_{I_1}, by placing the first piece C_j inside the current rectangle and let $B = \eta_1$;
6. Branch out of B and generate the list of offspring nodes B_ω;
7. Let $B = \min(\omega, |B_\omega|)$ nodes having the best MLDPs and corresponding to distinct corner positions and reset $B_\omega = \emptyset$;
8. Let η_2 be the node at position ω in B;

Improvement step.

9. Set `feasible` = BSLA(η_2, ω, $\overline{L}$, $\underline{L}$);
10. `if feasible`= `true then` $\overline{L}$ and L_{best} are updated if a better length is obtained by BSLA;
11. Set $\underline{L} = (\pi \times \sum_{i=1}^{n} r_i^2)/W$;
12. Set $j = j + 1$;

`while` $(j \leq m')$;

`Terminal phase`
`exit` with the best target length L_{best}.

Fig. 5. An Extended Beam Search-Based Algorithm (EBS)

6 Computational Results

The proposed algorithm (EBS) was coded in C language and run on a 3-GHz processor and 256 MB of RAM and tested on a set of eighteen instances. These instances represent the most known ones in the literature for the SPP. The first six instances, denoted by SY1, SY2, ..., SY6 were extracted from Stoyan and Yaskov [12] and their size varies from 20 to 100 pieces. The twelve other instances are taken from [1] and were generated by combining several instances of the first group, their size varies from 45 to 200 pieces (recall that for SPP, the pieces of each instance must be strongly heterogeneous). The results obtained by EBS are compared to the best and most known results of the literature obtained either by MHD (algorithms B1.0 and B1.5) [7], by the parallel MHD (B1.6) [9], by BSBIS and/or the multi-start strategy when it is associated with BSBIS [2], or by the SEP-MSBS algorithm [2].

As in [1, 2], the maximum beam width value ω does not exceed the constant 100 and the cumulative runtime is limited to thirty hours for all the algorithms

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

(MHD was also run with the same computation time limit). Furthermore, each algorithm can abort when the beam width limit is matched or when the time limit is attained. As discussed in Sect. 5.1, ω is fixed to $10 + 5 * i, i \in \mathbb{N}$ for EBS and ω varies in the discrete interval $\{1, \ldots, 100\}$ for BSBIS and SEP-MSBS. In addition, according to Sect. 5.2 and Fig. 5, the multi-start strategy is run with different values for the parameter α. Indeed, such a variation is made in the discrete interval $\{0.25, 0.5, 0.75, 1\}$.

Table 1. Behavior of EBS when varying the value of the parameter α

	$\alpha = 1$				$\alpha = 0.75$				$\alpha = 0.5$				$\alpha = 0.25$			
Inst.	L	ω^*	t_{ω^*}	$\overline{\omega}$	L	ω^*	t_{ω^*}	$\overline{\omega}$	L	ω^*	t_{ω^*}	$\overline{\omega}$	L	ω^*	t_{ω^*}	$\overline{\omega}$
SY1	17.0954	20	7620	45	17.0955	20	1508	75	17.1641	55	7700	85	17.1367	50	3460	100
SY2	14.4548	15	580	80	14.4548	15	116	100	14.4548	15	76	100	14.4548	15	37	100
SY3	14.4017	70	21885	70	14.4106	20	580	100	14.4106	20	383	100	14.4243	25	347	100
SY4	23.3538	10	4135	35	23.4316	55	26000	55	23.4317	55	17465	65	23.4737	55	8400	85
SY5	36.0061	10	–	10	36.0435	10	–	10	36.0045	15	11270	15	35.9748	15	64530	15
SY6	36.6629	10	–	10	36.6653	10	–	10	36.5573	15	88300	15	36.6358	15	64020	15
SY12	29.8148	20	95570	25	29.7344	35	11140	35	29.7202	35	35740	40	29.7626	40	24370	45
SY13	30.4547	15	71400	15	30.4193	25	42000	30	30.3989	35	54880	40	30.4354	30	19640	40
SY14	37.7244	10	91200	15	37.7829	20	58100	20	37.7129	15	23230	25	37.7195	20	20830	30
SY23	27.7574	30	38830	30	27.6931	35	30000	40	27.7178	40	26600	50	27.7179	40	13870	55
SY24	34.1511	15	30130	15	34.1314	25	42800	30	34.1087	30	42800	35	34.0972	35	29000	40
SY34	34.6744	10	57240	15	34.6794	25	35900	25	34.6794	25	42760	30	34.6807	25	20500	35
SY56	64.7876	10	–	10	64.6904	10	–	10	64.6904	10	–	10	64.7430	10	–	10
SY123	43.0930	15	47285	15	43.1130	15	66100	20	43.0917	15	45340	25	43.0411	20	39700	25
SY124	48.6101	15	66690	15	48.5358	15	49120	15	48.4892	15	67150	15	48.6714	15	45070	20
SY134	49.2739	15	64780	15	49.1996	10	76220	15	49.2000	10	52380	15	49.1733	15	60360	20
SY234	45.4586	15	46580	15	45.4576	15	67100	15	45.5780	10	27400	20	45.5167	15	30321	25
SY1–4	60.3346	10	–	10	60.1561	10	–	10	60.0036	10	72610	10	60.1225	10	72160	15

Table 1 summarizes the results reached by EBS when varying the parameter α. Column 1 displays the instance's name (additional informations for each considered instance are reported in Tab. 2). For each value of α, four informations are reported in Tab. 1: (i) the best solution (length of the final rectangle realized L), (ii) the beam width ω^* reaching L, (iii) the runtime t_{ω^*} (measured in seconds) when EBS is run with $\omega = \omega^*$, and (iv) the maximum beam width $\overline{\omega}$ explored by the algorithm according to the imposed computation time limit. Finally, the symbol "–" means that the runtime limit is attained for the first value of ω ($\omega = 10$) which appears clearly for large-sized instances.

