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
Galleria 2, CH-6928 Manno, 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 are interval numbers instead of fixed values.

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 it is used to drive optimization.

In this paper a branch and bound algorithm for the robust spanning tree problem is proposed. The method embeds 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 finally presented. They highlight the efficiency of the approach we propose.

Key words: Branch and bound, robust optimization, 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 are specified as interval numbers. 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.

* Corresponding author.
Email addresses: roberto@idsia.ch (R. Montemanni), luca@idsia.ch (L.M. Gambardella).

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-times solving algorithms (Kruskal [6] and Prim [9]) cannot be used.

A more complex optimization criterion has then to be chosen, and new approaches have to be developed. We adopt the *relative robustness criterion* (this criterion is discussed in Kouvelis and Yu [4], a book entirely devoted to robust discrete optimization), and we develop a new solving strategy.

A *relavite robust spanning tree* (*robust spanning tree* in the remainder of this paper, for short) is a spanning tree whose total cost minimizes the maximum deviation from the optimal spanning tree over all realizations of the edge costs.

The study has practical motivations, and in particular there is a main application 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 broadcasting whatever the situation in the network is (see Bertsekas and Gallagher [2] for a more detailed description of the problem).

In the literature it is possible to find other studies related to robust versions of the minimum spanning tree problem. Kozina and Perepelista [5] studied the minimum spanning tree problem with interval edge costs. They defined a relation order on the set of feasible solutions and generated a Pareto set.

In Yaman et al. [10] a mixed integer programming formulation and a preprocessing technique for the robust spanning tree problem are presented. This preprocessing technique identifies edges which must be in every robust spanning tree and edges which cannot lay in any of these trees, treating them consequently.

Kouvelis and Yu [4] conjectured that the robust spanning tree problem with interval data is $\mathcal{NP}$-hard, but no proof was presented.

In this paper a branch and bound algorithm for the robust spanning tree problem with interval data is presented. It is based on some results known from the literature, on a new lower bound and on some new reduction rules which work by exploiting some properties arising from the particular branching strategy adopted. The algorithm is inspired by the one for the *robust shortest path problem* described in Montemanni et al. [8].

It must be pointed out that another approach to the robust spanning tree problem, very similar to the one we propose, 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 and a less efficient branching strategy.

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

Accordingly to Yaman et al. [10], 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 described through the following definitions:

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

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

Definition 3 *A spanning tree t is said to be a* 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 robust spanning tree is a tree which should guarantee reasonably good performance under any possible arc costs configuration.

From the literature the following result is known:

Theorem 4 (Yaman et al. [10]) *Given a spanning tree t, a scenario r which makes the robust deviation maximum for t is the one where each edge $\{i, j\}$ on t has cost u_{ij} and each edge $\{k, h\}$ not on t has cost l_{kh}, i.e. $c_{ij}^r = u_{ij}$ $\forall \{i, j\} \in t$ and $c_{kh}^r = l_{kh}$ $\forall \{k, h\} \in E \backslash t$.*

PROOF. See Yaman et al. [10]. □

In the remainder of this paper we will refer to the scenario r derived from tree t as described in Theorem 4 as the scenario *induced* by tree t.

We will also refer to the cost of t minus the cost of a minimum spanning tree of the scenario induced by t as the *robustness cost* of t.

Theorem 4 is very important because it suggests a polynomial time procedure (see Prim [9]) for the evaluation of the robustness cost of a given spanning tree t. In practice it is enough to subtract the cost of the minimum spanning tree of the scenario induced by t from the cost, in the same scenario, of t.

Applying Theorem 4, Yaman et al. [10] modelled the robust spanning tree problem as a mixed integer programming formulation. In this formulation a *single-commodity* formulation and the dual of a *multi-commodity* formulation, both modelling the classic minimum spanning tree problem and both presented in Magnanti and Wolsey [7], are joined together.

To describe these formulations we first need to define the arc set $A = \{(i,j),(j,i)|\{i,j\} \in E\}$. It is the arc set associated with the edges set E.

In the multi-commodity formulation presented in [7] each $k \in V\backslash\{0\}$ defines a commodity: one unit of commodity k is originated at node 0 and has to be delivered to node k on digraph $D = \{V, A\}$. Letting f_{ij}^k be the flow of commodity k on arc (i,j), the multi-commodity formulation for the classic shortest path problem, where a fixed cost c_{ij} is associated to each edge $\{i,j\}$, is the following one:

$$
\begin{aligned}
&(F1)\ \text{Min} \sum_{\{i,j\}\in E} c_{ij}(y_{ij}+y_{ji}) && && (1)\\
&\text{s.t.} \sum_{(j,0)\in A} f_{j0}^k - \sum_{(0,j)\in A} f_{0j}^k = -1 && \forall k \in V\backslash\{0\} && (2)\\
&\sum_{(j,i)\in A} f_{ji}^k - \sum_{(i,j)\in A} f_{ij}^k = 0 && \forall i,k \in V\backslash\{0\}, i \neq k && (3)\\
&\sum_{(j,k)\in A} f_{jk}^k - \sum_{(k,j)\in A} f_{kj}^k = 1 && \forall k \in V\backslash\{0\} && (4)\\
&f_{ij}^k \leq y_{ij} && \forall (i,j) \in A, \forall k \in V\backslash\{0\} && (5)\\
&\sum_{\{i,j\}\in A} y_{ij} = |V|-1 && && (6)\\
&f_{ij}^k \geq 0 && \forall \{i,j\} \in A, \forall k \in V\backslash\{0\} && (7)\\
&y_{ij} \geq 0 && \forall \{i,j\} \in A && (8)
\end{aligned}
$$

