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]).

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

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

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

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%.

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

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

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

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)

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)