Noname manuscript No.
(will be inserted by the editor)

A tabu search algorithm for the minmax regret minimum spanning tree problem with interval data

Adam Kasperski · Mariusz Makuchowski · Paweł Zieliński

Received: date / Accepted: date

Abstract This paper deals with the strongly NP-hard minmax regret version of the minimum spanning tree problem with interval costs. The best known exact algorithms solve the problem in reasonable time for rather small graphs. In this paper an algorithm based on the idea of tabu search is constructed. Some properties of the local minima are shown. Exhaustive computational tests for various classes of graphs are performed. The obtained results suggest that the proposed tabu search algorithm quickly outputs the optimal solutions for the smaller instances, previously discussed in the existing literature. Furthermore, some arguments that this algorithm performs well also for larger instances are provided.

Keywords spanning tree · minmax regret · interval data · tabu search

1 Preliminaries

The minimum spanning tree problem is one of the most important and most extensively studied problems in combinatorial optimization. We are given a

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

Adam Kasperski (corresponding author)
Institute of Industrial Engineering and Management, Wrocław University of Technology
Wybrzeże Wyspiańskiego 27, 50-370, Wrocław, Poland
E-mail: adam.kasperski@pwr.wroc.pl

Mariusz Makuchowski
Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology
Wybrzeże Wyspiańskiego 27, 50-370, Wrocław, Poland
E-mail: mariusz.makuchowski@pwr.wroc.pl

Paweł Zieliński
Institute of Mathematics and Computer Science, Wrocław University of Technology
Wybrzeże Wyspiańskiego 27, 50-370, Wrocław, Poland
E-mail: pawel.zielinski@pwr.wroc.pl

connected and undirected graph $G = (V, E)$ with a cost c_e specified for each edge $e \in E$. A *spanning tree* T is a subset of precisely $|V| - 1$ edges of G that forms an acyclic subgraph of G. We will use $\mathcal{T}$ to denote the set of all the spanning trees of G. In the classical deterministic case we seek a spanning tree whose total cost is minimal. This problem has many applications and we refer the reader to [1] for a description of them. The minimum spanning tree is a well known example of a *matroidal problem*. Recall that a *matroid* is a system $(E, \mathcal{F})$, where E is a finite set of elements and $\mathcal{F}$ is a set of subsets of E closed under inclusion (if $A \in \mathcal{F}$ and $B \subseteq A$ then $B \in \mathcal{F}$) and having the so called *growth property* (if $A, B \in \mathcal{F}$, $|B| < |A|$, then there is $e \in A \setminus B$ such that $B \cup \{e\} \in \mathcal{F}$). The maximal, under inclusion, subsets in $\mathcal{F}$ are called *bases* and the minimal, under inclusion, subsets that are not in $\mathcal{F}$ are called *circuits*. It is easy to verify that all the bases of a given matroid have the same cardinality. A more detailed description of matroids can be found, for example, in [27].

Let E be the set of the edges of graph G and let $\mathcal{F}$ be the set of the subsets of the edges which form acyclic subgraphs (forests) of G. It is not difficult to show that the system $(E, \mathcal{F})$ is a matroid and in the literature it is called a *graphic matroid* [28]. The spanning trees of G form the bases of this matroid and the simple cycles in G form its circuits. From the matroidal structure of the problem it follows that a simple *greedy algorithm* can be used to obtain a minimum spanning tree (see, e.g., [9,20,28]). The greedy algorithm is shown in Figure 1. It picks the edges of G in nondecreasing order of their costs. An edge e is excluded if it forms a cycle with the edges previously selected.

GREEDY
Require: Graph $G = (V, E)$ with deterministic edge costs.
Ensure: A minimum spanning tree T of G.
1: Order the edges of G so that $c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m}$
2: $T \leftarrow \emptyset$
3: **for** $i \leftarrow 1$ **to** m **do**
4: **if** $T \cup \{e_i\} \in \mathcal{F}$ **then** $T \leftarrow T \cup \{e_i\}$
5: **end for**
6: **return** T

Fig. 1 A greedy algorithm for solving the minimum spanning tree problem in graph $G = (V, E)$ with deterministic costs.

A direct implementation of algorithm GREEDY runs in $O(|E| \log |E|)$ time and is known in the literature as Kruskal's algorithm [19]. For the minimum spanning tree problem Prim's and Boruvka's algorithms are also commonly used [9,29]. One of the fastest algorithms up to now is the one developed by Chazelle [8]. It runs in $O(|E|\alpha(|V|, |E|))$ time, where α is the inverse of Ackerman's function. This running time is very close to be linear with respect to $|E|$.

2 Minmax regret minimum spanning tree problem

We now consider the case in which the edge costs are not precisely known. We investigate the simple and appealing model of uncertainty introduced in [31]. Assume that the edge costs are only known to belong to closed intervals, that is for each edge $e \in E$ a range of possible costs $[\underline{c}_e, \overline{c}_e]$ is specified. If the cost of e is precisely known, then it is represented by a *degenerate* interval such that $\underline{c}_e = \overline{c}_e$. We make the assumption that the costs are unrelated, that is the value of each cost does not depend on the values of the remaining costs. Denote by Γ the Cartesian product of all the cost intervals. Then Γ forms a *scenario set*, i.e. the set of all possible realizations of the costs, called *scenarios*. Thus each scenario $S \in \Gamma$ is a vector of the costs, $(c_e^S)_{e \in E}$, which may appear with a positive but perhaps unknown probability. Let us define

$$F(T, S) = \sum_{e \in T} c_e^S.$$

The value of $F(T, S)$ is the cost of the spanning tree T under scenario S. Let us also define

$$F^*(S) = \min_{T \in \mathcal{T}} F(T, S).$$

Thus $F^*(S)$ is the cost of a minimum spanning tree under scenario S. The value of $F^*(S)$ can be efficiently obtained by applying one of the known algorithms for the deterministic minimum spanning tree problem. The *maximal regret* of a given spanning tree $T \in \mathcal{T}$ is defined in the following way:

$$Z(T) = \max_{S \in \Gamma} \{F(T, S) - F^*(S)\}. \tag{1}$$

Hence the maximal regret expresses the largest deviation of T from the optimum over all scenarios in Γ. A scenario that maximizes the right hand side of (1) is called the *worst case scenario* for T and a minimum spanning tree under the worst case scenario is called the *worst case alternative* for T. In this paper we focus on the following *minmax regret minimum spanning tree problem*:

$$\min_{T \in \mathcal{T}} Z(T). \tag{2}$$

We thus seek a spanning tree, called an *optimal minmax regret spanning tree*, which minimizes the maximal regret. The maximal regret is one of the robust criteria for choosing a solution under uncertainty. For a deeper description of various robust criteria, we refer the reader to [18].

It turns out that the value of the maximal regret $Z(T)$ of a given spanning tree T can be computed efficiently. Let us define scenario S_T as follows: $c^{S_T} = \overline{c}_e$ if $e \in T$ and $c^{S_T} = \underline{c}_e$ if $e \in E \setminus T$. In [31], it has been shown that S_T is the worst case scenario for T. We can thus express the maximal regret of T in the following way:

$$Z(T) = F(T, S_T) - F^*(S_T) = F(T, S_T) - F(T^*, S_T),$$

where T^* is a minimum spanning tree under S_T (i.e. T^* is a worst case alternative for T). Consequently, the time required for computing the value of $Z(T)$ is nearly the same as the time required for solving the deterministic minimum spanning tree problem. This important property is widely explored in the literature.

We now recall another problem, which is closely related to the minmax regret minimum spanning tree. A *distance* between two spanning trees T_1 and T_2 is defined as follows:

$$d(T_1, T_2) = |T_1 \setminus T_2| = |T_2 \setminus T_1|.$$

In the *central spanning tree problem*, we wish to find a spanning tree, called a *central tree*, whose maximal distance to all the other spanning trees is minimal. Namely, we seek an optimal solution to the following optimization problem:

$$\min_{T \in \mathcal{T}} \max_{T_1 \in \mathcal{T}} d(T, T_1).$$

The central spanning tree problem has been discussed in [3,7,10,14] and it has some applications, for example, in the analysis of transportation networks. It was proven in [7] that the central spanning tree problem is strongly NP-hard. We now show, following [5], that this problem is a special case of the minmax regret minimum spanning tree. Indeed, let graph $G = (V, E)$ be an instance of the central spanning tree problem and let us associate the interval cost $[0, 1]$ with each edge $e \in E$ of G. A simple verification shows (see [5]) that $Z(T) = \max_{T_1 \in \mathcal{T}} d(T, T_1)$ for any $T \in \mathcal{T}$. In consequence, every central tree is an optimal minmax regret spanning tree and vice versa. From this, it follows immediately that the minmax regret minimum spanning tree problem is strongly NP-hard even if all edges have $[0, 1]$ cost intervals [5,6]. Thus, there is no hope to provide a polynomial time algorithm for the problem. In Section 3, we will describe some attempts to solve it, which have been reported in the literature to date.

3 Previous methods of solving the problem

The first method for solving the problem was proposed in [31], where the authors formulated a mixed integer linear programming (MIP) model which, together with some preprocessing rules, was able to solve the instances of the problem with graphs having up to 25 nodes. Currently, these results can be improved to 30 nodes by using faster computers and the latest versions of the optimization software. For larger instances of the problem, the MIP approach appears to be very slow and useless in practice. By applying Bender's decomposition one can increase the efficiency of the MIP approach and solve the instances with graphs having up to 40 nodes [24]. Other exact algorithms, based on the branch and bound technique, were proposed in [4] and [23]. Using these algorithms, one can solve the problem for graphs having up to 40

nodes [4]. However, for the graphs with 40 nodes the branch and bound algorithm requires many hours to compute an optimal solution. Generally, if the number of nodes of the input graph is greater than 50, then solving the problem is a rather difficult task. Interestingly, the problem seems to be much harder than the minmax regret versions of some other network problems such as the shortest path (see, e.g. [22]), which can be solved for much larger graphs.

Recently, a fast heuristic based on a simulated annealing, which can be applied to larger graphs, has been proposed [26]. The author has shown the results of experiments for graphs having up to 50 nodes. For the central spanning tree problem some heuristics were proposed in [14] and the computational tests were performed for graphs having up to 200 nodes. These heuristics, however, cannot be applied directly to the more general minmax regret minimum spanning tree problem.

Fortunately, there exists a simple and efficient approximation algorithm for the problem, proposed in [17], which simply outputs a minimum spanning tree for a *midpoint scenario*. Namely, the scenario S such that $c_e^S = \frac{1}{2}(\underline{c}_e + \overline{c}_e)$ for all $e \in E$ is first constructed and then a minimum spanning tree T under S is determined. It can be shown [17] that $Z(T) \leq 2OPT$, where OPT is the maximal regret of an optimal minmax regret minimum spanning tree. Thus the algorithm is a 2-approximation one for the problem. We will denote it by AM and we will use it later in this paper. In [15,25], some computational tests suggested that AM may be additionally refined with a very small computational effort. Namely, it may be advantageous to compute a minimum spanning tree T_1 under the midpoint scenario and a minimum spanning tree T_2 under the *pessimistic scenario*, i.e. $S = (\overline{c}_e)_{e \in E}$, where all the costs are set to their upper bounds. Then the spanning tree with the smaller maximal regret is chosen (T_1 or T_2). This algorithm has also a performance ratio of 2, but on average it seems to perform better than AM (see [15]). We denote this algorithm as AMU. No additional approximation results for the problem are known. In particular, we do not know whether there is an approximation algorithm with a performance ratio better than 2. This is perhaps the most important open question concerning this problem.

4 Solving the problem by local search

One of the most successful methods of attacking large scale hard combinatorial optimization problems is *local search* [2,21]. Every local search technique is based on the concept of a *neighborhood function*, that is a mapping N, which for each solution X assigns a subset of solutions $N(X)$ that can be reached in one move starting from X. The set $N(X)$ is called a *neighborhood* of X. A local search algorithm starts from a feasible solution X_0 and performs a sequence of moves, which consist of choosing a solution $X_{i+1} \in N(X_i)$. By specifying a method of choosing a solution from the neighborhood and a stopping criterion we obtain a particular type of the local search algorithms such as: iterative improvement, simulated annealing, threshold acceptance or tabu search.