From Tab. 1, we can observe that the value of $\overline{\omega}$ increases when the value of α decreases. Of course, such a phenomenon can be explained by the fact that EBS tries less starting solutions and so, it allows to reach greater values for the beam width when considering the same runtime limit. On the one hand, the same table reveals that $\alpha = 1$, for the instances whose size $n < 40$, is able to provide the better results compared to those reached when fixing other values for α. On the other hand, EBS has a better behavior with $\alpha = 0.5$ when considering the large-scale instances. Indeed, such a tuning is able to improve several better solutions of the literature (indicated in Column 11 of Tab. 2). Then, the value $\alpha = 1$ (resp. $\alpha = 0.5$) remains a good compromise when applying the multi-start strategy, especially for the instances with $n < 40$ (resp. $n \geq 40$).

Table 2 displays the results provided by the proposed algorithm EBS. The solution quality is compared to those reached by the best algorithms of the literature. Columns 1 and 2 of the table report the instance's name and its size (n). Column 3 shows the number of piece types (m) of the instance, Column 4

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Table 2. Computational results of EBS on the problem instances of the literature

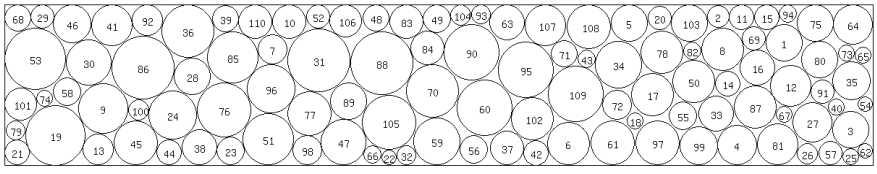
				MHD	B16	BSBIS		SEP-MSBS		Best Lit.	EBS	
Inst.	n	m	W	L	L	L	ω^*	L	ω^*	L	L	ω^*
SY1	30	30	9.5	17.291	17.247	17.2315	45	17.2070	50	17.2070	**17.0954**	20
SY2	20	20	8.5	14.535	14.536	14.6277	86	14.5287	24	14.4867	**14.4548**	15
SY3	25	25	9	14.470	14.467	14.5310	78	14.4616	44	14.4176	**14.4017**	70
SY4	35	35	11	23.555	23.717	23.6719	42	23.4921	66	23.4921	**23.3538**	10
SY5	100	99	15	36.327	**35.859**	36.0796	95	36.1818	22	35.859	36.0045	15
SY6	100	98	19	36.857	**36.452**	36.8456	85	36.7197	26	36.452	36.5573	15
SY12	50	48	9.5	30.067	–	29.7011	52	**29.6837**	61	29.6837	29.7202	35
SY13	55	54	9.5	30.891	–	30.6371	100	**30.3705**	68	30.3705	30.3989	35
SY14	65	65	11	38.265	–	38.0922	79	37.8518	63	37.8518	**37.7129**	15
SY23	45	45	9	28.270	–	27.8708	98	**27.6351**	89	27.6351	27.7178	40
SY24	55	54	11	34.605	–	34.5476	26	34.1455	49	34.1455	**34.1087**	30
SY34	60	59	11	34.901	–	34.9354	39	34.6859	43	34.6376	34.6794	25
SY56	200	193	19	69.979	–	64.7246	65	65.2024	6	64.7246	**64.6904**	10
SY123	75	72	9.5	43.626	–	43.2558	64	43.0306	25	42.9931	43.0917	15
SY124	85	82	11	49.335	–	48.8927	90	48.8411	35	48.8411	**48.4892**	15
SY134	90	88	11	49.721	–	49.3954	100	49.3362	27	49.3254	**49.2000**	10
SY234	80	78	11	45.8880	–	45.9526	83	45.6115	39	45.5576	45.5780	10
SY1234	110	105	11	61.906	–	60.2613	48	60.0564	25	60.0564	**60.0036**	10

displays the width (W) of the strip of each instance. Column 5 reports the best solution obtained by MHD-based algorithms (B1.0 and B1.5 of Huang et al.,[7]). Column 6 shows the solution value realized by the parallel version of B1.5 (Kubach et al., [9]). Columns 7 and 8 tally the length provided by the BSBIS algorithm [1] and its corresponding best beam width ω^* respectively. Columns 9 and 10 report the same information than those of Columns 7 and 8 but for SEP-MSBS [2]. Column 11 summarizes the best known solution of the literature realized by one of the algorithms displayed on the previous columns or reached when the multi-start strategy is used in [2]. Finally, Columns 12 and 13 display the length obtained by EBS and the corresponding beam width respectively.

