

# Comparison of the Exact and Approximate Algorithms in the Random Shortest Path Problem

Jacek Czekaj<sup>1</sup> and Lesław Socha<sup>2</sup>

<sup>1</sup>University of Silesia, Institute of Physics, 4 Uniwersytecka st., 40-007 Katowice, Poland  
jackens@math.us.edu.pl

<sup>2</sup>Cardinal Stefan Wyszyński University in Warsaw, Faculty of Mathematics and Natural Sciences. College of Sciences, 5 Dewajtis st., 01-815 Warszawa, Poland  
leslawsocha@poczta.onet.pl

## Abstract

The Random Shortest Path Problem with the second moment criterion is discussed in this paper. After the formulation of the problem, exact algorithms, based on general concepts for solving the Multi-objective Shortest Path Problem, are described. Next, several approximate algorithms are proposed. It is shown that the complexity of the exact algorithms is exponential, while the complexity of the approximated algorithms is only polynomial. Computational results for the exact and approximate algorithms, which were performed on large graphs, are shown.

## 1 Introduction

One of the basic problems in the operational research is the Classic Shortest Path Problem (shortly CSPP) that can be also interpreted as the problem of determination of the path from a given source point to the destination point with minimal transfer time.

In this paper we consider a routing problem to determine the path from a given point to the destination point with minimum transfer time. The simplest solution is to bring the problem to the CSPP.

The CSPP has been studied by many researchers. The first algorithms solving the CSPP were published by Dijkstra [5] for the single source problem and Bellmann [3], Floyd [6] for all pairs problem. There have been many variations and improvements in either analysis or special classes of the given graphs. For example Moffat, Takaoka [11] for the all pairs shortest path problem; Fredman, Tarjan [10] for the single source problem; Frederickson [7] for the single source problem in planar graphs; Johnson [9] for a sparse graphs. The detailed study and discussion of the CSPP one can find in monographs, for instance [4] or [1].

Unfortunately, in practice, the travel times are not known exactly. Therefore the travel time from one place to another should be treated as a random variable rather

than a single real number. Similar approach can be found in [12] and [13]. This is the main problem discussed in this paper.

In Sect. 2 we present an example showing that in the Random Shortest Path Problem (shortly RSPP) with the second moment criterion the subpath of the shortest path is not necessarily the shortest subpath. We also formally state the Random Shortest Path Problem with the second moment criterion. In Sect. 3 we present two exact algorithms solving RSPP with the second moment criterion. We show that these exact algorithms have exponential complexity. Therefore in the next section we present approximate algorithms which have only polynomial complexity. In Sect. 5 we show computational results of the presented exact and approximate algorithms, performed on large graphs. In the end of the paper some conclusions are given.

## 2 Problem Statement

Let  $G = (V, E)$  be a directed graph with a finite set of vertices  $V$  and a set of edges  $E \subseteq V \times V$ . We denote by  $s$  a source vertex and by  $t$  a destination vertex.

In general, in the Shortest Path Problem (shortly SSP) each edge  $e \in E$  is associated with some cost (sometimes called weight)  $c_e$ . The goal is to find the path, from the source vertex  $s$  to the destination vertex  $t$ , which minimizes a function of costs related with edges of the path.

In the CSSP this function is a sum, and the costs are real numbers (they usually express the distance between given vertices). There are many algorithms for solving the CSSP, for example, classical Bellman-Ford algorithm or classical Dijkstra algorithm. Since in the next sections we will adopt these algorithms to the Random Shortest Path Problem, therefore we present their pseudocodes below.

### Classical-Bellman-Ford-algorithm:

```

set  $c_v = \infty$  for all  $v \in V \setminus \{s\}$  and  $c_s = 0$ 
for  $i = 1$  to  $\#V - 1$  do
    relaxation_state = not_changed
    for all  $(u, v) \in E$  do
        Relax $(u, v)$ 
    if relaxation_state = not_changed then
        break
return path related with  $c_t$ 

```

### Classical-Dijkstra-algorithm:

```

set  $c_v = \infty$  for all  $v \in V \setminus \{s\}$  and  $c_s = 0$ 
 $Q = V$ 
while  $Q \neq \emptyset$  do
    choose  $u \in Q$  such that  $c_u = \min\{c_q \mid q \in Q\}$ 
     $Q = Q \setminus \{u\}$ 
    for all  $(u, v) \in E$  do
        Relax $(u, v)$ 
return path related with  $c_t$ 

```

**Relax**( $u, v$ ) denotes the following procedure:

**Relax**( $u, v$ ):  
**if**  $c_v > c_u + c_{(u,v)}$  **then**  
 $c_v = c_u + c_{(u,v)}$   
 $\pi_v = u$   
*relaxation\_state = changed*

The detailed description of the CSPP can be found for instance in [4].

In further consideration we will adopt the concept of multi-objective optimization methods. The Multi-objective Shortest Path Problem (shortly MOSPP) has been introduced by Hansen in [8]. In this version of the SSP, each edge of the graph is associated with a vector of real numbers (usually nonnegative). We recall the definition of the standard product order relation in the set  $\mathbb{R}^n$ :

$$\begin{aligned} [a_1, \dots, a_n] < [b_1, \dots, b_n] &\iff \\ \iff [a_1, \dots, a_n] \leq [b_1, \dots, b_n] \wedge [a_1, \dots, a_n] \neq [b_1, \dots, b_n], & \quad (1) \end{aligned}$$

where

$$[a_1, \dots, a_n] \leq [b_1, \dots, b_n] \iff a_1 \leq b_1 \wedge \dots \wedge a_n \leq b_n. \quad (2)$$

Operators  $\leq$  appearing in the right-hand side of the above equivalence denote the standard order relation in the set  $\mathbb{R}$ . If (1) holds then we say that the vector  $[a_1, \dots, a_n]$  dominates the vector  $[b_1, \dots, b_n]$ . In the MOSPP the goal is to find all paths from the source vertex  $s$  to the destination vertex  $t$  for which the sum of the vectors related with edges of the path is nondominated by the sum of the vectors related with edges of any other path.

Finally, in the RSPP each edge  $e \in E$  is associated with a random variable  $T_e$ . The goal is to find the path, from the source vertex  $s$  to the destination vertex  $t$ , which minimizes a function of random variables related with edges of the path, just like in the CSPP. In practice, the random variable  $T_e$ , with  $e = (u, v) \in E$ , expresses the travel time from the vertex  $u$  to the vertex  $v$ , or the distance between these vertices. Therefore we assume that every moment of the random variable  $T_e$  is nonnegative and finite. We also assume that all the random variables  $T_e$ , for  $e \in E$ , are independent. The notion of the shortest path can be defined in many different ways. It depends only on random variables related with edges of the graph.

For example, if we take as the shortest path, the path with minimal expected value of the sum of random variables related with edges of the path, then the RSPP reduces to the CSPP and can be solved with help of classical Dijkstra or classical Bellman-Ford algorithm. It follows from the equality  $\mathcal{E}[X_1 + \dots + X_n] = \mathcal{E}[X_1] + \dots + \mathcal{E}[X_n]$ , for any random variables  $X_1, \dots, X_n$ .

We note also that for any independent random variables  $X_1, \dots, X_n$  holds the equality  $\mathcal{V}[X_1 + \dots + X_n] = \mathcal{V}[X_1] + \dots + \mathcal{V}[X_n]$ , thus if we take as the shortest path, the path with minimal variance of the sum of random variables related with edges of the path, then the RSPP reduces to the CSPP, too.

However, the problem becomes more complicated when as the shortest path we take the path with a minimal sum of the variance and a square of the expected value of

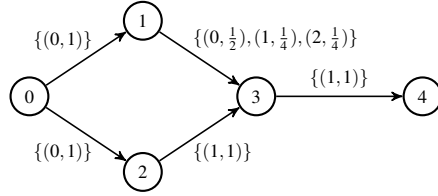


Figure 1: Example of graph, showing that a subpath of the shortest path is not necessarily the shortest subpath.

the sum of random variables related with edges of the path, i.e. the second moment of the sum of random variables related with edges of the path, because  $(\mathcal{E}[X])^2 + \mathcal{V}[X] = \mathcal{E}[X^2]$ . Similar criterion has been considered by Murthy and Sarkar in [12]. We show on a simple example that in this case the subpath of the shortest path is not necessarily the shortest subpath.

