

Solving Stochastic Path Problem: Particle Swarm Optimization Approach

Saeedeh Momtazi¹, Somayeh Kafi¹, and Hamid Beigy^{1,2}

¹ Computer Engineering Department, Sharif University of Technology, Tehran, Iran

² Institutes for Studies in Theoretical Physics and Mathematics (IPM), School of Computer Science, Tehran, Iran

{momtazi, kafi, beigy}@ce.sharif.edu

Abstract. An stochastic version of the classical shortest path problem whereby for each node of a graph, a probability distribution over the set of successor nodes must be chosen so as to reach a certain destination node with minimum expected cost. In this paper, we propose a new algorithm based on Particle Swarm Optimization (PSO) for solving Stochastic Shortest Path Problem (SSPP). The comparison of our algorithm with other algorithms indicates that its performance is suitable even by the less number of iterations.

Keywords: Particle Swarm Optimization, Stochastic Shortest Path Problem, Swarm Intelligence.

1 Introduction

Given a directed graph $G = (V; E)$, where $V = \{1, 2, \dots, n\}$ denotes the set of nodes and $E \subset V \times V$ specifies the set of edges, the deterministic shortest path problem is to select at each node j , a successor node $\mu(j)$ so that $(j, \mu(j))$ is an edge, and the path formed by a sequence of successor nodes starting node v_s as source node and terminates at destination node v_d has minimum cost (i.e. minimum sum of edge costs), over all paths that start at v_s and terminate at v_d .

The stochastic shortest path problem (SSPP) is a generalization of classic shortest path problem whereby at each node, we must select a probability distribution over all possible successor nodes, out of a given set of probability distributions. The stochastic graph G can be defined by a triple $G = (V; E; Q)$, where $n \times n$ matrix Q is the probability distribution describing the statistics of edge costs. In particular, the cost C_{ij} of edge (i, j) is assumed to be a random variable with q_{ij} as its probability density function. For a given edges' cost distributions and for a given source node, the path traversed as well as its cost are now random, but we wish that the path leads to destination node v_d with probability close to one and has minimum expected cost.

Solving the shortest path problem for deterministic graphs has an extensive background and its results have been found to be a very useful tool in a great variety of contexts [1]. By presence of uncertainties in many applications such as link failure and variable travel time due to the congestion in a variety of distributed systems, the stochastic graphs are known as a more realistic model.

Different classes of stochastic shortest path problems are considered in the literature. The first work known in this area is due to Frank [2] where the shortest path over a probabilistic graph is determined. The most common criterion to determine the shortest path is the one that optimizes the expected value of a utility function. This criterion stems from the formulation presented by Von-Neuman-Morgenstern where evaluations should be made under uncertainty [3]. Algorithms for linear and quadratic utility functions have been presented by Mirchandini and Soroush [3], Murthy and Sarkar [4], [5] and more recently by Deolinda Dias Rasteiro, Antonio Batel Anjo [6].

Ishwar Murthy and Sumit Sarkar [5] considered a stochastic shortest path problem which the costs of edges are independent random variables following a normal distribution. In this problem, the optimal path is one that maximizes the expected utility, where the utility function being piecewise linear and concave. Computational testing is done to evaluate the performance of their algorithms. Overall, their algorithms are very effective in solving large problems quickly.

Beigy and Meybodi introduce two algorithms based on distributed learning automata (DLA) for solving SSPP [7,8]. In the former [7], DLA is introduced and applied to the shortest path problem and in the latter [8], they introduce another DLA based algorithm which is faster than it. It requires fewer numbers of samples taken from the edges of the graph in order to decide which path from source to destination is shortest. The main difference between these algorithms is the definition of dynamic threshold, which is used for producing the reinforcement signal. It was shown that the shortest path is found with a probability as close as to unity by proper choice of the parameters of the proposed algorithms [14].

In this paper we introduce a new algorithm for solving the stochastic shortest path problem which is based on the particle swarm optimization method. Computer experiments are conducted to evaluate the performance of the proposed algorithm. The comparison of our algorithm with other algorithms indicates that its performance is suitable even by the less number of iterations.

The rest of this paper is organized as follows: Section 2 deals with a short summarization of particle swarm optimization (PSO) and its features. The proposed algorithm and preliminary experimental results are discussed in sections 3 and 4, respectively. Finally, in section 5, concluding remarks are made.

