A Simulated Annealing Approach for the Minmax Regret Path Problem

Francisco Pérez, **César A. Astudillo**, **Matthew Bardeen**, **Alfredo Candia-Véjar**
fperez@gmail.com, {castudillo, mbardeen, acandia}@utalca.cl
Universidad de Talca
Km 1. Camino a los Niches Curicó, Chile

August 5, 2012

Abstract

We propose a novel neighbor generation method for a Simulated Annealing (SA) algorithm used to solve the Minmax Regret Path problem with Interval Data, a difficult problem in Combinatorial Optimization. The problem is defined on a graph where there exists uncertainty in the edge lengths; it is assumed that the uncertainty is deterministic and only the upper and lower bounds are known. A worst-case criterion is assumed for optimization purposes. The goal is to find a path s-t, which minimizes the maximum regret. The literature includes algorithms that solve the classic version of the shortest paths problem efficiently. However, the variant that we study in this manuscript is known to be NP-Hard. We propose a SA algorithm to tackle the aforementioned problem, and we show that our implementation is able to find good solutions in reasonable times for large size instances. Furthermore, a known exact algorithm that utilizes a Mixed Integer Programming (MIP) formulation was implemented by using a commercial solver (CPLEX[1]). We show that this MIP formulation is able to solve instances up to 1000 nodes within a reasonable amount of time.

1 Introduction

In this work we study a variant of the well known Shortest Path (SP) problem. For the classical version of this problem, efficient algorithms have been known since 1959 (Dijkstra, 1959). Given a digraph $G = (V, A)$ (V is the set of nodes and A is the set of arcs) with non-negative arc costs associated to each arc and two nodes s and t in V, SP consists of finding a s-t path of minimum total cost. Dijkstra designed a polynomial time algorithm and from this, a number of other approaches have been proposed. Ahuja *et al.* present the different algorithmic alternatives to solve the problem (Ahuja et al., 1993).

Our interest is focused on the variant of shortest path problems where there exists uncertainty in the objective function parameters. In this SP problem, for each arc we have a closed interval defining the possibilities for the arc length. A *scenario* is a vector where each number represents one element of an arc length interval. The uncertainty model used here is the *minmax regret* approach (MMR), sometimes named *robust deviation*; in this model the problem is to find a feasible solution being α-optimal for any possible scenario with α as small as possible. One of the properties of the minmax regret model is that it is not as pessimistic as the (*absolute*) *minmax* model. This model

[1]Although popularly referred to simply as CPLEX, its formal name is IBM ILOG CPLEX Optimization Studio. For additional information, the interested reader may consult the following URL: `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`

in combinatorial optimization has largely been studied only recently: see the books by Kouvelis and Yu (Kouvelis and G., 1997), and Kasperski (Kasperski, 2008), as well as the recent reviews by Aissi *et al.* (Aissi et al., 2009) and Candia-Véjar *et al.* (Candia-Véjar et al., 2011). The later also mentions some interesting applications of the MMR model in the real world.

It is known that minmax regret combinatorial optimization problems with interval data (MMRCO) are usually NP-hard, even in the case when the classic problem is easy to solve; this is the case of the minimum spanning tree problem, shortest path problem, assignment problems and others; see Kasperski (2008) for details.

Exact algorithms for Minmax Regret Paths have been proposed by (Karasan et al., 2001; Kasperski, 2008; Montemanni and Gambardella, 2004, 2005). All these papers show that exact solutions for MMRP can be obtained by different methods and take into account several types of graphs and degrees of uncertainty. However, the size of the graphs tested in these papers was limited to a maximum of 2000 nodes.

In this context, our main contributions in this paper are the analysis of the performance of the CPLEX solver for a MIP formulation of MMRP, the analysis of the performance of known heuristics for the problem and finally the analysis of the performance of a proposed SA approach for the problem. For experiments we consider two classes of networks, random and a class of networks used in telecommunications and both containing different problem sizes. Instances containing from 500 to 20000 nodes with different degrees of uncertainty were considered.

In the next section, we present the formal definition of the problem and notation associated and in Section 3 a mathematical programming formulation for MMRP is presented. We also discuss the midpoint scenario and the upper limit scenario heuristics for MMRP in more detail and then present the general algorithm for SA. In Section 4 we formally define the neighborhood used for our SA approach. Details of our experiments and their results are analyzed in Section 5. Finally in Section 6, our conclusions and suggestions for future work are presented.