From Tab.2, we can remark that EBS outperforms MHD on all the instances and is able to improve several solution values compared to the results reached by B1.6 (such improvement is performed on instances SY1, SY2, SY3, and SY4). Note that B1.6 remains competitive for both instances SY5 and SY6 which contain 100 pieces each. In addition, EBS performs better than BSBIS in 17 occasions out of 18 (except for SY12 for which BSBIS remains competitive). EBS outperforms SEP-MSBS in 14 occasions out of 18. Finally, we can observe that EBS is able to improve the best known solutions of the literature in 10 values out of 18, which represents a percentage of 55.56% of the solutions.

Figure 6 displays the solution obtained by EBS on instance SY1–4 containing 110 circular pieces: the length L realized by EBS is equal to 60.0036, which corresponds to a new upper bound in the literature.

Fig. 6. Solution of EBS on SY1–4($n = 110$ pieces, $\alpha = 0.5$, and $L = 60.0036$)



Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

7 Conclusion

In this paper an extended beam search-based algorithm (EBS) is proposed for approximately solving the strip packing problem. EBS combines several strategies: beam search, look-ahead, and the multi-start. The performance of the proposed algorithm was evaluated on a set of problem instances of the literature, varying from small/medium instances (20 circles) to large-scale ones (200 circles). Our limited computational results showed that EBS is able to improve more than 50% of the best known solutions of the literature. For a future work, it remains interesting to see how the parallelization can be profitable for such a method, especially when solving large-scale problem instances.

References

1. Akeb, H., Hifi, M.: Algorithms for the circular two-dimensional open dimension problem. Int. Trans. in Oper. Res. **15** (2008) 685–704.
2. Akeb, H., Hifi, M., Negre, S.: An augmented beam search-based algorithm for the circular open dimension problem. Comput. Ind. Eng., **In Press**, doi:10.1016/j.cie.2011.02.009 (2011).
3. Baltacioglu, E., Moore, J.T., Hill, R.R.: The distributor's three-dimensional pallet-packing problem: a human intelligence-based heuristic approach. Int. J. Oper. Res. **1** (2006) 249–266
4. Birgin, E.G., Martinez, J.M., Ronconi, D.P.: Optimizing the packing of cylinders into a rectangular container: A nonlinear approach. Eur. J. Oper. Res. **160** (2005) 19–33.
5. George, J.A., George, J.M., Lamar, B.W.: Packing different-sized circles into a rectangular container. Eur. J. Oper. Res. **84** (1995) 693–712.
6. Hifi, M., M'Hallah, R.: Approximate algorithms for constrained circular cutting problems. Comput. Oper. Res., **31** (2004) 675–694.
7. Huang, W.Q., Li, Y., Akeb, H., Li, C.M.: Greedy algorithms for packing unequal circles into a rectangular container. J. Oper. Res. Soc. **56** (2005) 539–548.
8. Kim, K.H., Kim, J.B.: Determining load patterns for the delivery of assembly components under JIT systems. Int. J. Prod. Econ. **77** (2002) 25–38.
9. Kubach, T., Bortfeldt, A., Gehring, H.: Parallel greedy algorithms for packing unequal circles into a strip or a rectangle, Cent. Eur. J. Oper. Res., **17** (2009) 461–477.
10. Lewis, R., Song, S., Dowsland, K., Thompson, J.: An investigation into two bin packing problems with ordering and orientation implications. Eur. J. Oper. Res. **213** (2011) 52–65.
11. Ow, P.S., Morton, T.E: Filtered beam search in scheduling. Int. J. Prod. Res. **26** (1988) 35–62.
12. Stoyan Y.G., Yaskov, G.N.: Mathematical model and solution method of optimization problem of placement of rectangles and circles taking into account special constraints. Int. Trans. Oper. Res. **5** (1998) 45–57.
13. Sugihara K, Sawai M, Sano H, Kim DS, Kim D. Disk packing for the estimation of the size of wire bundle. Japan Jour. Ind. App. Math. **21** (2004) 259–278.
14. Wäscher, G., Haussner, H., Schumann, H.: An improved typology of cutting and packing problems. Eur. J. Oper. Res. **183** (2007) 1109–1130.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Progress Rate in Noisy Genetic Programming for Choosing λ

Jean-Baptiste Hoock[1] and Olivier Teytaud[1,2]

[1] TAO (Inria), LRI, UMR 8623(CNRS - Univ. Paris-Sud),
bat 490 Univ. Paris-Sud 91405 Orsay, France, firstname.name@lri.fr
[2] Dept. of Computer Science and Information Engineering, National University of Tainan, Taiwan

Abstract. Recently, it has been proposed to use Bernstein races for implementing non-regression testing in noisy genetic programming. We study the population size of such a $(1+\lambda)$ evolutionary algorithm applied to a noisy fitness function optimization by a progress rate analysis and experiment it on a policy search application.