2 Particle Swarm Optimization

From the beginning of 90's, new optimization technique researches using analogy of swarm behavior of natural creatures have been started [9]. Dorigo developed ant colony optimization (ACO) mainly based on the social insect, especially ant, metaphor [10]. Eberhart and Kennedy developed particle swarm optimization based on the analogy of swarm of bird and fish school [11]. PSO simulates the behaviors of bird flocking. Suppose the following scenario: a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. No bird knows where the food is, but at each iteration they know how far the food is. One of the best strategies to find the food is to follow the bird which is nearest to the food. For using this strategy, PSO considers each bird (particle) in the search space as a single solution.

All particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles. They are flown through the problem space by following the current optimum particles [12].

PSO is initialized with a group of random particles and then searches for optimum solution by updating generations. At each iteration, each particle is updated by following two "best" values. The first one is the best solution it has achieved so far and is called *pbest* and the second one is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population and is called *gbest*. Finally, each particle updates its velocity and positions using equations (1) and (2).

$$velocity(k+1) = w * velocity(k) + c1 * rand() * (pbest - p(k)) + c2 * (gbest - p(k)), \quad (1)$$

$$p(k+1) = p(k) + velocity(k+1), \quad (2)$$

where *velocity* is the particle's velocity, *p* is the current particle's position, *pbest* and *gbest* are defined as stated before. *rand()* is a random number between (0,1). *c1* and *c2* are learning factors. Usually *c1* = *c2* = 2. The inertia weight *w* is employed to control the impact of the previous history of velocities on the current velocity, thus to influence the trade-off between global and local exploration abilities of the *flying points*. The performance of each particle is measured according to a predefined fitness function, which is related to the problem being solved. In SSPP each particle is defined as a sequence of vertexes that represent a valid path in the stochastic graph and the fitness function is the cost of the path according to the cost of the edges. The detail of representation style for each particle, their fitness function, and its operators are discussed in the following section.

3 The Proposed Algorithm

In this section, we propose an algorithm based on particle swarm optimization for solving the SSPP. Our algorithm and its pseudo code are like the PSO. The novelty of the proposed algorithm is in definition of elements and operators. The pseudo code of the proposed algorithm is as follows:

```

Initialize particles
repeat
  for each particle u do
    Calculate fitness value of particle u, denoted by f(u)
    if (f(u) is better than f(pbest)) then
      Set u as the new pbest
    end
  Choose the particle with the best fitness value among all
  particles as the gbest
  for each particle u do
    Calculate particle velocity according equation (1)
    Update particle position according equation (2)
  end
until (maximum iterations or minimum error criteria is attained)

```

In the rest of this section we describe the elements and operators of this algorithm.

3.1 Position

Let $G = (V; E; Q)$ be the graph in which we are looking for shortest path from v_s to v_d . As we are looking for path, we can consider just sequences of $n-1$ nodes, all different nodes of graph except v_s , since we can add v_s in the first place of sequence, we don't use it during sequence generation. Also we just need the nodes that appear before v_d ; for this reason just the sequence of nodes that appear before v_d are considered. Such a sequence is here seen as a "position". So the search space is defined as the finite set of all positions.

3.2 Objective Function

Let us to define a position as $p = (v_1, v_2, \dots, v_{n-1})$. This position is *acceptable* only if for all pairs (v_i, v_{i+1}) are edges of the graph. In order to define the *cost* function, a classical way it to just complete the graph, by *virtual* edges with an arbitrary large cost value enough to be sure no solution could contain such them. Now, on each position, a possible objective function can simply be defined by

$$p(x) = \left\{ \sum_{i=1}^{k-1} w_{i,i+1} \mid v_k = v_d \right\}, \quad (3)$$

where $w_{i,i+1}$ is the cost of edge (v_i, v_{i+1}) . This objective function has a finite number of values and its global minimum is indeed on the best solution.

3.3 Velocity

We want to define an operator *velocity*; when this operator is applied to a position, it gives another position. So, here, it is a permutation of n elements, that is to say a list of transpositions. A *velocity* is then defined by

$$velocity = \{(i_k, j_k) \mid i_k, j_k \in \{v_1, \dots, v_n\}\} \quad (4)$$

which means exchanging numbers (i_1, j_1) , then numbers (i_2, j_2) ... and the last numbers (i_n, j_n) .

