Nikulin, Yury

Simulated annealing algorithm for the robust spanning tree problem

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 591

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 591

Simulated annealing algorithm for the robust spanning tree problem

Yury Nikulin

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

Abstract

This paper addresses the robust spanning tree problem with interval data, i.e. the case of classical minimum spanning tree problem when edge weights are not fixed but take their values from some intervals associated with edges. The problem consists in finding a spanning tree 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 [8], 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 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 a metaheuristic approach for solving the robust spanning tree problem.

Keywords: robust spanning tree, simulated annealing, uncertainty.

1 Introduction

We consider the special case of a minimum spanning tree problem where the edge costs (weights) are not fixed but take their values from some intervals. No stochastic distribution is given inside intervals. The interval function is defined as the sum of interval weights over all edges of feasible spanning tree. This problem was first mentioned in [9], where some questions concerning solvability and computational complexity were studied. Contrary to the classical minimum spanning tree problem which can be easily solved by the algorithms of Kruskal (1956) or Prim (1957), minimum spanning trees of the interval variant depend on weights realization and the optimal objective value generally is not unique. Therefore, the authors of [9] proposed to introduce the relation on the set of intervals, which gives the possibility to transform the problem into a special bicriteria counterpart. The Pareto set of the counterpart, which can be generated by standard multiobjective methods, is taken to be the solution of the interval problem. It was shown that the counterpart problem is intractable, and it follows that the interval problem is also very hard to solve.

The special interest motivated by telecommunications applications induces not to solve the interval spanning tree 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. [8] as well as the more recent papers [3], [4] and the annotated bibliography [18]). As it was indicated in [8], in many cases the robust equivalent of a polynomially solvable problem becomes NP-hard.

Robust spanning tree problem was originally formulated in [8] for the case where edge costs are taken from some set of scenarios. It was proven that the problem is NP-hard [5] if the number of scenarios is bounded. Furthermore, strong NP-hardness of the problem for unbounded number of scenarios has been shown.

The basic theoretical background for the robust spanning tree problem has been presented in [23]. Two different types of robustness were introduced: absolute and relative robustness. It was proven that the absolute robust spanning tree problem can be easily resolved, whereas the relative robust spanning tree problem is very hard to solve. A reformulation of the last problem as a specific mixed integer program was presented. The concepts of weak and strong edges were introduced as well as polynomial time algorithms for their recognition were described. It was shown how these concepts can be efficiently used in a preprocessing stage for solving the relative robust spanning tree problem. We will shortly sketch out the main results from [23]

in section 3. In the remainder of the present paper only relative robustness is considered, and the problem is commonly referred to as robust spanning tree problem.
One more evidence of the NP-hardness of the robust spanning tree problem was presented in [2], where the relation between the well-known central tree problem [21], which is NP-hard, and the robust spanning tree problem is detected. It was shown that the robust spanning tree problem is at least as hard as the central tree problem. Moreover, the problem preserves its hardness on complete graphs, even though a central tree can be found in polynomial time on such graphs. Therefore, it was concluded that both factors – cost interval structure and topological graph properties – exert an essential influence on making the problem intractable from a computational point of view. Two cases where the problem can be easily solved are described: 1) no edge intervals have an intersection 2) the graph is complete and all edge costs take their value from the same interval. The question whether there are some other cases when the problem can be easily resolved is still open and presents a promising avenue for future research.
A branch and bound procedure, which embeds extension of some results previously presented in the literature as well as some other innovations, are presented in [15]. The results were compared with earlier results obtained by solving a mixed-integer program in [8] and with those of the branch and bound algorithm presented in [1]. It was shown that the algorithm outperforms all others and can be regarded as the best known exact algorithm for solving the robust spanning tree problem. Nevertheless, the proposed algorithm has the time complexity $O(|V|^{|V|-1}|E|^2 \log |E|)$ and cannot solve problem instances with $|V| > 25$ in reasonable time, where $V(E)$ is the set of vertices (edges). In [13] and [14] a new exact method based on Benders decomposition was described with respect to the robust spanning tree and the robust shortest path problem respectively. It was shown that this approach gives very good computational results on all the benchmarks considered, and especially on those that were harder to solve for the methods previously known.
The rest of the paper is organized as follows. In section 2 we introduce basic notations and formulate the problem. Section 3 is devoted to the main theoretical results. How to apply the simulated annealing metaheuristic is described in section 4. The results of computational experiments are presented in section 5. Final remarks appear in section 6.

