Nikulin, Yury

Solving the robust shortest path problem with interval data using a probabilistic metaheuristic approach

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 597

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 597

Solving the robust shortest path problem with interval data using a probabilistic metaheuristic approach

Yury Nikulin

Working Paper, August 2005

Christian-Albrechts-Universität zu Kiel,
Institut für Betriebswirtschaftslehre,
Olshausenstr 40, 24118 Kiel, Germany,
nkln@lycos.com

Abstract

This paper addresses the robust shortest path problem with interval data, i.e. the case of classical shortest path problem with given source and sink when arc weights are not fixed but take their values from some intervals associated with arcs. The problem consists in finding a shortest path that minimizes so called robust deviation, i.e. deviation from an optimal solution under the worst case realization of interval weights. As it was proven in [9], the problem is NP-hard, therefore it is of great interest to tackle it with some metaheuristic approach, namely simulated annealing, in order to calculate an approximate solution for the large scale instances efficiently. We describe theoretical aspects and present the results of computational experiments. To the best of our knowledge, this is the first attempt to develop metaheuristic approach for solving the robust shortest path problem.

Keywords: shortest path problem, simulated annealing, uncertainty, robustness.

1 Introduction

The special interest motivated by telecommunications applications induces not to solve the interval shortest path problem itself, but to hedge against the worst-case realization (scenario) of problem parameters, which can be interpreted as given with uncertainty. Playing against worst-case scenario is commonly known as robust optimization (see, e.g. [16]). As it was indicated in [9], in many cases the robust equivalent of a polynomially solvable problem becomes NP-hard. This is true also for the problem considered in this paper.

We consider the special case of a shortest path problem on directed graphs where the arc costs (weights) are not fixed but take their values from some positive intervals. No stochastic distribution is given inside intervals. The interval function is defined as the sum of interval weights over all arcs of feasible shortest path. Our goal is to find a relative robust shortest path from start node (source) to destination node (sink) which minimizes the maximum deviation from the optimal shortest path over all realizations of arc costs. This case is principally different from the absolute robust shortest problem [9] which is to select a path for which the maximum path length taken across all possible scenarios is minimal – a variant which can be easily solved in case of interval data. In the remainder of the present paper only relative robustness is considered, and the problem is commonly referred to as the robust shortest path problem. This problem is significantly harder than the conventional shortest path problem that deals with fixed and static positive values of weights associated with every arc.

Contrary to the classical shortest path problem, which can be easily solved by Dijkstra's algorithm [4] in strongly polynomial time (for a survey of other algorithms see e.g. [1]), the robust shortest path problem depends on the realization of arc weights. As it was proven in [19], the robust shortest path problem with scenario representation, i.e. where the set of arc weight realizations is defined, is strongly NP-hard. In [19] the authors conjectured that the robust shortest path problem with interval data is also NP-hard, what has been later proven in [20]. Simultaneously, in [2] it was proven that the robust shortest path problem is NP-hard even if the bounds of all intervals of uncertainty belong to $(0, 1)$. The interval data minmax regret shortest path problem is NP-hard even if the network is directed, acyclic, and has a layered structure. Nevertheless it was shown that the problem is polynomially solvable in the practically important case where the number of arcs with uncertain lengths is fixed or is bounded by the logarithm of a polynomial function of the total number of arcs.

The basic theoretical background for the robust shortest problem has been presented in [6]. A reformulation of the robust shortest path problem as a special mixed integer program and