Note that the results of two such lists applied to any position are the same. To create the opposite of a velocity ($-velocity$) we can do the same transpositions as in *velocity*, but in reverse order.

3.4 Move

Let p be a position and *velocity* a velocity. The position $p' = p + velocity$ is found by applying the first transposition of *velocity* to p , then the second one to the result and etc.

For example consider $p = (1,2,3,4,5)$ and $velocity = \{(1,2)(2,3)\}$. By applying *velocity* to p , we obtain successively $(2,1,3,4,5)$ and $(3,1,2,4,5)$.

3.5 Subtraction

Let p_1 and p_2 be two positions. The difference of p_1 and p_2 is defined as *velocity*, found by a given algorithm, so that applying *velocity* to p_1 gives p_2 . The algorithm that we introduce for reaching this goal is as follow:

- Compare the elements of p_1 and p_2 one by one
- For each $v_x \in p_1$ and $v_y \in p_2$ which are in the same position in p_1 and p_2 and $v_x \neq v_y$, add (v_x, v_y) to *velocity*
- Apply the updated *velocity* to p_1

3.6 Addition

Let $velocity_1$ and $velocity_2$ be two velocities. In order to compute $velocity_1 \oplus velocity_2$, we construct a list which contains first the transpositions of $velocity_1$, followed by the ones of $velocity_2$. Optionally, we contract it to obtain a smaller equivalent velocity. In particular, this operation is defined so that

$$\| velocity_1 \oplus velocity_2 \| \leq \| velocity_1 \| + \| velocity_2 \|, \tag{5}$$

where \oplus is the addition operation between two *velocities*.

3.7 Multiplication

Let c be a real coefficient and *velocity* be a velocity. There are different cases, depending on the value of c . When c is zero, then we having $c * velocity = \phi$, when $c \in (0, 1]$, we just truncate *velocity*. So we select the first $c\%$ of *velocity* elements, and when c is greater than one, it means we have $c = k + c', k \in \mathbb{N}, c' \in [0, 1)$. So we define

$$c * velocity = \underbrace{velocity \oplus velocity \oplus \dots \oplus velocity}_{k \text{ times}} \oplus c' * velocity, \tag{6}$$

when c is negative, we write $c * velocity = (-c) * \neg velocity$ and we just have to consider one of the previous cases.

4 Experimental Result

To evaluate the performance of the proposed algorithm a set of experiment are conducted on three following stochastic graphs borrowed from [15]. The distribution of edge costs for these graphs are given in tables 1 through 3.

- Graph 1 is a graph with 10 nodes, 23 edges, $v_s = 1$, and $v_d = 10$
- Graph 2 is a graph with 10 nodes, 23 edges, $v_s = 1$, and $v_d = 10$
- Graph 3 is a graph with 15 nodes, 42 edges, $v_s = 1$, and $v_d = 15$

Table 1. Cost Distribution of Graph 1 with $v_s = 1$ and $v_d = 10$

Edge	Cost	Probability
(1,2)	7.0 7.3 9.4	0.2 0.5 0.3
(1,3)	2.5 3.5 8.2	0.5 0.4 0.1
(1,4)	4.2 4.8 6.1	0.2 0.3 0.5
(2,5)	2.6 3.1 5.5 8.8 9.0	0.1 0.2 0.4 0.2 0.1
(2,6)	5.8 7.0 9.5	0.3 0.3 0.4
(3,2)	1.5 7.3	0.4 0.6
(3,7)	6.5 7.4 7.5	0.4 0.5 0.1
(3,8)	5.9 7.2 9.8	0.6 0.3 0.1
(4,3)	2.1 3.2 8.5 9.8	0.3 0.2 0.3 0.2
(4,9)	8.9 9.6	0.7 0.3
(5,7)	3.2 4.8 6.7	0.2 0.2 0.6
(5,10)	6.3 6.9	0.5 0.5
(6,3)	6.6 8.5 9.8	0.8 0.1 0.1
(6,5)	0.6 1.5 3.9 5.8	0.1 0.4 0.3 0.2
(6,7)	0.2 4.8	0.4 0.6
(7,6)	6.1 6.3 8.5	0.2 0.3 0.5
(7,8)	1.6 1.8 4.0 5.2	0.2 0.3 0.3 0.2
(7,10)	1.6 3.4 7.1	0.1 0.5 0.4
(8,4)	9.0 9.6	0.5 0.5
(8,7)	2.1 4.6 8.5	0.3 0.4 0.3
(8,9)	1.7 4.9 5.3 6.5	0.1 0.4 0.4 0.1
(7,9)	0.3 3.0 5.0	0.1 0.4 0.5
(9,10)	0.6 1.2 5.4 6.6	0.1 0.1 0.3 0.5

