O P E R A T I O N S R E S E A R C H A N D D E C I S I O N S
No. 1 2014
DOI: 10.5277/ord140103

Mariusz MAKUCHOWSKI*

PERTURBATION ALGORITHM FOR A MINIMAX REGRET MINIMUM SPANNING TREE PROBLEM

The problem of finding a robust spanning tree has been analysed. The problem consists of determining a minimum spanning tree of a graph with uncertain edge costs. We should determine a spanning tree that minimizes the difference in costs between the tree selected and the optimal tree. While doing this, all possible realizations of the edge costs should be taken into account. This issue belongs to the class of NP-hard problems. In this paper, an algorithm based on the cost perturbation method and adapted to the analysed problem has been proposed. The paper also contains the results of numerical experiments testing the effectiveness of the proposed algorithm and compares it with algorithms known in the literature. The research is based on a large number of various test examples taken from the literature.

Keywords: *discrete optimization, robust optimization, perturbation algorithms, minimax regret*

1. Introduction

In the real world, we often have to make a decision when the parameters are unknown, e.g. a decision about purchasing a summer swimming costume without a guarantee of a warm summer. The quality of the decision made can be evaluated only with hindsight. What is more, it is only from such a perspective that we see what the optimal action would have been on our part. Since at the time of making a decision we are unable to evaluate its actual value, often indirect evaluations are used such as an analysis of the worst case scenario or expected value. In this paper, the criterion evaluating a given decision is the minimization of maximal regret. What we call regret is the difference between the value obtained from a decision made and the hypothetical value obtained for the decision which, with hindsight, would have been regarded

*Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland, e-mail: mariusz.makuchowski@pwr.edu.pl

as optimal. As we are unable to foresee the future, we try to make sure that the maximum regret that may happen to us is as small as possible.

Some easy problems, i.e. those belonging to the *P* class, with precise data become issues that belong to the class of *NP*-hard problems when the parameters are changed into imprecise ones and the criterion is transformed to minimizing the maximum possible regret. This is also the case for the *NP*-hard minimax regret minimum spanning tree problem examined in this paper, which derives from a classical easy minimum spanning tree problem.

2. Problem of finding a robust spanning tree

One of the fundamental problems of operations research is determining a minimum cost spanning tree. A spanning tree t of a connected, undirected graph $G = (V, E)$ is any acyclic connected subgraph of this graph which contains all the vertices from V. We will use Γ to denote the set of all the spanning trees of a given graph. The number of the spanning trees of a graph depends both on its size and structure. For complete graphs, it depends exponentially on the number of nodes in the graph and is given by:

$$\left|\Gamma\right| = \left|V\right|^{|V|-2} \tag{1}$$

For graphs with weighted edges, we can determine the cost of any spanning tree as the sum of the weights of its edges. The problem of finding a minimum cost spanning tree consists of choosing the tree with the lowest cost from all the trees spanning a given graph. This problem is solved in polynomial time by the algorithm proposed in [10, 13].

In the analysed problem, the weights of the edges are not known. For each edge $e \in E$, the range of its possible realizations is given by $\left[c_e^-, c_e^+\right]$. A specific realization of all the edge weights is called a scenario and denoted by S. We will use Ω to denote the set of all possible scenarios. The cost of an edge of weight $e \in E$ in the scenario $s \in \Omega$ will be denoted by $c_e^s \in \left[c_e^-, c_e^+\right]$. For a given tree $t \in \Gamma$, its cost depends on the scenario $s \in \Omega$ and is given by:

$$F\left(t, s\right) = \sum_{e \in t} c_e^s \tag{2}$$

For the scenario $s \in \Omega$ we can determine a minimum cost spanning tree, whose total cost is:

$$F^*(s) = \min_{t \in \Gamma} F(t, s) \tag{3}$$

For each tree $t \in \Gamma$ and scenario $s \in \Omega$ we can determine the regret which is the difference between the cost of a given tree and the cost of the minimum spanning tree for a given scenario:

$$Z(t, s) = F(t, s) - F^*(s) \tag{4}$$

The maximal regret of the tree $t \in \Gamma$ is defined in the following way:

$$Z(t) = \max_{s \in \Omega} Z(t, s) = \max_{s \in \Omega} \{F(t, s) - F^*(s)\} \tag{5}$$

The problem analysed in this paper is to find the tree $t^* \in \Gamma$ with the minimum value of the maximal regret, i.e.:

$$t^* \in \arg\min_{t \in \Gamma} Z(t) \tag{6}$$

3. Overview of the literature

The minimax regret minimum spanning tree problem (6) is strongly *NP*-hard [2]. In paper [3], a special case is analysed in which all the edge costs can have values from the interval [0, 1]. This subset of problems is called the central spanning tree problem and also belongs to the class of *NP*-hard problems.