a preprocessing technique based on weak path concept were presented. The concept of weak paths was first defined in [3]. In [6] it was shown how this concept can be efficiently used in a preprocessing stage for solving the robust shortest path problem. We will shortly sketch out the main results of [6] in section 2.
A branch and bound procedure extending of some previous results as well as some other innovations is presented in [12], [13]. The computational results obtained by the new branch and bound algorithm are presented and analyzed with respect to various benchmarks. The results were compared with earlier results obtained by solving a mixed integer program in [6]. It was shown that the algorithm is extremely efficient on random graphs, but has no good performance on layered acyclic graphs (mainly due to their specifics). Nevertheless, the proposed algorithm can solve problem instances on random graphs with the number of nodes $n \approx 500$ in reasonable time.
In [14] a new exact method based on Benders decomposition was described with regard to the robust shortest path problem. It was shown that this approach gives very good computational results on many classes of networks. The study of impact of arc density and network structure is additionally presented. It was concluded that the method based on Benders decomposition is the best one for networks with low arc density, while the branch and bound methods are the most promising ones in the case arc density essentially increases.
The rest of the paper is organized as follows. In section 2 we introduce the basic notation and formulate the problem. The main well-known theoretical results are also presented. How to apply the simulated annealing metaheuristic to the problem is described in section 3. The results of computational experiments are presented in section 4. Final remarks and conclusions appear in section 5.

2 Problem description and theoretical background

Let $G = (V, A)$ be a digraph, where V, $|V| = n$, is the set of nodes and A, $|A| = m$, $m \leq n(n-1)$, is the set of arcs. With each arc $(i, j) \in A$ we associate a cost interval $[l_{ij}, u_{ij}]$, $0 < l_{ij} < u_{ij} < c$, i.e. for each arc $(i, j) \in A$ its cost c_{ij} is not fixed and belongs to $[l_{ij}, u_{ij}]$. The cost upper bound c is given and fixed. No probability distribution is given inside the cost interval. A realization of all arc costs is called a scenario s. The set S is the set of all possible scenarios. The source node 1 and the destination node n are given. An ordered chain from 1 to n of alternating nodes and arcs such that every two neighboring nodes v_k and v_{k+1} are connected with arc $(v_k, v_{k+1}) \in A$ is said to be a path p from 1 to n. Moreover, if all the nodes in a chain are different, the path is called simple. Let P be the set of all paths from 1 to n. If $p \in P$ contains all the nodes of V, then it is called Hamiltonian. Let s be a realization of arc costs, i.e. $c_{ij}^s \in [l_{ij}, u_{ij}]$. We denote by c_p^s the total cost of path p in scenario s, which is defined as:

$$c_p^s := \sum_{(i,j) \in A_p} c_{ij}^s,$$

where A_p is the set of arcs that the path p consists of.
For every scenario $s \in S$ the shortest path problem is to find a path from 1 to n of minimum total cost over all paths $p \in P$. This problem can be easily solved by Dijkstra's algorithm [4], which processes nodes in nondecreasing order of their actual distances from the source node. At the beginning all nodes are given an infinite distance except the source which is given a distance 0. At each step we choose the next unlabelled node which is nearest to the source

and mark it, while updating the optimal distance to all its neighbors. The optimal distance of a neighbor is updated only if reaching it from the current labelling node gives a total path length that is shorter then its current distance. Doing so the algorithm constructs the so-called shortest path tree, which is a spanning tree rooted at source node where the shortest paths to all other nodes are determined. The shortest path to each node is then found by tracing the predecessor iteratively back to the source. One of the best implementations of Dijkstra's algorithm uses priority queue structure and has time complexity $O(n\ log(n+m))$. Observe that Dijkstra's algorithm can be correctly applied to the problem that has no negative arc costs, otherwise it terminates but does not provide the fitness of the algorithm, i.e. a proper optimal solution is unlikely to be found. In our case any possible scenario has only positive arc costs, so the Dijkstra algorithm can be used to calculate the shortest path for a particular arc cost realization.

The interval representation of arc costs may be interpreted as some sort of uncertainty for input data of the classical shortest path problem. The presence of uncertainty may be caused by different reasons: inaccuracy of initial data, non-adequacy of models to real processes, errors of numerical methods, errors of rounding off and other factors. So it appears to be important to identify a solution which is flexible under all realizations of problem parameters. It is of special interest to find a solution which provides the smallest changes of the result under worst possible scenario of distribution of problem parameters. The model with such a nice property is called robust counterpart problem, and a solution of the robust counterpart problem is generally known as robust solution.