In this section, our goal is to construct a fast tabu search algorithm for computing solutions of good quality for large instances of the minmax regret minimum spanning tree problem. The standard works on the tabu search technique and some of its applications to hard combinatorial optimization problems can be found in [11–13].

4.1 Neighborhood function and local minimum

In this paper, we use the fact that for the minmax regret minimum spanning tree problem there is a natural definition of the neighborhood function, which follows directly from the matroidal structure of the problem. Let us define

$$N(T) = \{T_1 \in \mathcal{T} : \ |T \setminus T_1| = 1\}.$$

Hence $N(T)$ is a *1-exchange neighborhood* and it consists of all the spanning trees differing from T in exactly a single edge. The neighborhood of a spanning tree T can be generated by means of the algorithm presented in Figure 2. This algorithm can be implemented efficiently by applying the techniques and data structures described, for example, in [1].

NEIGHBORHOOD
Require: Graph $G = (V, E)$, a spanning tree T of G.
Ensure: The neighborhood $N(T)$.
1: $N(T) \leftarrow \emptyset$
2: **for all** $\{i, j\} \in E \setminus T$ **do**
3: Determine the set of edges $\{f_1, \ldots f_k\}$ that are on the path from i to j in T
4: **for all** $f \in \{f_1, \ldots, f_k\}$ **do**
5: Add $T \cup \{i, j\} \setminus f$ to $N(T)$
6: **end for**
7: **end for**
8: **return** $N(T)$

Fig. 2 Generating the neighborhood $N(T)$ of a given spanning tree T.

The correctness of the algorithm NEIGHBORHOOD follows from the matroidal structure of the problem. For each edge $\{i, j\} \in E \setminus T$ the set $T \cup \{i, j\}$ contains the unique circuit (a simple cycle in G), which is formed by the edge $\{i, j\}$ and the unique path from i to j in T. A new spanning tree can be obtained by removing any edge on this path (see Figure 3). Clearly, $|N(T)| = O(|E||V|)$ is bounded by a polynomial in the problem size. In consequence, the neighborhood of T can be generated efficiently. Notice that our concept of the neighborhood is different from that in [26]. The approach used here is more time consuming. Nevertheless, as we will see from the computational results, our algorithm based on the neighborhood $N(T)$ will be also very fast. Furthermore, using some properties of the 1-exchange neighborhood $N(T)$, we will be able to compute efficiently the maximal regrets of many solutions from this neighborhood (see Section 4.3.6).

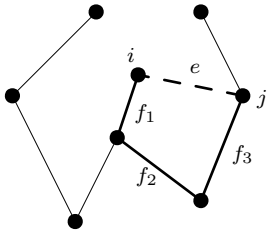

Fig. 3 A sample move: edge $e = \{i, j\}$ is added and an edge f on the path $\{f_1, f_2, f_3\}$ from i to j is removed.

A spanning tree T_{min} is a *local minimum* with respect to the 1-exchange neighborhood if there is no spanning tree $T \in N(T_{min})$ such that $Z(T) < Z(T_{min})$. Now an interesting question arises what is the quality of T_{min}, namely how far from optimum this tree is? We now construct an instance in which $Z(T_{min}) = 2OPT$, where OPT is the maximal regret of an optimal minmax regret spanning tree for this instance. Hence, the quality of a local minimum is not better than the quality of a solution returned by the 2-approximation algorithm AM.

Consider an instance of the problem given in Figure 4. The input graph $G = (V, E)$ is formed by a complete graph composed of nodes $1, 2, \ldots, m$; two nodes a, b with edges $\{a, 1\}$, $\{a, 2\}$, $\{b, 3\}$ and $\{b, 4\}$; nodes $5', \ldots, m'$ with edges $\{5', 5\}, \ldots, \{m', m\}$ and edges $\{b, 5'\}, \{5', 6'\}, \ldots, \{m', a\}$. The cost intervals of all the edges are $[0, 1]$. Let us focus on the spanning tree T shown in Figure 4a. This tree is composed of $2m - 3$ edges, thus $F(T, S_T) = 2m - 3$. If we remove all the edges of the tree T from G, then the resulting graph G' (see Figure 4b) is still connected. Hence, the worst case alternative T^* for T is such that $F(T^*, S_T) = 0$ and so $Z(T) = 2m - 3$. Consider now a spanning tree T_1, which is the result of a move from T. This move consists in adding an edge $e \in E \setminus T$ to T and removing an edge $f \in T$ from T. However, observe that it is not possible to disconnect graph G' by removing a single edge from G'. Hence, removing T_1 from G also forms a connected graph and consequently $F^*(S_{T_1}) = 0$. Now it is clear that $Z(T_1) = Z(T) = 2m - 3$ and T is a local minimum.

Consider now the spanning tree R presented in Figure 5a. This tree is an optimal minmax regret spanning tree for the sample instance. By removing R from G we obtain the graph G' (see Figure 5b) which is composed of $m - 1$ connected components. Hence, the worst case alternative R^* for R is such that $F(R^*, S_R) = m - 2$. In consequence, $Z(R) = 2m - 3 - m + 2 = m - 1$. It holds

$$\frac{Z(T)}{Z(R)} = \frac{2m - 3}{m - 1} \sim 2,$$

which is asymptotically equal to 2 for large m.

We are thus get the following result:

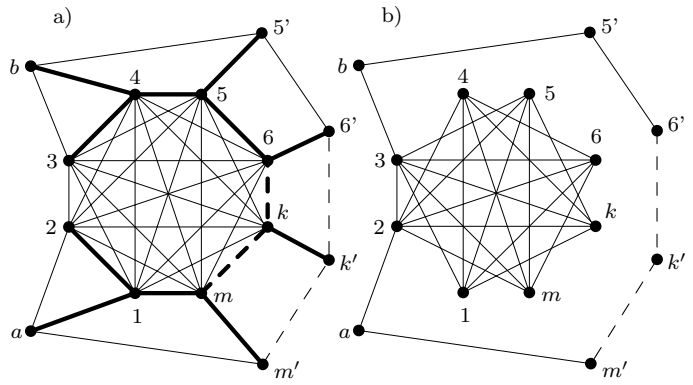

Fig. 4 A local minimum (in bold). All the cost intervals are $[0, 1]$.

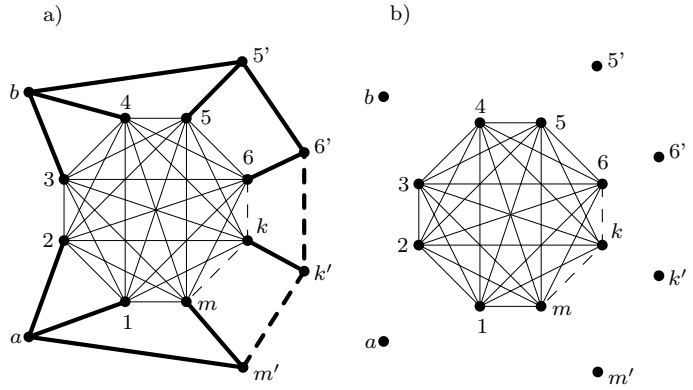

Fig. 5 The optimal minmax regret spanning tree (in bold). All the cost intervals are $[0, 1]$.

Theorem 1 *There exists a local minimum T_{min} with respect to the 1-exchange neighborhood such that $Z(T_{min}) \geq (2 - \varepsilon)OPT$ for any $\varepsilon > 0$ even if all the cost intervals are* $[0, 1]$.

Note that we have just found another hard example for the algorithms AM and AMU. A graph constructed in [15] contains multiedges, while the graph in Figure 4 does not. In consequence, the algorithms AM and AMU attain asymptotically the bound of 2 even if the input graph has no multiedges. Furthermore, both algorithms may output a 2-approximate solution which cannot be improved by using a single edge exchange. We conjecture that the class of graphs shown in Figure 4 is hard for any local search algorithm that uses the 1-exchange neighborhood. So, we use it to test our tabu search algorithm in the next sections. Finally, it is worth pointing out that for the deterministic minimum spanning tree problem, the 1-exchange neighborhood is *exact*. This means that every local minimum with respect to this neighborhood is also a

global one (see., e.g. [21]). Clearly, this is not the case for the minmax regret version of this problem.

4.2 Iterative improvement algorithm

The *iterative improvement* algorithm for the robust spanning tree problem is shown in Figure 6. We start with an initial spanning tree T (step 3) and perform a sequence of improving moves (steps 4-10) until the current spanning tree is a local minimum. We repeat this procedure, each time starting with a different spanning tree, until some stop criterion is satisfied. A returned spanning tree T_{best} is the best solution found and it is a local minimum. As we have shown in Section 4.1, the spanning tree T_{best} can be a factor of 2 away from the optimum. So the iterative improvement cannot improve the worst case performance of the approximation algorithm AMU and it is also a 2-approximation algorithm for the problem. Nevertheless, we will show later that this simple and fast algorithm returns good solutions for the instances discussed in literature.

ITERATIVE IMPROVEMENT
Require: Graph $G = (V, E)$ with interval costs of edges.
Ensure: A spanning tree T_{best} of G.
1: $Z_{best} \leftarrow \infty$
2: **while** stop criterion = **false do**
3: Compute an initial spanning tree T of G.
4: **repeat**
5: $impr \leftarrow$ ***false***
6: Find a spanning tree $T' \in N(T)$ of the smallest value of $Z(T')$
7: **if** $Z(T') < Z(T)$ **then**
8: $T \leftarrow T'$, $impr \leftarrow$ ***true*** */* A better solution has been found */*
9: **end if**
10: **until** $impr =$ ***false***
11: **if** $Z(T) < Z_{best}$ **then**
12: $Z_{best} \leftarrow Z(T)$
13: $T_{best} \leftarrow T$
14: **end if**
15: **end while**
16: **return** T_{best}

Fig. 6 The iterative improvement algorithm.

Later in this paper we will consider the following three versions of the iterative improvement:

1. *Single-run iterative improvement.* The initial spanning tree in step 3 is the one obtained by applying the algorithm AMU. The loop 2-12 is executed only once. So the algorithm returns a local minimum reached by a sequence of improving moves from a 2-approximate solution returned by AMU.
2. *Multi-run random iterative improvement.* The initial spanning tree in step 3 is chosen randomly. The stopping criterion is a fixed time of computations.

3. *Multi-run perturbed iterative improvement.* The initial spanning tree in step 3 is computed by a perturbed AMU algorithm. This algorithm perturbs the midpoint and pessimistic scenarios before executing AMU. The method of perturbing the costs under both scenarios is the same as in [30]. The stopping criterion is a fixed time of computations.

4.3 Tabu search algorithm

The iterative improvement algorithm can be refined by applying the idea of *tabu search* [11–13]. By using tabu search we avoid being trapped at local minima, which allows us to explore a larger part of the solution space. The tabu search algorithm for the robust spanning tree problem is shown in Figure 7. We now describe all its details.

TABU SEARCH
Require: Graph $G = (V, E)$ with interval costs of edges.
Ensure: A spanning tree T_{best} of G.
1: Preprocess the input graph G.
2: Compute an initial spanning tree T of G
3: $T_{best} \leftarrow T$, $Z_{best} \leftarrow Z(T)$
4: $TABU \leftarrow \emptyset$, $E^* \leftarrow T^*$, $k \leftarrow 0$
5: **while** stop criterion=**false do**
6: $\overline{N}(T) \leftarrow \{T_1 \in N(T)$: the move to T_1 is not forbidden **or** $Z(T_1) < Z_{best}\}$
7: Find a spanning tree $T_1 \in \overline{N}(T)$ of the smallest value of $Z(T_1)$
8: $k \leftarrow k + 1$
9: **if** $Z(T_1) < Z_{best}$ **then**
10: $T_{best} \leftarrow T_1$, $Z_{best} \leftarrow Z(T_1)$ /* *A better solution has been found* */
11: $E^* \leftarrow E^* \cup T_1^*$
12: $k \leftarrow 0$
13: **end if**
14: **if** $k > k_{max}$ **then**
15: Compute a random spanning tree T of $G = (V, E^*)$
16: **if** $Z(T) < Z_{best}$ **then**
17: $T_{best} \leftarrow T$, $Z_{best} \leftarrow Z(T)$ /* *A better solution has been found* */
18: **end if**
19: **Go to** line 4 /* *Restart the algorithm* */
20: **else**
21: $T \leftarrow T_1$ /* *Perform the move* */
22: Update $TABU$
23: **end if**
24: **end while**
25: **return** T_{best}

Fig. 7 The tabu search algorithm.