The first method for providing an exact solution to the problem examined was proposed in [15]. This method was based on mixed integer programming (MIP) and using it, the authors solved instances having up to 25 nodes in a graph. In [1, 11] algorithms based on the branch and bound method were proposed which were able to provide solutions for graphs having up to 20–40 nodes.

In the paper [8], a 2-approximation algorithm was proposed, whereas [12] presented an approximate improvement algorithm based on the method of simulated annealing. The most efficient, in terms of the quality of the solutions obtained for large instances (which cannot be solved exactly in reasonable time using algorithms), is the algorithm proposed in [7] based on the tabu search method.

A similar problem to the minimax regret minimum spanning tree was studied in [4], namely the problem of finding minimax regret minimum spanning arborescences with interval costs in directed graphs. In contrast to our problem, the problem

of finding minimax regret minimum spanning arborescences with interval costs turned out to be polynomially solvable for directed acyclic graphs.

A comprehensive treatment of the state of the art in robust discrete optimization, applications of this framework to several combinatorial problems and extensive references can be found in books [5, 9].

4. Perturbation algorithm

Perturbation algorithms are modifications of existing algorithms, most often construction algorithms. The idea of a perturbation algorithm consists of carrying out multiple perturbations of the input data and then solving the resulting problem with the help of a certain algorithm, called a base algorithm. The solution of the perturbed problem obtained in this way is tested on the original, unchanged input data. Because perturbations are random, a perturbation algorithm can perform a whole series of analogous iterations. After performing a fixed number of iterations, the algorithm stops and returns the best-found solutions (in terms of the values of the objective function for the original data). The pseudo-code of such a perturbation algorithm is shown in Fig. 1.

```
Step 1:    enter the data D for the problem

Step 2:    repeat steps 2a, 2b, 2c n-times

Step 2a:   randomly perturb the original data D creating data D'

Step 2b:   initiate Base Algorithm on perturbed data D' obtaining
           solution x'

Step 2c:   if for the original data D, x' is a better solution
           than the best solution found previously x* then set x*=x'

Step 3:    return the best solution found x*
```

Fig. 1. Pseudo-code of a perturbation algorithm

Undoubtedly, the basic element of each perturbation algorithm is a fast construction algorithm which provides high quality solutions. In the problem analysed, these will be the two algorithms, *AM* and *AU*, described in [6]. For discrete optimization problems, in which the input data (just like in the problem examined) is a graph with weighted edges, various variants of cost perturbations are applied. In the paper [14], various algorithms are proposed for perturbing all the weights of the edges. Moreover,

appropriate results of perturbation are given in terms of the intensification or diversification of calculations. Intensification of calculations is obtained by perturbing the costs in such a way that they promote the edges chosen in previous iterations of the algorithm, whereas diversification is obtained by perturbing the costs in the opposite way.

4.1. Perturbation of the data

The proposed perturbation of the data consists of changing the range of the weights of the edges from $\left[c_e^-, c_e^+\right]$ to $\left[c_e^-, c_e^*\right]$. With the probability $p = 0.2$, the new value c_e^* of the upper bound is drawn from the interval $\left[c_e^+, kc_e^+\right]$, where $k = 1.1$. Otherwise, it is not modified, i.e. $c_e^* = c_e^+$. The values p and k were chosen in an experimental way.

5. Numerical calculations

All the results presented, in particular the algorithm running times, were obtained on a PC equipped with a Core2Duo E6750 2.66GHz, 2GB RAM processor, running on a 32-bit Windows 7 system. The algorithms were programmed in the DEV C++ 4.9.9.1 environment.

5.1. Notation to describe algorithms

AR – an algorithm that provides a random solution.

AM – an algorithm that provides a solution which is a minimum cost spanning tree in the case of deterministic edge weights where the values of the edge weights are defined to be the midpoints of the intervals in the initial problem.

AU – an algorithm that provides a solution in the form of a minimum cost spanning tree in the case of deterministic edge weights where the values of the edge weights are determined to be the upper endpoints of the intervals in the initial problem.

AMU – returns the better solution from the two obtained by the algorithms *AM* and *AU*.