If arc (i,j) is a member of the active directed spanning tree, then variable y_{ij} defines a capacity for the flow of each commodity on arc (i,j). It is interesting to notice that, as shown in [7], y variables have integer values in all of the extreme points of the polytope defined by $F1$.

A mathematical formulation for the robust spanning tree problem with interval data, as described in Yaman et al. [10], is the following one:

$$
\begin{aligned}
(MIP)\ \text{Min} \quad & \sum_{\{i,j\}\in E} u_{ij}x_{ij} - \sum_{k\in V\setminus\{0\}} (\alpha_k^k - \alpha_0^k) - (|V|-1)\mu && && (9)\\
\text{s.t.} \quad & \sum_{(0,j)\in A} f_{0j} - \sum_{(j,0)\in A} f_{j0} = (|V|-1) && && (10)\\
& \sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = -1 && \forall i \in V\setminus\{0\} && (11)\\
& f_{ij} \le (|V|-1)x_{ij} && \forall \{i,j\}\in E && (12)\\
& f_{ji} \le (|V|-1)x_{ij} && \forall \{i,j\}\in E && (13)\\
& \sum_{\{i,j\}\in E} x_{ij} = |V|-1 && && (14)\\
& \sigma_{ij}^k \ge \alpha_j^k - \alpha_i^k && \forall (i,j)\in A, \forall k \in V\setminus\{0\} && (15)\\
& \sum_{k\in V\setminus\{0\}} \sigma_{ij}^k + \mu \le l_{ij} + (u_{ij}-l_{ij})x_{ij} && \forall \{i,j\}\in E && (16)\\
& \sum_{k\in V\setminus\{0\}} \sigma_{ji}^k + \mu \le l_{ij} + (u_{ij}-l_{ij})x_{ij} && \forall \{i,j\}\in E && (17)\\
& f_{ij} \ge 0 && \forall (i,j)\in A && (18)\\
& x_{ij} \in \{0,1\} && \forall \{i,j\}\in E && (19)\\
& \sigma_{ij}^k \ge 0 && \forall (i,j)\in A, \forall k \in V\setminus\{0\} && (20)\\
& \alpha_h^k \ge 0 && \forall h\in V, \forall k \in V\setminus\{0\} && (21)\\
& \mu \text{ unrestricted} && && (22)
\end{aligned}
$$

The first sum in the objective function and constraints (10) to (14) represent the single-commodity formulation described in [7] for a classic minimum spanning tree problem where $c_{ij} = u_{ij}\ \forall\{i,j\} \in E$. In this context, variable x_{ij} indicates whether edge $\{i,j\}$ is in the optimal tree (i.e. carries any flow) or not, while variable f_{ij} denotes the total flow on arc $(i,j) \in A$.

The second sum and the last term of the objective function, together with constraints (15) to (17) represent the dual of formulation $F1$, with each c_{ij} defined as $l_{ij} + (u_{ij} - l_{ij})x_{ij}$.

In this context, constraints (16) and (17) realize an edge costs alignment between the two formulations for the classic minimum spanning tree problem joined together in MIP.

Constraints (18) to (22) finally define the domains of the variables.

In the context of the robust spanning tree problem, x variables contain the edges of the robust spanning tree, while the solution of the dual of formulation $F1$ contains the cost of the minimum spanning tree in the scenario induced by the tree described by x variables, accordingly to Theorem 4.

3 The branch and bound algorithm RST

RST, a branch and bound algorithm for the Robust Spanning Tree problem with interval data, which works by creating and visiting a tree-search, is presented in this section.

Section 3.1 is dedicated to the introduction of the notation used. The elements of the algorithm are described in Sections 3.2 to 3.7. Finally in Section 3.8 the branch and bound algorithm is summarized with the help of a pseudo-code.

3.1 Notation

The notation adopted in the remainder of Section 3 is the following one:

- scenario u: scenario such that $c_{ij}^{u} = u_{ij}\ \forall\{i,j\} \in E$;
- $T(d)$: tree-search nodes contained in the tree-search subtree rooted in node d;
- $RC(t)$: robustness cost of spanning tree t. Accordingly to Theorem 4, it is obtained by subtracting the cost of the minimum spanning tree in the scenario induced by t from the cost of t in scenario u;
- $MST(B, in, out)$: defined scenario s as the scenario where $c_{ij}^{s} = u_{ij}\ \forall\{i,j\} \in B$ and $c_{ij}^{s} = l_{ij}\ \forall\{i,j\} \in E \backslash B$, $MST(B, in, out)$ is the cost of t, the spanning tree in 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.