4.3.1 Preprocessing

Before applying any algorithm to an instance of the problem it may be advantageous to use a preprocessing, proposed in [31]. We now briefly describe the

idea of this preprocessing. An edge $e \in E$ is called *weak* if it belongs to a minimum spanning tree under some scenario $S \in \Gamma$. An edge $e \in E$ is said to be *strong* if it belongs to a minimum spanning tree under all scenarios $S \in \Gamma$. It turns out that all the weak and strong edges can be efficiently detected [4,31, 16], which follows from the matroidal structure of the problem. Furthermore, a non-weak edge cannot be a part of an optimal minmax regret spanning tree and, in the absence of degeneracy ($\overline{c}_e > \underline{c}_e$ for all $e \in E$), there is an optimal minmax regret spanning tree which contains all strong edges. We can now use these two facts to reduce the problem size. Namely, one need not to explore the moves consisting of adding a non-weak edge or removing a strong edge (in the absence of degeneracy). We can thus mark all the non-weak and strong edges before executing the tabu search algorithm and use this information during the search process. This is done in line 1 of the algorithm (Figure 7). In Section 5.3, we will investigate the percent of the non-weak and strong edges for different classes of graphs.

4.3.2 Initial solution

In our algorithm we prefer to start with a random spanning tree. By performing some tests we have discovered that the use of AMU or its perturbed version to generate the initial solutions does not improve the final solutions returned by the tabu search algorithm but requires an additional computational effort. Furthermore, we will show later that the tabu search outperforms the iterative improvement for the classes of graphs for which the algorithm AMU is equivalent to computing a random spanning tree.

4.3.3 Tabu list

In the tabu search algorithm, we do not terminate at a local minimum and permit a move to a solution T_1 such that $Z(T_1) \geq Z(T)$. A simple way to avoid oscillations around the local minima is to store the information about the performed moves in the so called *tabu list*. This list contains a description of the moves which are forbidden for a certain number of iterations.

Let T be a current spanning tree. Suppose that we have performed a move by adding edge $e \in E \setminus T$ to T and removing edge f from T. Hence, we have obtained a spanning tree $T_1 = T \setminus \{f\} \cup \{e\}$ from $N(T)$. In order to avoid getting back to T, we add to the tabu list two elements: $(Add(f), IterAdd)$ and $(Drop(e), IterDrop)$. This means, that we forbid adding edge f to the current solution for $IterAdd$ iterations and dropping edge e from the current solution for $IterDrop$ iterations. In consequence, each move in which we add edge f or remove edge e is not allowed for a certain number of iterations. The values of $IterAdd$ and $IterDrop$ will be specified later.

4.3.4 Aspiration criterion

The *aspiration criterion* is a condition that allows us to perform a move even though it is forbidden by the tabu list. In our tabu search algorithm, we use the following simple aspiration criterion: we always permit a move that leads to a solution better than the current best solution.

The tabu list and the aspiration criterion allow us to define the modified neighborhood $\overline{N}(T)$ of a current solution T (see line 6 of the tabu search algorithm). Namely, we can now perform a move that is not forbidden by the tabu list or satisfies the aspiration criterion.

4.3.5 Long-term memory function

A tabu list together with an aspiration criterion are the so called *short-term memory functions* (see [13]) of a tabu search algorithm. This means that an information about the search process, stored in the tabu list, is lost after a small number of iterations. Because a search strongly tends to focus on locally good solutions, only a small region of a solution space may be explored.

In order to achieve a global diversification some *long-term memory functions* can be employed. In our tabu search algorithm, we incorporate the following method. We introduce a subset of the edges $E^* \subseteq E$. For the initial solution T and each time T improves the current best solution, we add to E^* all the edges that belong to T^* (i.e. to a worst case alternative of T). After a fixed number of iterations, during which the current best solution has not been improved, we restart the algorithm. This restart consists in generating a new random solution in the subgraph $G^* = (V, E^*)$ induced by E^*.

Observe that the distance between T and T^* (that is $|T \setminus T^*|$) is usually large. Therefore, we may expect that a new spanning tree, derived from the subgraph $G^* = (V, E^*)$, allows us to move the search process to another region of the solutions space. Moreover, a spanning tree composed of the edges of the worst case alternatives of the currently best solutions may be itself a good solution.

4.3.6 Some implementation details

It is clear that, contrary to the MIP and branch and bound approaches, the proposed tabu search algorithm can be applied to large (or even very large) instances of the problem. It follows from the fact that all the steps in the algorithm can be implemented to run in polynomial time. The most time consuming step is performed in line 7, where we seek a spanning tree in $\overline{N}(T)$ of a minimum value of the maximal regret. In this step we have to scan the whole neighborhood of the current spanning tree T. A direct computation of $Z(T_1)$ for $T_1 \in \overline{N}(T)$ requires solving the deterministic minimum spanning tree problem. We now show that it is not necessary to solve the deterministic problem for all $T_1 \in \overline{N}(T)$. In other words, one can compute the value of $Z(T_1)$ having some information about the spanning tree T. Before we proceed, we

recall some useful results obtained for matroidal problems in [16]. Let σ be a sequence of the edges of G and $pred(e, \sigma)$ stands for the subset of the edges which precede edge e in σ. Let us denote by T_σ the spanning tree determined by algorithm GREEDY (see Figure 1), in which σ is the order of the edges in line 1. The following proposition is useful:

Proposition 1 ([16]) *Let σ and ρ be two sequences of the edges such that $pred(e, \sigma) \subseteq pred(e, \rho)$ for a given edge $e \in E$. If $e \notin T_\sigma$, then $e \notin T_\rho$.*

Let T be a current spanning tree and let T^* be a worst case alternative for T. Recall that T^* is an optimal solution under scenario S_T. Suppose that we explore a solution that has been obtained by adding edge $e \in E \setminus T$ and removing edge $f \in T$, so that $T_1 = T \cup \{e\} \setminus \{f\}$. It holds

$$F(T_1, S_{T_1}) = F(T, S_T) + \overline{c}_e - \overline{c}_f.$$

We now focus on computing the value of $F^*(S_{T_1})$. Our goal is to quickly compute the spanning tree T_1^*, i.e. a minimum spanning tree under scenario S_{T_1}, having the spanning tree T^*. Let σ be the sequence of edges sorted in nondecreasing order of costs under scenario S_T. We get the spanning tree T^* by applying the algorithm GREEDY to σ (σ is the ordering of the edges in line 1). The scenario S_{T_1} is obtained from S_T by increasing the cost of e to $\overline{c}_e$ and decreasing the cost of f to $\underline{c}_f$. Therefore, to keep the order of the elements under S_{T_1}, we move the element e a number of positions to the right and we move the element f a number of positions to the left in σ. Let σ' denote the resulting new sequence of the elements. The spanning tree T_1^* can be obtained by applying the algorithm GREEDY to σ'. Assume that $e \notin T^*$. Since $pred(\sigma, e) \subseteq pred(\sigma', e)$, Proposition 1 implies $e \notin T_1^*$. Therefore, we must check only whether f belongs to T_1^*. This can be done as follows: if $f \in T^*$, then $T_1^* = T^*$; otherwise $T^* \cup \{f\}$ contains the unique simple cycle composed of the edges $\{f, g_1, \ldots, g_k\}$; if there is g_i whose cost under S_{T_1} is greater than $\underline{c}_f$, then $T_1^* = T^* \cup \{f\} \setminus \{g_i\}$; otherwise $T_1^* = T^*$. Thus, if $e \notin T^*$, then the maximal regret $Z(T_1)$ can be computed in $O(|V|)$ time, which is required to detect a cycle in $T^* \cup \{f\}$. If $e \in T^*$, then we solve the deterministic minimum spanning tree problem under scenario S_{T_1} to obtain T_1^*. However, this must only be done $|V| - 1$ times since $|T^*| = |V| - 1$ and one may use the sequence σ to speed up the calculations (it is not necessary to sort the edges each time T_1^* is computed).

The above technical considerations together with appropriate data structures allow us to scan the neighborhood of a given solution very efficiently.

5 The computational results

In this section we present the results of computational tests performed for various classes of graphs. There are three objectives, which we would like to test. First, we wish to investigate the efficiency of the preprocessing. Namely, we would like to know how many non-weak and strong edges one can expect in

the input graph. Some tests performed in the existing literature (see, e.g., [31]) suggest that the preprocessing may significantly reduce the problem size. We would like to confirm this observation. Next, we wish to test the performance of the very simple and fast approximation algorithm AMU, which computes optimal solutions for the midpoint and pessimistic scenarios and returns the better of them. Finally, we would like to investigate the performance of the iterative improvement and tabu search algorithms and compare them to the existing simulated annealing algorithm constructed in [26]. We would like to check when the usage of the simple iterative improvement is enough to obtain good solutions and when the more sophisticated tabu search method performs better. All the tests were carried out on a Pentium Core Duo 2.66 MHz computer equipped with 2GB RAM. The MIP models were solved by using CPLEX 12.1 solver.

5.1 The classes of graphs

We consider all the classes of graphs, which were previously discussed in the literature and two additional classes, which appears to be hard for local search algorithms with the 1-exchange neighborhood. Namely, we consider the following classes of graphs:

1. *Ya(l,u)-n*: Graph G is complete and has n nodes. For each edge $e \in E$, $\underline{c}_e$ is uniformly distributed in $[0, l]$ and $\overline{c}_e$ is uniformly distributed in $(\underline{c}_e, u]$. This class has been considered in [31].
2. *He1-n*: This class of graphs has been considered in [4]. Graph G represents a two-level network. The lower level consists of clusters (complete graphs) of 5 nodes whose edges are generated as in the class *Ya(10,10)-n*. The upper level links the clusters and these edges have higher costs, i.e. the class *Ya(10,10)-n* costs shifted by a constant. Notice that the graphs in this class are complete and n is the overall number of nodes in G, which is an integer multiple of 5.
3. *He2-n*: This class has been also discussed in [4] and it is similar to the class *He1-n*, except that the upper level links are organized as a binary tree. The graphs in this class are not complete.
4. *Mo(p)-n*: Graph G is complete and has n nodes. These nodes are randomly placed on a 50×50 greed. For all $e = \{i, j\} \in E$, $\underline{c}_e$ is randomly selected in $[d_{ij}(1-p), d_{ij})$ and $\overline{c}_e$ is randomly selected in $(\underline{c}_e, d_{ij}(1+p)]$, where d_{ij} is the Euclidean distance between i and j, and p is a distortion parameter. This class of graphs has been investigated in [23].
5. *Ka-n*: Graph G has n nodes and its structure is shown in Figure 4a. The interval costs of all edges are $[0, 1]$. Notice that the graphs in this class have "'hard"' local minima and only one global minimum, which is known in advance (see Figures 4 and 5).
6. *La-n*: Graph G has n nodes and is composed of three layers (see Figure 8). The first layer is formed by a complete graph $G_{n/2}$. The second layer is composed of $n/2 - 1$ nodes and each node in this layer is connected with

exactly two random nodes from the first layer. Finally, the third layer contains only one node which is connected with all the nodes from the second layer. All the cost intervals are $[0, 1]$. The graphs of this type appear in papers [6,7], where the complexity of the robust spanning tree problem was investigated. We have also tested the graphs with different proportions of the nodes in layers 1 and 2 and different number of edges linking these two layers. However, the described class *La-n* appeared to be the most difficult for our algorithms, so we decided to show the results only for this particular graph structure.