2 Problem description

Let $G = (V, E)$ be a connected graph, where V is the set of vertices and E is the set of edges. With each edge $(i, j) \in E$ we associate a cost interval $[l_{ij}, u_{ij}]$, $0 < l_{ij} < u_{ij}$, i.e. For each edge $(i, j) \in E$ its cost c_{ij} is not fixed and belongs to $[l_{ij}, u_{ij}]$. No probability distribution is given inside the cost interval. Let

$$T := \{t \mid t = (V, E_t)\}$$

represent the set of all spanning trees of graph G. It is well-known that: a tree contains a unique spanning tree, a cycle graph C_n contains n spanning trees, and a complete graph K_n contains n^{n-2} spanning trees (see e.g. [12]). The matrix tree theorem, also called Kirchhoff's matrix-tree theorem, states that the number of nonidentical spanning trees (skeletons) of a graph G is equal to any cofactor of the degree matrix of G minus the adjacency matrix of G (see e.g. [21]).
The total cost c_t of spanning tree $t \in T$ is defined as:

$$c_t := \sum_{(i,j)\in E_t} c_{ij}.$$

It is obvious that $c_t = [l_t, u_t]$ where $l_t := \sum_{(i,j)\in E_t} l_{ij}$ and $u_t := \sum_{(i,j)\in E_t} u_{ij}$.
Then interval spanning tree problem is to find the spanning tree of minimum "weight":

$$\min_{t\in T} c_t. \tag{1}$$

Let s be a realization of edge costs, i.e. $c_{ij}^s \in [l_{ij}, u_{ij}]$ is fixed for any $(i,j) \in E$. In the classical non-interval case, i.e. when some scenario s is given and fixed, the problem is formulated as follows: among all the spanning trees of a weighted and connected graph find one (possibly more) with the least total cost, which is called a minimum spanning tree:

$$\min_{t\in T} c_t^s, \tag{2}$$

where

$$c_t^s := \sum_{(i,j)\in E_t} c_{ij}^s.$$

The minimum spanning tree may not be unique. However, if the weights of all the edges are pairwise distinct, it is indeed unique. Problem (2) can be easily solved by two greedy strategies, known as Kruskal's [10] and Prim's [19] algorithms, with time complexity $O(|E|\log|E|)$ and $O((|V|+|E|)\log|V|)$, respectively.
Concerning the interval case the authors of [9] introduced a partial order on the set of intervals. They proved that the interval spanning tree problem (2) can be reduced to a biobjective counterpart, where simultaneous minimization of l_t as a first objective and u_t as a second one over all spanning trees $t \in T$ is needed. The Pareto set of the biobjective optimization problem corresponds to the set of minimum spanning trees of the interval spanning tree problem. It was shown that problem (2) is very hard to solve in general. One can find some interesting two-stage techniques based on a k-best solution strategy for solving biobjective minimum spanning tree problem in [20].
The interval representation of edge costs may be interpreted as some sort of uncertainty for input data of the classical minimum spanning tree 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 subject 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 spanning tree t in a scenario s the difference between total cost of t and the cost of minimum spanning tee in s represents the deviation for t in scenario s:

$$dev_t^s := c_t^s - \min_{t\in T} c_t^s.$$

Let S be the set of all scenarios. For a given spanning tree $t \in T$ the worst-case scenario s_t is a scenario for which the deviation for t is maximum over all scenarios $s \in S$, i.e.

$$s_t := \arg\max_{s \in S} dev_t^s.$$

Then the difference

$$dev_t^{s_t} := c_t^{s_t} - \min_{t \in T} c_t^{s_t}$$

represents the robust deviation of t.
A spanning tree t' is said to be a robust spanning tree if it has the smallest robust deviation

$$t' := \arg\min_{t \in T} dev_t^{s_t}$$

among all spanning trees.

3 Theoretical background

In order to keep the paper self-contained we start with some theoretical background, originally presented in [23].
The following proposition gives a worst-case scenario for a given spanning tree.