2 Notation and Problem Definition

Let $G = (V, A)$ be a digraph, where V corresponds to the set of vertices and A is conformed by a set of arcs. With each arc (i, j) in A we associate a non-negative cost interval $[c_{ij}^-, c_{ij}^+]$, $c_{ij}^- \leq c_{ij}^+$, i.e., there is uncertainty regarding the true cost of the arc (i, j), and whose value is a real number that falls somewhere in the above-mentioned interval. Additionally, we make no particular assumptions regarding the probability distribution of the unknown costs.

The Cartesian product of the uncertainty intervals $\left[c_{ij}^-, c_{ij}^+\right]$, $(i, j)\epsilon A$, is denoted as S and any element s of S is called a *scenario*; S is the vector of all possible realizations of the costs of arcs. c_{ij}^s, $(i, j)\epsilon A$ denote the costs corresponding to scenario s.

Let *P* be the set of all s-t paths in G. For each $X\epsilon P$ and $s\epsilon S$, let

$$F(s, X) = \sum_{(i,j)\epsilon X} c_{ij}^s, \tag{1}$$

be the cost of the s-t path X in the scenario s.

The classical s-t shortest path problem for a fixed scenario $s\epsilon S$ is

Problem OPT.PATH(s). Minimize $F(s, X) : X\epsilon P$.

Let $F^*(s)$ be the optimum objective value for problem *OPT.PATH(s).*

For any $X\epsilon P$ and $s\epsilon S$, the value $R(s, X) = F(s, X) - F^*(s)$ is called the *regret* for X under scenario s. For any $X\epsilon P$, the value

$$Z(X) = max_{s\epsilon S} R(s, X), \tag{2}$$

is called the *maximum* (or *worst-case*) *regret* for X and an optimal scenario $s*$ producing such a worst-case regret is called *worst-case scenario* for X. The minmax regret version of Problem *OPT.PATH(s)* is

Problem MMRP. Minimize $\{Z(X) : X \epsilon P\}$.

Let Z^* denote the optimum objective value for Problem *MMRP*.
For any $X \epsilon P$, the *scenario induced by* X, $s(X)$, for each $(i,j) \epsilon A$ is defined by

$$c_{ij}^{s(X)} = \begin{cases} c_{ij}^{+}, & (i,j)\, \epsilon X \\ c_{ij}^{-}, & \text{otherwise.} \end{cases} \tag{3}$$

Let $Y(s)$ denote an optimal solution to Problem *OPT.PATH(s).*

Lemma 1 (Karasan *et al.* (Karasan et al., 2001)). $s(X)$ is a worst-case scenario for X.

According to Lemma 1, for any $X \epsilon P$, the worst-case regret

$$\begin{aligned} Z(X) &= F(s(X), X) - F^*(s(X)) \\ &= F(s(X), X) - F(s(X), Y(s(X)), \end{aligned} \tag{4}$$

can be computed by solving just one classic SP problem according to the definition of $Y(S)$ given above.

3 Algorithms for MMRP

In this section we present both a MIP formulation for MMRP, which will be used to obtain an exact solution by using a solver CPLEX, and our SA approach for finding an approximate solution for the problem. Two simple and known heuristics based on the definition of specific scenarios are also presented.

3.1 A MIP Formulation for MMRP

Consider a digraph $G = (V, A)$ with two distinguished nodes s and t. According with the past section each arc $(i,j) \in A$ has interval weight $\left[c_{ij}^{-}, c_{ij}^{+}\right]$ and also has a binary variable x_{ij} associated expressing if the arc (i,j) is part of the constructed path. We use Kasperski's MIP formulation of MMRP (Kasperski, 2008), given as follows:

$$\min \sum_{(i,j) \in A} c_{ij}^{+} x_{ij} - \lambda_s + \lambda_t \tag{5}$$

$$\lambda_i \leq \lambda_j + c_{ij}^{+} x_{ij} + c_{ij}^{-}(1 - x_{ij}),\ (i,j) \in A, \lambda \in \mathbb{R} \tag{6}$$

$$\sum_{\{i:(j,i) \in A\}} x_{ji} - \sum_{\{k:(k,j) \in A\}} x_{kj} = \begin{cases} 1, & j = s \\ 0, & j \in V\text{-}\{s,t\} \\ -1, & j = t \end{cases} \tag{7}$$

$$x_{ij} \in \{0,1\}\,,\ for\ (i,j) \in A \tag{8}$$

The solver CPLEX is then used to solve the above MIP.