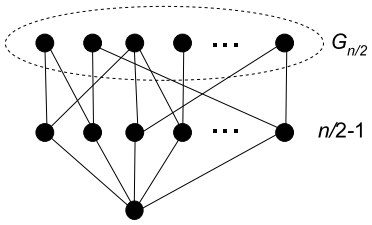

Fig. 8 The graph *La-n*. All the cost intervals are $[0, 1]$.

Notice that the classes *Ka-n* and *La-n* contain some instances of the strongly NP-hard central spanning problem, which is a special case of the minmax regret minimum spanning tree problem. For each particular class of graphs (with fixed n, p, l and u) we generated randomly 10 instances and performed all the tests using them. We divided the instances into two groups. The first group consists of the *easy instances*, i.e. the ones for which an optimal minmax regret spanning tree can be computed efficiently (we assumed that it can be computed within 1 hour) by using CPLEX or is known in advance, as in the class *Ka-n*. The easy instances are listed in Table 1.

Table 1 The easy instances.

Ya(10,10)-10, Ya(10,10)-20
Ya(15,15)-10, Ya(15,15)-20
Ya(20,20)-10, Ya(20,20)-20
Ya(10,20)-10, Ya(10,20)-20
Ya(15,30)-10, Ya(15,30)-20
Ya(20,40)-10, Ya(20,40)-20
He1-10
He2-10,He2-20,He2-30
Mo(0.15)-10, Mo(0.15)-20
Mo(0.50)-10, Mo(0.50)-20
Mo(0.85)-10, Mo(0.85)-20
Ka-10, . . . , Ka-100
La-10, La-20

Notice that the MIP model for the class *He1-n* is surprisingly hard to solve. We were unable to solve the instances, for which the number of nodes $n \geq 20$. Note also that all the instances in *Ka-n* are easy, because we know the global optimum for each such an instance.

The second group consists of the *hard instances*, for which an optimal solution cannot by computed efficiently by the usage of CPLEX. These instances are formed by the graphs having a large number of nodes and no efficient exact algorithm is known for most of them. Therefore, they require a different treatment and our aim is to show some arguments that the tabu search algorithm performs well for them. In the hard instances, the input graphs have up to 100 nodes, so they are much larger than the graphs previously examined in literature.

5.2 The parameters of tabu search

After performing some preliminary tests, we decided to fix the following parameters of the tabu search algorithm, which in our opinion give the best performance. The algorithm stops after performing 1000 iterations, where by iteration we mean the set of steps from line 6 to line 23 (see Figure 7). The restart is performed if after 400 iterations the current best solution has not been improved. Recall that the tabu list is composed of the elements $(Add(f), IterAdd)$ and $(Drop(e), IterDrop)$, where $IterAdd$ is the number of iterations for which we forbid adding the edge f to the current spanning tree and $IterDrop$ is the number of iterations for which we forbid removing the edge e from the current spanning tree. The values of the parameters $IterAdd$ and $IterDrop$ are crucial for the efficiency of our algorithm. In order to fix them we performed some computational tests. Namely, we executed the tabu search algorithm for 50 instances from the classes *Ka-60*, ..., *Ka-100* and the average deviations from the optimum reported for various $IterAddd$ and $IterDrop$ are shown in Table 2.

Table 2 The average percent deviations from optimum reported for different values of $IterAdd$ and $IterDrop$.

	$IterDrop$								
$IterAdd$	0	2	4	6	8	10	12	14	16
0	0.17	0.00	0.09	0.14	0.60	0.65	2.29	2.44	2.74
2	0.57	0.00	0.00	0.15	1.02	2.05	3.09	4.78	6.06
4	0.17	0.04	0.04	0.83	1.54	3.67	5.78	6.00	8.09
6	0.18	0.00	0.00	1.05	2.47	4.08	5.44	6.44	9.18
8	0.08	0.00	0.35	1.30	3.39	4.96	5.89	8.12	9.17
10	0.00	0.00	0.15	0.95	2.99	4.96	6.34	9.27	10.64
12	0.26	0.00	0.09	1.55	2.96	5.28	7.26	8.72	10.37
14	0.00	0.00	0.08	1.67	3.55	5.62	7.67	9.64	11.49
16	0.08	0.00	0.15	1.95	2.70	5.00	7.04	9.25	10.20

It is well seen that the value of the parameter $IterDrop$ should be rather small and its best value is 0,1 or 2. For these values of $IterDrop$, the best values of $IterAdd$ seem to be located in the range $6, \ldots, 16$. We decided to choose $IterAdd = 10$ and $IterDrop = 2$ for the hard instances. Some additional experiments suggested that for the easy instances a slightly better results can be obtained if $IterAdd = 10$ and $IterDrop = 0$. We chose the class *Ka-n* for the tests, because we know the optimum for each instance in this class. Furthermore, the assumed values of $IterAdd$ and $IterDrop$ turned out to be good also for all the remaining classes and the corresponding results will be shown in the next sections.

The parameters are the same for all the tested instances and, in particular, they do not depend on the size of an instance. There are two reasons of this. With the number of iterations assumed the algorithm is always fast. For the largest instances, with 100 nodes, its average running time is about 60 seconds. Furthermore, as we will see from the obtained results, the algorithm with these settings returned an optimal solution for all the easy instances and we conjecture that its performance is not much worse for the hard instances.

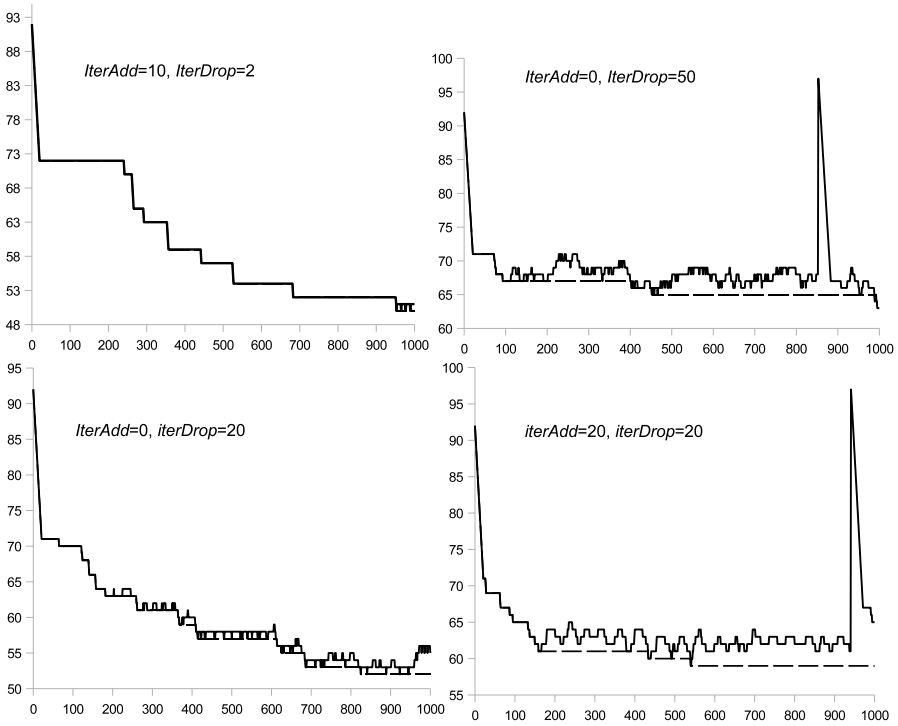

Fig. 9 Four sample executions of the tabu search algorithm for graph *Ka-100*. The dashed line represents the maximal regret of a best solution and the solid line represents the maximal regret of the current solution.

In Figure 9 four sample executions of the tabu search for various values of $IterAdd$ and $IterDrop$ are shown. As we can see, the algorithm starts with a random spanning tree and reaches a good solution within 1000 iterations. For the settings $IterAdd = 0$ and $IterDrop = 50$, between iterations 800 and 900, the restart can be observed after which a better solution is found.

5.3 The results of the preprocessing

We first check what happens in line 1 of the tabu search algorithm, i.e. how many non-weak and strong edges one can expect for each class of graphs. We investigate only the classes 1-4, since in the classes *Ka-n* and *La-n* all the edges are weak and there are no strong edges. The results obtained are shown in Table 3.

Table 3 The average percent of non-weak (nw.) and strong (str.) edges.

Class	nw.	str.	Class	nw.	str.	Class	nw.	str.
Ya(10,10)-10	46.00	1.33	*Ya(15,15)-10*	45.33	2.44	*Ya(20,20)-10*	47.56	5.11
Ya(10,10)-20	58.05	0.47	*Ya(15,15)-20*	59.74	0.74	*Ya(20,20)-20*	60.16	0.79
Ya(10,10)-30	65.79	0.11	*Ya(15,15)-30*	68.30	0.34	*Ya(20,20)-30*	68.07	0.28
Ya(10,10)-40	70.56	0.10	*Ya(15,15)-40*	72.33	0.14	*Ya(20,20)-40*	70.83	0.15
Ya(10,10)-50	73.71	0.08	*Ya(15,15)-50*	74.35	0.10	*Ya(20,20)-50*	73.84	0.04
Ya(10,10)-60	75.58	0.03	*Ya(15,15)-60*	76.02	0.03	*Ya(20,20)-60*	75.74	0.06
Ya(10,10)-70	78.05	0.04	*Ya(15,15)-70*	78.16	0.04	*Ya(20,20)-70*	77.16	0.04
Ya(10,10)-80	79.01	0.02	*Ya(15,15)-80*	78.64	0.05	*Ya(20,20)-80*	78.91	0.04
Ya(10,10)-90	80.18	0.02	*Ya(15,15)-90*	80.15	0.03	*Ya(20,20)-90*	80.02	0.02
Ya(10,10)-100	81.50	0.01	*Ya(15,15)-100*	81.21	0.02	*Ya(20,20)-100*	80.90	0.03
Ya(10,20)-10	21.56	0.89	*Ya(15,30)-10*	24.00	1.56	*Ya(20,40)-10*	26.22	2.00
Ya(10,20)-20	40.00	0.32	*Ya(15,30)-20*	44.37	0.53	*Ya(20,40)-20*	43.05	0.42
Ya(10,20)-30	50.94	0.09	*Ya(15,30)-30*	50.62	0.14	*Ya(20,40)-30*	54.07	0.16
Ya(10,20)-40	57.45	0.04	*Ya(15,30)-40*	57.65	0.10	*Ya(20,40)-40*	56.67	0.06
Ya(10,20)-50	61.75	0.05	*Ya(15,30)-50*	61.74	0.03	*Ya(20,40)-50*	62.97	0.07
Ya(10,20)-60	64.45	0.00	*Ya(15,30)-60*	65.99	0.02	*Ya(20,40)-60*	65.47	0.06
Ya(10,20)-70	68.02	0.02	*Ya(15,30)-70*	68.03	0.03	*Ya(20,40)-70*	67.64	0.01
Ya(10,20)-80	69.77	0.01	*Ya(15,30)-80*	70.93	0.01	*Ya(20,40)-80*	70.40	0.03
Ya(10,20)-90	71.90	0.01	*Ya(15,30)-90*	71.51	0.01	*Ya(20,40)-90*	71.98	0.01
Ya(10,20)-100	73.48	0.01	*Ya(15,30)-100*	73.02	0.02	*Ya(20,40)-100*	72.57	0.01
He1-10	53.78	4.22	*He2-10*	53.78	4.22			
He1-20	69.37	2.58	*He2-20*	60.78	3.83			
He1-30	77.31	1.91	*He2-30*	61.03	3.24			
He1-40	79.72	1.27	*He2-40*	61.65	3.02			
He1-50	82.40	1.00	*He2-50*	62.03	3.82			
He1-60	84.30	0.86	*He2-60*	61.39	3.42			
He1-70	86.00	0.68	*He2-70*	62.52	3.98			
He1-80	87.24	0.66	*He2-80*	61.78	3.27			
He1-90	87.98	0.58	*He2-90*	62.03	3.60			
He1-100	88.87	0.53	*He2-100*	61.90	3.59			
Mo(0.15)-10	71.33	14.22	*Mo(0.85)-10*	40.44	4.89	*Mo(0.50)-10*	58.89	9.56
Mo(0.15)-20	87.11	8.00	*Mo(0.85)-20*	59.84	2.05	*Mo(0.50)-20*	76.53	4.32
Mo(0.15)-30	91.22	5.13	*Mo(0.85)-30*	69.38	1.06	*Mo(0.50)-30*	83.86	2.76
Mo(0.15)-40	93.29	3.79	*Mo(0.85)-40*	75.15	0.95	*Mo(0.50)-40*	87.44	2.09
Mo(0.15)-50	94.84	3.02	*Mo(0.85)-50*	79.51	0.69	*Mo(0.50)-50*	90.68	1.64
Mo(0.15)-60	95.63	2.54	*Mo(0.85)-60*	81.58	0.52	*Mo(0.50)-60*	91.80	1.35
Mo(0.15)-70	96.18	2.14	*Mo(0.85)-70*	84.43	0.51	*Mo(0.50)-70*	92.88	1.16
Mo(0.15)-80	96.64	1.84	*Mo(0.85)-80*	86.70	0.34	*Mo(0.50)-80*	93.83	0.99
Mo(0.15)-90	96.95	1.64	*Mo(0.85)-90*	87.44	0.28	*Mo(0.50)-90*	94.14	0.82
Mo(0.15)-100	97.33	1.50	*Mo(0.85)-100*	88.85	0.32	*Mo(0.50)-100*	94.97	0.78