First it seems to be natural to give a definition of a robust solution as follows: an optimal solution is robust if it remains optimal under any realization of the input data. But this definition can hardly be regarded as desirable, because it is too restrictive. Most unlikely such a solution exists. Another definition may be considered more appropriate: our goal is to find a robust solution which minimizes maximum regret or relative deviation (minimizes possible consequences of worst-case scenario with respect to objective function).

Thus, for each path $p \in P$ in a scenario s the difference between the total cost c_p^s and the cost of the shortest path in s represents the deviation for p in scenario s:

$$dev_p^s := c_p^s - \min_{p' \in P} c_{p'}^s.$$

For a given path $p \in P$ the worst-case scenario s_p is a scenario for which the deviation for p is maximum over all scenarios $s \in S$, i.e.

$$s_p := \arg\max_{s \in S} dev_p^s.$$

Then the difference

$$dev_p^{s_p} := c_p^{s_p} - \min_{p' \in P} c_{p'}^{s_p}$$

represents the robust deviation of p.

A path p^0 is said to be a robust shortest path if it has the smallest robust deviation

$$p^0 := \arg\min_{p \in P} dev_p^{s_p}$$

among all paths from 1 to n.

In order to keep the paper self-contained we start with some theoretical background, originally presented in [6].

The following proposition gives a worst-case scenario for a given path.

Proposition 1 *[6] The scenario in which the costs of every arc on a path $t \in T$ is at its upper bound and the cost of every other arcs is at its lower bound is a worst-case scenario, i.e. $c_{ij}^{s_p} = u_{ij}\ \forall (i,j) \in A_p$ and $c_{ij}^{s_p} = l_{ij}\ \forall (i,j) \in A \setminus A_p$.*

In other words, for any path $p \in P$ the worst-case scenario s_p is uniquely determined and minimizes the robust deviation $dev_p^{s_p}$ over all possible scenarios. It means that we need to consider only a finite number of scenarios. However, the number of paths in a graph may grow exponentially with the number of nodes in the network.

Definition 1 *[6] A path p is a weak path if it is a shortest path for some (at least one) realization s of arc costs.*

The following theorem gives a characterization of weak paths.

Theorem 1 *[6] A path $p \in P$ is a weak path if and only if it is a shortest path when the cost of every arc on this path corresponds to its lower bound and the cost of all the remaining arcs correspond to their upper bounds.*

The following statements give us an idea about the construction of a robust shortest path.

Proposition 2 *[6] A robust shortest path is a weak path.*

Definition 2 *[6] An arc (i,j) is a weak arc if it is on one of the weak paths.*

In other words, every robust shortest path uses only weak arcs, i.e. all non-weak arcs are not considered at all. This observation leads to:

Proposition 3 *[6] A non-weak arc $a \in A$ can be deleted from graph G when solving a robust shortest path problem.*

Unfortunately, the last result cannot be used efficiently in a preprocessing stage of any algorithm solving the robust shortest path problem for arbitrary graph G due to the following result.

Proposition 4 *[6] Deciding whether a given arc is weak or not is NP-complete.*

In [6] the authors proposed a polynomial-in-time procedure to decide whether a given arc is weak or not for very specific classes of layered graphs. Later in [12] it was shown that the procedure essentially decreases calculation complexity on layered graphs however, evidently, cannot be extended on arbitrary graphs.