Table 2. Cost Distribution of Graph 2 with $v_s = 1$ and $v_d = 10$

Edge	Cost	Probability
(1,2)	3.0 5.3 7.4 9.4	0.2 0.2 0.3 0.3
(1,3)	3.5 6.2 7.9 8.5	0.3 0.3 0.2 0.2
(1,4)	4.2 6.1 6.9 8.9	0.2 0.3 0.2 0.3
(2,5)	2.6 4.1 5.5 9.0	0.2 0.2 0.4 0.2
(2,6)	5.8 7.0 8.5 9.6	0.3 0.3 0.2 0.2
(3,2)	1.5 2.3 3.6 4.5	0.2 0.2 0.3 0.3
(3,7)	6.5 7.2 8.3 9.4	0.5 0.2 0.2 0.1
(3,8)	5.9 7.8 8.6 9.9	0.4 0.3 0.1 0.2
(4,3)	2.1 3.2 4.5 6.8	0.2 0.2 0.3 0.3
(4,9)	1.1 2.2 3.5 4.3	0.2 0.3 0.4 0.1

Table 2. (continued)

(5,7)	3.2 4.8 6.7 8.2	0.2 0.2 0.3 0.3
(5,10)	6.3 7.8 3.4 9.1	0.2 0.2 0.4 0.2
(6,3)	6.8 7.7 8.5 9.6	0.4 0.1 0.1 0.4
(6,5)	0.6 1.5 3.9 5.8	0.2 0.2 0.3 0.3
(6,7)	2.1 4.8 6.6 7.5	0.2 0.4 0.2 0.2
(7,6)	4.1 6.3 8.5 9.7	0.2 0.3 0.4 0.1
(7,8)	1.6 2.8 5.2 6.0	0.2 0.3 0.3 0.2
(7,10)	1.6 3.4 8.2 9.3	0.2 0.3 0.3 0.2
(8,4)	7.0 8.0 8.8 9.4	0.2 0.2 0.2 0.4
(8,7)	2.1 4.6 8.5 9.6	0.4 0.2 0.2 0.2
(8,9)	1.7 4.9 6.5 7.8	0.2 0.2 0.2 0.4
(7,9)	3.5 4.0 5.0 7.7	0.1 0.2 0.4 0.3
(9,10)	4.6 6.4 7.6 8.9	0.4 0.1 0.2 0.3

Table 3. Cost Distribution of Graph 3 with $v_s = 1$ and $v_d = 15$

Edge	Cost	Probability
(1,2)	16 25 36	0.6 0.3 0.1
(1,3)	21 24 25 39	0.5 0.2 0.2 0.1
(1,4)	11 13 26	0.4 0.4 0.2
(2,11)	24 28 31	0.5 0.3 0.2
(2,5)	11 30	0.7 0.3
(2,6)	13 37 39	0.6 0.2 0.2
(3,2)	11 20 24	0.6 0.3 0.1
(3,7)	23 30 34	0.4 0.3 0.2
(3,8)	14 23 34	0.5 0.4 0.1
(4,3)	22 30	0.7 0.3
(4,9)	35 40	0.6 0.4
(4,12)	16 25 37	0.5 0.4 0.1
(5,13)	28 35 37 40	0.4 0.3 0.2 0.1
(5,15)	25 32	0.7 0.3
(5,10)	27 33 40	0.4 0.3 0.3
(5,7)	15 17 19 26	0.3 0.3 0.3 0.1
(6,5)	18 25 29	0.5 0.3 0.2
(6,13)	21 23	0.5 0.5
(6,7)	11 31 37	0.5 0.5 0.1
(6,3)	18 24	0.7 0.3
(7,10)	19 23 37	0.6 0.2 0.2

