STOCHASTIC SHORTEST PATH PROBLEM WITH UNCERTAIN DELAYS

Jianqiang Cheng[1], Stefanie Kosuch[2] and Abdel Lisser[1]
[1]*Laboratoire de Recherche en Informatique (LRI), Université Paris Sud - XI*
Bât. 490, 91405 Orsay Cedex, France
[2]*Institutionen för datavetenskap (IDA), Linköpings Universitet*
58183 Linköping, Sweden
{*cheng,lisser*}*@lri.fr, stefanie.kosuch@liu.se*

Keywords: Stochastic programming: Shortest path: Projected gradient algorithm: Active set methods: Branch-and-bound

Abstract: This paper considers a stochastic version of the shortest path problem, the Stochastic Shortest Path Problem with Delay Excess Penalty on directed, acyclic graphs. In this model, the arc costs are deterministic, while each arc has a random delay, assumed normally distributed. A penalty occurs when the given delay constraint is not satisfied. The objective is to minimize the sum of the path cost and the expected path delay penalty. In order to solve the model, a Stochastic Projected Gradient method within a branch-and-bound framework is proposed and numerical examples are given to illustrate its effectiveness. We also show that, within given assumptions, the Stochastic Shortest Path Problem with Delay Excess Penalty can be reduced to the classic shortest path problem.

1 INTRODUCTION

The Shortest Path (SP) problem is one of the best known combinatorial optimization problems and has been extensively studied for a long time ((Dijkstra, 1959), (Bellman, 1958), (Ford and Fulkerson, 1962)). The objective of SP problem is to find a path with minimum distance, time or cost between two specified vertices of a given graph. There is a surprising variety of real life applications, including operations research, robotics, transportation and communications, e.g. figuring out how to direct packets to a destination across a network.

In the deterministic SP problem, all the parameters (distances, time or cost) are known. However in real life, due to failure, maintenance or other reasons, different kinds of uncertainties are frequently encountered and must be taken into account, e.g. the traffic delay between two cities. In these cases, it is natural to model parameters by continuously or discretely distributed random variables, which turns the underlying problem into a stochastic optimization problem ((Sahinidis, 2004)). Till now, there have been many papers presenting the Stochastic Shortest Path Problem (SSPP). Hutsona and Shierb (Hutsona and Shierb, 2009), Mirchandani et al. (Mirchandani and Soroush, 1985) and Murthy et al. (Murthy and Sarkar, 1996) considered the problem of selecting a path which maximizes utility functions or minimizes cost functions in a stochastic network, where arc lengths are random. Ohtsubo (Ohtsubo, 2003; Ohtsubo, 2008) selects a probability distribution over the set of successor nodes and formulates such a problem as a Markov decision process. Provan (Provan, 2003), Polychronopoulos and Tsitsiklis (Polychronopoulos and Tsitsiklis, 1996) studied expected shortest paths in networks where information on arc cost values is accumulated as the graph is being traversed, while Nikolova (Nikolova et al., 2006) maximizes the probability that the path length does not exceed a given threshold value.

In this paper, we study a special SSPP, the Stochastic Shortest Path Problem with Delay Excess Penalty (SSPD). In this model, each arc has a deterministic cost and a random delay. Furthermore, we assume that the arc delays are independently normally distributed. The problem has a simple recourse formulation. That means we deal with the delays of the path by introducing a penalty which occurs in the case where the delay constraint is not satisfied. The objective is to minimize the sum of the cost and the expected path delay penalty. As the deterministic shortest path problem with delay is NP-hard ((Verweij et al., 2003)), it follows that the SSPD with normally distributed delays is NP-hard by choosing all variances of the arcs equal to 0.

SSPD has been previously studied (see (Verweij et al., 2003)). In this paper, the authors give near-optimal solutions with the Sample Average Approximation method. More precisely, the authors approximate the initial SSPD (with arbitrary delay distribution) with an SSPD with jointly discretely distributed delays.

We propose a Stochastic Projected Gradient method within a branch-and-bound framework to solve the SSPD on directed, acyclic graphs. The main idea of this algorithm is to search the set of feasible solutions (all directed paths from a source node s to a sink node t) using a depth-first search method on the given directed graph. In order to reject subspaces of the search space, lower bounds are computed by solving the corresponding relaxed problems with a Stochastic Projected Gradient method. Furthermore, we show that under a weak assumption, SSPD can be simplified into the classic shortest path problem and thus be solved in polynomial time.

The paper is organized as follows. SSPD is introduced in section 2. In section 3, we prove that SSPD can be significantly simplified into the classic shortest path problem in the case of a positive linear dependence between the arc costs and the means and variances of the arc delays. In section 4, the Stochastic Projected Gradient algorithm is presented and a specific branch-and-bound algorithm is designed. In section 5, some numerical examples are given to reveal the effectiveness of the approach. The conclusions are given in the last section.

2 SSPD FORMULATION

Let $\mathcal{G} = (V,A)$ be a simple and acyclic digraph, where $V = \{1,2,...,n\}$ represents the set of the nodes and $A \subseteq \{(v,w) : v,w \in V, v \neq w\}$ represents the set of arcs. Each arc $a \in A$ has an associated cost $c(a) > 0$ as well as an independently normally distributed delay with strictly positive mean represented by the random variable $\delta(a)$. We further assume for two distinct arcs a and a' that $\delta(a)$ and $\delta(a')$ are independent.