**Example 1.** Consider the graph depicted in Fig. 1 and let random variables  $T_{(0,1)}$ ,  $T_{(0,2)}$ ,  $T_{(1,3)}$ ,  $T_{(2,3)}$  and  $T_{(3,4)}$  have distributions presented in the picture. For any random variable  $T_e$ , the notation  $\{(x_1, p_1), \dots, (x_k, p_k)\}$  means that  $P(T_e=x_1) = p_1, \dots, P(T_e=x_k) = p_k$ . Calculating second order moments we obtain

$$\mathcal{E}[(T_{(0,1)}+T_{(1,3)})^2] = \frac{5}{4} \quad \text{and} \quad \mathcal{E}[(T_{(0,2)}+T_{(2,3)})^2] = 1, \quad (3)$$

therefore the shortest path between vertices 0 and 3 is the path  $\langle 0, 2, 3 \rangle$ , while the shortest path between vertices 0 and 4 is the path  $\langle 0, 1, 3, 4 \rangle$ . It follows from the equalities

$$\mathcal{E}[(T_{(0,1)}+T_{(1,3)}+T_{(3,4)})^2] = 3\frac{3}{4} \quad \text{and} \quad \mathcal{E}[(T_{(0,2)}+T_{(2,3)}+T_{(3,4)})^2] = 4. \quad (4)$$

Hence, it follows that the subpath  $\langle 0, 1, 3 \rangle$  of the shortest path  $\langle 0, 1, 3, 4 \rangle$  is not the shortest subpath. It means Bellman's "principle of optimality" does not hold!  $\square$

The above example shows that for solving the RSPP with the second moment criterion we cannot directly apply the methods effective for the CSPP.

Now we formulate the RSPP with the second moment criterion. Let

$$\mathcal{P} = \{\langle v_0, v_1, \dots, v_{n-1}, v_n \rangle \mid v_0 = s \wedge v_n = t \wedge (v_0, v_1), \dots, (v_{n-1}, v_n) \in E\} \quad (5)$$

be a set of all paths from the source vertex  $s$  to the destination vertex  $t$ . The goal is to find a path  $\langle v_0, v_1, \dots, v_{n-1}, v_n \rangle \in \mathcal{P}$  such that

$$\mathcal{E}\left[\left(\sum_{i=1}^n T_{(v_{i-1}, v_i)}\right)^2\right] = \min_{\langle u_0, u_1, \dots, u_{m-1}, u_m \rangle \in \mathcal{P}} \mathcal{E}\left[\left(\sum_{i=1}^m T_{(u_{i-1}, u_i)}\right)^2\right]. \quad (6)$$

In the next section we present exact algorithms for solving the RSPP.

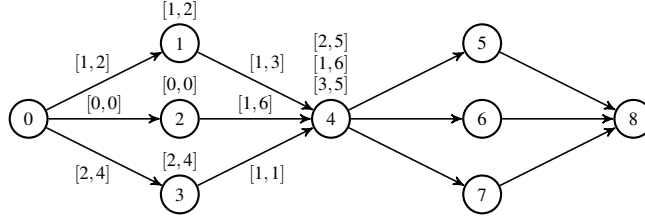


Figure 2: Illustration of cost vectors propagation.

### 3 Exact Algorithms

The second moment of the sum of independent random variables  $X_1, \dots, X_n$  can be expressed as follows

$$\mathcal{E} \left[ \left( \sum_{i=1}^n X_i \right)^2 \right] = \sum_{i=1}^n \mathcal{E}[X_i^2] + 2 \sum_{i=2}^n \mathcal{E}[X_i] \sum_{j=1}^{i-1} \mathcal{E}[X_j] \quad (7)$$

or

$$\mathcal{E} \left[ \left( \sum_{i=1}^n X_i \right)^2 \right] = \mathcal{V} \left[ \sum_{i=1}^n X_i \right] + \left( \mathcal{E} \left[ \sum_{i=1}^n X_i \right] \right)^2. \quad (8)$$

Therefore, to calculate the cost of a path we must remember the expected values and the second moments of all random variables  $T_e$ , for  $e \in E$ , or expected values and variances of all these random variables. We will refer to the representation [expected value, second moment] and [expected value, variance] shortly by EV-SM and EV-V, respectively.

In the next example we show a method for calculating costs of all possible paths from the source vertex  $s$  to the destination vertex  $t$ .

**Example 2.** Consider the graph depicted in Fig. 2 and let random variables related with the edges have vectors of expected value and variance presented in the picture. There is only one edge incoming to the vertex 1, thus  $[1, 2]$  is the only one cost vector of coming to the vertex 1 (this cost vector is denoted over vertex 1 on the graph). Similarly in the case of the vertices 2 and 3. There are three subpaths to the vertex 4:  $\langle 0, 1, 4 \rangle$ ,  $\langle 0, 2, 4 \rangle$ ,  $\langle 0, 3, 4 \rangle$  and the cost vectors of these subpaths are  $[2, 5]$ ,  $[1, 6]$  and  $[3, 5]$ , respectively. Note that the vector  $[3, 5]$  can be discarded, because if  $[\mu, \xi]$  is the cost vector related with some path from the vertex 4 to the destination vertex 8, then  $(\mu+3)^2 + (\xi+5)$  is a total cost of the path related with the vector  $[3, 5]$ , while a total cost of the path related with the vector  $[2, 5]$  is  $(\mu+2)^2 + (\nu+5)$ . In general, if  $[\mu_1, \nu_1]$  and  $[\mu_2, \nu_2]$  are cost vectors of two subpaths to the same vertex and  $[\mu_1, \nu_1] < [\mu_2, \nu_2]$  (conf. (1)), then the vector  $[\mu_2, \nu_2]$  can be discarded, because for any cost vector  $[\mu, \nu]$  related with a path from the given vertex to the destination vertex, the following inequality holds

$$(\mu_1 + \mu)^2 + (\nu_1 + \nu) < (\mu_2 + \mu)^2 + (\nu_2 + \nu). \quad (9)$$

Similar situation appears for the representation EV-SM, because if  $[\mu_1, \xi_1]$  and  $[\mu_2, \xi_2]$  are cost vectors of two subpaths to the same vertex and  $[\mu_1, \xi_1] < [\mu_2, \xi_2]$ , then

the vector  $[\mu_2, \xi_2]$  can be discarded, because for any cost vector  $[\mu, \xi]$  expanding the given subpaths to the paths to the destination vertex the following inequality

$$\xi_1 + \xi + 2\mu_1\mu < \xi_2 + \xi + 2\mu_2\mu \quad (10)$$

is satisfied.

Therefore, the RSPP with the second moment criterion reduce to the MOSPP, regardless of the data representation.

According to (7) and (8) the representation EV-V requires less multiplications than the representation EV-SM, this is very important statement for the time of computations. However, the representation EV-SM have some disadvantages, too. For example, let  $[1, 2]$  and  $[2, 1]$  be two vectors in the representation EV-V. These vectors are incomparable (since neither  $[1, 2] < [2, 1]$  nor  $[1, 2] > [2, 1]$  holds), thus the both vectors must be remembered. But in the representation EV-SM the corresponding vectors are  $[1, 2+1^2] = [1, 3]$  and  $[2, 1+2^2] = [2, 5]$ , thus the vector  $[2, 5]$  (and so the vector  $[2, 1]$ , too) does not have to be stored! Thus in the representation EV-SM elimination of dominated vectors appears more frequently.

In the next part of this section we discuss the solution of the RSPP with the second moment criterion with help of general methods for solving the MOSPP.

### 3.1 Extended Bellman-Ford Algorithm

Classical algorithms for the CSPP can be extended for the MOSPP. First, we show an extension of the classical Bellman-Ford algorithm. We associate with each vertex  $v \in V$  a list  $L_v$  of vectors  $[a, b] \in \mathbb{R}^2$ . Lists  $L_v$  for all  $v \in V \setminus \{s\}$  are initially empty and the list  $L_s$  related with the source vertex  $s$  contains the vector  $[0, 0]$  at the beginning.