1 Introduction

Genetic programming (GP) consists in automatically building a program for solving a given task. The fitness function quantifies the efficiency of a program for this task.
Non-regression testing consists in testing each new version of a program, in order to check that it is at least as good as the previous version. Non-regression testing is very difficult when the fitness function is noisy, as it is uncertain. Statistical tests have to take into account the high number of tests - this is a non-trivial issue. The present work originates in the tedious non-regression testing in a highly collaborative development, often poorly performed by humans. They don't care at the individual level of the global risk due to multiple testings. Tests had to be automatized, simultaneously with the development of improvements in a search module by automatic means in order to get rid of human biases in such developments.
Noisy GP is an important problem, related to many applications. In particular, direct policy search with symbolic controlers is noisy genetic programming, as well as the design of sorting algorithms faster on average on a huge number of instances. Bandits have been investigated very early for this problem[9, 7]. However, bandits address the problem of the load balancing between different possible mutations, but not the validation of the selection, i.e. the halting criterion for the offspring evaluation. Races[11] have the double advantage of considering the load balancing and the statistical validation. [11] considers the case in which we look for all good mutations. We might, in GP, be more interested in finding one good mutation, as μ good mutations do not necessarily cumulate to one better mutation. Yet, the case of selecting several good mutations is important also, but it was shown in [8] that there are frameworks in which $\mu = 1$ is more relevant and we focus on this case. In the case of

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

continuous optimization, races have been theoretically analyzed as a tool for ensuring optimal rates (within log factors) in [12]. A partial theoretical analysis, essentially ensuring consistency, was proposed for GP in [8]: we here extend this work by an analysis of progress rate. Importantly, the theoretical analysis proposes a choice for λ by optimisation of the progress rate.

Progress rate theory is classical in continuous optimization[2]. The progress rate will be here adapted to noisy GP. We will consider T (formally defined below), the number of fitness evaluations before finding a good mutation. The progress rate is then defined as $1/T$, the inverse time before finding a good mutation. We will then, following the continuous case, choose parameters that optimize the progress rate. Note that here, the progress rate is a success probability; this is equivalent to classical criteria [3] only in restricted settings (see assumptions in Section 3).

2 Framework and notations

Consider a program P, and a set M of possible mutations; let $P+m$ be the program after application of mutation m. Assume that the fitness is stochastic; $f(P+m)$ is a random variable with values in $[-1,1]$. In all the paper we consider maximization. We consider $(1+\lambda)$ genetic programming algorithms as in Alg. 1; this is a simple $(1+\lambda)$-algorithm[8].

RBGP algorithm.
Parameters: a risk level δ, a pop. size λ, a coefficient $c > 1$, a threshold $\epsilon > 0$.
Let $K = 1/(\sum_{n\geq 1} 1/n^c)$
Let P be a program to be optimized
Let M be a set of possible mutations on P
Let $n \leftarrow 0$
while There is some time left **do**
  while no mutation accepted **do**
    Let $n \leftarrow n+1$
    Randomly draw $pop = \{m_1, \ldots, m_\lambda\}$ in M
    Perform a Bernstein race with risk $\delta_n = K\delta/n^c$ on pop and threshold ϵ
    if A mutation m is selected **then**
      $P \leftarrow P+m$
    end if
  end while
end while

Algorithm 1: One-plus-lambda Racing-Based Genetic Programming. We assume $c > 1$ so that K is finite.

All this section is written assuming that the range of fitness values is bounded in absolute value by 1, i.e. when we compute the fitness of a

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

point we get an answer between -1 and 1. We assume that the fitness is unbiased, *i.e.* the expected value of the measurements is equal to the real fitness. This is standard in *e.g.* Monte-Carlo sampling, or in some randomized forms of Quasi-Monte-Carlo samplings.
Hoeffding and Bernstein bounds quantify the effect of noise on empirical averages. In noisy cases, the fitness $fitness(x)$ of a point x is unknown: we have only access to $y_1, \ldots, y_k$, k real numbers, if $fitness(x)$ has been approximated k times; a natural estimate is $\widehat{fitness}(x) = \frac{1}{k}\sum_{i=1}^{k} y_i$.
Hoeffding or Bernstein bounds state that with probability at least $1-\delta$, $|fitness(x) - \widehat{fitness}(x)| < deviation$, where $deviation$ is

$$deviation_{Hoeffding} = \sqrt{\log(2/\delta)/k}. \quad (1)$$

[1, 11] has shown the efficiency of using Bernstein's bound instead of Hoeffding's bound, in some settings. The deviation term is then:

$$deviation_{Bernstein} = \sqrt{\hat{\sigma}^2 2\log(3/\delta)/n} + 3\log(3/\delta)/k. \quad (2)$$

This equation depends on $\hat{\sigma}^2$, the empirical variance of the measurements $y_1, \ldots, y_k$. An interesting feature of this equation is that using $\hat{\sigma}^2$ and not the real variance σ^2 is not an approximation: the inequality is rigorous with $\hat{\sigma}^2$ (contrarily to many asymptotical confidence intervals).
The Bernstein race is a typical one, following [11], except that we want to validate one and only one mutation, because in our framework it is known [8] that two good mutations do not necessarily cumulate, in the sense that sometimes:

$$\begin{aligned} \mathbb{E}f(P+m_1) &> \mathbb{E}f(P) \\ \mathbb{E}f(P+m_2) &> \mathbb{E}f(P) \\ \text{and yet } \mathbb{E}f(P+m_1+m_2) &< \mathbb{E}f(P). \end{aligned}$$

In a $(1+\lambda)$-GP, we look for a mutation which provides a better success rate than the current parent, i.e. a higher fitness. However, we will here translate the fitness by substracting the fitness of the parent; i.e. with a fitness function with values in $[-1, 1]$ and a parent z, we define

$$fitness'(x) = (fitness(x) - fitness(z))/2;$$

this just doubles the number of calls to the fitness. Therefore, in the Bernstein race, and without loss of generality, we look for a mutation which gives a positive fitness, instead of a mutation with $fitness(x) > fitness(z)$. The Bernstein race is as presented in Alg. 2, and the *computeBounds* function is defined as shown in Alg. 3.
where # denotes the cardinal operator.
Important properties of Bernstein's races as above, and which hold with probability at least $1-\delta$, are the followings ([11]).
Properties of Bernstein races.