The *Stochastic Shortest Path Problem with Delay Excess Penalty (SSPD)* consists in finding a directed path between two given vertices s and t such that the sum of the cost and the expected delay cost is minimal. The delay cost is based on a penalty per time unit $d > 0$ that has to be paid whenever the total delay exceeds a given threshold $D > 0$.

SSPD can be formulated as a stochastic combinatorial optimization problem in the following way: Let $x \in \{0,1\}^{|A|}$ such that each component x_a of x represents an arc $a \in A$. For a directed path P, we define the corresponding $x = x(P)$ such that $x_a = 1$ if and only if $a \in P$. Then, SSPD can be mathematically formulated as follows:

$$\min_{x\in\{0,1\}^{|A|}} \mathbb{E}[\mathcal{J}(x,\delta)] := \sum_{a\in A} c(a)x_a + d\cdot\mathbb{E}[\sum_{a\in A}\delta(a)x_a - D]^+$$

$$s.t. \forall v \in V: \sum_{\substack{w\in V:\\(v,w)\in A}} x(v,w) - \sum_{\substack{w\in V:\\(w,v)\in A}} x(w,v) = \begin{cases} 1 \text{ if } v = s,\\ -1 \text{ if } v = t,\\ 0 \text{ else.}\end{cases}$$

where $[\cdot]^+ = max\{0,\cdot\}$, $\mathbb{E}[X]$ denotes the expectation of a random variable X and $\delta \in \mathbb{R}^{|A|}$ is the vector consisting of the random variables $\delta(a)$. Note that the constraints of SSPD are the common shortest path constraints.

SSPD can be compactly written as:

$$\text{(SSPD)} \quad \min_{x\in\{0,1\}^{|A|}} \quad \mathbb{E}[\mathcal{J}(x,\delta)] \tag{1a}$$

$$\text{s.t. } Mx = b \tag{1b}$$

where $M \in \mathbb{R}^{n\times|A|}$ is the *node-arc incidence matrix* (see (Ahuja et al., 1993)) and $b \in \mathbb{R}^n$, where all elements are 0 except the s-*th* and t-*th* element, which are 1 and -1, respectively.

Remark that we can drop arbitrarily one of the shortest path constraints of the SSPD in order to obtain a full rank matrix M.

In the case of independently normally distributed delays, the objective function of SSPD has a deterministic equivalent formulation and can thus be evaluated exactly (see (Kosuch and Lisser, 2010)) :

$$\min_{x\in\{0,1\}^{|A|}} \quad \sum_{a\in A} c(a)x_a + d\cdot[\hat{\sigma}\cdot f(\frac{D-\hat{\mu}}{\hat{\sigma}}) + (\hat{\mu}-D)(1-F(\frac{D-\hat{\mu}}{\hat{\sigma}}))] \tag{2}$$

where $\hat{\mu} = \sum_{a\in A}\mu(a)x_a$, $\hat{\sigma} = \sqrt{\sum_{a\in A}\sigma^2(a)x_a^2}$, in which $\mu(a)$ and $\sigma^2(a)$ are the mean and the variance of $\delta(a)$, respectively. $f(\cdot)$ and $F(\cdot)$ are density and cumulative distribution function of the standard normal distribution.

3 SIMPLIFIED SSPD

In some cases, SSPD can be simplified significantly. Here we introduce one special case, where, for each arc, the mean and variance of the arc delay is positively proportional to the cost of the arc. For instance, in a transportation network, it is often the case that the traffic delay time is proportional to the length of the road, i.e., the longer the road, the more traffic jams and thus the longer the delay. For these graphs, we can prove that the optimal solution of SSPD is the same as the one of its corresponding classic shortest path problem. More precisely, one just needs to determine the path such that the sum of the costs is minimized, which can be done in polynomial time. This can be used to obtain benchmarks for numerical tests.

First, we introduce the following lemma:

Lemma 1. *Let $Y := \sum_{a\in A} \delta(a)x_a$, then the objective function of SSPD is a nondecreasing function of the expectation and the variance of Y.*

Proof. We suppose that the expectation and the standard deviation of Y are μ and σ. Let $X = \frac{Y-\mu}{\sigma}$ and we have:

$$\begin{aligned} G(\mu,\sigma) &:= \mathbb{E}[[Y-D]^+] = \mathbb{E}[[\sigma X+\mu-D]^+] \\ &= \int_{\frac{D-\mu}{\sigma}}^{\infty} (\sigma x+\mu-D) f_X(x)dx \\ &= \sigma \int_{\frac{D-\mu}{\sigma}}^{\infty} x\cdot f_X(x)dx + (\mu-D)\int_{\frac{D-\mu}{\sigma}}^{\infty} f_X(x)dx \end{aligned}$$

where $f_X(\cdot)$ is the probability density function of X.