Modifying the procedure of the relaxation of edge we can solve the RSPP with the second moment criterion using the following Extended Bellman-Ford algorithm (shortly EBF):

**Extended-Bellman-Ford-algorithm:**

set  $L_v = \emptyset$  for all  $v \in V \setminus \{s\}$  and  $L_s = \langle [0, 0] \rangle$

**for**  $i = 1$  **to**  $\#V - 1$  **do**

*relaxation\_state = not\_changed*

**for all**  $(u, v) \in E$  **do**

**EBF-Relax** $(u, v)$

**if** *relaxation\_state = not\_changed* **then**

**break**

**return** path related with minimal value on the list  $L_t$

The procedure **EBF-Relax**( $u, v$ ) has the following pseudocode<sup>1</sup>:

```

EBF-Relax( $u, v$ ):
for all  $[\mu_u, \nu_u] \in L_u$  do
  set  $\mu_v^{new} = \mu_u + \mathcal{E}[T_{(u,v)}]$ ,  $\nu_v^{new} = \nu_u + \mathcal{V}[T_{(u,v)}]$  and  $\xi_v^{new} = (\mu_v^{new})^2 + \nu_v^{new}$ 
  vector_state = incomparable
  for all  $[\mu_v, \nu_v] \in L_v$  do
    set  $\xi_v = \mu_v^2 + \nu_v$ 
    if  $[\mu_v, \xi_v] > [\mu_v^{new}, \xi_v^{new}]$  then
       $L_v = L_v \setminus \langle [\mu_v, \nu_v] \rangle$ 
      vector_state = less
    if  $[\mu_v, \xi_v] \leq [\mu_v^{new}, \xi_v^{new}]$  then
      vector_state = greater_or_equal
    break
  if vector_state = less or vector_state = incomparable or  $L_v = \emptyset$  then
     $L_v = L_v \cup \langle [\mu_v^{new}, \nu_v^{new}] \rangle$ 
    relaxation_state = changed

```

Since, we have assumed that the expected values and the second moments of all random variables  $T_e$  for  $e \in E$ , are nonnegative, thus there is no cycle with negative cost in the graph.

We will justify, that for every vertex  $v \in V$ , after  $n$  iterations of the main loop, the list  $L_v$  contains all incomparable vectors related with all possible paths from the source vertex  $s$  to the vertex  $v$ , where  $n$  denotes the number of edges of the most numerous path from the source vertex  $s$  to the vertex  $v$ . In particular, after  $(\#V - 1)$  iterations of the main loop, the list  $L_t$  contains all incomparable vectors related with all possible paths from the source vertex  $s$  to the destination vertex  $t$  (because the number of edges of the most numerous path from the source vertex  $s$  to the destination vertex  $t$  can have  $(\#V - 1)$  edges at most.)

Assume first that  $v = s$ . The list  $L_s$  is initialized with  $\langle [0, 0] \rangle$ . There is only one simple path from the source vertex  $s$  to the vertex  $s$  i.e. the path  $\langle s \rangle$ , and the cost vector of that path is  $[0, 0]$ . Every other path from the source vertex  $s$  to the vertex  $s$  must contain a cycle, thus the cost vector of that path must be greater than or equal to  $[0, 0]$ . Thus, after zero iterations of the main loop, the list  $L_s$  contains all incomparable vectors related with all possible paths from the source vertex  $s$  to the vertex  $s$ .

Suppose now that a vertex  $v \neq s$  is achievable from the source  $s$  and let  $\langle v_0, \dots, v_n \rangle$  be the most numerous path from the source vertex  $s$  to the vertex  $v$  (i.e.  $v_0 = s$  and  $v_n = v$ ). At first iteration of the main loop, the procedure **EBF-Relax**( $v_0, v_1$ ) is executed, hence the list  $L_{v_1}$  contains the vector related with the edge  $(v_0, v_1)$ . At second iteration of the main loop, the procedure **EBF-Relax**( $v_1, v_2$ ) is executed, hence the vector related with the path  $\langle v_0, v_1, v_2 \rangle$  will be taken into consideration on the list  $L_{v_2}$ , i.e. it will be added to the list  $L_{v_2}$ , if it is not bigger than any different vector from this list, etc. Finally, after  $n$  iterations of the main loop, the procedure **EBF-Relax**( $v_{n-1}, v_n$ ) is executed, and the cost vector related with the path  $\langle v_0, \dots, v_n \rangle$  will be taken into

<sup>1</sup>For vectors in representation EV-V. The procedure **EBF-Relax**( $u, v$ ) for the representation EV-SM is similar.

consideration on the list  $L_{v_n}$ . All different simple paths from the source vertex  $s$  to the vertex  $v$  have not more than  $n$  edges, thus all cost vectors related with these paths will be also taken into consideration on the list  $L_v$  after  $n$  iterations of the main loop. All paths from the source vertex  $s$  to the vertex  $v$  having more than  $n$  edges must contain a cycle. But the cost vector of every cycle in the graph is nonnegative, thus the cost vector of a path containing a cycle is greater or equal to the cost vector of the path without cycles. Such a path is simple and as we have justified, the cost vectors related with that path, will be also taken into consideration on the list  $L_v$  after  $n$  iterations of the main loop.

Let us notice, that if the vertex  $v \neq s$  is not achievable from the source  $s$ , then after zero iterations of the main loop, the list  $L_v$  is empty. Since, there is no edge  $(u, v)$  in the graph, the list  $L_v$  will not change in the further iterations of the main loop.

### 3.2 Generic Label Correcting Algorithm

Another way to solve the RSPP with the second moment criterion is to use a Generic Label Correcting algorithm (shortly GLC) that is a kind of the generalization of the classical Dijkstra algorithm. In this algorithm relaxations of edges are coming in a little bit more natural order than in the EBF algorithm. We present a pseudocode of the GLC algorithm below:

**Generic-Label-Correcting-Algorithm:**  
 set  $L_v = \emptyset$  for all  $v \in V \setminus \{s\}$  and  $L_s = \langle [0, 0] \rangle$   
 $Q = \langle s \rangle$   
**while**  $Q \neq \emptyset$  **do**  
     select  $u \in Q$       //typically, with respect to the FIFO principle.  
      $Q = Q \setminus \langle u \rangle$   
     **for all**  $v \in \text{Successors}(u)$  **do**  
         *relaxation\_state = not\_changed*  
         **EBF-Relax** $(u, v)$   
         **if** *relaxation\_state = changed* **and**  $v \notin Q$  **then**  
              $Q = Q \cup \langle v \rangle$   
**return** path related with minimal value on the list  $L_t$

Note that it is possible to move searching of minimal value on the list  $L_t$  to the procedure of relaxation of edge what will save quite a lot of memory.

The GLC algorithm starts with  $Q = \langle s \rangle$  and  $L_s = \langle [0, 0] \rangle$ . As it has been discussed above, the list  $L_s$  contains all incomparable vectors related with all possible paths from the source vertex  $s$  to itself and any further relaxation can not change the contents of this list.

Execution of **EBF-Relax** $(s, v)$  for each  $v \in \text{Successors}(s)$  can change the contents of the list  $L_v$ , and in that case these changes can propagate to the lists related with successors of the vertex  $v$ . Therefore, if execution of **EBF-Relax** $(s, v)$  change the list  $L_v$ , then the vertex  $v$  must be added to the list  $Q$ . Hence cost vectors related with paths going through the vertex  $v$  are also updated. The GLC algorithm works as long as the list  $Q$  is not empty.

Since there is no cycle with negative cost vector, thus the GLC algorithm must stop, and after its execution, every list  $L_v$  contains all incomparable vectors related with all possible paths from the source vertex  $s$  to the vertex  $v$ .

We show how the GLC algorithm works on a simple example.

**Example 3.** Consider the graph depicted in Fig. 1. Vectors of expected values and variances related with edges  $(0,1)$ ,  $(0,2)$ ,  $(1,3)$ ,  $(2,3)$  and  $(3,4)$  are equal to  $[0,0]$ ,  $[0,0]$ ,  $[\frac{3}{4}, \frac{11}{16}]$ ,  $[1,0]$  and  $[1,0]$ , respectively.