- *Property 1. The number of evaluations in a Bernstein race*
  - *with population size #pop;*
  - *with parameters δ and ϵ;*

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

BernsteinRace(pop, δ, ϵ)
while $pop \neq \emptyset$ **do**
  for all $m \in pop$ **do**
    Let n be the number of simulations of mutation m.
    Simulate m n more times (*i.e.* now m has been simulated $2n$ times).
    //this ensures $nbTests(m) = O(\log(n(m)))$
    computeBounds(m, M, δ)
    if $lb(m) > 0$ **then**
      Return individual corresponding to mutation m.
    else if $ub(m) < \epsilon$ **then**
      $pop = pop \setminus \{m\}$ m is discarded.
    end if
  end for
end while
Return "no good individual in the offspring!"

Algorithm 2: Bernstein race for selecting good individuals in a population *pop*. M is the complete set of arms (global variable; see Alg. 1).

Function *computeBounds*(m, pop, δ)
Static internal variable: $nbTests(m)$, initialized at 0.
Let $n = nbTests(m)$.
Let r be the total reward over those n simulations.
$nbTests(m) = nbTests(m) + 1$
$lb(m) = \frac{r}{n} - deviation_{\text{Bernstein}}\left(\delta/(\#pop \times \pi^2 nbTests(m)^2/6), n\right)$.
$ub(m) = \frac{r}{n} + deviation_{\text{Bernstein}}\left(\delta/(\#pop \times 2^{\pi^2 nbTests(m)^2/6}), n\right)$.

Algorithm 3: Function for computing a lower and an upper bound on arm m with confidence $1 - \delta$, where *pop* is the complete set of arms.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

- *with a population such that, with* $p = \max(\epsilon, \max_{m \in pop} \mathbb{E}f(P+m))$*, all expected fitness values are in* $[-\Theta(\epsilon), p]$ *(possibly all fitness values* ≤ 0*, case in which there's no good mutation).*

is

$$Time(pop, \delta, \epsilon) = \Theta(Th(\#pop, \delta, \epsilon, p))$$

where

$$Th(\#pop, \delta, \epsilon, p) = \left(\log(\frac{\#pop \log(1/p)}{\delta}) \right) \max\left(\sigma^2/p^2, 1/p\right) \quad (3)$$

with: σ^2 *is an upper bound on the variance of fitness values:*

$$\sigma^2 = \sup_m \mathbb{E}(f(P+m) - \mathbb{E}f(P+m))^2.$$

- *Property 2. If a mutation is selected, then it has fitness* > 0*.*
- *Property 3. If there is a mutation with fitness* $\geq \epsilon$ *then the race will return a mutation.*

We will here focus on the case $\sigma^2 = \Theta(1)$ (which corresponds to the applicative framework in which variance does not decrease; this is consistent with many applicative fields, in particular when optimizing the parameters of a strategy with optimal success rate $< 100\%$ - yet, other cases might be considered in a future work), and therefore Eq. 3 can be replaced by Eq. 4 without significant loss:

$$Th(\#pop, \delta, \epsilon, p) = \left(\log(\frac{\#pop \log(1/p)}{\delta}) \right) /p^2. \quad (4)$$

Properties of RBGP. *Eq. 4 and properties of Bernstein races above lead to the following properties of RBGP, which hold (all simultaneously) with probability at least* $1-\delta$*: if there are*

- $n-1$ *iterations of RBGP in which all mutations have expected fitness* $\in [-\epsilon, 0]$*;*
- *and thereafter,* 1 *iteration of RBGP with at least one mutation with expected fitness* p *and all other mutations with expected fitness in* $[-\epsilon, p]$*.*

then

- *Property A: RBGP will not return a bad mutation (i.e. a mutation with fitness* ≤ 0*);*
- *Property B: If there is a mutation with fitness* $\geq \epsilon$ *then a mutation will be found;*
- *Properby C: And in that case the halting time is at most*

$$O\left(\sum_{i=1}^{n-1} Th(\lambda, K\delta/i^c, \epsilon, p) + Th(\lambda, K\delta/n^c, p) \right) \quad (5)$$

Eq. 5 will be central in the progress rate analysis below.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

3 Progress rate of Racing-based GP

We consider that randomly drawn mutation have fitness[1]:
- $q > 0$ with probability $f > 0$;
- ≤ 0 otherwise.

We will study the behavior of $(1 + \lambda)$-GP depending on q and f. The other relevant parameters are:
- λ (the offspring size);
- ϵ, the threshold of the Bernstein races (see Alg. 1);
- c, a parameter used in Alg. 1 and which is of moderate importance as shown below.