The objective function is nondecreasing for μ, provided that its partial derivative with respect to μ is non-negative:

$$\begin{aligned} \frac{\partial G(\mu,\sigma)}{\partial \mu} &= \sigma\cdot(-1)\frac{D-\mu}{\sigma} f_X(\frac{D-\mu}{\sigma})(\frac{-1}{\sigma}) \\ &+ \int_{\frac{D-\mu}{\sigma}}^{\infty} f_X(x)dx + (\mu-D)(-1) f_X(\frac{D-\mu}{\sigma})(\frac{-1}{\sigma}) \\ &= \int_{\frac{D-\mu}{\sigma}}^{\infty} f_X(x)dx \end{aligned}$$

Since the density function $f_X(x)$ is non-negative, we have $\int_{\frac{D-\mu}{\sigma}}^{\infty} f_X(x)dx \geq 0$. Therefore, $\mathbb{E}[[Y-D]^+]$ is a nondecreasing function of μ.

For σ, we also prove that the partial derivative with respect to σ is non-negative:

$$\begin{aligned} \frac{\partial G(\mu,\sigma)}{\partial \sigma} &= \int_{\frac{D-\mu}{\sigma}}^{\infty} x\cdot f_X(x)dx \\ &+ \sigma\cdot(-1)\frac{D-\mu}{\sigma} f_X(\frac{D-\mu}{\sigma})(\frac{D-\mu}{-\sigma^2}) \\ &+ (\mu-D)(-1) f_X(\frac{D-\mu}{\sigma})(\frac{D-\mu}{-\sigma^2}) \\ &= \int_{\frac{D-\mu}{\sigma}}^{\infty} x\cdot f_X(x)dx \end{aligned}$$

Let $B = \frac{D-\mu}{\sigma}$,

$$\frac{\partial G(\mu,\sigma)}{\partial \sigma} = \int_B^{\infty} x\cdot f_X(x)dx$$

When $B \geq 0$, we have $\int_B^{\infty} x\cdot f_X(x)dx \geq 0$.

As $X = \frac{Y-\mu}{\sigma}$, we have $\mathbb{E}(x) = 0$, i.e., $\int_{-\infty}^{\infty} x\cdot f_X(x)dx = 0$. When $B < 0$, and as $f(x) \geq 0$, we thus get:

$$\begin{aligned} &\int_{-\infty}^{0} x\cdot f_X(x)dx \leq \int_B^0 x\cdot f_X(x)dx \leq 0 \Longrightarrow \\ &0 = \mathbb{E}(X) = \int_{-\infty}^{0} x\cdot f_X(x)dx + \int_0^{\infty} x\cdot f_X(x)dx \\ &\leq \int_B^0 x\cdot f_X(x)dx + \int_0^{\infty} x\cdot f_X(x)dx \end{aligned}$$

From above, we have $\frac{\partial G(\mu,\sigma)}{\partial \sigma} \geq 0$. Therefore $\mathbb{E}[[Y-D]^+]$ is also a nondecreasing function of σ. Correspondingly, $\mathbb{E}[[Y-D]^+]$ is a nondecreasing function of σ^2, the variance of Y. □

We formulate the following assumptions:

(A1) *for each arc, the expectation and variance of the delay are positively proportional to the cost of the arc.*
(A2) *for two distinct arcs a and a', the delay $\delta(a)$ and delay $\delta(a')$ are distributed independently.*

Remark: there is no need to assume that the delay $\delta(a)$ is normally distributed.

Theorem 1. *Under assumption (A1) and assumption (A2), the SSPD is equivalent to the classic shortest path problem*

$$\begin{aligned} \min_{x\in\{0,1\}^{|A|}} &\sum_{a\in A} c(a)x_a \\ s.t.\ &Mx = b \end{aligned}$$

Proof. We suppose that the proportions of the expectation and the variance to the cost are $C1 > 0$ and $C2 > 0$, respectively. Let $Y = \sum_{a\in A} \delta(a)x_a$. As $x_a \in \{0,1\}$, i.e., $x_a^2 = x_a$ and given assumption (A2), we have

$$\mathbb{E}(Y) = \mathbb{E}(\sum_{a\in A}\delta(a)x_a) = \sum_{a\in A} x_a\mathbb{E}(\delta(a)) = C1\sum_{a\in A} c(a)x_a$$

$$\begin{aligned} Var(Y) &= Var(\sum_{a\in A}\delta(a)x_a) = \sum_{a\in A} x_a Var(\delta(a)) \\ &= C2\sum_{a\in A} c(a)x_a \end{aligned}$$

By Lemma 1, we get the conclusion that the problem $\min_{x\in S\subseteq\{0,1\}^{|A|}} \mathbb{E}[\mathcal{J}(x,\delta)]$ has the same optimal solution as $\min_{x\in S\subseteq\{0,1\}^{|A|}} \sum_{a\in A} c(a)x_a$. □

4 PROBLEM SOLVING METHOD

A specific branch-and-bound algorithm for SSPD is introduced in this section. The main idea of the algorithm is as follows: on the one hand, in order to get a lower bound to exclude some subsets of the solution space $\mathcal{P}$, we solve the corresponding relaxed, i.e., continuous version of SSPD (see subsection 4.1) with a Stochastic Projected Gradient method. On the other hand, instead of introducing a binary search tree for the branch-and-bound procedure, we directly use the given graph to browse $\mathcal{P}$ (i.e., the set of directed paths from s to t (see subsection 4.2)).