- The GLC algorithm starts with  $Q = \langle 0 \rangle$  and  $L_0 = \langle [0,0] \rangle$ , thus at first iteration we obtain  $v = 0$ ,  $Q = \langle 0 \rangle \setminus \langle 0 \rangle = \emptyset$  and  $\text{Successors}(0) = \langle 1, 2 \rangle$ .  
After execution of **EBF-Relax**(0,1) we obtain  $L_1 = \emptyset \cup \langle [0,0] \rangle = \langle [0,0] \rangle$  and  $Q = \emptyset \cup \langle 1 \rangle = \langle 1 \rangle$ .  
After execution of **EBF-Relax**(0,2) we obtain  $L_2 = \emptyset \cup \langle [0,0] \rangle = \langle [0,0] \rangle$  and  $Q = \langle 1 \rangle \cup \langle 2 \rangle = \langle 1, 2 \rangle$ .
- Since  $Q = \langle 1, 2 \rangle \neq \emptyset$ , thus at second iteration we obtain  $v = 1$ ,  $Q = \langle 1, 2 \rangle \setminus \langle 1 \rangle = \langle 2 \rangle$  and  $\text{Successors}(1) = \langle 3 \rangle$ .  
After execution of **EBF-Relax**(1,3) we obtain:  
 $L_3 = \emptyset \cup \langle [\frac{3}{4}, \frac{11}{16}] \rangle = \langle [\frac{3}{4}, \frac{11}{16}] \rangle$  and  $Q = \langle 2 \rangle \cup \langle 3 \rangle = \langle 2, 3 \rangle$ .
- Since  $Q = \langle 2, 3 \rangle \neq \emptyset$ , thus at third iteration we obtain  $v = 2$ ,  $Q = \langle 2, 3 \rangle \setminus \langle 2 \rangle = \langle 3 \rangle$  and  $\text{Successors}(2) = \langle 3 \rangle$ .  
After execution of **EBF-Relax**(2,3) we obtain:  
 $L_3 = \langle [\frac{3}{4}, \frac{11}{16}] \rangle \cup \langle [1,0] \rangle = \langle [\frac{3}{4}, \frac{11}{16}], [1,0] \rangle$  and  $Q = \langle 3 \rangle \cup \langle 3 \rangle = \langle 3 \rangle$ .
- Since  $Q = \langle 3 \rangle \neq \emptyset$ , thus at fourth iteration we obtain  $v = 3$ ,  $Q = \langle 3 \rangle \setminus \langle 3 \rangle = \emptyset$  and  $\text{Successors}(3) = \langle 4 \rangle$ .  
After execution of **EBF-Relax**(3,4) we obtain:  
 $L_4 = \emptyset \cup \langle [1\frac{3}{4}, \frac{11}{16}], [2,0] \rangle = \langle [1\frac{3}{4}, \frac{11}{16}], [2,0] \rangle$  and  $Q = \emptyset \cup \langle 4 \rangle = \langle 4 \rangle$ .
- Since  $Q = \langle 4 \rangle \neq \emptyset$ , thus at fifth iteration we obtain  $v = 4$ ,  $Q = \langle 4 \rangle \setminus \langle 4 \rangle = \emptyset$  and  $\text{Successors}(4) = \emptyset$ .
- Since  $Q = \emptyset$ , thus we check the list  $L_4$  that is  $L_4 = \langle [1\frac{3}{4}, \frac{11}{16}], [2,0] \rangle$ . As we can see,  $[1\frac{3}{4}, \frac{11}{16}]$  is the cost vector related with the shortest path.

### 3.3 Complexity

Unfortunately the time complexity and also the space complexity of the EBF algorithm and the GLC algorithm are exponential. We will show it on a simple example.

**Example 4.** Consider the graph depicted in Fig. 3(a) and let random variables related with all edges of the graph have vectors in the representation  $EV-V$  presented in the picture. Let

$$[0,0], [2^{\alpha_1}, 2^{2n+1-\alpha_1}], [0,0], [2^{\alpha_2}, 2^{2n+1-\alpha_2}], \dots, [0,0], [2^{\alpha_n}, 2^{2n+1-\alpha_n}] \quad (11)$$

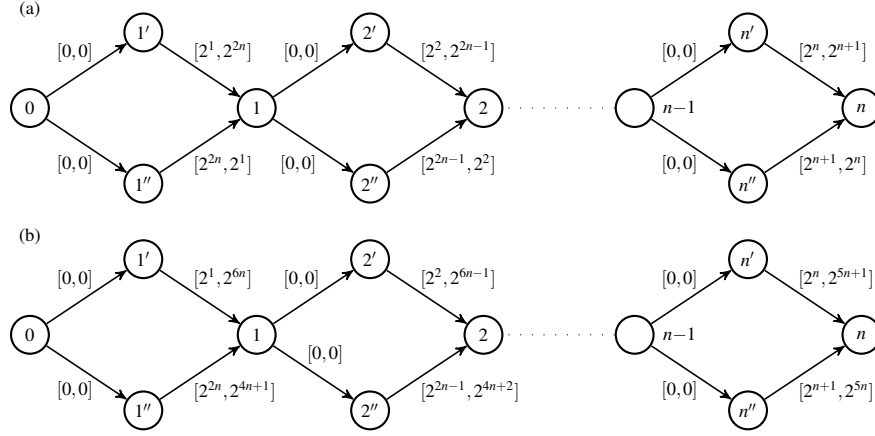


Figure 3: (a) Graph with  $3n + 1$  vertices and  $4n$  edges and cost vectors in representation EV-V. (Sums of exponents power of coordinates in all non-zero vectors in the graph are equal to  $2n + 1$ .) All of  $2^n$  paths in this graph are incomparable. (b) Graph with  $3n + 1$  vertices and  $4n$  edges and cost vectors in representation EV-SM. (Sums of exponents power of coordinates in all non-zero vectors in the graph are equal to  $6n + 1$ .) All of  $2^n$  paths in this graph are incomparable.

be vectors which coordinates are the expected values and the variances, related with the edges of a path from the source vertex 0 to the destination vertex  $n$ . The sum  $\sum_{i=1}^n 2^{\alpha_i}$  can be expressed as binary number  $(a_{2n}, a_{2n-1}, \dots, a_2, a_1, 0)_2$ . This binary number contains ones at positions  $\alpha_1, \dots, \alpha_n$  and zeros at the other positions, thus all of  $2^n$  possible cost vectors related with all possible paths from the source vertex 0 to the destination vertex  $n$  must be different. Furthermore, if we order these cost vectors increasingly by the expected values, then variances of these cost vectors will be ordered decreasing. Therefore all  $2^n$  possible cost vectors related with all possible paths from the source 0 to the destination  $n$  are incomparable.

Now, consider the graph depicted in Fig. 3(b) and let random variables related with all edges of the graph have vectors in the representation EV-SM presented in the picture. Let

$$[0, 0], [2^{\alpha_1}, 2^{6n+1-\alpha_1}], [0, 0], [2^{\alpha_2}, 2^{6n+1-\alpha_2}], \dots, [0, 0], [2^{\alpha_n}, 2^{6n+1-\alpha_n}] \quad (12)$$

be vectors whose coordinates are the expected values and the second moments, related with edges of some path from the source vertex 0 to the destination vertex  $n$ . From equation (7) it follows that the vector related with that path is equal to

$$\left[ \sum_{i=1}^n 2^{\alpha_i}, \sum_{i=1}^n 2^{6n+1-\alpha_i} + 2 \sum_{i=2}^n 2^{\alpha_i} \sum_{j=1}^{i-1} 2^{\alpha_j} \right]. \quad (13)$$

Similarly to the case of graph from Fig. 3(a), we order cost vectors of all  $2^n$  paths

increasingly by the expected values, and the factors  $\sum_{i=1}^n 2^{6n+1-\alpha_i}$  of cost vectors of these paths will be ordered decreasing.

Note that the minimal difference of the sum of the second coordinates of cost vectors of two different paths from the source vertex 0 to the destination vertex  $n$  is equal to  $2^{5n}$ .

