A branch and bound algorithm for the robust spanning tree problem with interval data

R. Montemanni*, L.M. Gambardella

Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)
Galleria 2, CH-6928 Manno-Lugano, Switzerland

Abstract

The robust spanning tree problem is a variation, motivated by telecommunications applications, of the classic minimum spanning tree problem. In the robust spanning tree problem edge costs lie in an interval instead of having a fixed value.

Interval numbers model uncertainty about the exact cost values. A robust spanning tree is a spanning tree whose total cost minimizes the maximum deviation from the optimal spanning tree over all realizations of the edge costs. This robustness concept is formalized in mathematical terms and is used to drive optimization.

In this paper a branch and bound algorithm for the robust spanning tree problem is proposed. The method embeds the extension of some results previously presented in the literature and some new elements, such as a new lower bound and some new reduction rules, all based on the exploitation of some peculiarities of the branching strategy adopted.

Computational results obtained by the algorithm are presented. The technique we propose is up to 210 faster than methods recently appeared in the literature.

Keywords: Branch and bound, robust optimization, interval data, spanning tree problem.

1 Introduction

This paper presents a branch and bound algorithm for a robust version of the minimum spanning tree problem where edge costs lie in an interval instead of having a fixed value. Each interval is used to model uncertainty about the real value of the respective cost, which can take any value in the interval, independently from the costs associated with the other edges of the graph.

Adopting the model described above, the classic optimality criterion of the minimum spanning tree problem (where a fixed cost is associated with each edge of the graph) does not apply anymore, and the classic polynomial-time algorithms (Kruskal [9] and Prim [11]) cannot be used. A more complex optimization criterion has then to be adopted. We have chosen the *relative robustness criterion* (see Kouvelis and Yu [7]).

The study has practical motivations, and in particular there are some applications in the field of telecommunications. Consider a supervisor node in a data network where transmission lines are subject to uncertain delays, that wants to send a control message to all other nodes in the network. The supervisor node generally wants to broadcast the message over a robust spanning tree, in order to have a relatively quick broadcast whatever the situation in the network is (see Bertsekas and Gallagher [3] for a more detailed

*Corresponding author. Tel. +41 91 610 8568. Fax: +41 91 610 8661. Email: roberto@idsia.ch

description of the problem). A second application concerns the design of communication networks where routing delays on edges are uncertain, since they depend on the network traffic. The ideal network guarantees good performance whatever is the real traffic, i.e. a robust spanning tree is desirable (see Kouvelis and Yu [7] for more details).

In the literature there are some other studies related to robust versions of the minimum spanning tree problem. Kozina and Perepelista [8] defined an order relation on the set of feasible solutions and generated a Pareto set. Aron and Van Hentenryck [2] proved that the problem is $\mathcal{NP}$-hard. In Yaman et al. [12] a mixed integer programming formulation and a preprocessing technique are presented.

The present paper describes a branch and bound algorithm, based on an adaptation of the method developed in Montemanni et al. [10] for the *robust shortest path problem*. The algorithm incorporates an extension of the preprocessing rules described in [12] and some new concepts which significantly contribute to the efficiency of the method.

Another branch and bound approach to the robust spanning tree problem has been independently developed by Aron and Van Hentenryck (see [1]). From an algorithmic point of view, the method they propose has weaker preprocessing and reduction rules, a less efficient branching strategy and shares the same lower bound.

In Section 2 the robust spanning tree problem with interval data is formally described. Section 3 is devoted to the presentation of the new branch and bound algorithm and its components. Section 4 is dedicated to computational results, while conclusions are presented in Section 5.