4.1 Solving the Relaxed SSPD

Since the continuous relaxation of SSPD (where $x \in \{0,1\}^{|A|}$ is replaced by $x \in [0,1]^{|A|}$) is a convex problem (see for example (Kosuch and Lisser, 2010)), we can solve it by using a Stochastic Projected Gradient method. The basic idea is as follows: at each iteration $k \geq 1$, we first estimate the gradient $\nabla_x \mathbb{E}[\mathcal{J}(x^{k-1},\delta)]$ by $\sum_{j=1}^{N} \nabla_x \mathcal{J}(x^{k-1},\overline{\delta}_k^j)/N$ (where x^{k-1} is the feasible solution vector computed in the previous iteration and $\overline{\delta}_k^j$, $j = 1,\ldots,N$, are N samples of the random vector δ, which are regenerated at the *k-th* iteration; see subsection 4.1.1). This gradient estimator is projected onto the null space of the matrix M. x^k is then computed as usual, i.e., x^{k-1} minus the projected gradient times the step size. In case we obtain negative components of x, we adapt (i.e., shorten) the step size (subsection 4.1.2).

The complete algorithm is given in Algorithm 1. In the following subsections we define the used variables and give further details on the algorithm.

4.1.1 Estimating the Gradient of j

$\mathcal{J}$ is differentiable everywhere except for those points x where $\sum_{a\in A} \delta(a)x_a - D = 0$. The set of all these points is a null set and can thus be neglected as the aim of all stochastic gradient algorithms is to approximate the gradient of the expectation of $\mathcal{J}$ via a sampling procedure. Therefore, we define the gradient of $\mathcal{J}$ as follows:

$$\nabla_x \mathcal{J}(x,\delta) = \begin{cases} c & \text{if } [\sum_{a\in A} \delta(a)x_a - D \leq 0] \\ c + d\cdot\delta & \text{otherwise} \end{cases}$$

So the estimator of the gradient $\nabla_x \mathbb{E}[\mathcal{J}(x^{k-1},\delta)]$ is

$$\nabla_x \hat{E}[\mathcal{J}(x^{k-1},\delta)] = \frac{\sum_{j=1}^{N} \nabla_x \mathcal{J}(x^{k-1},\overline{\delta}_k^j)}{N}$$

4.1.2 Projection and Update of x

At iteration $k \geq 1$, let $r^k := \nabla_x \hat{E}[\mathcal{J}(x^{k-1},\delta)]$ be the estimator of the gradient. Its projection on the null space of M is done by multiplying it with the projection matrix $T^M := \mathbb{I}_n - M^T(MM^T)^{-1}M$. Then, x is updated as follows:

$$x^k = x^{k-1} - \rho^k(T^M \cdot r^k)$$

where ρ^k is the step size given by a σ-sequence $(\rho^k)_{k\in\mathcal{N}}$[1].

However, the predefined step size ρ^k might be too large in the sense that we can obtain components of x^k that are negative.

In order to handle negative components, we proceed as follows: Let I_-^k be the index set of the strictly negative components of x^k. We then compute the maximum step size that keeps x^k in the feasible region by

$$\overline{\rho}^k = \min_{i\in I_-^k} \left\{ \frac{x_i^{k-1}}{(T^M \cdot r^k)_i} \right\}$$

and update x accordingly:

$$x^k = x^{k-1} - \overline{\rho}^k(T^M \cdot r^k)$$

Proposition 1. *Let x^0 be a feasible solution of the relaxed SSPD. Then, using the update procedure mentioned above, x^k remains feasible for the relaxed SSPD for all $k \geq 1$.*

Proof. Using the update procedure above, $Mx^k = M(x^{k-1} - \overline{\rho}^k(T^M \cdot r^k))$. As $T^M = \mathbb{I}_n - M^T(MM^T)^{-1}M$, we have: $Mx^k = Mx^{k-1}$. So provided that x^0 is a feasible solution, $Mx^k = b$ for all $k \geq 1$.

By using the step size $\overline{\rho}^k$ we assure that $x_i^k \geq 0$ for all $k = \{1,\ldots,n\}$.

Define $\overline{A} := \{a \in A | x^k(a) \neq 0\}$. First of all remark that due to constraints (1b) x^k defines a (positive) flow on $\overline{\mathcal{G}} = (V,\overline{A})$ with value 1. As $\mathcal{G}$ contains no directed cycles, we can now partition V in disjunct vertex sets $V_1,\ldots,V_\kappa$ such that

1. $\bigcup_{i=1}^{\kappa} V_i = V$
2. $V_1 = \{s\}$ and $V_\kappa = \{t\}$
3. Let $a = vw \in \overline{A}$. Then there exist $h, j \in \{1,\ldots,\kappa\}$ with $h < j$, s.t. $v \in V_h$ and $w \in V_j$.

Let $a = vw \in \overline{A}$ and $v \in V_h$, $h < \kappa$. Then there exists a cut $C_a = \bigcup_{i=1}^{h} V_i$ such that $a \in E(C_a,\overline{C_a})$, denoting the set of the edges between C_a and $\overline{C_a}$, and $E(\overline{C_a},C_a) = \emptyset$. As

[1] A σ-sequence is a sequence $(\rho^k)_{k\in\mathcal{N}}$ that satisfies $\lim_{k\to\infty} \rho^k = 0$ and $\sum_{k=0}^{\infty} \rho^k \to \infty$.

$$x(C_a,\overline{C_a}) := \sum_{\substack{v\in C_a\\ w\in\overline{C_a}}} x_{vw} = 1 \quad \text{and} \quad x_a > 0 \forall a \in \overline{A}$$

it follows that $x_a \leq 1$. This ends the proof. □

4.1.3 Active Set Methods

However, if $x_i^{k-1} = 0$ for a $k \geq 1$ and an index $i \in \{1,\dots,n\}$, we get the step size $\overline{\rho}^k = 0$. In this case, we are (and will keep) stuck on the current, probably non-optimal solution. To prevent this, we use the *active set method* (see (Luenberger and Ye, 2008)), which introduces a set of additional equality constraints, the so called *active set* $\mathcal{A}^k$. As this set is continuously updated, we use a superscript that indicates in which iteration the set $\mathcal{A}^k$ is active.