One can see that the number of the non-weak edges is very large. For the class *Mo(p)-n* over 90% of the edges may be non-weak. One can also expect several strong edges in each instance. It is interesting that the percent of the non-weak edges increases with the number of nodes and this holds true for each class of graphs. We can thus conclude that it is very advantageous to perform the preprocessing, because it allows us to avoid many unnecessary moves during the execution of the algorithm. Our results confirm the observations noticed previously in papers [4,23,31].

5.4 The results for the easy instances

In this section we discuss the easy instances, for which the optimal solutions are known. All these instances, except for the ones in the class *Ka-n*, were solved by CPLEX. The minimal, average and maximal computational times required by CPLEX to solve the instances are presented in Table 4. These results demonstrate that the MIP formulation is an inefficient method of solving the problem, which confirms the conclusions made previously in literature [4, 23,31]. The running time of CPLEX grows very fast with the problem size and, in practice, only very small instances can be solved in reasonable time.

Table 4 The minimal (min), average (av.), and maximal (max) computational times in seconds required by CPLEX to solve the problem.

	CPLEX time (s.)				CPLEX time (s.)		
Class	min	av.	max	Class	min	av.	max
Ya(10,10)-10	0.14	0.40	0.61	*Ya(15,15)-10*	0.19	0.49	1.00
Ya(10,10)-20	25.12	324.19	755.56	*Ya(15,15)-20*	40.78	213.40	794.11
Ya(20,20)-10	0.16	0.34	0.64	*Ya(10,20)-10*	0.19	0.49	0.87
Ya(20,20)-20	28.95	150.18	641.66	*Ya(10,20)-20*	21.23	112.11	176.55
Ya(15,30)-10	0.23	0.56	1.33	*Ya(20,40)-10*	0.19	0.49	1.26
Ya(15,30)-20	28.69	147.78	393.31	*Ya(20,40)-20*	29.83	103.39	202.51
He1-10	0.16	0.40	1.44	*He2-10*	0.14	0.41	1.45
				He2-20	4.84	24.18	48.59
				He2-30	25.46	604.37	1847.72
Mo(0.15)-10	0.05	0.24	0.78	*Mo(0.50)-10*	0.12	0.26	0.48
Mo(0.15)-20	7.18	31.78	113.90	*Mo(0.50)-20*	10.08	85.56	273.22
Mo(0.85)-10	0.19	0.33	0.76	*La-10*	0.12	0.28	0.59
Mo(0.85)-20	23.37	401.83	2413.62	*La-20*	191.04	1775.38	4931.49

Table 5 shows the application of AMU, the single-run iterative improvement, and the tabu search to the easy instances. We additionally consider the algorithm AR, which simply outputs a random spanning tree (recall that a random spanning tree is a starting solution for the tabu search). As we can see, the tabu search found an optimal solution for all the 340 easy instances (10 instances in each of 34 classes), even for all the instances of the class *Ka-n*. In column f(s.), the average time in seconds when an optimal solution was found is shown and this time can be compared to the average running time

of the algorithm shown in column t(s.). We can conclude that, for the easy instances, the tabu search algorithm finds an optimal solution very quickly. For the considered classes, it outperforms CPLEX as well as the branch and bound algorithms proposed in the literature [4,23]. The approximation algorithm AMU seems to perform quite well, except for the classes *Ka-n* and *La-n*. This is not surprising, because for the instances of these classes AMU behaves exactly the same as the random algorithm AR and AR is very poor as one could expect. If we omit the classes *Ka-n* and *La-n*, then the worst deviation reported for AMU over all the remaining 220 easy instances is 17.90%. Finally, observe that the single-run iterative improvement is not much worse than the tabu search, except for the classes *Ka-n* and *La-n* for which the tabu search is much better. For the class *La-n*, the single-run iterative improvement is only slightly better than the random algorithm AR, while the tabu search always finds an optimal solution.

Table 5 The minimum (min), average (av.) and maximum (max) percent deviation from optimum reported for the easy instances. For the classes *Ka-n* and *La-n*, AMU and AR are equivalent. The running time of the iter. impr. is less than 1 s. for each instance.

	AMU			Sng.-run it. impr.			AR			Tabu search				
Class	min	av.	max	min	av.	max	min	av.	max	min	av.	max	f(s.)	t(s.)
Ya(10,10)-10	0.00	2.27	7.38	0.00	0.00	0.00	52.56	113.94	162.56	0.00	0.00	0.00	0.00	0.04
Ya(10,10)-20	0.00	2.22	4.49	0.00	0.11	1.09	83.68	130.02	200.05	0.00	0.00	0.00	0.02	0.43
Ya(15,15)-10	0.00	0.94	5.33	0.00	0.27	2.69	67.57	109.59	156.17	0.00	0.00	0.00	0.00	0.04
Ya(15,15)-20	0.00	0.94	5.33	0.00	0.00	0.00	99.71	137.65	188.09	0.00	0.00	0.00	0.01	0.44
Ya(20,20)-10	0.00	2.69	8.30	0.00	0.00	0.00	41.56	92.32	207.78	0.00	0.00	0.00	0.00	0.03
Ya(20,20)-20	0.00	0.86	3.80	0.00	0.00	0.00	93.89	149.33	194.92	0.00	0.00	0.00	0.01	0.41
Ya(10,20)-10	0.00	0.50	3.03	0.00	0.00	0.00	67.82	141.74	302.79	0.00	0.00	0.00	0.00	0.09
Ya(10,20)-20	0.00	0.43	1.40	0.00	0.00	0.00	131.37	164.67	206.84	0.00	0.00	0.00	0.01	0.55
Ya(15,30)-10	0.00	2.23	11.58	0.00	0.00	0.00	80.92	111.79	233.39	0.00	0.00	0.00	0.00	0.08
Ya(15,30)-20	0.00	0.77	2.86	0.00	0.00	0.00	149.79	199.96	236.80	0.00	0.00	0.00	0.01	0.45
Ya(20,40)-10	0.00	1.01	3.79	0.00	0.00	0.00	79.78	131.41	165.79	0.00	0.00	0.00	0.00	0.05
Ya(20,40)-20	0.00	0.25	1.70	0.00	0.00	0.00	126.96	217.75	312.70	0.00	0.00	0.00	0.01	0.42
He1-10	0.00	4.59	10.19	0.00	0.47	4.73	50.08	448.15	1388.16	0.00	0.00	0.00	0.00	0.02
He2-10	0.00	4.59	10.19	0.00	0.47	4.73	50.08	448.15	1388.16	0.00	0.00	0.00	0.00	0.02
He2-20	0.00	3.91	10.39	0.00	0.28	2.78	218.29	472.34	912.52	0.00	0.00	0.00	0.00	0.10
He2-30	0.00	2.14	7.02	0.00	0.19	1.88	207.06	510.81	817.52	0.00	0.00	0.00	0.00	0.23
Mo(0.15)-10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	413.86	1849.10	0.00	0.00	0.00	0.00	0.01
Mo(0.15)-20	0.00	1.93	17.90	0.00	0.00	0.00	0.00	175.25	770.43	0.00	0.00	0.00	0.00	0.02
Mo(0.50)-10	0.00	0.64	5.86	0.00	0.00	0.00	128.62	254.97	506.71	0.00	0.00	0.00	0.00	0.01
Mo(0.50)-20	0.00	1.94	6.23	0.00	0.00	0.00	86.71	190.54	299.77	0.00	0.00	0.00	0.00	0.06
Mo(0.85)-10	0.00	0.14	1.42	0.00	0.00	0.00	136.86	277.51	486.63	0.00	0.00	0.00	0.00	0.02
Mo(0.85)-20	0.00	2.56	7.40	0.00	0.00	0.00	185.15	303.58	477.71	0.00	0.00	0.00	0.00	0.19
Ka-10	-	-	-	0.00	30.00	60.00	60.00	74.00	80.00	0.00	0.00	0.00	0.00	0.03
Ka-20	-	-	-	30.00	43.00	70.00	70.00	81.00	90.00	0.00	0.00	0.00	0.01	0.25
Ka-30	-	-	-	13.33	39.33	66.67	73.33	83.33	93.33	0.00	0.00	0.00	0.08	0.88
Ka-40	-	-	-	20.00	42.50	65.00	80.00	85.50	90.00	0.00	0.00	0.00	0.25	2.27
Ka-50	-	-	-	32.00	42.40	52.00	80.00	84.40	92.00	0.00	0.00	0.00	0.99	4.68
Ka-60	-	-	-	23.33	37.00	53.33	83.33	86.67	93.33	0.00	0.00	0.00	1.99	9.06
Ka-70	-	-	-	20.00	35.14	57.14	82.86	88.57	94.29	0.00	0.00	0.00	3.77	15.03
Ka-80	-	-	-	20.00	35.00	45.00	82.50	87.75	92.50	0.00	0.00	0.00	9.09	25.16
Ka-90	-	-	-	24.44	34.89	46.67	86.67	89.56	91.11	0.00	0.00	0.00	14.57	36.19
Ka-100	-	-	-	28.00	38.00	46.00	82.00	86.80	92.00	0.00	0.00	0.00	31.37	53.52
La-10	-	-	-	14.29	22.86	33.33	28.57	44.05	50.00	0.00	0.00	0.00	0.00	0.02
La-20	-	-	-	20.00	22.57	28.57	26.67	29.38	35.71	0.00	0.00	0.00	0.09	0.24

All the easy instances were also solved by using the multi-run random and perturbed iterative improvement algorithms (see Section 4.2). In order to compare these algorithms to the tabu search, we fixed their running time to be approximately the same as the running time of the tabu search. For all the considered classes of graphs, except for $Ka-n$ and $La-n$, the performance of both multi-run iterative improvement algorithms is the same as the performance of the tabu search, i.e. an optimal solution was always found. However, for the classes $Ka-n$ and $La-n$ the performance of the tabu search is much better as we can see in Table 6. For the class $Ka-n$ with $n \geq 40$, the multi-run iterative improvement algorithm had never found an optimal solution. Notice that for both $Ka-n$ and $La-n$ the perturbed and random versions of the iterative improvement algorithm are equivalent.

Table 6 The minimum (min), average (av.) and maximum (max) percent deviation from optimum reported for the easy instances in the classes *Ka-n* and *La-n* by using different versions of the iterative improvement algorithm and the tabu search.