PMU – returns the better solution from the solutions obtained by the algorithms *AM* and *AU* for the perturbed data.

Algorithms denoted by X_n, $X \in \{AR, AMU, PMU\}$, perform n steps. In each step, they call algorithm X only once, obtaining in this way a set of n solutions. The ultimate solution returned by the algorithm X_n is the best solution from this set.

Algorithms denoted by $X + LS$ function in two phases. First, algorithm X, which provides a starting solution, is called. Then the obtained solution is improved by the local improvement algorithm LS. The latter phase involves a few steps. In each step i, a basic solution B_i is defined. In the former step, the basic solution is the initial solution obtained in the first phase of the algorithm. In every step of this phase, the LS algorithm generates the neighborhood of the basic solution $N(B_i)$. Then the best solution Y_i is selected from this neighborhood. If the selected solution Y_i is better than the current base solution B_i, the algorithm proceeds to the next step $i + 1$, in which the solution Y_i becomes the base solution, i.e. $B_{i+1} := Y_i$. If in step i the solution Y_i is not better than the basic solution B_i, then the algorithm stops and returns its current basic solution B_i.

The neighborhood of the current solution (spanning tree of the given graph G) used in the improvement algorithm LS is the set of spanning trees of the graph G differing in exactly one edge from the current solution, i.e.:

$$N(t) = \left\{ t' \in \Gamma : \left| E(t) \setminus E(t') \right| = 1 \right\} \qquad \text{for } t \in \Gamma \tag{7}$$

5.2. Test examples

The presented algorithms were compared using examples generated according to the concepts of researchers who have been analyzing this problem. The first two classes of examples *Ya*1, *Ya*4 were proposed in [15]. This paper describes the construction of six classes of instances, from which, for the purpose of this research, classes 1 and 4 were selected. Two further classes of instances *He*1, *He*2 have been described in [1]. The next two classes *Mo*1, *Mo*3 were generated according to a formula contained in [11], where from the 3 classes of the instances proposed classes 1 and 3 were selected. In the literature [5, 7], other well-known difficult test examples (*Ka*, *La*) are presented concerning such a specific case as instances of a central spanning tree. In such instances, all the weights in graphs are described by the same interval [0, 1]. The difficulty for these instances results only from the topology of the graph, and not, as in previous instances, from the different range of edge weights.

Each class consists of 8 groups and each group has 10 instances. The graphs in a group have the same number of vertices, which varies from 10 to 80. Hence, in the study we used a total of 640 test examples.

5.3. Reference solutions

For each instance, there was a reference solution of the value Z_{ref}, which is used to determine regret. The method of obtaining a reference solution depended on the instance. For instances from class *Ka*, the values Z_{ref} were obtained through the analytical derivation of optimal solutions. For instances from other classes, we tried to obtain an exact solution using the CPLEX [16] solver. If the solver was not able to find an optimal solution within a few hours, the instance was then solved with the help of numerically intensive calculations performed by approximation algorithms of the *TS* type. The best of the solutions obtained was chosen as the reference one.

5.4. Method of evaluating the algorithms

A single base study of a selected algorithm for a particular instance of the problem provides information about r – the average regret of the obtained solution with respect to the reference solution, d – standard deviation of the regret, and time – average run time of the algorithm. Each algorithm is run k times for each instance and the Z_{ref} value is calculated to determine regret. Each time we run algorithms, the pseudo-random number generator starts in a different state. This study provides k solutions accompanied with the corresponding set of regret values and the average run time of the algorithm. Then, the following values are determined:

$$r = \left(av\left(Z\right) - Z_{\text{ref}}\right)\frac{100\%}{Z_{\text{ref}}} \tag{8}$$

$$d = \sqrt{av\left(Z^2\right) - av\left(Z\right)^2}\ \sqrt{\frac{k}{k-1}}\frac{100\%}{Z_{\text{ref}}} \tag{9}$$

5.5. Test 1

In this test, the algorithms AR_{1000}, *AMU*, and PMU_{10} were compared with one another. The study involved finding solutions to a set of 640 instances (8 classes in 8 groups of 10 instances) and the derivation of the values of r, d and time for each combination of algorithm and instance. The mean values $av(r)$, $av(d)$, av(time) for each of the classes for the algorithms discussed are presented in Table 1.