2 Problem description

The robust spanning tree problem with interval data is defined on a graph $G = \{V, E\}$, where V is a set of vertices and E is a set of edges. An interval $[l_{ij}, u_{ij}]$, with $0 \leq l_{ij} < u_{ij}$, is associated with each edge $\{i, j\} \in E$. The problem is formally described through the following definitions:

Definition 1. *A* scenario *s is a realization of edge costs, i.e. a cost $c^s_{ij} \in [l_{ij}, u_{ij}]$ is fixed $\forall \{i, j\} \in E$.*

Definition 2. *The* robust deviation *for a spanning tree t in a scenario s is the difference between the cost of t in s and the cost of the minimum spanning tree in s.*

Definition 3. *A spanning tree t is said to be a* relative robust spanning tree *if it has the smallest (among all spanning trees) maximum (among all possible scenarios) robust deviation.*

A scenario can be seen as a snapshot of the network situation, while a relative robust spanning tree (*robust spanning tree* for short) is a tree which minimizes the maximum deviation from the optimal spanning tree over all realizations of the edge costs.

The following result (see [12]) is used by the method we propose:

Given a spanning tree t, a scenario $s(t)$ which makes the robust deviation maximum for t is the one where $c^{s(t)}_{ij} = u_{ij}$ $\forall \{i, j\} \in t$ and $c^{s(t)}_{kh} = l_{kh}$ $\forall \{k, h\} \in E \backslash t$.

In the remainder of this paper we will refer to the scenario $s(t)$ as the scenario *induced* by tree t. We will also refer to the cost of t in $s(t)$ minus the cost in $s(t)$ of a minimum spanning tree of $s(t)$ as $RCost(t)$, the *robustness cost* of t.

A polynomial-time procedure for the evaluation of the robustness cost of a given spanning tree t arises. It simply works by subtracting the cost of the minimum spanning tree in scenario $s(t)$ (Prim [11]) from the cost, in the same scenario, of t.

3 The branch and bound algorithm BB-RST

BB-RST, a Branch and Bound algorithm for the Robust Spanning Tree problem with interval data, is presented in this paper. It is an adaptation of an algorithm recently presented in Montemanni et al. [10]. The elements of the algorithm are described in Sections 3.1 to 3.5. A pseudo-code for the method is presented in Section 3.6, together with a study of the computational complexity of the algorithm.

3.1 Structure of the search-tree node

From now on, we will refer to the search-tree subtree rooted in node d as $SubT(d)$. Each node d of the search-tree is then identified by the following elements:

- $in(d)$: list of edges. The edges contained in $in(d)$ must appear in all of the spanning trees associated with the search-tree nodes of $SubT(d)$;
- $out(d)$: list of edges. The edges contained in $out(d)$ are forbidden for all of the spanning trees associated with the search-tree nodes of $SubT(d)$;
- $t(d)$: spanning tree associated with the search-tree node d. $t(d)$ is a spanning tree with minimum cost in *scenario* $s(u)$ (i.e. the scenario where $c_{ij}^{s(u)} = u_{ij}\ \forall\{i,j\} \in E$) among those which respect the limitations imposed by edge sets $in(d)$ and $out(d)$, i.e. $t(d)$ must contain the edges in $in(d)$ and cannot include the edges in $out(d)$;
- $lb(d)$: lower bound for the robustness cost of the spanning trees associated with the search-tree nodes in $SubT(d)$. The calculation of the lower bound is described in Section 3.4.

3.2 Preprocessing

The rules described in the following sections are to be applied before the branch and bound procedure starts, in order to simplify the problem.

3.2.1 Rule *P1*

Yaman et al. [12] proved that a robust spanning tree contains only edges which lie on some minimum spanning tree for some scenario (*weak edges*). They consequently do not consider non-weak edges when looking for a robust spanning tree. A stronger result is given in this paper.

Theorem 1. *A non-weak edge e can be deleted from graph G when solving a robust spanning tree problem.*

Proof. From [12] we know that e will not be in any robust spanning tree t of G. The minimum spanning tree in scenario $s(t)$ is a weak tree by definition, and consequently it cannot contain edge e.

We can conclude that e can be deleted from E because it will be neither in any robust tree nor in the minimum spanning trees of the scenarios induced by them. □

Theorem 1 is stronger because it states that it is possible to delete a non-weak edge instead of simply not considering it when looking for a robust spanning tree.

3.2.2 Rule *P2*

Observation 1. *An edge which lies on some minimum spanning tree for each scenario* (strong edge)*, can be inserted into the edge set* $in(r)$*, where* r *is the root of the search-tree.*

Observation 1 follows directly from [12] and from the branching rule adopted by BB-RST (see Section 3.3).