	Sng.-run			Mult.-run rand. (pert.)				Tabu search			
Class	min	av.	max	min	av.	max	t(s.)	min	av.	max	t(s.)
Ka-10	0.00	30.00	60.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.03
Ka-20	30.00	43.00	70.00	0.00	4.00	10.00	0.44	0.00	0.00	0.00	0.25
Ka-30	13.33	39.33	66.67	0.00	9.33	13.33	1.16	0.00	0.00	0.00	0.88
Ka-40	20.00	42.50	65.00	5.00	13.00	20.00	2.62	0.00	0.00	0.00	2.27
Ka-50	32.00	42.40	52.00	12.00	17.20	24.00	4.96	0.00	0.00	0.00	4.68
Ka-60	23.33	37.00	53.33	13.33	19.33	23.33	8.90	0.00	0.00	0.00	9.06
Ka-70	20.00	35.14	57.14	14.29	18.29	25.71	14.93	0.00	0.00	0.00	15.03
Ka-80	20.00	35.00	45.00	15.00	22.25	30.00	22.50	0.00	0.00	0.00	25.16
Ka-90	24.44	34.89	46.67	15.56	19.78	24.44	32.52	0.00	0.00	0.00	36.19
Ka-100	28.00	38.00	46.00	16.00	22.60	28.00	47.51	0.00	0.00	0.00	53.52
La-10	14.29	22.86	33.33	0.00	0.00	0.00	0.12	0.00	0.00	0.00	0.02
La-20	20.00	22.57	28.57	6.67	8.95	14.29	0.34	0.00	0.00	0.00	0.24

5.5 The results for the hard instances

In this section we consider the hard instances, for which the optimal solutions cannot be computed efficiently by using CPLEX. For each hard instance, we define a *reference solution* as the best one obtained by executing multiple times the tabu search algorithm, with different random starting solutions and different settings. Hence each reference solution is the best heuristic solution available for each particular instance. For all the hard instances we computed the deviations from the reference solutions. The results are presented in Tables 7-10.

The tabu search algorithm found the reference solution for almost all tested instances. The worst performance was reported for the class *La-n*. However, even for this artificial and hard class of graphs, the tabu search was able to improve the starting solutions significantly in contrast to the iterative improvement. Notice, however, that the single-run iterative improvement is not worse than the tabu search for the classes *Ya(u,l)-n* and *Mo(p)-n*. This suggests that

Table 7 The minimum (min), average (av.) and maximum (max) percent deviation from the reference solutions reported for the class *Ya(l,u)-n*. The running time of the iter. impr. is less than 1 s. for each instance.

	AMU			Sng.-run it. impr.			AR			Tabu search				
Class	min	av.	max	min	av.	max	min	av.	max	min	av.	max	f(s.)	t(s.)
Ya(10,10)-30	0.00	0.78	2.48	0.00	0.00	0.00	118.97	161.24	198.12	0.00	0.00	0.00	0.03	1.27
Ya(10,10)-40	0.00	0.93	1.98	0.00	0.00	0.02	152.95	196.75	243.16	0.00	0.00	0.00	0.21	2.88
Ya(10,10)-50	0.22	0.51	0.94	0.00	0.00	0.00	183.83	221.81	263.87	0.00	0.00	0.00	0.24	5.61
Ya(10,10)-60	0.00	0.57	1.29	0.00	0.00	0.00	196.37	237.39	296.28	0.00	0.00	0.00	0.52	9.81
Ya(10,10)-70	0.00	0.23	0.53	0.00	0.00	0.00	229.02	261.95	314.80	0.00	0.00	0.00	0.97	15.29
Ya(10,10)-80	0.00	0.30	0.78	0.00	0.00	0.00	238.83	290.45	322.68	0.00	0.00	0.00	2.53	35.29
Ya(10,10)-90	0.00	0.16	0.73	0.00	0.00	0.00	239.66	298.26	343.76	0.00	0.00	0.00	6.04	45.24
Ya(10,10)-100	0.00	0.10	0.26	0.00	0.00	0.00	292.37	320.49	344.53	0.00	0.00	0.00	5.36	57.11
Ya(15,15)-30	0.00	1.22	3.76	0.00	0.27	2.69	117.39	172.91	297.81	0.00	0.00	0.00	0.03	1.22
Ya(15,15)-40	0.02	0.56	2.61	0.00	0.00	0.00	160.55	194.47	237.15	0.00	0.00	0.00	0.10	2.82
Ya(15,15)-50	0.00	0.57	1.64	0.00	0.00	0.00	176.39	224.38	257.68	0.00	0.00	0.00	0.34	7.86
Ya(15,15)-60	0.00	0.25	1.08	0.00	0.00	0.00	198.03	246.53	277.20	0.00	0.00	0.00	0.52	9.98
Ya(15,15)-70	0.00	0.44	1.73	0.00	0.00	0.00	208.09	252.90	299.34	0.00	0.00	0.00	1.37	22.04
Ya(15,15)-80	0.00	0.29	0.63	0.00	0.00	0.00	253.55	281.65	305.32	0.00	0.00	0.00	2.54	36.07
Ya(15,15)-90	0.00	0.18	0.43	0.00	0.00	0.00	277.06	313.27	336.99	0.00	0.00	0.00	3.65	44.42
Ya(15,15)-100	0.00	0.16	0.58	0.00	0.00	0.00	316.48	342.19	376.14	0.00	0.00	0.00	5.50	62.04
Ya(20,20)-30	0.05	1.36	4.12	0.00	0.00	0.00	112.55	181.59	240.29	0.00	0.00	0.00	0.03	1.22
Ya(20,20)-40	0.27	0.77	1.16	0.00	0.04	0.44	160.43	208.83	270.90	0.00	0.01	0.07	0.24	2.76
Ya(20,20)-50	0.00	0.50	1.34	0.00	0.00	0.00	178.35	226.99	264.38	0.00	0.00	0.00	0.35	8.27
Ya(20,20)-60	0.00	0.33	0.79	0.00	0.00	0.00	199.99	233.92	272.02	0.00	0.00	0.00	0.50	9.87
Ya(20,20)-70	0.00	0.26	1.13	0.00	0.00	0.00	220.16	247.86	294.96	0.00	0.00	0.00	1.42	22.03
Ya(20,20)-80	0.01	0.25	0.60	0.00	0.00	0.00	236.96	294.14	354.38	0.00	0.00	0.00	2.56	37.22
Ya(20,20)-90	0.00	0.22	0.75	0.00	0.00	0.00	244.81	319.43	355.97	0.00	0.00	0.00	3.76	45.07
Ya(20,20)-100	0.00	0.16	0.57	0.00	0.00	0.00	255.47	318.82	366.94	0.00	0.00	0.00	5.39	58.08
Ya(10,20)-30	0.00	0.33	1.50	0.00	0.02	0.16	190.88	235.58	287.95	0.00	0.00	0.00	0.04	1.60
Ya(10,20)-40	0.00	0.39	1.42	0.00	0.00	0.00	224.84	272.12	353.27	0.00	0.00	0.00	0.13	3.71
Ya(10,20)-50	0.00	0.21	0.80	0.00	0.00	0.00	259.45	308.20	368.10	0.00	0.00	0.00	0.46	10.45
Ya(10,20)-60	0.00	0.15	0.45	0.00	0.00	0.00	249.68	333.29	386.49	0.00	0.00	0.00	0.69	12.41
Ya(10,20)-70	0.00	0.09	0.38	0.00	0.00	0.00	341.63	384.97	440.05	0.00	0.00	0.00	1.81	26.25
Ya(10,20)-80	0.00	0.09	0.52	0.00	0.00	0.00	393.28	429.39	471.34	0.00	0.00	0.00	3.30	43.44
Ya(10,20)-90	0.00	0.06	0.31	0.00	0.00	0.00	363.81	439.26	497.86	0.00	0.00	0.00	5.02	60.24
Ya(10,20)-100	0.00	0.04	0.13	0.00	0.00	0.00	412.97	456.80	522.31	0.00	0.00	0.00	7.03	72.85
Ya(15,30)-30	0.00	0.21	1.03	0.00	0.00	0.00	184.71	243.54	342.82	0.00	0.00	0.00	0.04	1.61
Ya(15,30)-40	0.00	0.42	1.17	0.00	0.00	0.00	231.64	297.89	410.65	0.00	0.00	0.00	0.13	3.37
Ya(15,30)-50	0.00	0.12	0.48	0.00	0.00	0.00	240.53	311.30	352.21	0.00	0.00	0.00	0.47	9.85
Ya(15,30)-60	0.00	0.19	0.77	0.00	0.00	0.00	305.63	358.19	456.39	0.00	0.00	0.00	0.67	12.36
Ya(15,30)-70	0.00	0.07	0.26	0.00	0.00	0.00	330.49	386.50	439.27	0.00	0.00	0.00	1.71	26.24
Ya(15,30)-80	0.00	0.05	0.17	0.00	0.00	0.00	363.36	436.15	512.61	0.00	0.00	0.00	3.14	41.76
Ya(15,30)-90	0.00	0.07	0.28	0.00	0.00	0.00	357.38	447.26	521.52	0.00	0.00	0.00	4.75	56.05
Ya(15,30)-100	0.00	0.04	0.25	0.00	0.00	0.00	459.59	498.36	554.28	0.00	0.00	0.00	7.07	73.88
Ya(20,40)-30	0.00	0.36	1.97	0.00	0.00	0.00	181.33	246.90	295.96	0.00	0.00	0.00	0.04	1.52
Ya(20,40)-40	0.00	0.11	0.55	0.00	0.00	0.00	184.47	281.99	376.21	0.00	0.00	0.00	0.13	3.52
Ya(20,40)-50	0.00	0.08	0.47	0.00	0.00	0.00	252.11	328.89	407.05	0.00	0.00	0.00	0.43	9.51
Ya(20,40)-60	0.00	0.08	0.32	0.00	0.00	0.00	300.95	371.84	426.07	0.00	0.00	0.00	0.64	11.86
Ya(20,40)-70	0.00	0.08	0.33	0.00	0.00	0.00	339.78	404.53	494.57	0.00	0.00	0.00	1.80	27.36
Ya(20,40)-80	0.00	0.06	0.16	0.00	0.00	0.00	349.46	412.48	451.09	0.00	0.00	0.00	3.11	42.74
Ya(20,40)-90	0.00	0.07	0.27	0.00	0.00	0.00	392.29	458.14	535.22	0.00	0.00	0.00	4.94	56.92
Ya(20,40)-100	0.00	0.04	0.20	0.00	0.00	0.00	362.92	483.75	543.70	0.00	0.00	0.00	7.05	73.44

every local minimum for these two classes is, in most cases, the global one and it is enough to compute such a local minimum. The single-run iterative improvement behaves a little bit worse for the classes *He1-n* and *He2-n*. Finally, the tabu search is much better than the single-run iterative improvement for the class *La-n*. In fact, for *La-n* the single-run iterative improvement is not much better than the random algorithm AR. Observe also that, except for the

Table 8 The minimum (min), average (av.) and maximum (max) percent deviation from the reference solutions reported for the classes *He1-n* and *He2-n*.The running time of the iter. impr. is less than 1 s. for each instance.