Proposition 1 *[23] The scenario in which the costs of every edge on a spanning tree $t \in T$ is at its upper bound and the cost of every other edge is at its lower bound is a worst-case scenario, i.e. $c_{ij}^{s_t} = u_{ij}\ \forall (i,j) \in E_t$ and $c_{ij}^{s_t} = l_{ij}\ \forall (i,j) \in E \setminus E_t$.*

In other words, for any spanning tree $t \in T$ the worst case scenario s_t is uniquely determined.

Definition 1 *[23] A spanning tree t is a weak tree if it is a minimum spanning tree for some realization s of edge costs. An edge e is a weak edge if it belongs to some weak tree.*

The following theorem gives a characterization of weak trees and edges.

Theorem 1 *[23] A spanning tree $t \in T$ is a weak tree if and only if it is a minimum spanning tree when the cost of every edge on this tree corresponds to its lower bound and the costs of the other edges correspond to their upper bounds.*
Edge e is a weak edge if and only if there exists a minimum spanning tree using edge e where the cost of edge e is at its lower bound and the cost of the remaining edges are at their upper bounds.

Using theorem 1 we can recognize efficiently, i.e. in polynomial time, whether edge e is a weak edge in graph G by applying a slightly modified version of Kruskal's algorithm which is in favor of choosing edge e in case of ties.
The following statements give us an idea about the construction of a robust spanning tree.

Proposition 2 *[23] A robust spanning tree is a weak tree.*

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

Proposition 3 *[15] A non-weak edge e can be deleted from graph G when solving a robust spanning tree problem.*

The last result can be used efficiently in a preprocessing stage of any algorithm solving the robust spanning tree problem.

Definition 2 *[23] An edge e is a strong edge if it belongs to some minimum spanning tree for any realization s.*

Theorem 2 *[23] An edge e is a strong edge if and only if there exists a minimum spanning tree using edge e when the cost of edge e is at its upper bound and the cost of the remaining edges are at their lower bounds.*

Proposition 4 *[23] There exists a robust spanning tree such that every strong edge in the graph belongs to the tree.*

As in the case of weak edges we can recognize strong edges very efficiently using again a slightly modified version of Kruskal's algorithm. The strong edges can be detected and labelled in a preprocessing stage. They also have to be ultimately included in a solution of the robust spanning tree problem.

4 Simulated Annealing

Simulated annealing (SA) or hill climbing is a generic probabilistic heuristic approach originally proposed in [6] and [7] 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 [11].
Now let us precisely describe how to apply SA to the robust spanning tree problem.
As it was already mentioned in section 3, we can reduce the set of edges E in a preprocessing stage by deleting all non-weak edges and consider graph $G' := (V, E')$ with a new edge set E', $\varphi = |E'|$, which does not contain non-weak edges.

- *Search space.* Any subset of the edge set E' can be represented by a boolean vector $\pi \in \{0,1\}^{\varphi}$, such that $\pi_i = 1$ if edge e_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 edges describing a connected graph. The subsets which describe unconnected graphs 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 an acyclic graph or not. The reason of this is the following. As far as any connected graph with cycles contains some spanning tree and the value of objective of such graph is always greater then the value of objective for the spanning tree, then this spanning tree 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 connected graph.

- *Initial solution.* Initially, we choose $\pi^0 = (1, 1, \ldots, 1) \in \{0,1\}^\varphi$ as starting point. Alternatively, an initial search point can be generated randomly, but nevertheless the connectivity of the graph described by the point has to be guaranteed.

- *Neighbourhood search moves.* Let π be the current search point. We randomly chose $i \in \{1, \ldots, \varphi\}$ such that π_i does not represent a strong edge. Then we construct the neighbor search point π' by inverting π_i, i.e. $\pi' = (\pi'_1, \pi'_2, \ldots, \pi'_\varphi)$ 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. [22]), but not unusual for local search algorithms that accept only better solutions at each iteration. For example, in [17] the authors analyze randomized local search for minimum spanning tree problem with 1-bit and 2-bit flips. This algorithms do not select new search points which are worse than the old one. Observe that the application of this strategy to the robust spanning tree problem might be an interesting direction for future research.