3.3 Branching strategy

The root r of the search-tree is the node with $out(r) = \emptyset$, $in(r) = \{e \mid e \text{ is a strong edge}\}$ and $lb(r) = 0$. Initially r will be the only node contained in S, the set of the nodes to be examined. During the execution of BB-RST, variable $ubTree$ will contain the best spanning tree (in terms of robustness) found so far.

At each iteration of BB-RST, the not yet examined node d with the smallest value of $lb(d)$ is selected and, if d is not a leaf of the search-tree (i.e. $in(d) \neq t(d)$), an edge $e \in t(d) \backslash in(d)$ is selected (as described in Section 3.3.1) in order to create two new search-tree nodes. The first new search-tree node, d', has $in(d') = in(d)$ and $out(d') = out(d) \cup \{e\}$. The second one, d'', has $in(d'') = in(d) \cup \{e\}$ and $out(d'') = out(d)$. The reduction rules described in Section 3.5 are applied to d'' in order to enlarge the arc set $out(d'')$. The values of $lb(d')$ and $lb(d'')$ are calculated as described in Section 3.4. If $lb(d') \geq RCost(ubTree)$ then node d' is cut, if $lb(d'') \geq RCost(ubTree)$ then node d'' is cut. It is easy to see that $t(d'') = t(d)$, while the computation of $t(d')$ is described in Section 3.3.1. In case $RCost(t(d')) < RCost(ubTree)$, $ubTree$ is updated and every $p \in S$ such that $RCost(t(p)) \geq RCost(ubTree)$ is extracted from S.

3.3.1 Selection of edge e and calculation of $t(d')$

The method we propose for the selection of edge e automatically produces $t(d')$. This is a key factor in the performance obtained by algorithm BB-RST. The strategy we use is based on the following theorem, which is a generalization of a result presented in Gabow [6].

Theorem 2. *Let* $H = \{V_H, E_H\}$ *be a given interval graph and let* s *be a scenario. Let* t *be the spanning tree with minimum cost in scenario* s *among those which include the edges of a given edge set* B*. Let* e *be an edge in* $t \backslash B$*. If we define* $F = \{g \in E_H \backslash \{e\} \mid t \backslash \{e\} \cup \{g\}$ *is a spanning tree*$\}$ *and* $f = argmin_{g \in F} \{c_g^s\}$*, then, if* $F \neq \emptyset$*,* $t \backslash \{e\} \cup \{f\}$ *is a spanning tree of graph* $K = \{V_H, E_H \backslash \{e\}\}$*, and it has the minimum cost in scenario* s *among the spanning trees of* K *which include the edges in the edge set* B*.*

Proof. Let $t' = t \backslash \{e\} \cup \{f\}$. Suppose t' does not have minimum cost in scenario s among trees on H which include the edges in B. Then there exists $g \in t' \backslash B$ and $h \in E_H$ such that $t'' = t' \backslash \{g\} \cup \{h\}$ is a spanning tree of H, t'' includes the edges of B and t'' has a smaller cost than t' in scenario s, i.e. $c_h^s < c_g^s$. We derive a contradiction below.

Let $t \backslash \{e\}$ consist of the two trees t_a and t_b, i.e. edge e joins t_a and t_b. Edge f also joins t_a and t_b by definition.

Edge h also joins t_a and t_b. For if not, assume without loss of generality that h joins two vertices of t_a. This implies (definition of h) that also g is in t_a. As $c_h^s < c_g^s$, the tree $t \backslash \{g\} \cup \{h\}$ would be a spanning tree and would cost less than t in scenario s. This contradiction proves h joins t_a and t_b.

Now we show edge $g \in t_a \cup t_b$. Since both e and h join t_a and t_b, and because of the definition of f, we have:

$$c_f^s \leq c_h^s < c_g^s \tag{1}$$

So $g \neq f$ and $g \in t' \backslash \{f\} = t_a \cup t_b$.