	AMU			Sng.-run it. impr.			AR			Tabu search				
Class	min	av.	max	min	av.	max	min	av.	max	min	av.	max	f(s.)	t(s.)
He1-20	0.00	4.64	10.56	0.00	0.00	0.00	375.76	825.85	1173.97	0.00	0.00	0.00	0.00	0.16
He1-30	1.05	2.65	6.76	0.00	0.00	0.00	418.43	907.80	1230.08	0.00	0.00	0.00	0.00	0.35
He1-40	0.11	3.02	5.74	0.00	0.00	0.00	795.08	1162.99	1630.44	0.00	0.00	0.00	0.02	0.81
He1-50	0.52	3.19	7.38	0.00	0.00	0.00	1014.72	1461.61	2358.88	0.00	0.00	0.00	0.14	2.34
He1-60	0.82	3.66	7.75	0.00	0.00	0.00	1219.48	1502.14	2138.16	0.00	0.00	0.00	0.32	2.41
He1-70	1.33	2.78	4.49	0.00	0.03	0.28	1152.68	1610.04	2185.50	0.00	0.00	0.00	0.32	5.65
He1-80	0.81	1.93	4.76	0.00	0.02	0.22	1439.18	1777.22	2496.94	0.00	0.00	0.00	0.93	8.30
He1-90	0.81	2.75	8.20	0.00	0.00	0.00	1419.15	1675.45	2085.29	0.00	0.00	0.00	0.81	9.92
He1-100	1.04	2.72	5.22	0.00	0.02	0.23	1248.96	1651.72	1991.89	0.00	0.00	0.00	2.24	12.63
He2-40	1.08	2.79	4.89	0.00	0.12	1.16	295.80	671.27	1106.72	0.00	0.00	0.00	0.04	0.50
He2-50	1.02	2.93	5.87	0.00	0.09	0.90	340.53	701.43	1124.78	0.00	0.03	0.31	0.20	0.79
He2-60	0.96	2.89	5.49	0.00	0.09	0.88	589.19	727.84	1061.20	0.00	0.00	0.00	0.09	1.23
He2-70	1.41	3.56	6.64	0.00	0.25	1.76	551.87	750.19	1125.34	0.00	0.00	0.00	0.09	1.94
He2-80	0.79	2.86	6.10	0.00	0.07	0.69	508.46	670.64	847.94	0.00	0.00	0.00	0.21	2.30
He2-90	1.74	3.27	5.67	0.00	0.10	0.63	552.87	677.75	897.97	0.00	0.03	0.31	0.38	2.93
He2-100	1.40	3.43	4.77	0.00	0.07	0.59	534.71	682.41	912.68	0.00	0.00	0.00	0.43	4.32

Table 9 The minimum (min), average (av.) and maximum (max) percent deviation from the reference solutions reported for the class *Mo(p)-n*.The running time of the iter. impr. is less than 1 s. for each instance.

	AMU			Sng.-run it. impr.			AR			Tabu search				
Class	min	av.	max	min	av.	max	min	av.	max	min	av.	max	f(s.)	t(s.)
Mo(0.15)-30	0.00	0.91	5.93	0.00	0.00	0.00	1.77	103.67	209.77	0.00	0.00	0.00	0.00	0.07
Mo(0.15)-40	0.00	0.25	1.67	0.00	0.00	0.00	94.85	205.27	404.48	0.00	0.00	0.00	0.01	0.10
Mo(0.15)-50	0.00	0.71	4.61	0.00	0.00	0.00	74.41	181.35	409.33	0.00	0.00	0.00	0.00	0.14
Mo(0.15)-60	0.00	0.25	2.25	0.00	0.00	0.00	34.03	154.56	240.00	0.00	0.00	0.00	0.01	0.19
Mo(0.15)-70	0.00	0.38	1.67	0.00	0.00	0.00	90.21	141.32	203.01	0.00	0.00	0.00	0.02	0.31
Mo(0.15)-80	0.00	1.07	3.27	0.00	0.00	0.00	80.90	160.18	245.22	0.00	0.00	0.00	0.02	0.45
Mo(0.15)-90	0.00	1.17	4.33	0.00	0.00	0.00	143.46	189.81	299.77	0.00	0.00	0.00	0.03	0.67
Mo(0.15)-100	0.00	0.57	4.43	0.00	0.00	0.00	102.77	190.33	276.22	0.00	0.00	0.00	0.05	0.78
Mo(0.50)-30	0.00	0.98	5.92	0.00	0.00	0.00	177.81	289.59	438.08	0.00	0.00	0.00	0.01	0.19
Mo(0.50)-40	0.00	2.44	5.13	0.00	0.00	0.00	150.32	261.07	416.11	0.00	0.00	0.00	0.01	0.41
Mo(0.50)-50	0.60	2.84	5.63	0.00	0.00	0.00	149.17	230.78	369.63	0.00	0.00	0.00	0.02	0.65
Mo(0.50)-60	0.16	2.30	4.11	0.00	0.00	0.00	139.54	237.99	322.41	0.00	0.00	0.00	0.03	0.98
Mo(0.50)-70	0.00	2.94	6.51	0.00	0.00	0.00	187.05	256.28	359.17	0.00	0.00	0.00	0.06	1.53
Mo(0.50)-80	0.30	3.45	8.84	0.00	0.00	0.00	179.63	234.66	311.72	0.00	0.00	0.00	0.09	2.25
Mo(0.50)-90	0.57	3.24	5.43	0.00	0.00	0.00	235.77	286.50	353.49	0.00	0.00	0.00	0.17	3.36
Mo(0.50)-100	0.77	2.10	3.81	0.00	0.00	0.03	216.43	247.33	281.41	0.00	0.00	0.00	0.42	4.20
Mo(0.85)-30	0.01	1.56	4.00	0.00	0.00	0.00	311.78	438.17	630.02	0.00	0.00	0.00	0.02	0.63
Mo(0.85)-40	0.31	2.41	4.19	0.00	0.00	0.00	334.85	450.97	628.65	0.00	0.00	0.00	0.04	1.25
Mo(0.85)-50	0.21	1.38	3.44	0.00	0.01	0.09	323.71	491.38	616.45	0.00	0.00	0.00	0.20	2.17
Mo(0.85)-60	0.50	1.78	3.25	0.00	0.01	0.12	329.70	505.74	707.51	0.00	0.00	0.00	0.37	3.75
Mo(0.85)-70	0.89	2.38	5.40	0.00	0.00	0.00	418.85	545.98	729.56	0.00	0.00	0.00	0.32	5.41
Mo(0.85)-80	0.34	1.67	2.98	0.00	0.01	0.12	407.23	501.53	615.86	0.00	0.00	0.00	0.54	8.33
Mo(0.85)-90	0.90	2.01	3.20	0.00	0.00	0.00	453.52	560.05	640.01	0.00	0.00	0.00	0.91	13.36
Mo(0.85)-100	0.68	1.71	2.81	0.00	0.02	0.14	414.62	553.35	677.20	0.00	0.00	0.00	1.83	13.98

class *La-n*, the approximation algorithm AMU returns good solutions in most cases. Therefore, it may be a good choice if a solution for a large graph must be computed quickly.

We also executed the multi-run random and perturbed iterative improvement algorithms (see Section 4.2) for all the hard instances. The running time of both algorithms was fixed to be approximately the same as the running

Table 10 The minimum (min), average (av.) and maximum (max) percent deviation from the reference solutions reported for the class *La-n*. The algorithms AMU and AR are equivalent. The running time of the iter. impr. is less than 1 s. for each instance.

	AMU			Sng.-run it. impr.			AR			Tabu search				
Class	min	av.	max	min	av.	max	min	av.	max	min	av.	max	f(s.)	t(s.)
La-30	-	-	-	12.50	22.23	28.57	20.83	27.96	38.10	0.00	2.73	9.52	0.23	0.81
La-40	-	-	-	15.15	20.40	26.67	18.18	23.89	30.00	0.00	0.31	3.13	0.89	1.67
La-50	-	-	-	17.07	19.24	23.08	19.51	22.23	25.64	0.00	1.01	2.56	0.61	3.03
La-60	-	-	-	18.37	21.68	28.89	20.41	23.78	31.11	0.00	1.94	6.67	2.14	5.31
La-70	-	-	-	15.52	20.24	25.93	18.97	22.18	27.78	0.00	1.24	3.57	4.31	8.32
La-80	-	-	-	16.42	21.53	23.81	17.91	23.09	25.40	0.00	0.00	0.00	6.70	11.77
La-90	-	-	-	17.33	19.77	22.22	18.67	21.13	23.61	0.00	0.27	1.37	13.82	17.28
La-100	-	-	-	15.29	17.25	20.99	16.47	18.45	22.22	0.00	1.07	3.53	16.50	25.98

time of the tabu search. For all the classes of graphs discussed in the existing literature, i.e. $Ya(u,l)-n$, $Mo(p)-n$, $He1-n$ and $He2-n$, both multi-run iterative improvement algorithms gave us the same results as the tabu search, i.e. for each particular instance a reference solution was almost always found (with a few exceptions). However, in most cases the tabu search was able to find a reference solution faster than the iterative improvements. Perhaps, this follows from the fact that the multi-run iterative improvement algorithms generate a lot of random initial solutions, which is time consuming, while the tabu search generates an initial solution only several times when performing the restart. More interesting results were obtained for the class $La-n$. For the instances in this class the tabu search algorithms outperforms the multi-run iterative improvement and the results are shown in Table 11. Notice that for the class $La-n$ the random and perturbed versions of the iterative improvement algorithm are equivalent.

Table 11 The minimum (min), average (av.) and maximum (max) percent deviation from optimum reported for the hard instances in the class *La-n* by using different versions of the iterative improvement algorithm and the tabu search.

	Sng.-run			Mult.-run rand. (pert.)				Tabu search			
Class	min	av.	max	min	av.	max	t(s.)	min	av.	max	t(s.)
La-30	12.50	22.23	28.57	8.33	15.85	23.81	0.80	0.00	2.73	9.52	0.81
La-40	15.15	20.40	26.67	9.38	15.40	20.00	1.68	0.00	0.31	3.13	1.67
La-50	17.07	19.24	23.08	12.20	15.52	17.95	3.80	0.00	1.01	2.56	3.03
La-60	18.37	21.68	28.89	14.29	18.39	24.44	5.26	0.00	1.94	6.67	5.31
La-70	15.52	20.24	25.93	15.79	17.82	22.22	10.07	0.00	1.24	3.57	8.32
La-80	16.42	21.53	23.81	13.43	18.92	22.22	12.71	0.00	0.00	0.00	11.77
La-90	17.33	19.77	22.22	14.67	17.77	19.44	20.06	0.00	0.27	1.37	17.28
La-100	15.29	17.25	20.99	12.94	15.94	18.52	29.25	0.00	1.07	3.53	25.98

Now an important question arises how far the reference solutions from the corresponding global optima are. We tried to provide an answer by performing an additional experiment in which we computed lower bounds for all the hard instances. Notice that a trivial lower bound can be obtained by using the algorithm AM. If this algorithm returns a spanning tree T with a maximal regret equal to $Z(T)$, then the lower bound is $Z(T)/2$. This lower bound can

be improved by executing the branch and bound algorithm proposed in [5]. We implemented a slightly modified version of this algorithm and applied it to all the hard instances. For each instance we executed the branch and bound algorithm for 30 seconds and reported the lower bound obtained. The assumed running time of the branch and bound algorithm seems to be small. However, increasing it does not significantly increase the lower bounds obtained but significantly increases the computer resources required. Notice that in [5] the branch and bound algorithm was executed for small graphs (with up to 40 nodes). For graphs having more than 40 nodes the algorithm converges very slowly and consumes a lot of computer memory to store a search tree (in fact, without a sophisticated implementation, the memory is quickly exhausted). The results are shown in Table 12.

For each class of graphs we computed the minimum, average and maximum gap, i.e. the quantity $(Z(T) - LB)/LB)$, where T is the reference solution and LB is the lower bound. The largest gaps, which are close the the trivial gap 100%, were reported for the class *Ya(l,u)-n*. For the remaining classes the test confirmed a good performance of the tabu search algorithm for the graphs with up to 40 nodes (for the class *Mo(0.15)-n* this performance was good for all the graphs in this class). In particular, the gaps equal to 0 mean that the tabu search found an optimal solution for some (or all) instances within a class. For the graphs with more than 40 nodes the gaps are larger which, however, may be a consequence of the weak lower bounds returned by the branch and bound algorithm.

After analyzing the results of the tests presented in this section, we conjecture that the reference solutions for the hard instances are nearly optimal (or even optimal in many cases). For the hard instances the tabu search algorithm is still very fast. It finds a reference solution usually in a few seconds. It clearly outperforms CPLEX and is much easier to implement than the branch and bound algorithm.