Table 3. (continued)

(7,8)	15 22 24	0.3 0.3 0.3
(7,6)	12 23 31	0.5 0.3 0.2
(8,7)	14 34 39	0.6 0.2 0.2
(8,14)	14 15 27 32	0.3 0.3 0.2 0.2
(8,9)	13 31 32	0.8 0.1 0.1
(8,4)	13 23 34	0.4 0.3 0.3
(9,7)	10 17 20	0.6 0.3 0.1
(9,10)	16 18 36 39	0.3 0.3 0.2 0.2
(9,15)	12 13 25 32	0.4 0.3 0.2 0.1
(9,14)	19 24 29	0.4 0.3 0.3
(10,13)	14 20 25 32	0.3 0.3 0.2 0.2
(10,15)	15 19 25	0.4 0.3 0.3
(10,14)	23 34	0.9 0.1
(11,13)	13 31 25	0.6 0.3 0.1
(11,5)	18 19 20 23	0.3 0.3 0.3 0.1
(11,6)	10 19 39	0.5 0.4 0.1
(12,8)	15 36 39	0.5 0.3 0.2
(12,9)	16 22	0.7 0.3
(12,14)	10 13 18 34	0.3 0.3 0.3 0.1
(13,15)	12 31	0.9 0.1
(14,15)	14 19 32	0.5 0.3 0.2

The number of iterations for each run is set to 50. We ran the algorithm 12 times for each graph by 25, 50, 100 and 150 swarm sizes. Tables 4 through 6 show the results of our experiments in each graph. Each row of table shows the swarm size.

Table 4. The Result of Algorithm on Graph 1 with Different Swarm Sizes

Swarm Size	Converged Path	Cost of the Converged Path	Number of Convergence	Percentage of Convergence
25	1 3 7 10	14.369	8	67.33%
	1 4 9 10	18.482	2	16.0%
	1 3 8 7 10	19.81	1	8.33%
	1 3 2 5 7 10	22.171	1	8.33%
50	1 3 7 10	14.92	12	100%
100	1 3 7 10	15.061	12	100%
150	1 3 7 10	14.873	12	100%

Table 5. The Result of Algorithm on Graph 2 with Different Swarm Sizes

Swarm Size	Converged Path	Cost of the Converged Path	Number of Convergence	Percentage of Convergence
25	1 4 9 10	15.6710	4	33.33%
	1 3 7 10	18.1515	3	25.0%
	1 2 5 10	17.5327	4	33.33%
	1 3 8 7 10	22.6778	1	8.33%
50	1 4 9 10	15.3545	11	91.66%
	1 2 5 10	14.7778	1	8.33%
100	1 4 9 10	15.7380	12	100%
150	1 4 9 10	15.3053	11	91.66%
	1 3 2 5 10	17.4222	1	8.33%

Table 6. The Result of Algorithm on Graph 3 with Different Swarm Sizes

Swarm Size	Converged Path	Cost of the Converged Path	Number of Convergence	Percentage of Convergence
25	1 2 5 15	63.292	4	33.33%
	1 4 9 15	68.0357	2	16.66%
	1 4 3 8 14 15	89	1	8.33%
	1 4 3 8 9 15	92.5714	1	8.33%
	1 3 7 10 15	94.2222	1	8.33%
	1 4 12 14 15	86.864	2	16.66%
50	1 2 5 15	61.1428	8	66.6%
	1 4 9 15	63.6461	4	33.3%
100	1 2 5 15	62.7442	6	50.0%
	1 4 9 15	68.0933	2	16.66%
	1 4 12 9 15	62	2	16.66%
	1 3 8 14 15	77.6667	1	8.33%
	1 4 12 14 15	65.4423	1	8.33%
150	1 2 5 15	63.0834	9	75.0%
	1 4 9 15	67.5215	3	25.0%

In tables 7 and 8, we compare the result of our algorithm on the above graphs by the results of the algorithms reported in [7][8][13]. Each row of the table shows the results for each graph. Columns contain paths and costs identified by our algorithm and other algorithms. The values in columns are mean of the values in runs of the algorithm which coverage to the correct path.