Assume now, without loss of generality, that $g \in t_a$. Now let $t_a \backslash \{g\}$ consist of two new trees, t_c and t_d. Edge e is incident to one of these trees, say t_c. Since $t\backslash\{e\}\backslash\{g\} \cup \{f\} \cup \{h\}$ is a spanning tree, either f or h is incident to t_c. So, either $t\backslash\{g\} \cup \{f\}$ or $t\backslash\{g\} \cup \{h\}$ is a spanning tree. But equation (1) implies that both of these spanning trees have a cost less than t in scenario s, a contradiction to the hypothesis. Thus the original assumption is false and t' has minimum cost in scenario s among trees that include the edges in B. □

Using Theorem 2, it is possible to derive the following result.

Corollary 1. *Given a search-tree node d and selected an edge $e \in t(d)\backslash in(d)$, then the spanning tree in the node d' is $t(d') = t(d)\backslash\{e\} \cup \{f\}$, where $f = argmin_{h \in F_e} \left\{ c_h^{s(u)} \right\}$ and $F_e = \{h \in E\backslash out(d)\backslash\{e\} \mid t\backslash\{e\} \cup \{h\}$ is a spanning tree$\}$.*

Proof. We observe that the edges contained in $out(d)$ cannot be contained in any spanning tree associated with the search-tree nodes in $SubT(d)$. Then we can define, in the context of Theorem 2, $H = \{V, E\backslash out(d)\}$ and scenario s as scenario $s(u)$. Consequently $t = t(d)$. Now, given $e \in t(d)\backslash in(d)$, Theorem 2 states that $t' = t(d)\backslash\{e\} \cup \{f\}$, with f defined as in the hypothesis, is a spanning tree with minimum cost in scenario $s(u)$ (among those that include the edges in $in(d) = in(d')$) for the graph $K = \{V, E\backslash out(d')\}$. We can then conclude that $t(d') = t'$. □

The theoretical result of Corollary 1 is used to select edge e in an intelligent way.

$$e := \arg\max_{g \in t(d)\backslash in(d)} \min_{h \in A_g} \left\{ c_h^{s(u)} - c_g^{s(u)} \right\} \tag{2}$$

where $A_g = \{h \in E\backslash out(d)\backslash\{g\}$, such that $t\backslash\{g\} \cup \{h\}$ is a spanning tree of $G\}$. The edge entering in $t(d')$ is then $f = \text{argmin}_{h \in A_e} \left\{ c_h^{s(u)} \right\}$. The cost in scenario $s(u)$ of $t(d')$ is greater than or equal to the cost of $t(d)$ in the same scenario because $c_h^{s(u)} - c_e^{s(u)} \geq 0$ by definition.

The evaluation of (2) is very quick, because checking whether the substitution of an edge in a spanning tree produces another spanning tree is a very easy task from a computational point of view. Selection rule (2) guarantees that $t(d')$ has the highest possible cost in scenario $s(u)$. This encourages the pruning of $SubT(d')$.

3.4 Lower bound

The lower bound we present is based on the same idea as that described in Montemanni et al. [10].

To better describe the lower bound, we need to give the following definitions. Scenario $s(B)$ is the scenario where $c_{ij}^{s(B)} = u_{ij}\ \forall\{i,j\} \in B$ and $c_{ij}^{s(B)} = l_{ij}\ \forall\{i,j\} \in E\backslash B$. $CostST(s, in, out)$ is the cost of t, the spanning tree in scenario s with the minimum cost such that t includes all the edges in the edge set in and t does not contain the edges in the edge set out.

Given a search-tree node d, we can then define:

$$lb(d) := CostST(s(E), in(d), out(d)) - CostST(s(E\backslash out(d)), \emptyset, \emptyset) \tag{3}$$

Practically $lb(d)$ is the cost in scenario $s(u)$ of $t(d)$ minus the cost of the minimum cost spanning tree of the scenario where the edges in $out(d)$ are at their lower bounds and the other edges are at their upper bounds. A theoretical justification for the definition given in (3) is guaranteed by the following result:

Theorem 3. $CostST(s(E), in(d), out(d)) - CostST(s(E\backslash out(d)), \emptyset, \emptyset) \leq RCost(t(p))\ \ \forall p \in SubT(d)$