3.2 Basic Heuristics for MMRP

Two basic heuristics for MMRP are known; in fact these heuristics are applicable to any MMRCO problem. These heuristics are based on the idea of specifying a particular scenario and then solving a classic problem using this scenario. The output of these heuristics are feasible solutions for the MMRCO problem (Candia-Véjar et al., 2011; Conde and Candia, 2007; Kasperski, 2008; Montemanni et al., 2007; Pereira and Averbakh, 2011a,b).

First we mention the midpoint scenario, s^M defined as $s^M = [(c_e^+ + c_e^-)/2]$, $e \in A$. The heuristic based on the midpoint scenario is described in Algorithm *HM*.

Algorithm HM(G,c)

Input: Network G, and interval costs function c
Output: A feasible solution Y for MMRP.
1: **for all** $e \in A$ **do**
2: $\quad c_e^{s^M} = (c_e^+ + c_e^-)/2$
3: **end for**
4: $Y \leftarrow OPT(s^M)$
5: **return** $Y, Z(Y)$

Algorithm HU(G,c)

Input: Network G, and interval costs function c
Output: A feasible solution Y for MMRP.
1: **for all** $e \in A$ **do**
2: $\quad c_e^{s^U} = c_e^+$
3: **end for**
4: $Y \leftarrow OPT(s^U)$
5: **return** $Y, Z(Y)$

We refer to the heuristic based on the midpoint scenario as HM. The other heuristic based on the upper limit scenario will be denoted by HU and is described in Algorithm *HU*.

The heuristics HU and HM have been designed for rapidly obtaining feasible solutions. These heuristics find a solution by solving the corresponding classic problem only twice. The first is the computation of the solution Y in the specific scenario, s^M for HM or s^U for HU, and the second is the computation of $Z(Y)$ (see steps 4 and 5 in Algorithm *HM* and Algorithm *HU*). These heuristics have been used in an integrated form by the sequential computing of the solutions given by HM and HU and using the best. In the evaluation of heuristics for MMR problems, some experiments have shown that if these heuristics are considered as an initial solution for a heuristic, improved solutions are not easy to achieve, please refer to Montemanni *et al.* (Montemanni et al., 2007), Pereira and Averbakh (Pereira and Averbakh, 2011a,b) and Candia-Véjar *et al.* (Candia-Véjar et al., 2011) for a more detailed explanation.

3.3 Simulated Annealing for MMRP

Simulated Annealing (SA) is a very traditional metaheuristic, see Dréo et al. (2006) for details. A generic version of SA is specified in Kirkpatrick et al. (1983).

We shall now describe the main concepts and parameters used within the context of the MMRP problem,

Search Space A subgraph S of the original graph G is defined such that this subgraph contains a s-t path. In S a classical s-t shortest path subproblem is solved, where the arc costs are chosen taking the upper limit arc costs. Then, the optimum solution of these problem is evaluated for acceptation.

Initial Solution The initial solution Y_0 is obtained applying the heuristic HU to the original network S^0 The regret $Z(Y_0)$ is then computed.

Initial Temperature Different values for the initial temperature were tested and $t_0 = 1$ was used for all experiments.

Cooling Programming A geometric descent of the temperature was according to parameter beta. Several experiments were performed and after to consider the trade-off between the regret of the solution and time needed to compute it, β was fixed as 0.94 for all experiments.

Internal Loop This loop is defined by a parameter L and depended on the size of the instances tested. After initial experiments, L was fixed at 25 for instances from 500 nodes to 10000 nodes. For instances with 20000 nodes the L was fixed at 50.

Neighborhood Search Moves Let S^i be the subgraph of G considered at the iteration i and let x^i be the solution given by the search space at the iteration i. Then we generate a new subgraph S^{i+1} of G from S^i changing the status of some components of the vector characterizing S^i. The number of components is managed by a parameter λ and a feasible solution is obtaining searching S^{i+1} (according with the definition of Search Space) which must be tested for acceptation.

Acceptation Criterion A standard probabilistic function is used to determine if a neighboring solution is accepted or not.

Termination Criterion A fixed value of temperature (final temperature t_f) is used as termination criterion with $t_f = 0.01$.

Our definition of Neighborhood Search Moves is new, but takes inspiration from that described by Nikulin. In his paper (Nikulin, 2007), he applied this to the interval data minmax regret spanning tree problem.