For $k \geq 1$ let $I_0^{k-1} := \{i\,|\,x_i^{k-1} = 0\}$. Then the active set for iteration k is defined as:

$$\mathcal{A}^k = \{\,[x_i = 0]\,|\,i \in I_0^{k-1}\}$$

Now, instead of projecting r^k on the null space of the matrix M, we project it on the null space of a matrix Z^k: This matrix consists of the matrix M enlarged by $|I_0^{k-1}|$ rows that correspond to the equality constraints in $\mathcal{A}^k$:

$$\begin{aligned} Z_i^k &= M_i && \text{for} \quad i = 1,\dots,n \\ Z_i^k &= e_{\tau^{k-1}(n-i)} && \text{for} \quad i = n+1,\dots,n+|I_0^{k-1}| \end{aligned}$$

where $\{\tau^{k-1}(1),\dots,\tau^{k-1}(|I_0^{k-1}|)\} = I_0^{k-1}$ and e_i is the *i-th* row of the n-dimensional identity matrix. Remark that Z^k might have linearly dependent rows. In this case the projection matrix can be computed as $T^k = I - (Z^k)^T(Z^k(Z^k)^T)^+Z^k$ where $(Z^k)^+$ is the unique Moore-Penrose pseudoinverse of Z^k.

If the computed projected gradient is zero, we might have obtained a local optimum of the deterministic variant of problem (1) with delay vectors $\overline{\delta}_k^j, j = 1...N$ and additional equality constraints given by $\mathcal{A}^k$. In this case we compute Lagrange multipliers as follows:

$$\lambda = -(Z^k(Z^k)^T)^+Z^kr^k$$

If all the multipliers associated with the constraints in $\mathcal{A}^k$ are positive, we have reached the optimal solution of the deterministic variant of problem (1) with delay vectors $\overline{\delta}_k^j, j = 1...N$. In this case we stop the algorithm. Otherwise, we remove the constraint with the most negative multiplier from $\mathcal{A}^k$ and start a new iteration.

Clearly, as the algorithm is stochastic, it might take some additional time to meet the stopping criterion that multipliers are positive, even though the

Algorithm 1 : Stochastic Projected Gradient Algorithm

```
• Initialization: Given constants $\varepsilon$, $K_1$, $K_2$ and a $\sigma$-sequence $(\rho^k)_{k\in\mathcal{N}}$. Choose $x^0$ feasible for SSPD (1) (for example using depth-first search on the graph). Set $k = 1$.
• For all $a \in A$ draw $N$ samples $\overline{\delta}_k^j(a), j = 1,\dots,N$ of $\delta(a)$ according to its normal distribution.
• Determine $Z^k$ and let $T^k$ be the matrix for projection on the null space of $Z^k$. Compute the approximated gradient $r^k := \sum_{j=1}^N \nabla_x \mathcal{J}(x^{k-1},\overline{\delta}_k^j)/N$.
  If $\mathbf{T^k \cdot r^k = 0}$: Compute the Lagrange multipliers of the current equality constraints.
  – If all multipliers associated with the constraints in $\mathcal{A}^k$ are positive, STOP.
  – Else delete the constraint from $\mathcal{A}^k$ having the most negative associated multiplier. Set $k = k+1$ and start a new iteration.
  Else: Update $x^k$ as follows: $x^k = x^{k-1} - \rho^k(T^k \cdot r^k)$
  – If $\min_{\mathbf{i}\in\{\mathbf{1},\dots,\mathbf{n}\}} \mathbf{x_i^k} < \mathbf{0}$: Define $I_-^k = \{i\,|\,x_i^k < 0\}$ and compute a new step size: $\overline{\rho}^k = \min_{i\in I_-^k}\left\{\frac{x_i^{k-1}}{(T^k\cdot r^k)_i}\right\}$.
    Update $x^k$ as follows: $x^k = x^{k-1} - \overline{\rho}^k(T^k \cdot r^k)$
  – If $k > K_1$ and $|\mathbb{E}[\mathcal{J}(x^k,\delta)] - \mathbb{E}[\mathcal{J}(x^{k-K_2},\delta)]| < \varepsilon$, STOP.
    Otherwise Set $k = k+1$ and start a new iteration.
```

current solution is (near) optimal. That is why we add an additional stopping criterion: if there is no significant improvement in the objective value, we stop the algorithm.