3 Simulated Annealing

Simulated annealing (SA) or hill climbing is a generic probabilistic heuristic approach originally proposed in [7] and [8] for global optimization. Usually, SA locates a "good" approximation of the global optimum of a given objective function z in a large search space. At each iteration, SA considers some neighbors of the current solution (search point) π, and probabilistically chooses either to accept a new solution π' or keeping π. The probabilities are chosen so that the problem ultimately tends to move to solutions with better objective function value. Typically this process is repeated until a solution which is "good enough" has been determined,

or until a given time limit has been reached. SA uses several basic concepts: neighborhood, probabilistic acceptance of a new neighborhood solution, parameter (temperature) dependent acceptance probability, cooling schedule, termination criterion.
In order to apply SA to a particular problem, we must specify the search space, the neighborhood search moves, the acceptance probability function, the cooling schedule and the termination criterion. These choices can significantly affect the method's effectiveness. Unfortunately, there is no unique choice that will be good for all problems, and there is no general way to find the best choice for a given problem [10].
Now let us precisely describe how to apply SA to the robust shortest path problem with interval data.

- *Search space.* Any subset of the arc set A can be represented by a boolean vector $\pi \in \{0,1\}^m$, such that $\pi_i = 1$ if arc a_i belongs to the current subset and $\pi_i = 0$ otherwise. Thus, a vector π represents a current point in the search space. Obviously, the search space of the algorithm is now presented by the variety of subsets of arcs describing a graph that has a path from the source node 1 to the destination node n. The subsets which describe graphs such that there exists no path from 1 to n are not feasible. We have to exclude such points from the search space by defining their objective values $z(\pi) = \infty$. It is necessary to emphasize that in each iteration we do not check whether the search point represents a simple path. The reason for this is the following. As far as any feasible graph contains a shortest path from 1 to n, and the value of objective of such graph is always greater then the value of the shortest path itself, then this shortest path will be most likely detected later during execution of SA. The only condition that has to be satisfied for any search point is that it should always describe a graph with at least one path from 1 to n.

- *Initial solution.* Initially, we find a shortest path in the scenario where every arc (i,j) has its largest cost u_{ij} in order to determine a good starting point π^0 for SA algorithm. Alternatively, an initial search point can be generated randomly, but nevertheless the feasibility of the graph described by the point has to be guaranteed.

- *Neighbourhood search moves.* Let π be the current search point. We randomly chose i from the list of arcs available for selection. This list contains all the arcs which are in the current solution as well as all arcs adjacent to them. Then we construct the neighbor search point π' by inverting π_i, i.e. $\pi' = (\pi'_1, \pi'_2, \ldots, \pi'_m)$ where $\pi'_j = \pi'_j$, if $j \neq i$, and $\pi'_j = 1 - \pi_j$ otherwise.

  This formalization is quite natural and best suited for probabilistic and evolutionary metaheuristics like SA and GA that allow small deterioration of the current solution in order to get amelioration afterwards (see e.g. [18]).

  Two cases are possible: 1) we invert 0 to 1, and 2) we invert 1 to 0. Let us consider these two cases in detail.

  Case 1. Let $\mathcal{A}$ represent an arc subset describing the current solution π. Case 1 corresponds to adding arc $a_i := (\tilde{i}, \tilde{j})$ to the arc set of graph $h := (V, \mathcal{A})$ described by the current search point π. Obviously, a new search point π' represents graph $h' := (V, \mathcal{A}')$, where $\mathcal{A}' := \mathcal{A} \cup \{a_i\}$. Since h is a feasible graph, h' also contains at least one path from 1 to n, therefore additional checking for path existence is not necessary.

According to Proposition 1, the worst-case scenario for solution π is the following: $c_{ij}^{s_h} = u_{ij}\ \forall (i,j) \in \mathcal{A}$ and $c_{ij}^{s_h} = l_{ij}\ \forall (i,j) \in A \setminus \mathcal{A}$, whereas solution π' has the following worst-case scenario $s_{h'}$: $c_{ij}^{s_{h'}} = u_{ij}\ \forall (i,j) \in \mathcal{A}'$ and $c_{ij}^{s_{h'}} = l_{ij}\ \forall (i,j) \in A \setminus \mathcal{A}'$. In other words, $s_{h'}$ can be obtained from s_h by replacing cost $c_{\tilde{i}\tilde{j}}^{s_h} = l_{\tilde{i}\tilde{j}}$ of arc a_i with new cost $c_{\tilde{i}\tilde{j}}^{s_{h'}} = u_{\tilde{i}\tilde{j}}$. Then the cost charge Δ of performing a neighbourhood search move can be calculated as:

$$\Delta_{\pi \to \pi'} := z(\pi') - z(\pi) =$$

$$= c_{h'}^{s_{h'}} - \min_{p \in P} c_p^{s_{h'}} - c_h^{s_h} + \min_{p \in P} c_p^{s_h} =$$

$$= u_{\tilde{i}\tilde{j}} + \min_{p \in P} c_p^{s_h} - \min_{p \in P} c_p^{s_{h'}}.$$

Thus, in order to calculate the cost charge Δ we have to calculate the difference of the cost of minimum shortest path in scenarios s_h and $s_{h'}$.

Case 2. This case corresponds to deleting arc $a_i := (\tilde{i}, \tilde{j})$ from the arc set of graph $h := (V, \mathcal{A})$ described by the current search point π. Obviously, a new search point π' represents graph $h' := (V, \mathcal{A}')$, where $\mathcal{A}' := \mathcal{A} \setminus \{a_i\}$. Additional checking for path existing of h' is necessary. If h' does not contain any path from 1 to n, then the neighbourhood search move fails. Otherwise, similar to Case 1, worst-case scenario $s_{h'}$ can be obtained from s_h by replacing cost $c_{\tilde{i}\tilde{j}}^{s_h} = u_{\tilde{i}\tilde{j}}$ of arc a_i with new cost $c_{\tilde{i}\tilde{j}}^{s_{h'}} = l_{\tilde{i}\tilde{j}}$. Then the cost charge Δ of performing a neighbourhood search move can be calculated as:

$$\Delta_{\pi \to \pi'} := z(\pi') - z(\pi) =$$

$$= c_{h'}^{s_{h'}} - \min_{p \in P} c_p^{s_{h'}} - c_h^{s_h} + \min_{p \in P} c_p^{s_h} =$$

$$= -u_{\tilde{i}\tilde{j}} + \min_{p \in P} c_p^{s_h} - \min_{p \in P} c_p^{s_{h'}}.$$

As in Case 1, in order to calculate Δ we have to calculate the difference of the costs of shortest pathes in scenarios s_h and $s_{h'}$.

Note that graph feasibility can be easily examined using breadth-first search.

- *Acceptance probability rule.* We will accept a new solution π' instead of π with probability

$$P(\pi, \pi', T_q) = \min \left\{ 1, \exp\left(-\frac{z(\pi') - z(\pi)}{T_q} \right) \right\}.$$

- *Cooling schedule.* The initial cooling temperature T_0 depends on φ and the largest possible arc cost $c_{max} := \max_{(i,j) \in A} u_{ij}$, namely $T_0 := 100 \cdot n \cdot c_{max}$. Annealing schedule is defined according to the following rule: $T_q := \alpha^q \cdot T_0$, where $\alpha := 0.95$.

- *Termination criterion.* One has the freedom to introduce different stopping criteria. Typically, SA is repeated until the system reaches a state which is "good enough", or until a given time limit has been reached. The annealing temperature decreases to (nearly) zero short before termination. For computational purposes we define the termination criterion: $T_q \leq \gamma$, where γ is equal to 0.001.

In order to improve efficiency of the algorithm for practical calculation we used gradient descent method incorporated into simulated annealing. Let L be the parameter that determines the number of successful moves that will be considered at each temperature level. A larger value increases the optimization time, but tends to yield solutions with a narrower spread around the global optimum. The main idea of gradient descent method is very simple: among L possible moves we choose the one with minimal value of Δ. The suitable value of L is determined experimentally.