Table 1. Mean values *av*(*r*), *av*(*d*), *av*(time) for each of the classes for the algorithms discussed

Class of instants	AR_{1000}			*AMU*		
	av(*r*) [%]	*av*(*d*) [%]	*av*(time) [ms]	*av*(*r*) [%]	*av*(*d*) [%]	*av*(time) [ms]
*Ya*1	243.47	7.85	73	0.97	0.00	4
*Ya*2	257.97	9.41	84	0.27	0.00	16
*He*1	2235.00	72.88	95	3.31	0.00	27
*He*2	1035.56	67.03	63	3.21	0.00	28
*Mo*1	15307.57	617.32	103	0.69	0.00	35
*Mo*3	865.34	36.27	111	1.73	0.00	43
Ka	68.03	2.70	90	82.46	3.97	43
La	20.58	0.85	90	26.71	1.46	47
Total	2504.19	101.79	88	14.91	0.00	30
Class of instants	PMU_{10}			PMU_{100}		
	av(*r*) [%]	*av*(*d*) [%]	*av*(time) [ms]	*av*(*r*) [%]	*av*(*d*) [%]	*av*(time) [ms]
*Ya*1	0.72	0.17	35	0.52	0.06	338
*Ya*2	0.17	0.05	47	0.13	0.01	350
*He*1	2.52	0.70	58	1.67	0.22	362
*He*2	2.19	0.71	30	1.42	0.22	54
*Mo*1	0.27	0.26	65	0.08	0.05	368
*Mo*3	1.17	0.24	73	0.86	0.13	377
Ka	76.21	3.25	46	71.05	2.34	70
La	24.04	1.96	50	21.03	0.86	72
Total	13.41	0.92	51	12.10	0.49	249

5.6. Test 2

Another test shows the effect of a change in the type of perturbation on the quality of an algorithm. For this purpose, a further two versions of the perturbation algorithm: PMU^a and PMU^b were created. In these algorithms a new value c_e^* of the upper bound of the range of weights is drawn for each edge of a tree. The upper bounds are perturbed by algorithm PMU^a in a way that draws c_e^*, the new value of the upper bound, from the interval $[c_e^+, 1.01c_e^+]$. Using algorithm PMU^b, the range of the values of c_e^* drawn increases with the number of perturbations performed:

$$c_e^* \in \left[c_e^+, \left(1 + 0.1 \frac{i}{maxIter} \right) c_e^+ \right]$$

where i is the number of the current iteration and *maxIter* – the number of the last iteration in the algorithm. The parameters in these relationships were selected in an experimental way. The test results for the described methods of perturbation are shown in Table 2.

Table 2. Test results for the described methods of perturbation

Class of instants	PMU_{10}^{a}			PMU_{10}^{b}		
	av(*r*) [%]	*av*(*d*) [%]	*av*(time) [ms]	*av*(*r*) [%]	*av*(*d*) [%]	*av*(time) [ms]
*Ya*1	0.88	0.03	35	0.70	0.11	35
*Ya*2	0.25	0.01	46	0.16	0.02	47
*He*1	2.65	0.21	58	2.63	0.49	58
*He*2	2.53	0.31	31	2.33	0.50	31
*Mo*1	0.52	0.04	66	0.22	0.12	66
*Mo*3	1.59	0.02	74	1.19	0.18	73
Ka	76.88	3.56	48	77.17	3.60	47
La	24.88	1.71	52	25.10	1.87	51
Total	13.77	0.74	51	13.69	0.86	51

5.7. Test 3

In the latter test, solutions are created with the use of a top-down method obtained from the solutions provided by the algorithms presented in the former test. The mean values *av*(*r*), *av*(*d*) and *av*(time) for each of the classes of algorithms discussed, $AR_{1000} + LS$, $AMU + LS$, $PMU_{10} + LS$, and $PMU_{100} + LS$ are presented in Table 3.

Table 3. Mean values *av*(*r*), *av*(*d*), *av*(time) for each of the classes for the algorithms discussed in Section 5.7

Class of instants	$AR_{1000} + LS$			$AMU + LS$		
	av(*r*) [%]	*av*(*d*) [%]	*av*(time) [ms]	*av*(*r*) [%]	*av*(*d*) [%]	*av*(time) [ms]
*Ya*1	0.01	0.02	1177	0.01	0.00	59
*Ya*2	0.00	0.00	1218	0.00	0.00	56
*He*1	0.07	0.10	801	0.07	0.00	76
*He*2	0.17	0.20	166	0.19	0.00	44
*Mo*1	0.00	0.00	655	0.00	0.00	46
*Mo*3	0.00	0.00	934	0.00	0.00	99
Ka	44.62	8.20	191	39.70	10.41	201
La	19.40	2.03	109	21.51	2.24	72
Total	8.03	1.32	656	7.69	1.58	82