4 Neighborhood Structure for the MMRP problem

Given the importance of the neighborhood structure in our proposed method, we have dedicated this section to explain it in detail. We start by defining the Local Search (LS) mechanism. Subsequently we detail the concepts of neighborhood structure and Search Space. After that, we explicitly describe an architectural model for obtaining new candidate solution by restricting the original search space.

4.1 Local Search (LS)

Local Search (LS), described in Algorithm *local-search*, is a search method for a CO problem P with feasible space $\mathcal{S}$. The method starts from an initial solution and iteratively improves it by replacing the current solution with a new candidate, which is only marginally different. During this initialization phase, the method selects an initial solution Y from the search space $\mathcal{S}$. This selection may be at random or taking advantage of some *a priori* knowledge about the problem.

An essential step in the algorithm is the acceptance criterion, i.e., a neighbor is identified as the new solution if its cost is strictly less in comparison to the current solution. This cost is a function assumed to be known and is dependent on the particular problem. The algorithm terminates when no improvements are possible, which happens when all the neighbors have a higher (or equal) cost when compared to the current solution. At this juncture, the method outputs the current solution as the best candidate. Observe that, at all iteration steps, the current solution is the best solution found so far. LS is a sub-optimal mechanism, and it is not unusual that the output will be far from the

optimum. The literature reports many algorithms that attempt to overcome the hurdles encountered in the original LS strategy. The interested reader may consult (Luke, 2009) for a survey.

Algorithm local-search(S,cost$(\cdot)$,$N(\cdot)$)

Input:
(i) S, A search space.
(ii) cost$(\cdot)$, a cost function.
(iii) $N(\cdot)$ a neighborhood function.

1: $Y \leftarrow$ A starting solution from $\mathcal{S}$.
2: **while** $\exists Y' \in N(Y)$ such that cost(Y') $<$ cost(Y) **do**
3: $\quad Y \leftarrow Y'$
4: **end while**

Algorithm neighbor-induction($\mathcal{R}$)

Input: $\mathcal{R}$, a subspace of the original search space $\mathcal{S}$.
Output: Y', the new candidate solution.

1: $\mathcal{R}' \leftarrow$ subspace-perturbation($\mathcal{R}$)
2: $Y' \leftarrow$ generate-candidate($\mathcal{R}'$)

4.2 Neighbor Induction Via Subspace Perturbation

Typically in LS, almost all cases of neighborhood structures are analogous to the k-opt method explained above, in the sense that a candidate solution is obtained by applying a slight modification to the previous candidate. A fundamentally different philosophy is the one of using sub-spaces to *induce* candidate solutions. In this model, the new candidate is not obtained directly from a previous solution. Rather the candidate is obtained by an indirect step, which consists in perturbing a sub-space in a LS fashion so as to obtain a new subspace which is marginally different in comparison to the former. Finally, the new subspace is employed to derive the new candidate solution. This concept adds an extra layer in the architectural model for defining the neighborhood structure. The method is detailed in Algorithm *neighbor-induction*, which generalizes the method presented in (Nikulin, 2007). The author of (Nikulin, 2007), in the first step apply local transformations to a connected graph (sub-space) to obtain a new graph which is also connected (new sub-space). In the second step, they calculate the differences in the regret between the original and modified candidate solutions.

4.3 MMRP Neighborhood

Our proposed solution for the MMRP Neighborhood retains the idea of using bitmap strings to represent (and restrict) the search space. We start by defining a bitmap string with cardinality $|A|$, such that $\pi(j) = 1$ if edge a_j belongs to the current subset, and $\pi_i(j) = 0$ otherwise. Further, $\pi(j)$ denotes the bit j of the bitmap vector. The full process for creating a new search space is detailed in Algorithm *MMRP-subspace-perturbation*.

At each iteration, a predetermined fraction of arcs from the original subspace are inverted, i.e., they are set to 1 (added) if they were not present in π or set to 0 (deleted) otherwise. This fraction is controlled by the parameter γ, and directly relates the concept of exploration and exploitation mentioned earlier. Small values for γ lead to slight perturbations of the current subspace, i.e., the resultant subspace will be only marginally different from the subspace currently being examined. This configuration favors the exploitation of the current solution. In contrast, large values for γ produce strong perturbations of the subspace, producing subspaces which are expected to be much different from the subspace currently being perturbed, which favors the exploration of unvisited

regions in the original search space. Exploratory test on a variety of datasets have show evidence that a suitable value for γ depends on the dataset being tested and particularly its size.