Table 7. Comparison of Our Algorithm with Dijkstra's by Running on Three Graphs

Graphs	PSO Shortest Path	PSO Shortest Cost	Dijkstra Shortest Path	Dijkstra Shortest Cost
Graph 1	1 3 7 10	14.84535	1 3 7 10	15.22
Graph 2	1 4 9 10	15.49471	1 4 9 10	16.1
Graph 3	1 2 5 15	62.46404	1 2 5 15	64.5

Table 8. Comparison of Our Algorithm with DLA Algorithm by Running on Three Graphs

Graphs	PSO Shortest Path	PSO Shortest Cost	DLA Shortest Path	DLA Shortest Cost
Graph 1	1 3 7 10	14.84535	1 3 7 10	15.1
Graph 2	1 4 9 10	15.49471	1 4 9 10	15.7
Graph 3	1 2 5 15	62.46404	1 2 5 15	63.2

An important point of our algorithm in comparison with DLA is the difference between numbers of iterations for each algorithm. In all cases we run the algorithm for 50 iterations and most of them reach the correct path, but DLA reach these result by 9769, 9846 and 9718 iterations for graphs 1, 2 and 3, respectively.

5 Conclusion

In this paper we proposed a new method for solving the stochastic shortest path problem. Since the population based techniques have good effects in stochastic applications, we use one of these methods. Our approach is based on particle swarm optimization which is an important parts of swarm intelligence by focused on the birds behavior. For implementing this idea we simulated all operators which are used in the PSO equation, by some suggested simple algorithms. These operators are move, addition, subtraction, multiplication. For evaluating our proposed algorithm, the result of it is compared with Dijkstra[13] and DLA[7][8] algorithms.

References

1. Deo, N., Pang, C.: Shortest Path Algorithms: Taxonomy and Annotation. Networks 14, 275–323 (1984)
2. Frank, H.: Shortest paths in probabilistic graphs. Operations Research 17, 583–599 (1969)
3. Loui, R.P.: Optimal paths in graphs with stochastic or multidimensional weights. Communications of the ACM 26, 670–676 (1983)
4. Murthy, I., Sarkar, S.: A relaxation based pruning technique for a class of stochastic shortest path problems. Transportation Science 30, 220–236 (1996)
5. Murthy, I., Sarkar, S.: Stochastic shortest path problems with piecewise linear concave utility functions. Management Science 44, 125–136 (1998)
6. Rasteiro, D.M., Anjo, A.B.: Metaheuristics for stochastic shortest path problem. In: Proceedings of 4th MetaHeuristics International Conference (MIC), Porto, Portugal (2001)

7. Meybodi, M.R., Beigy, H.: Solving Stochastic Shortest Path Problem Using Distributed Learning Automata. In: Proceedings of 6th Annual International Computer Society of Iran Computer Conference CSICC, Isfahan, Iran, February 2001, pp. 70–86 (2001)
8. Beigy, H., Meybodi, M.R.: A New Distributed Learning Automata Based Algorithm For Solving Stochastic Shortest Path Problem. In: Proceeding of Joint Conference on Information Sciences (JCIS), North Carolina, USA, pp. 339–343 (2002)
9. Fukuyama, Y.: Fundamentals of particle swarm optimization techniques. In: IEEE PES Tutorial on Modern Heuristic Optimization Techniques with Application to Power Systems, ch. 5 (January 2002)
10. Colomi, A., Dorigo, M., Maniezzo, V.: Distributed Optimization by Ant Colonies. In: Proceeding of First European Conference on Artificial Life, pp. 134–142. MIT Press, Cambridge (1991)
11. Kennedy, J., Eberhart, R.: Particle Swarm Optimization. In: Proceedings of IEEE International Conference on Neural Networks (ICNN), vol. IV, Perth, Australia, pp. 1942–1948 (1995)
12. Eberhart, R.C., Kennedy, J.: A New Optimizer using Particle Swarm Theory. In: Proceeding of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, pp. 39–43 (1995)
13. Dijkstra, E.W.: A note on two problems in connection with graphs. *Numerische Mathematik*, pp. 269–271 (1959)
14. Beigy, H., Meybodi, M.R.: Utilizing Distributed Learning Automata to Solve Stochastic Shortest Path Problems. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 14, 591–615 (2006)
15. Alexopoulos, C.: State Space Partitioning Methods for Stochastic Shortest Path Problems. *Networks* 30, 9–21 (1997)