Proof. By definition of function *RCost* we have:

$$RCost(t(p)) = CostST(s(E), in(p), out(p)) - CostST(s(t(p)), \emptyset, \emptyset) \tag{4}$$

As $in(d) \subseteq in(p)\ \forall p \in SubT(d)$ and $out(d) \subseteq out(p)\ \forall p \in SubT(d)$ by definition of the branching rule (see Section 3.3), we have:

$$CostST(s(E), in(d), out(d)) \leq CostST(s(E), in(p), out(p)) \quad \forall p \in SubT(d) \tag{5}$$

We also know that $t(p) \subseteq E \backslash out(p)\ \forall p \in SubT(d)$. If we consider this together with the definition of function $CostST$, we have:

$$CostST(s(E \backslash out(d)), \emptyset, \emptyset) \geq CostST(s(t(p)), \emptyset, \emptyset) \quad \forall p \in SubT(d) \tag{6}$$

We can conclude:

$$CostST(s(E), in(d), out(d)) - CostST(s(E \backslash out(d)), \emptyset, \emptyset) \leq RCost(t(p)) \quad \forall p \in SubT(d) \tag{7}$$

□

3.5 Reduction rules

In this section some reduction rules, used when a new search-tree node d is created, are described. The rules increase the dimension of $out(d)$. This is a crucial factor for algorithm BB-RST, because the lower bound $lb(d)$ is tighter when $out(d)$ is larger (see Section 3.4).

3.5.1 Rule *R1*

Observation 2. *Given a node d of the search-tree, if* $\{i, j\}, \{j, k\} \in in(d)$ *then* $\{i, k\}$ *can be inserted into* $out(d)$*.*

Observation 2 holds because $t(p)$ is a tree, and it cannot contain cycles. $\{i, j\}, \{j, k\} \in t(p)\ \ \forall p \in SubT(d)$ by definition of $in(d)$. Consequently $\forall p \in SubT(d)\ \ \{i, k\}$ cannot be in $t(p)$. Then $\{i, k\}$ can be inserted into $out(d)$. The rule is an adaptation of that described in Montemanni et al. [10].

3.5.2 Rule *R2*

Reduction rule $R2$ is based on a generalization of the preprocessing rule $P1$, presented in Section 3.2.1. To describe it, we define an $in(d)$*-spanning tree* as a spanning tree which includes all the edges of $in(d)$, and an $in(d)$*-weak edge* as an edge which lies on some $in(d)$-spanning tree which has minimum cost in some scenario.

Theorem 4. *Give a node d of the search-tree, if edge e is a non-*$in(d)$*-weak edge, then e can be inserted into* $out(d)$*.*

Proof. Each robust spanning tree is a *weak tree*, i.e. a minimum spanning tree for some scenario (Yaman et al. [12]). Each weak tree that includes the edges of $in(d)$ (if it exists) must be a $in(d)$*-weak tree*, i.e. must have, for some scenario, minimum cost among spanning trees which include the edges contained in $in(d)$. Then only $in(d)$-weak trees could be robust spanning trees. Consequently all non-$in(d)$-weak edges can be inserted into $out(d)$, because they will not appear in any robust spanning tree. □

The following characterization for $in(d)$-weak edges can then be given.

Theorem 5. *Given a search-tree node d, edge e is a $in(d)$-weak edge if and only if there exists a $in(d)$-spanning tree of minimum cost which uses edge e when the cost of edge e is at its lower bound and the costs of the remaining edges are at their upper bounds.*

Proof. If there exists a $in(d)$-spanning tree that uses edge e and has minimum cost (among $in(d)$-spanning trees) in the scenario defined in the hypothesis, then edge e is a $in(d)$-weak edge by definition.

We prove the converse by showing that if there does not exist a minimum $in(d)$-spanning tree that uses edge e for the stated scenario, then e cannot be a $in(d)$-weak edge.