  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 $\mathcal{E}$ represent an edge subset describing the current solution π. Case 1 corresponds to adding edge $e_i := (\tilde{i}, \tilde{j})$ to the edge set of graph $h := (V, \mathcal{E})$ described by the current search point π. Obviously, a new search point π' represents graph $h' := (V, \mathcal{E}')$, where $\mathcal{E}' := \mathcal{E} \cup \{e_i\}$. Since h is a connected graph, h' is also connected, therefore additional checking for connectivity is not necessary. According to Proposition 1, the worst-case scenario for solution π is the following: $c_{ij}^{s_h} = u_{ij}\ \forall (i,j) \in \mathcal{E}$ and $c_{ij}^{s_h} = l_{ij}\ \forall (i,j) \in E' \setminus \mathcal{E}$, whereas solution π' has the following worst-case scenario $s_{h'}$: $c_{ij}^{s_{h'}} = u_{ij}\ \forall (i,j) \in \mathcal{E}'$ and $c_{ij}^{s_{h'}} = l_{ij}\ \forall (i,j) \in E \setminus \mathcal{E}'$. 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 edge e_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:

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

$$= c_{h'}^{s_{h'}} - \min_{t \in T} c_t^{s_{h'}} - c_h^{s_h} + \min_{t \in T} c_t^{s_h} =$$

$$= u_{\tilde{i}\tilde{j}} + \min_{t \in T} c_t^{s_h} - \min_{t \in T} c_t^{s_{h'}}.$$

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

Case 2. This case corresponds to deleting edge $e_i := (\tilde{\imath}, \tilde{\jmath})$ from the edge set of graph $h := (V, \mathcal{E})$ described by the current search point π. Obviously, a new search point π' represents graph $h' := (V, \mathcal{E}')$, where $\mathcal{E}' := \mathcal{E} \setminus \{e_i\}$. Additional checking for connectivity of h' is necessary. If h' is disconnected, 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 $c_{\tilde{\imath}\tilde{\jmath}}^{s_h} = u_{\tilde{\imath}\tilde{\jmath}}$ of edge e_i with new cost $c_{\tilde{\imath}\tilde{\jmath}}^{s_{h'}} = l_{\tilde{\imath}\tilde{\jmath}}$. Then the cost charge Δ of performing a neighbourhood search move can be calculated as:

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

$$= c_{h'}^{s_{h'}} - \min_{t \in T} c_t^{s_{h'}} - c_h^{s_h} + \min_{t \in T} c_t^{s_h} =$$

$$= -u_{\tilde{\imath}\tilde{\jmath}} + \min_{t \in T} c_t^{s_h} - \min_{t \in T} c_t^{s_{h'}}.$$

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

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

- *Acceptance probability rule.* We use Metropolis acceptance rule, i.e. we 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 cost $c_{max} := \max_{(i,j) \in E'} u_{ij}$, namely $T_0 := 100 \cdot \varphi \cdot c_{max}$. 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 in practical calculation we use 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. For instances with small number of vertices (up to 10) we define this parameter equals to 10, for medium instances (with 15-20 vertices) we put $L = 30$ whereas for instances (the number of vertices is from 20 till 50) we put $L = 50$. The main idea of gradient descent method is very simple: among L possible moves we choose the one with minimal value of Δ.

5 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 1.7 GHz clockpulse and 512 Mb RAM.[1]

In order to evaluate the algorithm we use the family of benchmarks similar to [15]. Nevertheless, the instances considered are not identical due to random creation of parameters. We consider complete graphs with 10, 25 and 50 vertices. For each edge $(i,j) \in E$, we generate the lower bound l_{ij} uniformly at random from interval $(0,20)$, and the upper bound u_{ij} from $(l_{ij},40)$. We compare the running time with the result of [15], where the fastest exact algorithm recently known has been presented. We also have to take into account that for the results presented in [15] a Pentium II PC 400 MHz with 128 Mb RAM was used, whereas our machine is considered to be $12-15$ times faster. Therefore, we introduce the ratio $r = 0.07$, which is used for proper correlation of the results. The total running time τ (rounded average values) in seconds is presented in Table 1. In the first column the original running time of a branch and bound algorithm for the robust spanning tree problem (BB-RST) is presented, whereas in the second column this running time is correlated with the ratio r. As one can see, the SA algorithm can be effectively applied for large instances where all the known exact techniques are time consuming.