4.2 Branch-and-bound Framework

Definition 1. *Let P be a directed path. We say that an arc $a = (v,w)$ has its origin in P, if $v \in P$ but $a \notin P$. For a path P we define the set of all arcs that have their origin on P as O_P.*

The branch-and-bound algorithm can be stated as follows: First we solve the relaxed version of the overall problem, which gives us a solution $\tilde{x}$ of the relaxation as well as a first lower bound LB. We then begin to search for a feasible binary solution by plunging the graph (see phase 4). The obtained directed path P from s to t together with the corresponding lower bound $LB(P) = LB$ are stored in a pool of waiting s-t-paths $\mathcal{L}$. In addition, we store the value of $\tilde{x}$ for all arcs $a \in O_P$ in a variable $x_P(a)$. The solution value of SSPD given by P is our first upper bound and it is stored in the variable UB.

Then, each further iteration of our branch-and-bound algorithm consists of (up to) five phases:

Phase 1: Selecting a branch
If $\mathcal{L}$ is empty, the algorithm terminates. Output UB. Otherwise, we select a path $P \in \mathcal{L}$ such that $LB(P) = \min_{Q\in\mathcal{L}} LB(Q)$.

Phase 2: Selecting an arc
If no arc in O_P is left that has not already been examined (i.e., $\max\{x_P(b)|b \in O_P\} = -1$, see phase 5), we delete P from $\mathcal{L}$, end the iteration and go to phase 1. Otherwise, we go to the first vertex v on P such that there still exists at least one arc $(v,w) \in O_P$ that has not already be examined (i.e., such that $\max\{x_P(b)|\exists w \in A : b = (v,w)\} \neq -1$). We then choose the arc a such that $x_P(a) = \max\{x_P(b)|\exists w \in A : b = (v,w)\}$. If adding a to the sub-path s-P-v leads to a non-feasible solution, we reject a (i.e., set $x_P(a) = -1$) and choose another arc in O_P by repeating phase 2.

Phase 3: Calculating a lower bound
Let $a = (v,w)$ be the arc chosen in phase 2. Consider the relaxed subproblem of SSPD obtained by fixing the first part of the s-t-path to s-P-(v,w). Solving this subproblem gives us a lower bound $\widetilde{LB}$ for the corresponding binary solution and a solution vector $\tilde{x}$. If $\widetilde{LB} < UB$ we go to phase 4. Otherwise, we reject a (set $x_P(a) = -1$) and choose a new arc (phase 2).

Phase 4: Plunging
Find a new s-t-path P' containing the sub-path s-P-(v,w): Starting from vertex w, we always add the outgoing arc with the highest value of $\tilde{x}$.

Phase 5: Storage and Update
Path P' is stored together with the corresponding lower bound $LB(P') = \widetilde{LB}$ in the pool of waiting paths $\mathcal{L}$. In addition, we define for all arcs $a \in O_{P'}$ the value $x_{P'}(a)$ as follows: For all arcs a that have their origin on s-P-v $x_{P'}(a)$ is set to -1. For the rest of the outgoing arcs, we store the corresponding component of $\tilde{x}$. $x_P(vw)$ is set to -1.
If the solution value of SSPD given by P' is lower than the current upper bound UB, we update UB.

5 NUMERICAL EXAMPLES

The Stochastic Projected Gradient method as well as the branch-and-bound algorithm were implemented in Matlab and all tests ran on a Pentium(R)D @ 3.00 GHz with 2.0 GB RAM.

We considered five directed, acyclic graphs for our tests with $(|V|,|A|)$ equal to $(23,40)$, $(50,167)$, $(75,215)$, $(100,351)$ and $(100,573)$, respectively. Among the five networks, the first is taken from another paper (see (Ji, 2005)), while the other four are modified graphs of the OR-library (see (Beasley, 2010)).

5.1 The Continuous SSPD

We compare our Stochastic Projected Gradient method with Matlab's optimization toolbox: To solve the convex, deterministic reformulation of SSPD (2), we use Matlab's fmincon function and set, as in our algorithm, an active set method as optimization algorithm. Note that Matlab uses a deterministic gradient strategy while we use a stochastic one.

As test instances, we generated the parameters for the five networks as follows: the penalty d is 10, D is set to the mean of the delay of the shortest path, the expectation $\mu(a)$ and the variance $\sigma^2(a)$ are generated uniformly on the intervals $[\bar{c}, 2\bar{c}]$ ($\bar{c}$ is the median of all the costs) and $[\sigma^2(c), 4*\sigma^2(c)]$ ($\sigma^2(c)$ is the variance of all the costs), respectively. We run our algorithm as well as Matlab 10 times on each instance. The results are shown in Table 1, where we compare our Stochastic Projected Gradient method with Matlab's optimization toolbox in terms of average CPU time in seconds and the mean of best solution value found. We also give the relative performance ratio computed as

$$\mathrm{PR} = \frac{v_{SGrad} - v_{Matlab}}{v_{SGrad}}$$

where v_{SGrad} is the mean of the best solution values obtained with the Stochastic Projected Gradient algorithm and v_{Matlab} the mean of the ones obtained with Matlab. As SSPD is a minimization problem and both methods give a feasible solution, negative ratios indicate that our algorithm found a better solution while positive ones show that Matlab performed better. To give detailed information about the numerical tests, we list the percentage of all 10 runs where the Stochastic Projected Gradient algorithm finds a better solution than Matlab's optimization algorithm. Note that for the test of graph $(100,573)$, there are 4 runs out of 10 where Matlab did not finish in 20 minutes. However, with our method it took at most 140 seconds to get a solution. These instances are omitted in the computation of the average values given in Table 1.