The different cases under analysis are (i) $q = \epsilon$ (ii) $q >> \epsilon$. We will investigate the running time, i.e. the number T of fitness evaluations before finding a good mutation with probability at least $1 - 2\delta$. It is already known[8] that with probability at least $1 - \delta$,
- if there is a good mutation ($fitness \geq q$), it will be found;
- no bad mutation ($fitness < 0$) will be selected. Possibly, the race replies that it did not find any good mutation.

We will show (i) that a too mild rejection threshold ϵ has bad effects (section 3.1); (ii) that a good tuning provides significant improvements (section 3.2); (iii) that there is a parameter free version with ensures that there's no infinite loop (section 3.3).

3.1 Too mild rejection: $q >> \epsilon$

Here, the precision required for rejecting a bad mutation is very small in front of the quality of good mutations. The following result shows that in that case the choice of λ is crucial: λ should be of the order

$$\log(\delta)/\log(1-f) \tag{6}$$

Theorem 1: rejection pressure too small (ϵ too small). *Assume that $q > \epsilon$. Then,*
- *If $\lambda \geq \lceil \frac{\log(\delta)}{\log(1-f)} \rceil$, then $T = \Theta(\lambda \log(\lambda/(q\delta))/q^2)$.*
- *If $\lambda \leq \frac{\log(1/2)}{\log(1-f)}$, then $T = \Omega(\log(1/(f^2\epsilon\delta)/\epsilon^2)$.*

Proof: Eq. 5 shows that the computational cost is

$$O\left(\sum_{i=1}^{n-1} Th(\lambda, K\delta/i^c, \epsilon, q) Th(\lambda, K\delta/n^c, \epsilon, q)\right). \tag{7}$$

where n is the number of iterations before a good mutation is in an offspring.
The basic fact for the following is that the probability, for an offspring, to contain no good mutation is exactly $(1-f)^\lambda$.
We distinguish the two cases of the theorem:

[1] We recall that fitnesses are translated as explained in section 2 so that the parent has fitness 0 and therefore "good" mutations are mutations with value > 0.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

– If $\lambda \geq \lceil \frac{\log(\delta)}{\log(1-f)} \rceil$, then
$$(1-f)^{\lambda} \leq \delta.$$
This implies that with probability at least $1-\delta$, there's at least one good mutation in the first offspring. Therefore, $n = 1$; this and Eq. 7 yield the expected result.
– If $\lambda \leq \frac{\log(1/2)}{\log(1-f)}$, then
$$(1-f)^{\lambda} \geq \frac{1}{2}.$$
Therefore with probability at least $\frac{1}{2}$, there's no good mutation in the first offspring. Then, Eq. 7 (the first term) is enough for showing that
$$T = \Omega(\log(1/(f^2\epsilon\delta)/\epsilon^2)$$
which is the expected result. □

Remark: In the case $q >> \epsilon$, this theorem implies that $\lambda \geq \lceil \frac{\log(\delta)}{\log(1-f)} \rceil$ is much better than $\lambda \leq \frac{\log(1/2)}{\log(1-f)}$.

3.2 Well tuned parameters: $q = \epsilon$

Theorem 2: population size with well tuned parameters. *Assume that $q = \epsilon > 0$. Then,*
$$T = \Theta(\frac{\lambda \log(\lambda \log(\frac{1}{\epsilon})/\delta)}{(1-(1-f)^{\lambda})\epsilon^2}). \tag{8}$$

Proof: We use Eq. 5, which shows that
$$O\left(\sum_{i=1}^{n-1} Th(\lambda, K\delta/i^c, \epsilon, q) + Th(\lambda, K\delta/n^c, \epsilon, q)\right). \tag{9}$$
Then, we compute n, the number of offsprings before we have a good mutation. A good mutation is found in an offspring with probability $1-(1-f)^{\lambda}$. Therefore, n is a geometric random variable with parameter $z^{-1} = \left(1-(1-f)^{\lambda}\right)$: its expected value is $\Theta(z)$, and its standard deviation is $\Theta(z)$. So, with probability $1-\delta$, $n = \Theta(1/\left(1-(1-f)^{\lambda}\right)$. This and Eq. 9 lead to
$$T = \Theta\left(\sum_{i=1}^{n} Th(\lambda, K\delta/i^c, \epsilon, q)\right). \tag{10}$$
This is equivalent to
$$T = \Theta\left(nTh(\lambda, K\delta/n^c, \epsilon, q)\right). \tag{11}$$
which in turn is equivalent to Eq. 8 for a fixed c. □

Remark: Theorem 2 provides an evaluation of the cost
$$T = \Theta(\frac{\lambda \log(\lambda \log(\frac{1}{\epsilon})/\delta)}{(1-(1-f)^{\lambda})\epsilon^2}).$$
If we neglect all logarithmic factors,

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

- this is linear as a function of λ if $\lambda \simeq 1/n$, i.e. the overall cost is linear as a function of $1/(f\epsilon^2)$.
- this is linear as a function of $1/(f\epsilon^2)$ if $\lambda = 1$.

We therefore see that the population size does not matter a lot when the parameter ϵ is chosen so that it nearly matches q.