Once the subspace is determined, the algorithm ensures that there exists a path between s and t. If so π' is accepted, otherwise we reject it and randomly generate a new version of π' following the same scheme.

The overall algorithm starts with the entire search space by setting all the bits of the vector π to 1.

Algorithm 1 Algorithm MMRP-subspace-perturbation(π,γ)

Input:

i) π, a bitmap string with cardinality $|A|$, such that $\pi(j) = 1$ if edge a_j belongs to the current subset, and $\pi(j) = 0$ otherwise.
ii) γ, the fraction of arcs from the original subspace which are to be flipped.

Output:

i) π', a bitmap string with cardinality $|A|$, such that $\pi'(j) = 1$ if edge a_j belongs to the current subset, and $\pi'(j) = 0$ otherwise.

Method:

repeat
  $\pi' \leftarrow \pi$
  for $k = 1 \rightarrow \gamma$ **do**
    $j \rightarrow$ RANDOM$(0, |\pi'|)$
    if $\pi'(j) = 0$ **then**
      $\pi'(j) \leftarrow 1$
    else
      $\pi'(j) \leftarrow 0$
    end if
  end for
until the graph induced by π' contains at least one path from s to t.

Observe that, in our definition of neighborhood, a subspace is not restricted to connected graphs, i.e., a subspace may (or may not) possess unconnected components. For this reason, we must check at all iterations that at least a single s-t path exists. Note that the unconnected components may become connected depending on the stochastic properties of the environment.

Once the auxiliary graph is determined, we obtain a new candidate solution from it. This process is detailed in Algorithm *MMRP-generate-candidate*. In our proposition, the new candidate solution, i.e., a new path, is obtained by a heuristic criterion. We decided to apply the HU method mentioned earlier. Other avenues involve the replacement of this criterion, using HM instead of HU, or selecting the path with the minimum cost between the two heuristics. We then calculate the regret of this path with a classical SP algorithm over the original graph, then using it to determine whether or not to accept the new subspace.

Algorithm 2 Algorithm MMRP-generate-candidate(π')

Input:

i) π', a bitmap string with cardinality $|A|$, such that $\pi'(j) = 1$ if edge a_j belongs to the current subset, and $\pi'(j) = 0$ otherwise.

Output:

i) Y', the new candidate solution.

Method:

$Y' \leftarrow \text{HU}(\pi')$

With this method, we are able to tailor the percentage of arcs we flip when generating a neighbor candidate, enabling us to find the correct balance between exploration and exploitation. The result of this, however, is that we can no longer use the delta between the regrets as our acceptation criteria. Instead we have calculate the regret via a heuristic method. For MMRP this compromise is acceptable, as we know of linear time algorithms for calculating the two shortest paths required for the calculation of the HU and HM heuristics.

5 Experimental Results and Analysis

All the algorithms were implemented in C and the experiments were conducted on a computer with a 2.3 GHz Intel i3 processor and 3 Gb of RAM.

Experiments with two classes of instances were performed, Karasan instances and a set of random instances. We test how well a standard MIP formulation performs in solving both sets of graphs, using those solutions to benchmark the relative performance of the HU, HM, and SA approaches.

5.1 Karasan Instances

The Karasan graphs represent a network topology where the nodes are organized in w layers. They are named here as K-n-c-d-w, where n is the number of nodes, each cost interval has form $\left[c_{ij}^-, c_{ij}^+\right]$ where a random number $c_{ij} \in [1, c]$ is generated and $c_{ij}^- \in [(1-d)c_{ij}, (1+d)c_{ij}]$, $c_{ij}^+ \in \left[c_{ij}^- + 1, (1+d)c_{ij}\right]$ ($0 < d < 1$) are generated and w is the number of layers (Montemanni and Gambardella, 2004).

5.2 Random Graphs

Random graphs were defined in Montemanni *et al.* (Montemanni and Gambardella, 2004) and they are named as R-n-c-δ where n is the set of nodes, interval costs are randomly generated in such a way that $c_{ij}^+ \leq c$, $\forall (i,j) \in A$, $0 \leq c_{ij}^- \leq c_{ij}^+$, $\forall (i,j) \in A$ and δ defines the density of the graph.