Class of instants	$PMU_{10} + LS$			$PMU_{100} + LS$		
	$av(r)$ [%]	$av(d)$ [%]	av(time) [ms]	$av(r)$ [%]	$av(d)$ [%]	av(time) [ms]
*Ya*1	0.00	0.01	86	0.00	0.00	390
*Ya*2	0.00	0.00	85	0.00	0.00	389
*He*1	0.06	0.00	106	0.06	0.00	410
*He*2	0.19	0.00	45	0.20	0.00	68
*Mo*1	0.00	0.00	77	0.00	0.00	389
*Mo*3	0.00	0.00	126	0.00	0.00	427
Ka	43.44	9.58	178	44.74	9.08	179
La	20.61	2.68	71	19.62	2.05	88
Total	8.04	1.53	97	8.08	1.39	293

5.8. Commentary on the numerical calculations

In instances *Ka* and *La*, in which all the ranges of the edge weights are the same, the algorithms *AMU*, AR_2, and PMU_1 are equivalent. This is due to the fact that the algorithm *AMU* is influenced only by these intervals. Each edge gets the same priority, which determines the order in which they are attached to a tree. In algorithm *AR*, priority is ascribed at random. In the case of identical priority, this order is not defined and can be either random or fixed in any way.

For the instances *Ka* and *La* mentioned above, the algorithms AR_{1000}, PMU_{100}, PMU_{10}, and *AMU* choose the best solution from among the randomly selected 1000, 200, 20 and 2 solutions. For classes *Ka* and *La*, the theoretical predictions concerning the quality of the solutions $AR_{1000}{:}av(r) < PMU_{100}{:}av(r) < PMU_{10}{:}av(r) < AMU{:}av(r)$ coincide exactly with our numerical studies.

In the case of instances from the classes *Ya*, *He*, *Mo*, although algorithm AR_{1000} generates a large number of solutions, it generates much weaker solutions than algorithm *AMU*. Algorithms PMU_{10} and PMU_{100} generate 10 and 100 different solutions, respectively, with the quality of each of them being close to the quality of the solution obtained using *AMU*. For classes *Ya*, *He*, *Mo*, both the theoretical and experimental relationships between the quality of the solutions are as follows: $PMU_{100}{:}av(r) < PMU_{10}{:}av(r) < AMU{:}av(r)$.

Perturbation algorithms of the PMU_n type provide better solutions than the current best construction algorithm *AMU*. Generating better solutions is burdened with the cost of the longer run time of the algorithm PMU_n in comparison to the runtime of algorithm *AMU*. Moreover, the designer can manipulate the run time of the algorithm PMU_n by changing the parameter n. By increasing the value of n we can improve the quality of the solutions obtained at the expense of increasing the run time of an algorithm.

During the study, a number of various ways of perturbing data were performed. These methods differ in the amount of data perturbed and intensity of perturbation. In addition, the intensity of data perturbation in successive steps of the algorithm was either stable or underwent some changes. Changes in the perturbation intensity depended on the information obtained in previous solutions, or varied linearly. The results for the two methods of perturbation used are presented in Table 2. The quality of the algorithms PMU^a and PMU^b is not better than that of algorithm *PMU*. Other methods of perturbation did not give any significant improvement with respect to perturbations used in the algorithm *PMU*.

The use of a top-down algorithm *LS* in the problem analyzed gives a significant improvement in the solutions obtained by algorithms AR_{1000}, *AMU*, PMU_{10} and PMU_{100}. For instances from the classes *Ya*, *He*, *Mo*, the quality of our solutions is outstanding. The relative error with respect to the reference solutions is less than 0.1%, even if the error for the initial solutions was more than 10 000%. There is no evidence that any of the tested initial algorithms is more efficient than the others. Since the quality of the solutions $AR_{1000} + LS{:}av(r) \approx AMU + LS{:}av(r) \approx PMU_{10} + LS{:}av(r) \approx PMU_{100} + LS{:}av(r)$ is similar, we propose to use the computationally fastest solution, which is algorithm *AMU* + *LS*.

In the case of instances *Ka* and *La* solved by the local improvement algorithm *LS*, the initial algorithms AR_{1000}, *AMU*, PMU_{10}, and PMU_{100} differ only in the number of randomly generated solutions from which the best one is chosen. The lack of a positive correlation between the quality of the initial solution and the quality of the final solution leads us to use algorithm *AMU* which, for the instances *Ka* and *La*, is equivalent to random choice of any solution.