3.3 No prior knowledge

The analysis above has the weakness that it requires some knowledge on the fitnesses of possible mutations, in order to choose ϵ, the parameter used in the rejection rule. The main risk is a too strong rejection: $q << \epsilon$ would lead to the rejection of the best mutations. We here investigate results possible with no knowledge at all.

BernsteinRace(pop, δ, ϵ)
while $pop \neq \emptyset$ **do**
  for all $m \in pop$ **do**
    Let n be the number of simulations of mutation m.
    Simulate m, n more times
      (*i.e.* now m has been simulated $2n$ times).
      //this ensures $nbTests(m) = O(\log(n(m)))$
    computeBounds(m, M, δ)
    if $lb(m) > 0$ **then**
      Return individual corresponding to mutation m.
    else if $average\ fitness(m) < 0$ **then**
      $pop = pop \setminus \{m\}$ m is discarded.
    end if
  end for
end while
Return "no good individual in the offspring!"

Algorithm 4: Variant of Bernstein race: a pattern is rejected as soon as its average fitness is below 0.

Theorem 3: population size with $q << \epsilon$. *Consider RBGP with the Bernstein race as in Alg. 4. Then, with probability $1-\delta$, no bad mutation is accepted, and if the probability of a good mutation is positive, then*

$$T < \infty. \tag{12}$$

Proof: *The fact that no bad mutation is accepted (with probability at least $1 - \delta$) follows the same lines as in other cases, as the acceptance criterion has not been modified.*
We have to show $T < \infty$. This is proved as follows:

- *A mutation with fitness $> q$ is selected infinitely often, as such a mutation is in the population with positive probability by assumption.*
- *Such a mutation has a non-zero probability of having always a positive empirical average.* □

4 Experimental results

The theoretical results above for choosing the population size are rather preliminary; the population size can be chosen optimally only if we have many informations. On the other hand, it proposes a criterion different from classical Bernstein races: acceptance is based on the same criterion as usual Bernstein races, but rejection is based on a simple naive empirical average, and with this criterion we have T finite without any prior knowledge. We here experiment rules a bit more complicated than algorithms above.

We have already tried the algorithm on the program MoGo (a software of Go) without real success against the full version of the software. We have decided to compare with another testbed. The testbed is Monte-Carlo Tree Search (MCTS [4, 6]) on the game NoGo [5]. It is a two-player board game. It is a variant of the game of Go. The rule is the following : the first player which captures one or several stone(s) has lost and the pass move is forbidden. This game has been designed by the Birs seminar on games as a nice challenge for game developpers. In all our experiments, we have worked on the size 7x7 of the game.

The baseline is the program NoGo (adapted from MoGo[10]) without mutation. In our experiments, we have added some rules in order to reject more rapidly some bad mutations, combining Algo. 2 (which requires some knowledge on the distributions) and Algo. 4 (which requires no knowledge). This leads to Algo. 5, which is somehow a combination of these two algorithms, empirically developped for our problem.

$BernsteinRace(pop, \delta, \epsilon)$
while $pop \neq \emptyset$ **do**
  for all $m \in pop$ **do**
    Let n be the number of simulations of mutation m.
    Simulate m n more times (*i.e.* now m has been simulated $2n$ times).
    //this ensures $nbTests(m) = O(\log(n(m)))$
    $computeBounds(m, M, \delta)$
    if $lb(m) > 0$ **then**
      Return individual corresponding to mutation m.
    else if $average\ fitness(m) < 0.004$ or ($average\ fitness < 0.006$ and $n > 10^5$) **then**
      $pop = pop \setminus \{m\}$ m is discarded.
    end if
  end for
end while
Return "no good individual in the offspring!"

Algorithm 5: Empirically modified version of Bernstein race for our problem. It is essentially Alg. 4, with a bit more of rejection for fastening the algorithm.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