From Table 1, we observe that our algorithm is better than Matlab's optimization toolbox in terms of CPU time. Moreover, the CPU time of our algorithm does not exceed 70% of the CPU time Matlab takes.

(Nodes, Arcs)	Algor.	CPU time (s)	Best Val.	PR (%)	Perc-entage
(23,40)	N=1	0.36	216	18.06	0%
(23,40)	N=10	0.42	205	13.66	0%
(23,40)	N=100	0.40	205	13.66	0%
(23,40)	Matlab	1.50	177	-	-
(50,167)	N=1	2.72	13135	17.88	0%
(50,167)	N=10	2.72	13302	18.91	0%
(50,167)	N=100	2.78	13155	18.01	0%
(50,167)	Matlab	47.54	10786	-	-
(75,215)	N=1	6.62	14461	17.76	30%
(75,215)	N=10	6.42	14363	17.20	20%
(75,215)	N=100	6.52	13446	11.55	40%
(75,215)	Matlab	156.60	11893	-	-
(100,351)	N=1	30.08	7464	-36.90	70%
(100,351)	N=10	30.02	7471	-36.77	70%
(100,351)	N=100	30.34	7488	-36.46	70%
(100,351)	Matlab	133.26	10218	-	-
(100,573)	N=1	129.02	13241	10.90	50%
(100,573)	N=10	138.72	10847	-8.77	83%
(100,573)	N=100	140.26	10021	-17.73	100%
(100,573)	Matlab	200.70	11798	-	-

Table 1: Results of solving the Continuous SSPD

With respect to the produced solutions, for the two large graphs ($n = 100$), the Stochastic Projected Gradient algorithm finds the best solutions, while Matlab performs better on the small graphs with $n <= 75$. Moreover, with $N = 100$ samples our algorithm produces better solutions than Matlab in 81% of the runs on the graphs with 100 vertices. To resume, for the instances above, the Stochastic Projected Gradient algorithm performs better than Matlab's optimization toolbox when graph sizes are large, both in terms of CPU time and produced solutions. For the small graphs, Matlab gives better solutions but takes 3 times more CPU time than the Stochastic Projected Gradient algorithm.

Regarding the number of samples N, we observe no big difference between taking 10 samples and 100. However, in general, taking 100 samples gives better results than taking 1 sample: for instances 1, 3 and 5, better solutions are found with 100 samples, while the results are comparable for the other two instances. Notice that the number of samples N has no significant influence on the CPU time, although the computation load is clearly heavier at each iteration with $N = 100$ than with $N = 1$. However, recall that the stopping criterion is not a fixed number of iterations but merely based on a measure of convergence. This explains, that on some instances the algorithm takes less time when considering 100 samples per iteration than when considering only one.

5.2 The Combinatorial SSPD

In this section, we present our numerical results of the above mentioned branch-and-bound algorithm. For the five networks, we run two cases: in the first one, the expectation and the variance of the delay are directly proportional to the cost of the arc (instances SSPD1a - SSPD5a); in the second one, they are not proportional to the cost (instances SSPD1b - SSPD5b). For the first case, by Theorem 1 in section 3, we get the conclusion that the optimal solution of SSPD can be obtained in polynomial time by solving a classic shortest path problem. This provides us with a benchmark for the solution given by our algorithm. However, this doesn't suit to the second case.

Based on the numerical results for the continuous relaxation of SSPD, we set the sample number N to 10 for both cases. The other parameters are generated as follows: for the first case, the penalty d (the delay penalty per time unit) is 10, the expectation and the variance of the delay for each arc a are $\mu(a) = 10 * c(a)$ and $\sigma^2(a) = \mu(a)/9$, respectively, and the delay threshold D is set to the mean of the delay of the shortest path. For the second case, we use the same instances as the continuous SSPD, i.e., the parameters are the same as the parameters in the continuous one. In our numerical tests, we run each instance ten times. The results are shown in Table 2 and Table 3, respectively. For each of the 10 instances, Table 2 gives the mean of the solution value obtained with the branch-and-bound algorithm, the benchmark (the optimal value), the average number of considered nodes, i.e., the number of times a lower bound is calculated during the algorithm, the average CPU time in seconds and the gap, i.e., the relative difference between the solution value of the best solution provided by our algorithm and the benchmark; while Table 3 gives the mean of the solution value obtained with the branch-and-bound algorithm, the average number of considered nodes, the average CPU time in seconds.

From Table 2 and Table 3, we observe that for the first four instances the CPU time of our branch-and-bound algorithm does not exceed 400 seconds. Given the NP-hardness of the problem, the size of the graphs and number of s-t-paths (47, 88828, 810631 and up to more than 276 million for the graph with 100 vertices) the CPU times are very small compared to other branch-and-bound approaches (see e.g. (Kosuch and Lisser, 2010)). This is of course due to the quite high number of pruned subspaces that can be seen from the low number of considered nodes (that, on average, does not exceed 20 for all instances). Although, due to the approximative nature of the Stochastic Projected Gradient algorithm, it is theoretically possible that our branch-and-bound algorithm prunes subspaces that contain an optimal solution, the solutions we get are optimal for the first case, where the optimal solutions are known (i.e., all gaps are 0).