In the case of multiple starts in the improvement algorithm, we suggest using a perturbation algorithm with a sufficiently large perturbation, so that future solutions differ from one another. The feature of providing diverse solutions is also possessed by algorithm *AR*. However, due to the longer process of improving the solutions provided by *AR* , algorithm PMU_n is preferred.

6. Summary

If we use only a single algorithm to generate a final solution, we suggest using the algorithm PMU_n. This algorithm provides better quality solutions than *AMU*, the best known construction algorithm in the literature. Additionally, there is the possibility of extending the run time of algorithm PMU_n by increasing the parameter n, in order to obtain further improvement in the quality of solutions.

If we apply a single algorithm to generate a starting solution for a single run of the improvement algorithm, we propose algorithm *AMU*. The quality of the solutions gen-

erated by the improvement algorithm does not differ from the ones generated using the other initial algorithms. The time used to obtain initial solutions accompanied by the performance-enhancing procedure is shorter than for the other initial algorithms.

In the case of applying an algorithm to generate initial solutions for improvement algorithms using multiple starts, we propose algorithm PMU_n with a small value n and a large perturbation. The perturbations should be strong enough to provide different starting solutions with a high probability.

Acknowledgements

This work was supported by the Polish Committee for Scientific Research, grant N N206 492938.

References

[1] ARON I., VAN HENTENRYCK P., *A constraint satisfaction approach to the robust spanning tree with interval data*, Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence, Edmondton, Canada, 2002.

[2] ARON I., VAN HENTENRYCK P., *On the complexity of the robust spanning tree problem with interval data*, Operations Research Letters, 2003, 32, 36–40.

[3] BEZRUKOV S., KADERALI F., POGUNTKE W., *On central spanning trees of a graph*, Lecture Notes Computer Science, 1996, 1120, 53–58.

[4] CONDE E., CANADIA A., *Minimax regret spanning arborescences under uncertain costs*, European Journal of Operational Research, 2007, 182 (2), 561–577.

[5] KASPERSKI A., *Discrete optimization with interval data: Minimax regret and fuzzy approach*, Studies in Fuzziness and Soft Computing, Springer-Verlag, Berlin 2008, 228.

[6] KASPERSKI A., KOBYLAŃSKI P., KULEJ M., ZIELIŃSKI P., *Minimizing maximal regret in discrete optimization problems with interval data*, [in:] *Issues in Soft Computing Decisions and Operations Research*, O. Hryniewicz, J. Kacprzyk, D. Kuchta (Eds.), Akademicka Oficyna Wydawnicza EXIT, Warsaw 2005, 193–208.

[7] KASPERSKI A., MAKUCHOWSKI M., ZIELIŃSKI P., *A tabu search algorithm for the minimax regret minimum spanning tree problem with interval data*, Journal of Heuristics, 2012, 18 (4), 593–625.

[8] KASPERSKI A., ZIELIŃSKI P., *An approximation algorithm for interval data minmax regret combinatorial optimization problems*, Information Processing Letters, 2006, 97 (5), 177–180.

[9] KOUVELIS P., YU G., *Robust discrete optimization and its applications*, Kluwer Academic Publishers, 1997.

[10] KRUSKAL J.B., *On the shortest spanning subtree of graph and the traveling salesman problem*, Proc. Amer. Math. Soc., 1956, 7, 48–50.

[11] MONTEMANNI R., GAMBARDELLA L.M., *A branch and bound algorithm for robust spanning tree problem with interval data*, Operations Research Letters, 2005, 161, 771–779.

[12] NIKULIN Y., *Simulated annealing algorithm for the robust spanning tree problem*, Journal of Heuristics, 2007, 14, 391–402.

[13] PRIM R.C., *Shortest connection networks and some generalizations*, Bell System Technical Journal, 1957, 36, 1389–1401.

[14] RIBEIRO C., UCHOA E., WERNECK R., *A hybrid GRASP with perturbations for the Steiner problem in graphs*, Technical Report, Computer Science Department, Catholic University of Rio de Janeiro, 2002.

[15] YAMAN H., KARASAN O.E., PINAR M.C., *The robust spanning tree with interval data*, Operations Research Letters, 2001, 29, 31–40.

[16] IBM ILOG, CPLEX Optimizer, <http://www.ibm.com/software/commerce/optimization//cplex-optimizer>.

Received 4 May 2013
Accepted 23 January 2014