Consider Kruskal's algorithm [9] which sorts all edges in non-decreasing order of their costs, and define $\mathcal{L}$, the set of edges chosen to form a minimum spanning tree. Kruskal's algorithm proceeds as follows. Initially, the set $\mathcal{L}$ is empty. The algorithm examines each edge in the sorted order in turn and checks whether adding the current edge to set $\mathcal{L}$ creates a cycle with the edges already in $\mathcal{L}$. If it does not, the current edge is added to $\mathcal{L}$, otherwise it is discarded. The algorithm stops when there are $n-1$ edges in $\mathcal{L}$. In case of ties in the sorted order, an edge may be chosen arbitrarily from amongst those with least cost.

We modify Kruskal's algorithm slightly. First we initialize $\mathcal{L} = in(d)$, forcing the edges of $in(d)$ to be included in the spanning tree returned by the algorithm. We also ask that in case of ties, the algorithm favors edge e over other edges to add to $\mathcal{L}$. With this last modification, it can be shown that if e is not on the spanning tree found by the algorithm for a particular scenario, then it is not on any minimum $in(d)$-spanning tree for that scenario.

Let s be the scenario defined in the hypothesis, i.e. with cost on edge e at its lower bound and all other costs at their upper bounds. We now show that if e is not on any minimum $in(d)$-spanning tree for scenario s, then it cannot be on any minimum $in(d)$-spanning tree under any scenario, and so it must be that edge e is not a $in(d)$-weak edge.

Let $\mathcal{L}^{s'}$ denote the spanning tree returned by the modified algorithm applied to the graph under scenario s'. If e is not on any minimum $in(d)$-spanning tree for scenario s, then it is not on the spanning tree found by the algorithm for scenario s, so either $|\mathcal{L}^s|$ reaches $n-1$ before e is encountered in the sorted order, or adding e to $\mathcal{L}^s$ at the point it was encountered in the sorted order would introduce a cycle. Let C denote the edges in such a cycle. Now suppose there is a scenario s' such that $e \in \mathcal{L}^{s'}$. Let D denote the set of edges $e' \in C \backslash \{e\}$ such that $e' \notin \mathcal{L}^{s'}$. Clearly $D \neq \emptyset$ and $C \backslash D \subseteq \mathcal{L}^s$. For each edge $e' \in D$, since it was not added to $\mathcal{L}^{s'}$, it must be that e' forms a cycle with edges already in $\mathcal{L}^{s'}$: let $C_{e'}$ denote the edges in such a cycle. Now it is not hard to see that $(C \backslash D) \cup \left(\bigcup_{e' \in D} C_{e'} \backslash \{e'\}\right)$ induces a cycle with all edges in the set $\mathcal{L}^{s'}$. This is a contradiction, since $\mathcal{L}^{s'}$ forms a tree, so e cannot lie on a minimum $in(d)$-spanning tree for scenario s' found by the algorithm, for any scenario s'.

Furthermore, by our modification of Kruskal's algorithm, we have that e cannot lie on any minimum $in(d)$-spanning tree under any scenario s', and hence e cannot be a $in(d)$-weak edge.

□

The proof of Theorem 5 suggests a procedure with computational complexity $O(|E| \log |E|)$ (modification of Kruskal's algorithm [9]) to check whether an edge e is $in(d)$-weak or not. This procedure is analogous to that used to check whether an edge is weak (strong) or not (see [12]).

3.6 Pseudo-code and computational complexity

The branch and bound algorithm BB-RST is summarized in Figure 1, where a pseudo-code is presented.

01: Apply rules $P1$ and $P2$ and initialize r; // r is the root of the search-tree
02: $S := \{r\}$; // S is the set of search-tree nodes to be visited
03: *ubTree* $:= t(r)$; // *ubTree* is the best tree retrieved so far
04: While $(S \neq \emptyset)$
05: Select $d \in S$ such that $lb(d) \leq lb(p)\ \forall p \in S$ and extract d from S;
06: If(d is not a search-tree leaf)
07: Select $e \in t(d) \backslash in(d)$ and create d' and d'' as described in Section 3.3;
08: Apply rules $R1$ and $R2$ to d'';
09: Calculate $lb(d')$ and $lb(d'')$;
10: Eventually add d' and d'' to S and update *ubTree*;
11: Return *ubTree*;

Figure 1: A pseudo-code for the branch and bound algorithm BB-RST.