Comparing the performance of our algorithm on the instances of the first case (Table 2) with that on the instances for the second case (Table 3), we see that the algorithm considers slightly more nodes in the second case. We think that this is due to the initial plunging that, for instances where the delays are positively proportional, produces a ”relatively better” solution. This allows the algorithm to prune even more subspaces. On the other hand, the arverage of caculating of the lower bound, i.e., the ratio between the time and the nodes, are nearly same for the same graphs with $|V| > 50$ in both cases, the proportional and general case, which indicates a sort of robustness.

Instances	(Nodes, Arcs)	Best Val.	Opt. Val.	No.of nodes	CPU time (s)	Gap (%)
SSPD1a	(23,40)	41	41	6	1.54	0.00
SSPD2a	(50,167)	530	530	5	13.79	0.00
SSPD3a	(75,215)	625	625	15	146.85	0.00
SSPD4a	(100,351)	231	231	9	274.54	0.00
SSPD5a	(100,573)	110	110	15	2066.30	0.00

Table 2: Computational results for instances with proportional delays

6 CONCLUSION

In this paper, we study and solve a stochastic version of the shortest path problem with a penalty for exceeded delay. The underlying graph is assumed to be directed and acyclic. We prove that in some cases the obtained Stochastic Shortest Path Problem with Delay Excess Penalty can be greatly simplified by reformulating it as the classic shortest path problem, which can be solved in polynomial time.

Instances	(Nodes, Arcs)	Best Val.	No.of nodes	CPU time (s)
SSPD1b	(23,40)	250	18	7.00
SSPD2b	(50,167)	16182	16	65.34
SSPD3b	(75,215)	15909	15	127.96
SSPD4b	(100,351)	9842	13	372.43
SSPD5b	(100,573)	10795	39	5464.30

Table 3: Computational results for instances

To solve the problem in general we propose to use a branch-and-bound framework to search the set of feasible paths. Lower bounds are obtained by solving the corresponding linear relaxation which in turn is done using a Stochastic Projected Gradient algorithm involving an active set method. Numerical examples are given to illustrate the effectiveness of the obtained algorithm. Concerning the resolution of the continuous relaxation, our Stochastic Projected Gradient algorithm clearly outperforms the Matlab optimization toolbox on large graphs. Moreover, for instances where the cost of an arc is positively proportional to the mean and variance of its delay, our branch-and-bound algorithm indeed finds the optimal solution.

For the future work, we can generalize the assumption on the distribution and our approach should be easily extendable to more general distributions, and maybe also to more general graphs. Concerning the special case for which we have shown the Stochastic Shortest Path Problem with Delay Excess Penalty to be equivalent to the classic shortest path problem, it might be possible to weaken the underlying assumptions in order to obtain this same result for a larger class of instances.

REFERENCES

Ahuja, R., Magnanti, T., and Orlin, J. (1993). *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, New Jersey.

Beasley, J. (2010). Or-library. `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`.

Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematic*, 16(87-90).

Dijkstra, E. (1959). A note on two problems in connection with graphs. *Numerische Mathematik*, 1(269-271).

Ford, L. and Fulkerson, D. (1962). *Flows in Networks*. Princeton University Press, Princeton.

Hutsona, K. and Shierb, D. (2009). Extended dominance and a stochastic shortest path problem. *Computers and Operations Research*, 36(584-596).

Ji, X. (2005). Models and algorithm for stochastic shortest path problem. *Applied Mathematics and Computation*, 170(503-514).

Kosuch, S. and Lisser, A. (2010). Upper bounds for the 0-1 stochastic knapsack problem and a b&b algorithm. *Annals of Operations Research*, 176(77-93).

Luenberger, D. and Ye, Y. (2008). *Linear and Nonlinear Programming*. Springer, New York, 3rd edition.

Mirchandani, P. and Soroush, H. (1985). Optimal paths in probabilistic networks: a case with temporary preferences. *Computers and Operations Research*, 12(365-381).

Murthy, I. and Sarkar, S. (1996). A relaxation-based pruning technique for a class of stochastic shortest path problems. *Transportation Science*, 30(220-236).

Nikolova, E., Kelner, J., Brand, M., and Mitzenmacher, M. (2006). Stochastic shortest paths via quasi-convex maximization. In *Proceedings of European Symposium of Algorithms*, pages 552–563. Springer.

Ohtsubo, Y. (2003). Minimization risk models in stochastic shortest path problems. *Mathematical Methods of Operations Research*, 57(79-88).

Ohtsubo, Y. (2008). Stochastic shortest path problems with associative accumulative criteria. *Applied Mathematics and Computation*, 198(1)(198-208).

Polychronopoulos, G. and Tsitsiklis, J. (1996). Stochastic shortest path problems with recourse. *Networks*, 27(133-143).

Provan, J. (2003). A polynomial-time algorithm to find shortest paths with recourse. *Networks*, 41(115-125).

Sahinidis, N. (2004). Optimization under uncertainty: state-of-the-art and opportunities. *Computers and Chemical Engineering*, 28(971-983).

Verweij, B., Ahmed, S., Kleywegt, A., Nemhauser, G., and Shapiro, A. (2003). The sample average approximation method applied to stochastic routing problems: A computational study. *Computational Optimization and Applications*, 24(2-3)(289-333).