Moreover, the maximal value of the factor  $2 \sum_{i=2}^n 2^{\alpha_i} \sum_{j=1}^{i-1} 2^{\alpha_j}$  can be estimated as follows:

$$\begin{aligned} 2 \sum_{i=2}^n 2^{\alpha_i} \sum_{j=1}^{i-1} 2^{\alpha_j} &\leq 2 \sum_{i=2}^n 2^{n+i} \sum_{j=1}^{i-1} 2^{n+j} = 2 \sum_{i=2}^n 2^{n+i} \cdot 2^{n+i} = 2^{2n+1} \sum_{i=2}^n 4^i = \\ &= 2^{2n+1} \cdot \frac{4^{n+1} - 4^2}{4 - 1} < 2^{2n+1} \cdot \frac{4^{n+1}}{2} = 2^{4n+2} < 2^{5n} \end{aligned} \quad (14)$$

for any  $n > 2$ . It is easy to see that an estimation (14) holds also for  $n = 1$  (then  $0 < 2^5$ ) and for  $n = 2$  (then  $2 \cdot 2^3 \cdot 2^4 < 2^{10}$ ). Inequality (14) guarantees that order of the second coordinates of cost vectors of all paths will be preserved. Hence it follows that all vectors related with all possible paths from the source vertex 0 to the destination vertex  $n$  are incomparable.

Graphs depicted in Fig. 3(a) and Fig. 3(b) show that the pessimistic complexity of the EBF and the GLC algorithm are exponential, regardless of the data representation EV-V or EV-SM.

## 4 Approximate Algorithms

As we have shown in the previous section, the time complexity and the space complexity of the presented exact algorithms are exponential. Therefore, in this section we consider approximate algorithms, which have polynomial complexity. The first presented algorithm is a kind of modification of the classical Bellman-Ford algorithm. The second one is a simple single criterion approximation, and the last one is a variant of an approximate algorithm for the MOSPP from [15].

### 4.1 Extended Bellman-Ford Algorithm with Fixed Capacity

The classical Bellman-Ford algorithm fails in the case of the RSPP, because in this algorithm, each vertex is connected with only one currently the best cost related with the shortest path to this vertex. The EBF algorithm solving the MOSPP presented in the previous section can be used in the case of the RSPP, because it assigns with each vertex the list of all nondominated cost vectors related with all paths to this vertex. Unfortunately, as we have shown in the previous section this algorithm has an exponential complexity. It is natural to consider a modification of the classical Bellman-Ford algorithm which assigns with each vertex the list of fixed number  $k$  of cost vectors. Obviously, such an algorithm is only an approximate algorithm. It is easy to extend the graph shown in Fig. 1, in such a way that the worst subpath will be a subpath of the shortest path.

We present a pseudocode of this modification of the Extended Bellman-Ford algorithm below:

**Extended-Bellman-Ford-Algorithm-with-Fixed-Capacity:**

set  $L_v = \emptyset$  for all  $v \in V \setminus \{s\}$  and  $L_s = \langle [0, 0] \rangle$

**for**  $i = 1$  **to**  $\#V - 1$  **do**

$relaxation\_state = not\_changed$

**for all**  $(u, v) \in E$  **do**

**EBF-FC-Relax** $(u, v)$

**if**  $relaxation\_state = not\_changed$  **then**

**break**

**return** path related with  $L_t$

The procedure **EBF-FC-Relax** $(u, v)$  has the following pseudocode<sup>2</sup>:

**EBF-FC-Relax** $(u, v)$ :

**for all**  $[\mu_u, \xi_u] \in L_u$  **do**

set  $\mu_v^{new} = \mu_u + \mathcal{E}[T_{(u,v)}]$  and  $\xi_v^{new} = \xi_u + \mathcal{E}[T_{(u,v)}^2] + 2\mu_u \mathcal{E}[T_{(u,v)}]$

**if**  $\#L_v < k$  **then**

$L_v = L_v \cup \langle [\mu_v^{new}, \xi_v^{new}] \rangle$

$relaxation\_state = changed$

**else**

find  $[\mu_v, \xi_v] \in L_v$  such that  $\xi_v = \max\{\xi \mid [\mu, \xi] \in L_v\}$ .

**if**  $\xi_v > \xi_v^{new}$  **or**  $(\xi_v = \xi_v^{new}$  **and**  $\mu_v > \mu_v^{new})$  **then**

$L_v = L_v \setminus \langle [\mu_v, \xi_v] \rangle \cup \langle [\mu_v^{new}, \xi_v^{new}] \rangle$

$relaxation\_state = changed$

We show how the Extended Bellman-Ford Algorithm with Fixed Capacity (shortly EBF-FC) works on a simple example.

**Example 5.** Consider the graph depicted in Fig. 1. Suppose that the edges are ordered as follows:  $(0, 1)$ ,  $(0, 2)$ ,  $(1, 3)$ ,  $(2, 3)$  and  $(3, 4)$ . Vectors of expected values and the second moments related with these edges are equal to  $[0, 0]$ ,  $[0, 0]$ ,  $[\frac{3}{4}, \frac{5}{4}]$ ,  $[1, 0]$  and  $[1, 0]$  respectively. Assume first that  $k = 1$  (conf. the pseudocode).

- The EBF-FC algorithm starts with  $L_0 = \langle [0, 0] \rangle$  and  $L_1 = L_2 = L_3 = L_4 = \emptyset$ .
- After execution of **EBF-FC-Relax** $(0, 1)$  we obtain  $L_1 = \langle [0+0, 0+0+2 \cdot 0 \cdot 0] \rangle = \langle [0, 0] \rangle$ .
- After execution of **EBF-FC-Relax** $(0, 2)$  we obtain  $L_2 = \langle [0+0, 0+0+2 \cdot 0 \cdot 0] \rangle = \langle [0, 0] \rangle$ .
- After execution of **EBF-FC-Relax** $(1, 3)$  we obtain  $L_3 = \langle [0+\frac{3}{4}, 0+\frac{5}{4}+2 \cdot 0 \cdot \frac{3}{4}] \rangle = \langle [\frac{3}{4}, \frac{5}{4}] \rangle$ .

<sup>2</sup>For vectors in representation EV-SM. The procedure **EBF-FC-Relax** $(u, v)$  for the representation EV-V is similar.

- Since  $k = 1$ ,  $[0+1, 0+1+2 \cdot 0 \cdot 1] = [1, 1]$  and  $\frac{5}{4} > 1$ , thus we have  $L_3 = \langle [\frac{3}{4}, \frac{5}{4}] \rangle \setminus \langle [\frac{3}{4}, \frac{5}{4}] \rangle \cup \langle [1, 1] \rangle = \langle [1, 1] \rangle$  after execution of **EBF-FC-Relax**(2, 3).
- After execution of **EBF-FC-Relax**(3, 4) we obtain  $L_4 = \langle [1+1, 1+1+2 \cdot 1 \cdot 1] \rangle = \langle [2, 4] \rangle$ .
- Nothing is changing after the next iteration of the main loop, thus  $[2, 4]$  is the cost vector related with the returned path. Let us note that we have shown in Example 1 that the cost of the shortest path is not equal to 4, but it is equal to  $3\frac{3}{4}$ .

Suppose now that  $k = 2$ , i.e. we take into account the Lists  $L_v$  for  $v \in V$  with the capacity of 2 cost vectors.