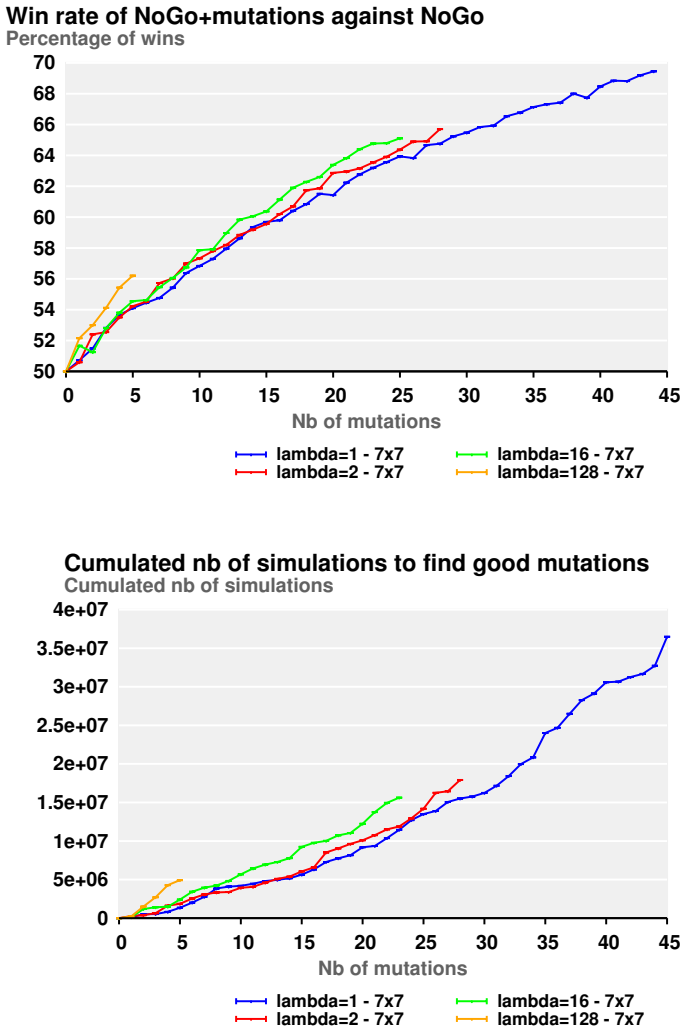


Fig. 1. Top: win rate of NoGo+mutations against the baseline (i.e. NoGo without mutation). Win rates are given with a precision of +/-0.3%. **Bottom:** the "running time" (measured by the cumulated number of simulations) for finding a good mutation.

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Fig. 1 shows a slightly better result for $\lambda = 16$, for a fixed number of mutations compared to $\lambda = 1$ and $\lambda = 2$; but more extensive experiments (possibly on toy datasets) are required; results are nearly the same for all λ in our real-world case. In all cases, we could get a nice curve, with a very significant (almost 70%) success rate against the baseline, which is still clearly increasing (yet, in a slower manner).
Another question is about the best population size λ in our experiments. It seems that we have generally a good mutation with frequency 1/15. Using Eq. 6, with $\delta = 0.05$ and assuming $f \simeq 1/15$, we get $\lambda \simeq \log(0.05)/\log(1 - 1/15) \simeq 43$. Therefore our analysis suggests a population size $\lambda \simeq 43$.

5 Conclusion

The use of Bernstein races for rigorously performing non-regression testing was already proposed in [8]. We here investigate the natural question of the choice of the population size λ, and the modification of Bernstein races when no prior knowledge is available:

- **Choosing the population size.** The good news is that we find a formula for optimaly choosing λ, equal to $\log(\delta)/\log(1 - f)$ where f is the frequency of good mutations and δ the risk level chosen by the user. Unfortunately, f is unlikely to be known unless the fitness improvement q that one can expect from good mutations is nearly known, and in this case the user is likely to choose ϵ of the order of q, and we show that in this case there is little to win by a good choice of λ: $\lambda = 1$ performs at least nearly as well as all values of λ.
- **What if we have no prior knowledge ?** We could propose a modified Bernstein race (Alg. 4) which has the advantage that it always converges ($T < \infty$), independently of all parameters.

In the experiments, we heuristically combined our various tools for optimizing the performance, proposing Algo. 5. Importantly, we got a very significant result on a new game, NoGo, recently proposed by the Birs seminar on games - the curve shows a regular improvement, for each parametrization of the algorithm. Importantly, we made this work with applications in minds; further investigations, on toy datasets for convenience and clarity, are necessary - so that we can see experimental results with confidence intervals, bridging the gap between our maths and our real-world experiments.

Acknowledgments.

The authors would like to thank Fabien Teytaud for the help to read the article. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

1. J.-Y. Audibert, R. Munos, and C. Szepesvari. Use of variance estimation in the multi-armed bandit problem. In *NIPS 2006 Workshop on On-line Trading of Exploration and Exploitation*, 2006.
2. A. Auger and N. Hansen. Reconsidering the progress rate theory for evolution strategies in finite dimensions. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 445–452, New York, NY, USA, 2006. ACM.
3. H.-G. Beyer. *The Theory of Evolution Strategies*. Natural Computing Series. Springer, Heidelberg, 2001.
4. G. Chaslot, J.-T. Saito, B. Bouzy, J. W. H. M. Uiterwijk, and H. J. van den Herik. Monte-Carlo Strategies for Computer Go. In P.-Y. Schobbens, W. Vanhoof, and G. Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91, 2006.
5. C.-W. Chou, O. Teytaud, and S.-J. Yen. Revisiting Monte-Carlo Tree Search on a Normal Form Game: NoGo. In *EvoGames 2011*, volume 6624 of *Lecture Notes in Computer Science*, pages 73–82, Turino, Italy, Apr. 2011. Springer-Verlag.
6. R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, 2006.
7. J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.*, 2(2):88–105, 1973.
8. J.-B. Hoock and O. Teytaud. Bandit-Based Genetic Programming. In *13th European Conference on Genetic Programming*, Istanbul, Turkey, 2010. Springer.
9. J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Evolution*. MIT Press, Massachusetts, 1992.
10. C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong. The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in games*, pages 73–89, 2009.
11. V. Mnih, C. Szepesvári, and J.-Y. Audibert. Empirical Bernstein stopping. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 672–679, New York, NY, USA, 2008. ACM.
12. P. Rolet and O. Teytaud. Bandit-based Estimation of Distribution Algorithms for Noisy Optimization: Rigorous Runtime Analysis. In *Lion4*, Venice Italie, 2010.

Index of Authors

Editors: Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, Marc Schoenauer

Index of authors