$\|V\|$	BB-RST	BB-RST	SA-RST
10	0.28	0.02	0.065
25	484	34	0.55
50	not presented	not presented	1.85

Table 1: Running time

We addressed four benchmark examples with 5, 10, 15 and 20 vertices. We used the mixed integer reformulation of the robust spanning tree problem given in [23], encoded the model with AMPL-language and solved the instances by ILOG CPLEX 7.0.0 solver in order to get an exact solution for the instances. 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. After running the algorithm $s = 100$ times, we calculate the number s_1 of successful runs of SA, i.e. where our metaheuristic algorithm was able to find the optimal solution, and the number s_2 of satisfactory runs of SA, i.e. where the approximate solution is very close to optimal one (the gap between optimal and approximate values of objective is less than the mean cost of one edge in the robust tree; for large instances the gap defines $2\% - 5\%$ relative error of objective calculation). Then $p = \frac{s_1+s_2}{s}$ 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.

$\|V\|$	s	s_1	s_2	τ	p
5	100	82	10	0.030 sec	92%
10	100	10	48	0.065 sec	58%
15	100	12	66	0.220 sec	78%
20	100	8	56	0.410 sec	64%

Table 2: SA efficiency

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

6 Conclusion

In this paper we propose a simulated annealing metaheuristic for the robust spanning tree problem where edge costs are not fixed but take their values from predefined intervals. During preprocessing we efficiently recognize the strong and weak edges, i.e. the edges that belong to some spanning tree for any or for some realization of edge costs, respectively. This recognition allows us to reduce the processing time of the main algorithm. For the simulated annealing algorithm we use boolean vector representation of the edge set and 1-1 flip of vector components as search moves. We compare our algorithm with the best procedures recently known. We are convinced that our approach can be considered as efficient to be used for large instances when finding optimal solution with exact methods takes too much time. We are also optimistic that using 2-2 flip together with 1-1 flip may slightly improve the algorithm, therefore it might be a promising area for the future research.

Acknowledgements. The author is grateful to Andreas Drexl for careful reading of the paper and for numerous suggestions that drastically improved the presentation. The author is also thankful to Harm Brand for algorithm encoding.

References

[1] I. Aron and P. van Hentenryck, A constraint satisfaction approach to the robust spanning tree problem with interval data. Computer Science Department, Brown University, Maz 2002, Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence, August 1 – 4, 2002, Edmonton, Canada.

[2] I. Aron and P. van Hentenryck, On the complexity of the robust spanning tree problem with interval data. Operations Research Letters 32 (2004) 36 – 40.

[3] A. Ben-Tal and A. Nemirovski, Robust solutions to uncertain programs. Operations Research Letters 25 (1999) 1 – 13.

[4] A. Ben-Tal and A. Nemirovski, Robust solutions of linear programming problems contaminated with uncertain data. Mathematical Programming 88 (2000) 411 – 424.

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

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

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

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

[9] G. Kozina and V. Perepelitsa, Interval spanning tree problem: solvability and computational complexity. Interval Computing 1 (1994) 42 – 50.

[10] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society 7 (1956) 48 – 50.

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

[12] O.Melnikov, R. Tyshkevich, V. Yemelichev and V. Sarvanov. Lectures on graph theory. BI Wissenschaftsverlag. Mannheim - Zurich. 1994.

[13] R. Montemanni, A Benders decomposition approach for the robust spanning tree problem with interval data. European Journal of Operational Research (to appear).

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

[15] R. Montemanni and L. Gambardella, A branch and bound algorithm for the robust spanning tree problem with interval data. European Journal of Operational Research 161 (2005) 771 – 779.

[16] J. Mulvey, R. Vanderbei and S. Zenios, Robust optimization of large-scale systems. Operations Research 43 (1995) 264 – 281.

[17] F. Neumann and I. Wegener, Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. In: Deb et al. (Eds.): GECCO 2004, LNCS 3102, Springer, Berlin, Germany, pages 713 – 724.

[18] 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.

[19] R.C. Prim, Shortest connection networks and some generalizations, Bell Systems Technology Journal 36 (1957) 1389 – 1401.

[20] S. Steiner and T. Radzik, Solving the biobjective minimum spanning tree problem using a k-best algorithm. Department of Computer Science King's College London, Technical Report TR-03-06, 2003.

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

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

[23] H. Yaman, O. Karasan and M. Pinar, The robust spanning tree problem with interval data. Operations Research Letters 29 (2001) 31 – 40.