- The EBF algorithm starts with  $L_0 = \langle [0, 0] \rangle$  and  $L_1 = L_2 = L_3 = L_4 = \emptyset$ .
- After execution of **EBF-FC-Relax**(0, 1) we obtain  $L_1 = \langle [0+0, 0+0+2 \cdot 0 \cdot 0] \rangle = \langle [0, 0] \rangle$ .
- After execution of **EBF-FC-Relax**(0, 2) we obtain  $L_2 = \langle [0+0, 0+0+2 \cdot 0 \cdot 0] \rangle = \langle [0, 0] \rangle$ .
- After execution of **EBF-FC-Relax**(1, 3) we obtain  $L_3 = \langle [0+\frac{3}{4}, 0+\frac{5}{4}+2 \cdot 0 \cdot \frac{3}{4}] \rangle = \langle [\frac{3}{4}, \frac{5}{4}] \rangle$ .
- Since  $k = 2$ , thus we obtain  $L_3 = \langle [\frac{3}{4}, \frac{5}{4}] \rangle \cup \langle [0+1, 0+1+2 \cdot 0 \cdot 1] \rangle = \langle [\frac{3}{4}, \frac{5}{4}], [1, 1] \rangle$  after execution of **EBF-FC-Relax**(2, 3).
- Since  $[\frac{3}{4}+1, \frac{5}{4}+1+2 \cdot \frac{3}{4} \cdot 1] = [1\frac{3}{4}, 3\frac{3}{4}]$ ,  $[1+1, 1+1+2 \cdot 1 \cdot 1] = [2, 4]$  and  $[1\frac{3}{4}, 3\frac{3}{4}] < [2, 4]$ , thus after execution **EBF-FC-Relax**(3, 4) we obtain  $L_4 = \langle [1\frac{3}{4}, 3\frac{3}{4}] \rangle$ .
- Nothing is changing after the next iteration of the main loop, thus  $[1\frac{3}{4}, 3\frac{3}{4}]$  is the cost vector related with the returned path. As we have shown in Example 1, this is the cost vector of the shortest path.

The above example shows that the EBF-FC algorithms is not an exact algorithm.

Note that the complexity of the EBF-FC algorithm is polynomial. Moreover, it is equal to  $O(k \cdot \#V \cdot \#E)$ , because the main loop can take at most  $\#V \cdot \#E$  iterations and each iteration is a simple execution of the **EBF-FC-Relax** procedure which takes at most  $k$  operations (list  $L_u$  can have at most  $k$  vectors).

## 4.2 Extended Bellman-Ford Algorithm with Smart Indexing

The second proposed approximate algorithm is a variant of the algorithm presented in [15]. It is another modification of Bellman-Ford algorithm.

Let  $\mu_v^{min} = \mu_v^1 < \dots < \mu_v^r = \mu_v^{max}$  be all possible expected values related with paths from the source vertex  $s$  to some vertex  $v \in V$ . (Note that  $\mu_v^i$  can be related with many different paths.) Suppose that the currently the best path from the source vertex  $s$  to the vertex  $v$  has the expected value equal to  $\mu_v^i$ . If we store the second moment (or

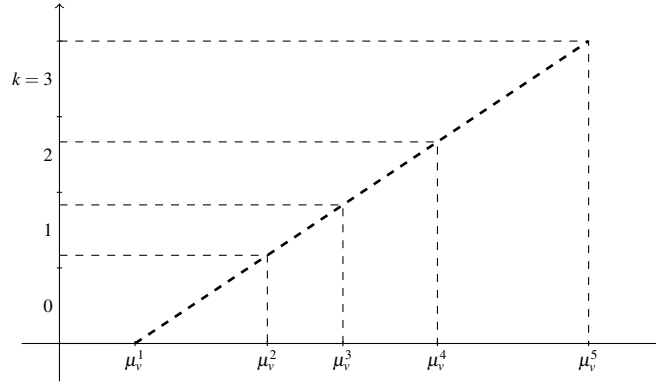


Figure 4: Illustration of the formula (15).

variance) of that path at  $i$ -th position in array of  $k$  elements, then we obtain an exact algorithm. Of course this algorithm is very "expensive" because we have to calculate position  $i$  and all possible values  $\mu_v^1, \dots, \mu_v^r$ . Since the number  $r$  of such values grows exponentially with respect to the distance between the source vertex  $s$  and the vertex  $v$ , we have to remember exponentially growing lists of values  $\mu_v^1, \dots, \mu_v^r$ .

Notice, that for each vertex  $v$  we can simply estimate (or even calculate) minimal and maximal expected values  $\mu_v^{min}$  and  $\mu_v^{max}$ . If we fix the size of lists related with each vertices to  $k$ , then we can store second moment (or variance) of that path at the position

$$\left[ k \cdot \frac{\mu_v^{new} - \mu_v^{min}}{\mu_v^{max} - \mu_v^{min}} \right] \quad (15)$$

if its expected value is equal to  $\mu_v^{new}$ . (For small  $k$  the variance or the second moment related with different paths will be written in the same position of the array more frequently.)

#### **Extended-Bellman-Ford-Algorithm-with-Smart-Indexing:**

use classical Bellman-Ford algorithm or classical Dijkstra algorithm to calculate minimal and maximal expected values  $\mu_v^{min}$  and  $\mu_v^{max}$  for all  $v \in V$

set  $\Pi_v^i = \langle \infty, \infty, null \rangle$  for all  $v \in V$ ,  $i = 0, \dots, k$  and  $\Pi_s^0 = \langle 0, 0, s \rangle$

**for**  $i = 1$  **to**  $\#V - 1$  **do**

$relaxation\_state = not\_changed$

**for all**  $(u, v) \in E$  **do**

**EBF-SI-Relax** $(u, v)$

**if**  $relaxation\_state = not\_changed$  **then**

**break**

**return** path related with  $\Pi_i$

The procedure **EBF-SI-Relax**( $u, v$ ) has the following pseudocode<sup>3</sup>:

**EBF-SI-Relax**( $u, v$ ):  
**for**  $i = 0$  **to**  $k$  **do**  
  let  $\langle \mu_u^i, \xi_u^i, \pi_u^i \rangle = \Pi_u^i$   
  **if**  $\pi_u^i \neq t$  **then**  
     $\mu_v^{new} = \mu_u^i + \mathcal{E}[T_{(u,v)}]$   
     $\xi_v^{new} = \xi_u^i + \mathcal{E}[T_{(u,v)}^2] + 2\mu_u^i \mathcal{E}[T_{(u,v)}]$   
    **if**  $\mu_v^{min} \neq \mu_v^{max}$  **then**  
       $j = \lceil k \cdot \frac{\mu_v^{new} - \mu_v^{min}}{\mu_v^{max} - \mu_v^{min}} \rceil$   
    **else**  
       $j = 0$   
    let  $\langle \mu_v^j, \xi_v^j, \pi_v^j \rangle = \Pi_v^j$   
    **if**  $\pi_v^j = \text{null}$  **or**  $\xi_v^j > \xi_v^{new}$  **then**  
       $\Pi_v^j = \langle \mu_v^{new}, \xi_v^{new}, u \rangle$   
       $\text{relaxation\_state} = \text{changed}$

To illustrate the performance of the above algorithm we present a simple example.

**Example 6.** Consider the graph depicted in Fig. 1. Let  $k = 1$ , i.e.  $k + 1 = 2$  is the array dimension.

- Calculating minimal and maximal expected values we obtain  $\mu_0^{min} = \mu_0^{max} = 0$ ,  $\mu_1^{min} = \mu_1^{max} = 0$ ,  $\mu_2^{min} = \mu_2^{max} = 0$ ,  $\mu_3^{min} = \frac{3}{4}$ ,  $\mu_3^{max} = 1$ ,  $\mu_4^{min} = 1\frac{3}{4}$  and  $\mu_4^{max} = 2$ .
- Since  $\mu_0^{min} = \mu_0^{max} = 0$ , thus after execution **EBF-SI-Relax**(0, 1) we obtain  $\Pi_1^0 = \langle 0, 0, 0 \rangle$ .
- Similarly, after execution of **EBF-SI-Relax**(0, 2) we obtain  $\Pi_2^0 = \langle 0, 0, 0 \rangle$ .
- After execution of **EBF-SI-Relax**(1, 3) we obtain  $\mu_v^{new} = \frac{3}{4}$ ,  $\xi_v^{new} = \frac{5}{4}$  and  $j = 0$ , thus  $\Pi_3^0 = \langle \frac{3}{4}, \frac{5}{4}, 1 \rangle$ .
- After execution of **EBF-SI-Relax**(2, 3) we obtain  $\mu_v^{new} = 1$ ,  $\xi_v^{new} = 1$  and  $j = 1$ , thus  $\Pi_3^0 = \langle 1, 1, 2 \rangle$ .
- After execution of **EBF-SI-Relax**(3, 4), for  $i = 0$ , we obtain  $\mu_v^{new} = 1\frac{3}{4}$ ,  $\xi_v^{new} = 3\frac{3}{4}$  and  $j = 0$ , thus  $\Pi_3^0 = \langle 1\frac{3}{4}, 3\frac{3}{4}, 3 \rangle$ . For  $i = 1$  we have  $\mu_v^{new} = 2$ ,  $\xi_v^{new} = 4$  and  $j = 1$ , thus  $\Pi_3^1 = \langle 2, 4, 3 \rangle$ .
- Nothing is changing after the next iteration of the main loop, thus  $[1\frac{3}{4}, 3\frac{3}{4}]$  is the cost vector related with the returned path. As we have shown in Example 1, this is the cost vector of the shortest path.