Table 1: Definition of the problems.

	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
L^k	$[0, 10)$	$[0, 15)$	$[0, 20)$	$[0, 10)$	$[0, 15)$	$[0, 20)$
U^k	$(l_e, 10]$	$(l_e, 15]$	$(l_e, 20]$	$(l_e, 20]$	$(l_e, 30]$	$(l_e, 40]$

The pseudo-code also helps to estimate the computational complexity of algorithm BB-RST.

Theorem 6. *The computational complexity of algorithm BB-RST is* $O(|V|^{|V|-1}|E|^2 \log |E|)$.

Proof. Given a search-tree node d, the spanning tree $t(d)$ cannot be associated with the nodes in $SubT(d')$ because one of the edges of $t(d)$ is in $out(d')$. $t(d)$ can be associated with at most $|V|-1$ search-tree nodes of a chain of d''-type nodes, with the corresponding in sets incrementally increasing from $\emptyset$ to $t(d)$. Since there are no more than $|V|^{|V|-2}$ spanning trees for a graph G (Cayley [4]), the search-tree nodes examined in lines $05 - 10$ of the pseudo-code are no more than $(|V| - 1)|V|^{|V|-2}$. The most expensive operation carried out for each search-tree node is the one at line 08, and has a computational complexity of $O(|E|^2 \log |E|)$ (Yaman et al. [12]). The computational complexity of algorithm BB-RST is consequently $O(|V|^{|V|-1}|E|^2 \log |E|)$ □

It is important to point out that notwithstanding this very high computational complexity, algorithm BB-RST is very efficient in practice (see Section 4).

4 Computational results

We test our method on the same families of problems used in Yaman et al. [12] and on some new problems we have generated.

In [12] complete graphs with $|V| = 10$, 15, 20 and 25 are considered. For problems with $|V| = 10$, 15 and 20 six sets of five problems each are generated, accordingly to the following definition. Each set k is defined by two intervals L^k and U^k, summarized in Table 1. For each problem of set k, $\forall e \in E$, l_e is randomly selected from L^k and u_e from U^k. For the case $|V| = 25$ only five test problems from set 1 are considered, because of the long computation times.

We compare our results with those obtained by solving the mixed integer program described in [12], preprocessed as described in the same paper and with those of the branch

Table 2: Computational results 1.

$\|V\|$	Yaman et al. [12]	Yaman et al. new	Aron and Van Hentenryck [1]	BB-RST
10	1.96	0.85	0.28	0.04
15	33.09	286.47	4.60	1.84
20	693.88	>2016.28	69.95	56.46
25	2027.60	>3600.00	610.45	484.24

and bound algorithm presented in Aron and Van Hentenryck [1], where a different implementation of the procedure to retrieve weak (and B-weak) edges is also proposed.

We randomly generated the families of benchmarks with the same specifications used in [12], and we repeated exactly the same tests presented there. The problems considered are not necessarily the same used in [12], because of the presence of a random factor in their creation. For the results presented in [12], a Pentium II PC 450 MHz and ILOG CPLEX[1] 6.5.1 were used, while for this paper a Pentium II PC 400 MHz and CPLEX 6.0 are used. The results presented in [1] are obtained on a Sun Sparc 440 MHz computer, which is about 2.5 times faster than the our machine (Dongarra [5]). This ratio will be used to compare the results with those of BB-RST. For each family of problems, in Table 2 we present the results reported in [12] (*Yaman et al. [12]*), the results obtained by the method described in [12] in our tests (*Yaman et al. new*), the results presented in [1] for the equivalent of our branch and bound algorithm (*Aron and Van Hentenryck [1]*) and the results obtained by algorithm BB-RST (*BB-RST*). The maximum computation time for each run has been fixed at 3600 seconds. After this time the computation is stopped, unless it has already terminated. This happened for some tests summarized in the last two rows of the third column of Table 2 (entries containing ">"). In these cases a contribute of 3600 seconds has been counted in order to obtain the averages reported in the table.