5.3 Results and Analysis

Instances with 1000, 10000 and 20000 nodes were considered. For a fixed number of nodes, instances with differing number of arcs were considered. The parameters used by SA (after tuning) were defined as follows: Initial temperature was fixed by $T_0 = 1$ and final temperature was fixed by $T_f = 0.1$, the parameter λ was defined as $\lambda = 0.1$ for graphs with 1000 until 10000 nodes and

by $\lambda = 0.01$ for graphs with 20000 nodes, the cooling programming was defined by $\beta = 0.94$, the internal loop was defined by $L = 25$ for instances with 1000 and 10000 nodes. For instances with 20000 nodes the parameter was defined by $L = 50$.

CPLEX was able to solve at the optimum, instances with 1000 nodes. For instances with 10000 and 20000 nodes, memory problems only permit feasible solutions to be found. In the tables, when the optimal value for the instance is known, we refer to it as GAP. When CPLEX can not find the optimal solution, we give the gap estimated by CPLEX (denoted as Est. Gap) and the gap between the best known value found by CPLEX and that found by the heuristic (denoted as GAP*).

Table 1 shows that the exact algorithm leads to larger values for the gaps as the complexity of the instances increases. With respect to the heuristics HM and HU, it is clear that HU has a better performance than HM and the gap of solutions found by HU is always small.

From Table 2 it is noted first that SA is able to obtain feasible solutions in no more than 22 minutes for all instances. Only a single instance with 20000 nodes took 1245 seconds (on average) and the other solutions were obtained within 433 seconds or less. We observe that in most instances SA was able to improve the solution given by the heuristic and all the instances achieved solutions with small gaps with respect to the best solution given by CPLEX. For the most part, there is very little variation in both the running time and the quality of the solution when using SA. However, for two instances, one with 10000 nodes and the other with 20000 nodes, the running time had very significant differences.

For random networks CPLEX was able to optimally solve all the instances. Our results for random instances confirm the results commented in others papers. These classes of instances are not complicated for exact algorithms. HU has better performance than HM and almost always finds an optimal solution. In two particular cases HM and HU only find good feasible solutions. In these cases, SA was not be able to improve on the solution provided by HU.

6 Conclusions and Future Work

A novel neighbor generation method for a SA algorithm was proposed and used for solving the interval data minmax regret path problem. The performance of this method was compared with the performance of two known simple and effective heuristics for MMRCO problems; the optimal solution (in most cases) was provided by CPLEX from a linear integer programming formulation known for MMRP. Two classes of instances were considered for experimentation; random graphs and Karasan graphs. For random graphs, the optimal solutions were always obtained by CPLEX in reasonable times. The heuristic using the upper bound scenario outperforms that using the midpoint scenario in this problem and almost always finds the optimal solutions.

For Karasan graphs, the optimal solution was always found by CPLEX for instances with 1000 nodes. All the instances with 10000 and 20000 nodes had estimated gaps between 2.07% and 9.17% within the time limit. Also, in some cases CPLEX was not able to find the optimum because of memory overflow errors.

According to our experiments, there are many cases where the SA improves on the result of the HU and HM algorithm. In particular, in the case when the cardinality of the datasets is large. This is an important feature of our proposed SA algorithm, that is, it consistently improves on the best of two basic heuristics for the broad range of Karasan instances we tested. This shows that the method of perturbing the search space during the neighbor generation process in the SA algorithm is an effective means of generating better solutions than the two heuristic algorithms.

For future work, when using SA in Minmax regret problems, it would be ideal to consider more variety on some parameters when defining the test instances e.g., the parameter c. Also, the neighborhood scheme used here could be considered when using different metaheuristics, like Genetic Algorithms, to solve Minmax Regret combinatorial optimization problems.

References

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*. Prentice Hall, Upper Saddle River, NJ.

Aissi, H., Bazgan, C., and Vanderpooten, D. (2009). Min-max and min-max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427–438.

Candia-Véjar, A., Alvarez-Miranda, E., and Maculan, N. (2011). Minmax regret combinatorial optimization problems: an algorithmic perspective. *RAIRO Op. Research*, 45(2):101–129.

Conde, E. and Candia, A. (2007). Minimax regret spanning arborescences under uncertain costs. *European Journal of Operational Research*, 182(2):561–577.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

Dréo, J., Pétrowski, A., Siarry, P., and Taillard, E. (2006). *Metaheuristics for Hard Optimization*. Springer.