<sup>3</sup>For vectors in representation EV-SM. The procedure **EBF-SI-Relax**( $u, v$ ) for the representation EV-V is similar.

Note that the complexity of the above algorithm is polynomial. Moreover, it is equal to  $O(k \cdot \#V \cdot \#E)$ , because the main loop can take at most  $\#V \cdot \#E$  iterations and each iteration is a simple execution of the **EBF-SI-Relax** procedure which took at most  $k$  operations.

### 4.3 Extended Bellman-Ford Algorithm with Rounded Values

The next presented approximate algorithm is a simple modification of the EBF algorithm in which we round values of the vectors before the comparison. We call this algorithm Extended Bellman-Ford Algorithm with Rounded Values (shortly EBF-RV). We present a pseudocode of the EBF-RV algorithm below:

**Extended-Bellman-Ford-Algorithm-with-Rounded-Values:**

use classical Bellman-Ford algorithm or classical Dijkstra algorithm to calculate/estimate minimal and maximal expected values and variances  $\mu_v^{min}$ ,  $\mu_v^{max}$ ,  $v_v^{min}$  and  $v_v^{max}$  for all  $v \in V$

set  $\xi_v^{min} = (\mu_v^{min})^2 + v_v^{min}$  and  $\xi_v^{max} = (\mu_v^{max})^2 + v_v^{max}$  for all  $v \in V$

set  $L_s = \langle [0, 0] \rangle$  and  $L_v = \emptyset$  for all  $v \in V \setminus \{s\}$

**for**  $i = 1$  **to**  $\#V - 1$  **do**

*relaxation\_state* = *not\_changed*

**for all**  $(u, v) \in E$  **do**

**EBF-RV-Relax** $(u, v)$

**if** *relaxation\_state* = *not\_changed* **then**

**break**

**return** path related with minimal value on the list  $L_t$

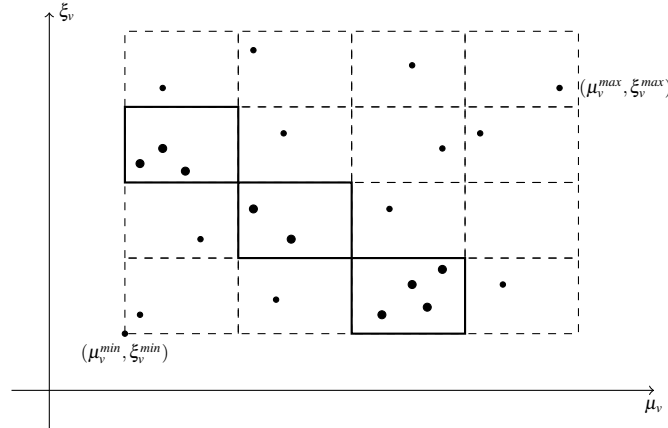


Figure 5: Illustration of the idea of the EBF-RV algorithm. All the vectors in the highlighted rectangle will be treated as the same vector by the EBF-RV algorithm.

The procedure **EBF-RV-Relax**( $u, v$ ) has the following pseudocode<sup>4</sup>:

**EBF-RV-Relax**( $u, v$ ):  
**for all**  $[\mu_u, \xi_u] \in L_u$  **do**  
  set  $\mu_v^{new} = \mu_u + \mathcal{E}[T_{(u,v)}]$  and  $\xi_v^{new} = \xi_u + \mathcal{E}[T_{(u,v)}^2] + 2\mu_u \mathcal{E}[T_{(u,v)}]$   
  set  $[\widehat{\mu}_v^{new}, \widehat{\xi}_v^{new}] = \left[ \left\lceil k \cdot \frac{\mu_v^{new} - \mu_v^{min}}{\mu_v^{max} - \mu_v^{min}} \right\rceil, \left\lceil k \cdot \frac{\xi_v^{new} - \xi_v^{min}}{\xi_v^{max} - \xi_v^{min}} \right\rceil \right]$   
  *vector\_state = incomparable*  
  **for all**  $[\mu_v, \xi_v] \in L_v$  **do**  
    set  $[\widehat{\mu}_v, \widehat{\xi}_v] = \left[ \left\lceil k \cdot \frac{\mu_v - \mu_v^{min}}{\mu_v^{max} - \mu_v^{min}} \right\rceil, \left\lceil k \cdot \frac{\xi_v - \xi_v^{min}}{\xi_v^{max} - \xi_v^{min}} \right\rceil \right]$   
    **if**  $[\widehat{\mu}_v, \widehat{\xi}_v] > [\widehat{\mu}_v^{new}, \widehat{\xi}_v^{new}]$  **then**  
       $L_v = L_v \setminus \langle [\mu_v, \xi_v] \rangle$   
      *vector\_state = less*  
    **if**  $[\widehat{\mu}_v, \widehat{\xi}_v] \leq [\widehat{\mu}_v^{new}, \widehat{\xi}_v^{new}]$  **then**  
      *vector\_state = greater\_or\_equal*  
      **break**  
  **if** *vector\_state = less* **or** *vector\_state = incomparable* **or**  $L_u = \emptyset$  **then**  
     $L_v = L_v \cup \langle [\mu_v^{new}, \xi_v^{new}] \rangle$   
    *relaxation\_state = changed*

Note that the complexity of the above algorithm is exponential.

#### 4.4 Single Criterion Approximation Algorithm

As it was mentioned in the Introduction, if we take as the shortest path, the path with the minimal expected value or the variance of the sum of the random variables related with edges of the path, then the RSPP reduces to the CSPP and can be solved with help of the classical Dijkstra algorithm or classical Bellman-Ford algorithm.

This natural observation can be used in the whole series of approximate algorithms.

We can forget for a moment about one of the criteria (expected value or variance) and calculate the shortest path with respect to the other criterion. Next, we can remove from the graph the edge which is the most damaging for the first criterion or indeed cost of the path, and repeat the procedure on the depreciated graph.

If there is no path from the source vertex  $s$  to the destination vertex  $t$  in the depreciated graph, then the algorithm stops and returns this from examined path which has minimal actual cost (second moment).

<sup>4</sup> For vectors in representation EV-SM. The procedure **EBF-RV-Relax**( $u, v$ ) for the representation EV-V is similar.

**Single-Criterion-Approximation-algorithm:**

**while** exists a path from the source vertex  $s$  to the destination vertex  $t$  **do**  
  use classical Bellman-Ford algorithm or classical Dijkstra algorithm to find the  
  shortest path  $p = \langle v_0, \dots, v_n \rangle \in \mathcal{P}$  with respect to expected value  
  //or variance, or the second moment  
  find  $(v_{j-1}, v_j) \in p$  such that  $\mathcal{V}[T_{(v_{j-1}, v_j)}] = \max_{i=1, \dots, n} \mathcal{V}[T_{(v_{i-1}, v_i)}]$   
  //or  $\mathcal{E}[T_{(v_{j-1}, v_j)}] = \max_{i=1, \dots, n} \mathcal{E}[T_{(v_{i-1}, v_i)}]$   
  //or  $\mathcal{E}[T_{(v_{j-1}, v_j)}^2] + \sum_{\substack{k=1 \\ k \neq j}}^n \mathcal{E}[T_{(v_{j-1}, v_j)}] \mathcal{E}[T_{(v_{k-1}, v_k)}] =$   
 $\max_{i=1, \dots, n} \mathcal{E}[T_{(v_{i-1}, v_i)}^2] + \sum_{\substack{k=1 \\ k \neq i}}^n \mathcal{E}[T_{(v_{i-1}, v_i)}] \mathcal{E}[T_{(v_{k-1}, v_k)}]$   
   $E = E \setminus \{(v_{j-1}, v_j)\}$   
  **if** the path  $p$  have cost lower than the path  $b$  **or**  $b$  is undefined **then**  
   $b = p$   
**return** path  $b$

To illustrate the performance of the Single Criterion Approximation algorithm (shortly SCA) we present the case of searching of the shortest path with respect to the variance and removing edge with the greatest expected value.