4 Computational experiments

An experimental version of the SA algorithm has been coded using Java Development Kit (JDK), version 5.0, Update 2.0, and implemented on a Pentium IV machine with 1.5 GHz clockpulse and 512 Mb RAM.[1]

Benchmark description. In order to evaluate the algorithm we use the family of benchmarks similar to [12] – [13]. Nevertheless, the instances considered are not identical due to random creation of parameters. We consider the family of random graphs $R-n-c-\delta$ that contain n vertices and $\lfloor \delta \cdot n(n-1) \rfloor$ arcs. Arcs are generated between randomly chosen pairs of vertices and interval costs are randomly set up in such a way that $0 < l_{ij} \leq u_{ij} \leq c \; \forall (i,j) \in A$. We generate the lower bound l_{ij} uniformly at random from interval $(0, \lfloor \frac{c}{2} \rfloor]$, and the upper bound u_{ij} from (l_{ij}, c).

Running time of SA. We compare the running time with the results of [6], [12], [13] and [14], where the fastest exact algorithms recently known have been presented.

The correlated running time τ (average value) in seconds is presented in Table 1. The first column represents the network. In the second column KPY_{MIP} the original running time of the algorithm (mixed integer program reformulation and direct solving with CPLEX) described in [6] is presented, while columns MG_{BB} and MGD_{IBB} summarize the performance of the major and improved methods based on branch and bound strategy proposed in [12] and [12], respectively. The two columns MG_{BD} and MG_{IBD} are devoted to the running time of the two most powerful exact algorithms which are based on Benders decomposition and reported in [14]. The last column N_{SA} presents the result of simulated annealing metaheuristic algorithm discussed in this paper.

Networks	KPY_{MIP}	MGD_{BB}	MGD_{IBB}	MG_{BD}	MG_{IBD}	N_{SA}
$R-500-100-0.01$	1.668	0.789	0.465	0.444	0.444	1.4
$R-500-100-0.10$	7.215	2.052	0.204	1.614	1.495	2.1
$R-900-1000-0.50$	857.155	-	48.316	862.168	140.141	15.3
$R-900-1000-0.90$	-	-	185.648	-	266.799	26.7

Table 1: Running time

Efficiency of SA. Due to the stochastic nature of our algorithm, a single run of it on a given instance is meaningless. A big number of repetitions is needed to give a representative view of the efficiency. The other complication is that the optimal solution of the problem is not known for large instances. Therefore we used three different benchmarks which have relatively small number of nodes and their construction is similar to the graph depicted in [6]. We first solve these benchmarks with CPLEX in order to get an exact optimal solution. Then we apply SA with standard parameters to these benchmarks. After running the SA algorithm $\zeta = 100$

[1]The code and the benchmark instances can be obtained from the author upon request.

times, we calculate the number ζ_1 of successful runs of SA, i.e. where the approximate solution is very close to optimal one (the gap between lower bound and approximate values of objective is less than the mean cost of one arc in the robust path; for large instance the gap defines $2\% - 5\%$ relative error of objective calculation). Then $p = \frac{\zeta_1+\zeta_2}{\zeta}$ gives a probability of success given a time measure τ. Thus, the pairs (τ, p) characterize efficiency of the algorithm with given parameters. These data are summarized in Table 2.

$\|V\|$	ζ	ζ_1	ζ_2	τ	$p\%$
10	100	63	9	0.15 sec	72%
20	100	45	20	0.18 sec	65%
30	100	36	32	0.2 sec	68%

Table 2: SA efficiency

5 Conclusion

In this paper we propose a simulated annealing metaheuristic for the robust shortest path problem where arc costs are not fixed but take their values from predefined intervals. For the simulated annealing algorithm we use boolean vector representation of the arc set and 1-1 flip of vector components as search moves. We compare our algorithm with the best procedures currently known. In spite of SA is generally to be used to find solutions of high quality in long runs and it is not oriented on quick performance, the algorithm we proposed is intended on speed as well as on quality of the output. It seems that the algorithm can be treated as quite effective (in most cases it finds a solution which represents a good approximation to the optimal one) and fast enough (the number of iterations and running time of the algorithm are reasonable). We are convinced that our approach can be considered as rather efficient to be used for large instances (networks with a thousand nodes and more as well as very dense graphs with δ greater than 0.5) when finding optimal solution with exact methods takes too much time.