Table 2 shows that, as expected, using the new approach described in this paper strongly improves the results presented in [12]. BB-RST is much faster than the method suggested in [12], even when the results of BB-RST are compared with those reported in [12], which are obtained, as pointed out before, on a much faster computer. BB-RST is also more efficient than the equivalent branch and bound algorithm described in [1]. Since, as observed before, the two methods share the same lower bound, but our approach has branching strategy, preprocessing and reduction rules with a sounder theoretical justification, this result was expected.

A second set of benchmarks is considered. All the problems are complete graphs and have 20 vertices randomly placed on a 50×50 grid. $\forall\{i,j\} \in E$, l_{ij} is randomly selected in $[ed_{ij}(1-p), ed_{ij})$ and u_{ij} in $(l_{ij}, ed_{ij}(1+p)]$, where ed_{ij} is the euclidian distance between i and j, and p is a distortion parameter. We consider three different values for p: 0.15, 0.50 and 0.85. For each of these values of p, five problems have been generated and solved with the technique proposed in [12] (*Yaman et al.*) and with the algorithm described in this paper (*BB-RST*). The average of the results obtained, this time on a SUNW Ultra-30 computer (with CPLEX 6.0), are presented in Table 3. The gap between the computation times of the two methods is much wider than in Table 2. This happens because BB-RST takes advantage of the euclidian structure of the problems. Both the algorithms work better when p is high because spanning trees have, in this case, more scattered robustness costs.

[1] www.cplex.com.

Table 3: Computational results 2.

p	Yaman et al.	BB-RST
0.15	320.91	1.53
0.50	61.41	0.73
0.85	33.74	0.46

5 Conclusion

The robust spanning tree problem, which can be used to model in mathematical terms some telecommunications problems, has been studied in this paper.

A branch and bound algorithm for this problem has been described. It combines the extension of results previously presented in the literature with some new elements, such as a new lower bound which works by exploiting some properties connected with the ad-hoc branching rule we have developed. Some new reduction rules, again based on the exploitation of some peculiarities of the branching rule adopted, contribute to speed up the method.

Computational results prove that the algorithm we propose is very competitive. It strongly improves the results obtained by methods that have recently appeared in the literature.

Acknowledgements

This work was co-funded by the European Commission IST project "MOSCA: Decision Support System For Integrated Door-To-Door Delivery: Planning and Control in Logistic Chain", grant IST-2000-29557. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] I. Aron and P. Van Hentenryck. A constraints satisfaction approach to the robust spanning tree problem with interval data. In preparation. Computer Science Department, Brown University, May 2002.

[2] I. Aron and P. Van Hentenryck. On the complexity of the robust spanning tree with interval data. *Operations Research Letters*, to appear.

[3] D.P. Bertsekas and R. Gallagher. *Data Networks*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[4] A. Cayley. A theorem on trees. *Quarterly Journal of Pure and Applied Mathematics*, 23:376–378, 1889.

[5] J.J. Dongarra. Performance of various computers using standard linear algebra software in a fortran environment. Technical Report CS-89-85, University of Tennessee, July 2003.

[6] H.N. Gabow. Two algorithms for generating weighted spanning trees in order. *SIAM Journal on Computing*, 6(1):139–150, March 1977.

[7] P. Kouvelis and G. Yu. *Robust Discrete Optimization and its applications*. Kluwer Academic Publishers, 1997.

[8] G.L. Kozina and V.A. Perepelista. Interval spanning trees problem: solvability and computational complexity. *Interval Computations*, 1:42–50, 1994.

[9] J.B. Kruskal. On the shortest spanning subtree of a graph and the travelling salesman problem. *Preceedings of the American Mathematical Society*, 7:48–50, 1956.

[10] R. Montemanni, L.M. Gambardella, and A.V. Donati. A branch and bound algorithm for the robust shortest path problem with interval data. *Operations Research Letters*, to appear.

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

[12] H. Yaman, O.E. Karaşan, and M.Ç. Pinar. The robust spanning tree problem with interval data. *Operations Research Letters*, 29:31–40, 2001.