**Example 7.** Consider the graph depicted in Fig. 1.

- The shortest path with respect to the variance is the path  $p = \langle 0, 2, 3, 4 \rangle$ . Since  $\mathcal{E}[T_{(0,2)}] = 0$ ,  $\mathcal{E}[T_{(2,3)}] = 1$  and  $\mathcal{E}[T_{(3,4)}] = 1$ , thus (for example) we have  $E = E \setminus \{(2,3)\}$ . The path  $b$  is undefined, thus  $b = p = \langle 0, 2, 3, 4 \rangle$ .
- The shortest path with respect of expected value is now the path  $p = \langle 0, 1, 2, 4 \rangle$ . Since  $\mathcal{E}[T_{(0,1)}] = 0$ ,  $\mathcal{E}[T_{(1,3)}] = \frac{3}{4}$  and  $\mathcal{E}[T_{(3,4)}] = 1$ , thus we have  $E = E \setminus \{(3,4)\}$ . The cost of the path  $p$  is lower that the cost of the path  $b$ , thus  $b = p = \langle 0, 1, 3, 4 \rangle$ .
- There is no path from the source verice 0 to the destination vertex 4, therefore the algorithm returns the path  $b = \langle 0, 1, 3, 4 \rangle$ . Note that we have shown in Example 1 that  $b$  is the shortest path.

Note that the complexity of the SCA algorithm is polynomial. Moreover, it is equal to  $O(\#E^2 \cdot \#V)$ , because the main loop can take at most  $\#E$  iterations (if  $E = \emptyset$  then the main loop must finish) and the classical Bellman-Ford algorithm or the classical Dijkstra algorithm have the complexity equal to  $O(\#V \cdot \#E)$  and the finding of the edge  $(v_{j-1}, v_j)$  can take at most  $(\#V - 1)$  operations (path  $p$  must be simple, thus it can contain at most  $(\#V - 1)$  edges).

In the next section we presents computational results of tests of the proposed algorithms.

	EBF	EBF-FC-2	EBF-FC-5	EBF-RV-50	EBF-SI-20	EBF-SI-50	SCA-22	SCA-2E	SCA-2V	SCA-E2	SCA-EE	SCA-EV	SCA-V2	SCA-VE	SCA-VV
$t_1$	1.000	<b>0.073</b>	<b>0.293</b>	<b>0.191</b>	<b>0.415</b>	<i>1.144</i>	0.136	0.082	0.114	0.081	0.101	0.041	0.096	0.069	0.080
$t_2$	1.000	<b>0.066</b>	<b>0.250</b>	<b>0.109</b>	<b>0.357</b>	<i>0.911</i>	0.936	2.086	0.480	2.326	0.431	1.872	0.533	2.012	0.470
$t_3$	1.000	<b>0.097</b>	<b>0.320</b>	<b>0.102</b>	<b>0.510</b>	<i>1.081</i>	—	—	—	—	—	—	—	—	—
$r_1$	1.000	<b>1.000</b>	<b>1.002</b>	<i>7.219</i>	<b>1.000</b>	<b>1.002</b>	2.321	2.321	2.321	2.321	2.398	2.437	2.321	2.321	2.321
$r_2$	1.000	<b>1.000</b>	<b>1.000</b>	<i>9.797</i>	<b>1.000</b>	<b>1.000</b>	2.179	2.179	2.179	2.480	2.845	2.574	2.179	2.179	2.179
$r_3$	1.000	<b>1.000</b>	<b>1.000</b>	<i>12.283</i>	<b>1.000</b>	<b>1.000</b>	—	—	—	—	—	—	—	—	—

Table 1: Comparison of the time of execution and precision of the presented algorithms.  $t_1, t_2, t_3$  — average factors of times of execution of given algorithms for three different groups of graphs;  $r_1, r_2, r_3$  — average factors of precision of given algorithms for three different groups of graphs; Specification of the  $i$ -th group of graphs (for  $i = 1, 2, 3$ ):  $\#V = 10000i$ ,  $\#Successors(v) = 10i$  for all vertex  $v \in V \setminus \{t\}$ .  $0 \leq \mathcal{E}[T_e] \leq 10$  and  $0 \leq \mathcal{E}[T_e^2] \leq 5100$  for all edge  $e \in E$ .

## 5 Computational Results

We performed our tests on quite large randomly generated graphs. All these tests were made on Intel Celeron Mobile 1400MHz with 768MB RAM, working under control of Linux operating system with kernel version 2.6.11-6. Table 1 presents factors of times of execution and factors of precision of given algorithms for three different groups of graphs. For example, for the first group of graphs, the EBF-FC-2 algorithm took average 0.073 of time of execution of the exact algorithm (EBF). Every group contained 10 graphs. The  $i$ -th group of graphs (for  $i = 1, 2, 3$ ) had  $\#V = 10000i$ ,  $\#Successors(v) = 10i$  for all vertex  $v \in V \setminus \{t\}$ .  $0 \leq \mathcal{E}[T_e] \leq 10$  and  $0 \leq \mathcal{E}[T_e^2] \leq 5100$  for all edge  $e \in E$ . Results better than the results of algorithm EBF are in bold, and worse results are highlighted in italics.

## 6 Conclusions

The Random Shortest Path Problem with the second moment criterion was discussed in this paper. It was shown that the complexity of the presented exact algorithms are exponential, while the complexity of the presented approximated algorithms is only polynomial.

Computational results for the exact and approximate algorithms were performed on large graphs. The results show that the approximate algorithms works very well, especially the EBF-FC algorithm which works really fast and produces exact results usually.

## References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Inc., Englewood Cliffs (1993)
2. Alexopoulos, Ch.: State Space Partitioning Methods for Stochastic Shortest Path Problems. *Networks* 30, 9-21 (1997)
3. Bellman, R.: On a routing problem, *Quarterly of Applied Mathematics* 16, 87-90 (1958)
4. Cormen, T.H., Leiserson, Ch.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge/McGraw-Hill Book Company, Boston (1990).
5. Dijkstra, E.W.: A note on two problems in connection with graphs. *Nuerische Mathematik* 1, 269-271 (1959)
6. Floyd, R.W.: Algorithm 97: Shortest path, *Comm. ACM* 5, 345 (1962)
7. Frederikson, G.N.: Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.* 16, 1004-1022 (1987)
8. Hansen, P.: Bicriterion Path Problems. In: Proc. 3rd Conf. Multiple Criteria Decision Making — Theory and Applications, Lecture Notes in Economics and Mathematical Systems 117, pp. 109-127, Springer, Heidelberg (1979)
9. Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM* 24, 1-13 (1977)
10. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their use in improved network optimization problems. *Journal of the ACM* 34, 596-615 (1987)
11. Moffat, A., Takaoka, T.: An all pairs shortest path algorithm with expected time  $O(n^2 \log n)$ . *SIAM J. Comput.* 16, 1023-1031 (1987)
12. Murthy, I., Sakar, S.: A Relaxation-Based Pruning Technique for a Class of Stochastic Shortest Path Problems. *Transportation Science* 30, 220-236 (1996)
13. Murthy, I., Sarkar, S.: Exact algorithms for the stochastic shortest path problem with a decreasing deadline utility function. *European Journal of Operational Research* 103, 209-229 (1997)
14. Skriver, A.J.V., Anderson, K.A.: A label correcting approach for solving bi-criterion shortest-path problems. *Computers & Operations Research* 27, 50-524 (2000)
15. Tsaggouris, G., Zaroliagis, Ch.D.: Improved FPTAS for Multiobjective Shortest Paths with Applications. Technical report no. TR 20050703 (July 2005).
16. Tsaggouris, G., Zaroliagis, Ch.D.: Multiobjective Optization: Improved FPTAS for Shortest Paths and Non-linear Objectives with Applications. Technical report no. TR 2006/03/01 (March 2006).