Karasan, O., Pinar, M., and Yaman, H. (2001). The robust shortest path problem with interval data. Technical report, Bilkent University.

Kasperski, A. (2008). *Discrete Optimization with Interval Data*, volume 228 of *Studies in Fuzziness and Soft Computing*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kouvelis, P. and G., Y. (1997). *Robust discrete optimization and its applications*. Kluwer Academic Publishers.

Luke, S. (2009). *Essentials of Metaheuristics*. Lulu. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

Montemanni, R., Barta, J., Mastrolilli, M., and Gambardella, L. M. (2007). The Robust Traveling Salesman Problem with Interval Data. *Transportation Science*, 41(3):366–381.

Montemanni, R. and Gambardella, L. M. (2004). An exact algorithm for the robust shortest path problem with interval data. *Computers & Operations Research*, 31(10):1667–1680.

Montemanni, R. and Gambardella, L. M. (2005). The robust shortest path problem with interval data via Benders decomposition. *4or*, 3(4):315–328.

Nikulin, Y. (2007). Simulated annealing algorithm for the robust spanning tree problem. *Journal of Heuristics*, 14(4):391–402.

Pereira, J. and Averbakh, I. (2011a). Exact and heuristic algorithms for the interval data robust assignment problem. *Computers & Operations Research*, 38(8):1153–1163.

Pereira, J. and Averbakh, I. (2011b). The robust set covering problem with interval data. *Annals of Operations Research*, pages 1–19. DOI: 10.1007/s10479-011-0876-5.

Table 1: The results of the analyses for Karasan graphs instances ranging from 7500 to 20000 nodes. The columns show statistical information for the MILP, HM and HU schemes, respectively. In these instances, GAP* for HM, HU, and SA are calculated from the best solution found by CPLEX, rather than the optimal solution.

	MILP			HM			HU		
Instances	$Z(x)$	T(sec)	Est. GAP(%)	$Z(x)$	T(sec)	GAP*(%)	$Z(x)$	T(sec)	GAP*(%)
K-1000-200-0.9-a-5	2854	23.30	0.00	2966	0.01	3.92	2918	0.01	2.24
K-1000-200-0.9-b-5	2463	12.70	0.00	2657	0.01	7.88	2482	0.01	0.77
K-1000-200-0.9-a-18	227	3.00	0.00	240	0.02	5.73	231	0.02	1.76
K-1000-200-0.9-b-18	250	3.90	0.00	255	0.02	2.00	251	0.02	0.40
K-1000-200-0.9-a-25	134	4.00	0.00	145	0.02	8.21	134	0.02	0.00
K-1000-200-0.9-b-25	125	4.70	0.00	127	0.02	1.60	126	0.02	0.80
K-1000-200-0.9-a-50	48	4.10	0.00	50	0.04	4.17	49	0.04	2.08
K-1000-200-0.9-b-50	56	2.80	0.00	62	0.04	10.71	56	0.04	0.00
K-1000-200-0.9-a-100	23	4.10	0.00	23	0.08	0.00	25	0.08	8.70
K-1000-200-0.9-b-100	17	3.70	0.00	17	0.08	0.00	17	0.08	0.00
K-10000-200-0.9-a-4	36091	7201.78	5.41	37912	0.06	5.05	37277	0.06	3.29
K-10000-200-0.9-c-4	36539	7202.01	7.00	38362	0.08	4.99	37685	0.08	3.14
K-10000-200-0.9-a-8	10083	7199.62	3.38	10490	0.11	4.04	10337	0.11	2.52
K-10000-200-0.9-c-8	10781	7200.46	4.46	11372	0.13	5.48	10940	0.13	1.47
K-10000-200-0.9-a-15	3147	7199.32	2.64	3322	0.18	5.56	3190	0.18	1.37
K-10000-200-0.9-c-15	3019	7198.59	2.07	3218	0.21	6.59	3039	0.23	0.66
K-10000-200-0.9-a-30	1018	7198.53	5.34	1074	0.33	5.50	1028	0.33	0.98
K-10000-200-0.9-b-30	1005	7197.98	5.31	1055	0.33	4.98	1014	0.33	0.90
K-10000-200-0.9-a-100	–	7200.00	–	184	1.06	–	172	1.05	–
K-10000-200-0.9-b-100	–	7200.00	–	172	1.05	–	166	1.06	–
K-20000-200-0.9-a-3	118661	7200.15	9.17	123705	0.11	4.25	124588	0.11	4.99
K-20000-200-0.9-c-3	118751	7201.10	9.39	123314	0.13	3.84	124170	0.13	4.56
K-20000-200-0.9-a-5	50917	7199.79	7.19	53141	0.17	4.37	51902	0.17	1.93
K-20000-200-0.9-c-5	50868	7201.39	6.75	53107	0.19	4.40	51735	0.19	1.70
K-20000-200-0.9-a-8	21516	7199.21	5.36	22613	0.26	5.10	21913	0.24	1.85
K-20000-200-0.9-c-8	21219	7202.52	5.24	22270	0.23	4.95	21499	0.23	1.32
K-20000-200-0.9-a-20	3798	7200.00	4.39	4040	0.53	6.37	3831	0.50	0.87
K-20000-200-0.9-b-20	3729	7200.00	5.34	3947	0.51	5.85	3757	0.51	0.75
K-20000-200-0.9-a-50	–	7200.00	–	917	1.14	–	864	1.13	–
K-20000-200-0.9-b-50	–	7200.00	–	868	1.14	–	838	1.19	–