Acknowledgements. The author is grateful to Andreas Drexl for useful suggestions and valuable remarks. The author is also thankful to Harm Brand for algorithm encoding.

References

[1] R. Ahuja, T. Magnanti and J. Orlin, Network flows: theory, algorithms and applications, Prentice-Hall, New Jersey, 1993.

[2] I. Averbakh and V. Lebedev, Interval data minmax regret network optimization problems. Discrete Applied Mathematics 138 (2004) 289 – 301.

[3] M. Demir, B. Tansel and G. Scheuenstuhl, The network 1-median location with Interval Data: A parameter space approach, IIE Transactions (to appear).

[4] E.W. Dijkstra, A note on two problems in connection with graphs. Numerische Mathematik 1 (1959) 269 – 271.

[5] M. Garey and D. Johnson, Computers and intractability. A guide to the theory of NP-completeness, Freeman and Co, San Francisco 1979.

[6] O. Karasan, M. Pinar and H. Yaman, The robust shortest path problem with interval data, Computers and Operations Research (revised, 2004: available online: *www.optimization – online.org/DB_HTML/2001/08/361.html*).

[7] S. Kirkpatrick, Optimization by simulated annealing – quantitative studies. Journal of Stat. Phys. 34 (1984) 975 – 986.

[8] S. Kirkpatrick, C. Gelatt and M. Vecchi, Optimization by simulated annealing. Science 220 (1983) 671 – 680.

[9] P. Kouvelis and G.Yu, Robust discrete optimization and its applications. Kluwer Academic Publishers, Norwell, M.A. 1997.

[10] P. van Laarhoven and E. Aarts. Simulated annealing: theory and applications. Reidel, Dordrecht, Holland 1987.

[11] E.Q.V. Martins and J.L.E. dos Santos. A new shortest paths ranking algorithm. Investigacao Operacional 20 (2000) 47 – 62.

[12] R. Montemanni and L.M. Gambardella, An exact algorithm for the robust shortest path problem with interval data. Computers and Operations Research 31 (2004) 1667 – 1680.

[13] R. Montemanni, L.M. Gambardella and A.V. Donati, A branch and bound algorithm for the robust shortest path problem with interval data, Operations Research Letters 32 (2004) 225 – 232.

[14] R. Montemanni and L.M. Gambardella, The robust shortest path problem with interval data via Benders decomposition. 4OR (to appear).

[15] Y. Nikulin, Robustness in combinatorial optimization and scheduling theory: an annotated bibliography, Manuskripte aus den Instituten für Betriebswirtschaftslehre No. 583, Christian-Albrechts-Universität zu Kiel, Germany 2004. (available online: *www.optimization – online.org/DB_FILE/2004/11/995.pdf*)

[16] Y. Nikulin, Simulated annealing algorithm for the robust spanning tree problem, Manuskripte aus den Instituten für Betriebswirtschaftslehre No. 591, Christian-Albrechts-Universität zu Kiel, Germany 2005.

[17] K. Thulasiraman and M.N.S. Swamz, Graphs: theory and algorithms, Wiley, New York, 1992.

[18] I. Wegener, Simulated annealing beats metropolis in combinatorial optimization. Electronic Colloquium on Computational Complexity, Report No. 8, Dortmund, Germany 2004.

[19] G. Yu and J. Yang, On the robust shortest path problem, Computers and Operations Research 25 (1998) 457 – 468.

[20] P. Zielinski, The computational complexity of the relative robust shortest path problem with interval data, European Journal of Operational Research 158 (2004) 570 – 576.