5.5.1 A comparison with the simulated annealing algorithm

In [26] a simulated algorithm for the minmax regret minimum spanning tree problem has been constructed which can also quickly provide solutions for large instances. We now compare the results obtained in [26] with the results obtained by our algorithm. In order to do this, we performed similar experiments as in [26]. In [26] only one class of graphs was considered, namely *Ya(20,40)-n*. For each $n \in \{5, 10, 15, 20\}$ we have generated 10 instances and for each instance we have executed the tabu search algorithm 10 times, each time starting from a different random spanning tree. Hence, for each n we obtained 100 solutions. We performed two experiments. In the first one the algorithm terminated after performing 40 iterations and in the second the algorithm terminated after performing 1000 iterations. The remaining settings are the same as in the previous sections. The obtained results are shown in Table 13.

Table 12 The gaps for the hard instances.

Class	% min	% av	%max	Class	% min	% av.	%max
Ya(10,10)-30	78.83	89.12	95.70	*Ya(15,15)-30*	36.24	79.36	92.76
Ya(10,10)-40	84.44	89.31	94.83	*Ya(15,15)-40*	83.06	87.97	92.49
Ya(10,10)-50	86.03	90.59	96.70	*Ya(15,15)-50*	84.85	89.34	92.26
Ya(10,10)-60	84.41	89.54	91.99	*Ya(15,15)-60*	85.14	88.29	90.11
Ya(10,10)-70	86.20	90.21	94.17	*Ya(15,15)-70*	86.16	88.52	92.76
Ya(10,10)-80	83.05	89.19	94.09	*Ya(15,15)-80*	84.07	88.15	91.18
Ya(10,10)-90	84.93	88.22	92.30	*Ya(15,15)-90*	84.57	88.58	93.95
Ya(10,10)-100	86.09	88.68	93.55	*Ya(15,15)-100*	87.57	89.21	92.28
Ya(20,20)-30	81.77	91.87	99.74	*Ya(10,20)-30*	84.08	92.48	97.56
Ya(20,20)-40	74.58	89.65	97.24	*Ya(10,20)-40*	83.08	89.42	96.42
Ya(20,20)-50	74.58	89.65	97.24	*Ya(10,20)-50*	83.87	89.62	94.30
Ya(20,20)-60	88.76	90.92	95.11	*Ya(10,20)-60*	83.39	87.54	92.54
Ya(20,20)-70	82.01	89.73	94.85	*Ya(10,20)-70*	81.73	87.84	92.23
Ya(20,20)-80	85.32	88.26	91.74	*Ya(10,20)-80*	85.70	88.36	92.73
Ya(20,20)-90	84.77	87.99	92.20	*Ya(10,20)-90*	81.37	87.44	90.86
Ya(20,20)-100	81.82	86.96	94.89	*Ya(10,20)-100*	86.90	90.21	95.29
Ya(15,30)-30	81.19	86.90	89.66	*Ya(20,40)-30*	73.80	87.64	93.05
Ya(15,30)-40	82.18	88.47	94.19	*Ya(20,40)-40*	83.54	87.86	91.28
Ya(15,30)-50	83.37	89.11	94.44	*Ya(20,40)-50*	85.42	89.32	93.32
Ya(15,30)-60	86.12	90.70	96.93	*Ya(20,40)-60*	86.89	90.70	95.41
Ya(15,30)-70	83.66	88.09	92.62	*Ya(20,40)-70*	83.72	88.86	93.78
Ya(15,30)-80	84.40	89.30	94.32	*Ya(20,40)-80*	82.38	87.77	92.56
Ya(15,30)-90	82.05	87.83	93.24	*Ya(20,40)-90*	83.97	87.46	91.67
Ya(15,30)-100	86.77	89.99	94.00	*Ya(20,40)-100*	83.42	88.39	91.36
He1-20	0.00	0.00	0.00	*He2-40*	12.62	43.24	69.65
He1-30	0.00	10.25	37.66	*He2-50*	33.97	65.16	89.91
He1-40	34.03	47.03	74.83	*He2-60*	65.01	86.38	97.27
He1-50	60.64	79.40	97.73	*He2-70*	66.11	90.07	96.48
He1-60	85.61	93.01	98.38	*He2-80*	88.51	94.48	98.44
He1-70	91.41	94.61	97.38	*He2-90*	89.27	93.70	96.57
He1-80	90.91	96.23	98.40	*He2-100*	90.89	93.39	97.25
He1-90	84.85	94.72	98.40				
He1-100	90.09	94.73	97.95				
Mo(0.15)-30	0.00	0.00	0.00	*Mo(0.50)-30*	0.00	0.99	9.90
Mo(0.15)-40	0.00	0.00	0.00	*Mo(0.50)-40*	0.00	26.60	43.21
Mo(0.15)-50	0.00	0.00	0.00	*Mo(0.50)-50*	28.41	41.74	65.40
Mo(0.15)-60	0.00	0.00	0.00	*Mo(0.50)-60*	38.54	70.78	99.59
Mo(0.15)-70	0.00	0.00	0.00	*Mo(0.50)-70*	62.27	88.42	99.28
Mo(0.15)-80	0.00	3.89	19.05	*Mo(0.50)-80*	83.33	93.00	99.39
Mo(0.15)-90	0.00	7.93	41.83	*Mo(0.50)-90*	84.60	92.58	98.28
Mo(0.15)-100	0.00	6.59	28.83	*Mo(0.50)-100*	92.67	95.90	98.46
Mo(0.85)-30	17.14	42.32	75.26	*La-30*	44.83	56.55	65.52
Mo(0.85)-40	65.67	85.60	97.65	*La-40*	53.85	61.54	69.23
Mo(0.85)-50	84.14	91.07	93.81	*La-50*	59.18	63.67	67.35
Mo(0.85)-60	83.88	92.30	97.59	*La-60*	52.54	61.69	66.10
Mo(0.85)-70	85.95	91.92	97.95	*La-70*	56.52	63.77	68.12
Mo(0.85)-80	86.75	93.13	97.22	*La-80*	59.49	62.53	69.62
Mo(0.85)-90	88.99	92.37	95.41	*La-90*	61.80	65.13	68.54
Mo(0.85)-100	89.17	93.17	96.79	*La-100*	63.64	68.89	71.72

We calculated the number S_1 of successful runs, i.e. when the tabu search found the optimal solution, and the number S_2 of satisfactory runs, when the relative error of the obtained solution is not greater than 5%. The percent of the successful and satisfactory runs is denoted by P. We also measured the average computation times τ for each n. As we can see in Table 13, our algorithm outperforms the simulated annealing for the tested instances. It found an optimal solution in each run (with only 1 exception) even if the

Table 13 A comparison with the simulated annealing algorithm proposed in [26].

	Tabu Search 40 iter.				Tabu Search 1000 iter.				Simulated Annealing			
	S_1	S_2	τ	P	S_1	S_2	τ	P	S_1	S_2	τ	P
Ya(20,40)-5	100	0	0.002	100%	100	0	0.01	100%	82	10	0.030	92%
Ya(20,40)-10	99	1	0.005	100%	100	0	0.06	100%	10	48	0.065	58%
Ya(20,40)-15	100	0	0.012	100%	100	0	0.22	100%	12	66	0.220	78%
Ya(20,40)-20	100	0	0.020	100%	100	0	0.42	100%	8	56	0.410	64%

number of iterations was 40. Our algorithm also returns good solutions for other classes of graphs, not discussed in [26].

6 Conclusions

In this paper, we have discussed the minmax regret version of the minimum spanning tree problem, in which the uncertain edge costs are modeled by closed intervals. We have defined a natural neighborhood function and investigated its theoretical properties. We have proposed an algorithm based on the idea of tabu search, which is an alternative method for the exact algorithms proposed in the existing literature. These exact algorithms seem to be inefficient for large graphs, having more than 40 nodes. The proposed tabu search algorithm is fast and the computational tests presented suggest that it outputs the solutions of good quality even for large graphs. We conjecture that the performance of the tabu search will be quite good for most of the problems arising in practice. Finding a hard class of graphs for the tabu search algorithm is an interesting subject of further research. We also concluded that for the classes of graphs discussed previously in the literature the iterative improvement algorithms also return good solutions. For many instances even the single-run iterative improvement returns an optimal or nearly optimal solution.

References

1. Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows, theory, algorithms and applications. Prentice Hall, New Jersey
2. Ahuja RK, Ergun O, Orlin J, Punnen AP (2002) A survey of large-scale neighborhood search techniques. Discrete Applied Mathematics 123: 75–102
3. Amoia A, Cottafava G (1971) Invariance properties of central trees. IEEE Trans. Circuit Theory, CT-18: 465–467
4. Aron I, van Hentenryck P (2002) A constraint satisfaction approach to the robust spanning tree with interval data. Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence. Edmonton, Canada
5. Aron I, van Hentenryck P (2004) On the complexity of the robust spanning tree problem with interval data. Operations Research Letters 32: 36–40
6. Averbakh I, Lebedev V (2004) Interval data minmax regret network optimization problems. Discrete Applied Mathematics 138: 289–301
7. Bezrukov S, Kaderali F, Poguntke W (1996) On central spanning trees of a graph. Lectures Notes in Computer Science 1120: 53–58
8. Chazelle B (2000) A minimum spanning tree algorithm with inverse-Ackermann type complexity, J. ACM, 47: 1028-1047

9. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. Second edition. The MIT Press and McGraw-Hill Book Company
10. Deo D (1966) A central tree. IEEE Trans. Circuit Th. CT-13: 439–440
11. Glover F (1989) Tabu Search Part I. ORSA Journal on Computing 1(3): 190–206
12. Glover F (1990) Tabu Search Part II. ORSA Journal on Computing 2(1): 4–32
13. Glover F, Laguna M (1997) Tabu Search. Kluwer, Norwell, MA
14. Bang-Jensen J, Nikulin J. (2010) Heuristics for the central tree problem. Journal of Heuristics 16: 633–651
15. Kasperski A, Kobylański P, Kulej M, Zieliński P (2005) 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, 193–208
16. Kasperski A, Zieliński P (2007) On combinatorial optimization problems on matroids with ill-known weights. European Journal of Operational Research 177: 851–864
17. Kasperski A, Zieliński P (2006) An approximation algorithm for interval data minmax regret combinatorial optimization problems. Information Processing Letters 97(5): 177–180
18. Kouvelis P, Yu G (1997) Robust discrete optimization and its applications. Kluwer Academic Publishers, Boston
19. Kruskal JB (1956) On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Amer. Math. Soc. 7: 48–50
20. Lawler EL (1976) Combinatorial optimization: networks and matroids. Holt, Rinehart and Winston, New York
21. Michiels W, Aarts E, Korts J (2007) Theoretical aspects of local search. Springer, Berlin
22. Montemanni R, Gambardella LM, Donati AV (2004) A branch and bound algorithm for the robust shortest path problem with interval data. Operations Research Letters, 32(3): 225–232.
23. Montemanni R, Gambardella LM (2004) A branch and bound algorithm for the robust spanning tree problem with interval data. Operations Research Letters 161: 771–779
24. Montemanni R (2006). A Benders decomposition approach for the robust spanning tree problem with interval data. European Journal of Operational Research 174(3): 1479–1490
25. Montemanni R, Barta J, Gambardella L (2007) The robust traveling salesman problem with interval data. Transportation Science 41: 366–381
26. Nikulin Y (2008) Simulated annealing algorithm for the robust spanning tree problem. Journal of Heuristics 14: 391–402
27. Oxley JG (1992) Matroid Theory. Oxford University Press, New York
28. Papadimitriou C, Steiglitz K (1998) Combinatorial optimization, algorithms and complexity. Dover Publications Inc., Mineola, New York
29. Prim RC (1957). Shortest connection networks and some generalizations. Bell System Technical Journal 36: 1389–1401
30. Ribeiro C, Uchoa E, Werneck R (2002) A hybrid GRASP with perturbations for the Steiner problem in graphs. Informs Journal on Computing. 14: 228–246
31. Yaman H, Karasan OE, Pinar MC (2001) The robust spanning tree with interval data. Operations Research Letters 29: 31–40