Table 2: The results of the analyses for Karasan graphs instances ranging from 7500 to 20000 nodes. The columns show statistical information about the SA, when the decaying factor for the temperature was 0.94. For each instance, the average between 10 runs is reported.

	SA(0.94)					
Instances	$Z(x)$	GAP*(%)	σ(GAP*)	T(sec)	$\min T(sec)$	$\max T(sec)$
K-1000-200-0.9-a-5	2897.6	0.02	0.005	2.40	1.90	6.30
K-1000-200-0.9-b-5	2476.9	0.01	0.002	3.80	2.00	12.10
K-1000-200-0.9-a-18	229.4	0.01	0.009	4.10	4.00	4.40
K-1000-200-0.9-b-18	250.7	0.00	0.002	18.20	4.10	143.60
K-1000-200-0.9-a-25	134.0	0.00	0.000	4.70	4.80	5.60
K-1000-200-0.9-b-25	126.0	0.01	0.000	4.70	4.80	5.40
K-1000-200-0.9-a-50	49.0	0.02	0.000	8.00	8.20	8.60
K-1000-200-0.9-b-50	56.0	0.00	0.000	8.20	8.40	8.50
K-1000-200-0.9-a-100	25.0	0.08	0.000	17.20	17.70	18.00
K-1000-200-0.9-b-100	17.0	0.00	0.000	17.50	17.70	17.80
K-10000-200-0.9-a-4	37093.6	2.70	0.210	42.60	40.80	44.26
K-10000-200-0.9-c-4	37510.6	2.61	0.090	44.13	40.88	56.67
K-10000-200-0.9-a-8	10318.5	2.28	0.150	55.49	54.57	56.80
K-10000-200-0.9-c-8	10913.8	1.22	0.070	56.70	54.48	64.25
K-10000-200-0.9-a-15	3189.1	1.32	0.050	152.55	90.30	705.71
K-10000-200-0.9-c-15	3037.7	0.62	0.050	94.55	91.80	108.85
K-10000-200-0.9-a-30	1028.0	0.97	0.000	139.13	138.22	139.72
K-10000-200-0.9-b-30	1011.9	0.68	0.280	135.76	134.94	136.67
K-10000-200-0.9-a-100	172.0	–	–	432.87	429.46	435.10
K-10000-200-0.9-b-100	165.9	–	–	420.23	417.38	422.80
K-20000-200-0.9-a-3	124075.2	4.30	0.100	97.12	95.07	100.83
K-20000-200-0.9-c-3	123652.0	3.96	0.120	124.98	120.74	129.00
K-20000-200-0.9-a-5	51846.2	1.79	0.060	118.81	116.74	121.61
K-20000-200-0.9-c-5	51649.5	1.51	0.070	144.04	136.16	147.49
K-20000-200-0.9-a-8	21890.9	1.71	0.080	116.21	114.54	117.63
K-20000-200-0.9-b-8	21483.5	1.23	0.060	140.51	117.86	265.97
K-20000-200-0.9-a-20	3829.3	0.82	0.090	222.05	219.57	223.66
K-20000-200-0.9-b-20	3754.1	0.67	0.080	246.83	233.27	299.18
K-20000-200-0.9-a-50	864.0	–	–	1244.95	457.37	8196.27
K-20000-200-0.9-b-50	–	–	0.000	0.00	–	0.00