3.2 Structure of the tree-search node

Each node d of the tree-search constructed by the algorithm is 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 tree-search nodes of $T(d)$;
- $out(d)$: list of edges. The edges contained in $out(d)$ are forbidden for all of the spanning trees associated with the tree-search nodes of $T(d)$;
- $t(d)$: spanning tree associated with the tree-search node d. $t(d)$ is a spanning tree with minimum cost in scenario u 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 tree-search nodes in $T(d)$. The calculation of the lower bound is described in Section 3.6.

3.3 Preprocessing

The results presented in this section are used to preprocess the problem, i.e. to reduce its dimension.

The preprocessing rules described are mainly those originally proposed in Yaman et al. [10]. These rules are based on some theoretical results, presented in the following subsections.

3.3.1 Rule P1

The rule described in this section is based on a result presented in [10] which permits to identify edges which will never be in a robust tree. We extend this result in order to conclude that these edges not only will never be in such a tree, but even that they can be eliminated from graph G.

We first need the following definitions.

Definition 5 *A spanning tree is a* weak tree *if it is a minimum spanning tree for some scenarios.*

Definition 6 *Edge* e *is a* weak edge *if it lies on some weak tree.*

The following characterization for weak edges is given.

Theorem 7 (Yaman et al. [10]) *Edge* e *is a weak edge if and only if there exists a minimum spanning tree using 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. See Yaman et al. [10]. □

Theorem 7 implies a practical criterion to identify weak edges. This criterion is based on a modification of Kruskal's algorithm [6] for the classic minimum spanning tree problem. In practice it is enough to modify the algorithm described in [6] in such a way that it favors edge e to enter the spanning tree retrieved (see Yaman et al. [10] for more details).

To verify whether an edge is weak or not has consequently a computational complexity of $O(|E| \log |E|)$.

In Yaman et al. [10] the following result is also presented.

Theorem 8 (Yaman et al. [10]) *A robust spanning tree is a weak tree.*

PROOF. See Yaman et al. [10]. □

Theorem 8 is very important because it implies that a robust spanning tree contains weak edges only. This result is used in Yaman et al. [10] to fix $x_{ij} = 0$ in formulation MIP for each non-weak edge $\{i, j\}$.

A stronger result, which provides a more powerful preprocessing criterion, can be derived.

Proposition 9 (Rule *P1*) *A non-weak edge e can be deleted from graph G when solving a robust spanning tree problem.*

PROOF. From Theorem 8 we know that e will not be in any robust spanning tree t of G. We now consider the scenario r induced by t. The minimum spanning tree in scenario r 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 not be neither in any robust tree nor in the minimum spanning trees in the scenarios induced by them. □

Proposition 8 is stronger than Theorem 8 because it permits to delete a non-weak edge instead of simply inserting it into the edge set $out(r)$ associated with the root of the tree search (see Section 3.4).

3.3.2 Rule P2

The reduction rule described in this section permits to identify edges which will be in every robust spanning tree.

We first need the following definitions.

Definition 10 *Edge e is a* strong edge *if it lies on a minimum spanning tree for all realizations of edge costs.*

An interesting characterization of strong edges is provided by the following theorem.

Theorem 11 (Yaman et al. [10]) *Edge e is a strong edge if and only if there exists a minimum spanning tree using edge e when the cost of edge e is at its upper bound and the costs of the remaining edges are at their lower bounds.*

PROOF. See Yaman et al. [10]. $\square$

Theorem 11 implies a practical criterion to identify strong edges in polynomial time. This criterion is based on a modification of Kruskal's algorithm [6] for the classic minimum spanning tree problem similar to the one described in Section 3.3.1 for the identification of (non-)weak edges.

The following result is very important, because it connects strong edges to robust spanning tress.

Theorem 12 (Yaman et al. [10]) *There exists a robust spanning tree such that every strong edge in the graph lies on the tree.*

PROOF. See Yaman et al. [10]. $\square$

In Yaman et al. [10] the result of Theorem 12 is used to fix in formulation MIP $x_{ij} = 1$ for each strong edge $\{i, j\}$.

We use a very similar approach, i.e. we force each strong edge to be included in every spanning tree examined by the branch and bound algorithm. In Proposition 13 it will be formalized how we obtain this.

Proposition 13 (Rule *P2*) *A strong edge e can be inserted in the edge set $in(r)$, associated with the root of the tree-search (see Section 3.4).*

PROOF. Follows directly from Theorem 12 and from the branching rule adopted (see Section 3.5). $\square$

3.4 Root of the tree-search

The root r of the tree search constructed by RST is the node with $out(r) = \emptyset$, $in(r) = \{e \mid e \text{ is a strong edge}\}$ (accordingly to Section 3.3) and $lb(r)$ calculated as described in Section 3.6. Initially r will be the only node in the set of the nodes to be examined.

RST has a variable ub which contains the cost of the best spanning tree (in terms of robustness) encountered so far. In the beginning $ub = RC(t(r))$.

At each iteration of RST, the not yet examined node with the smallest value of $lb(d)$, d, is selected and, if d is not a leaf of the tree-search (i.e. $in(d) \neq t(d)$), an edge $e \in t(d) \backslash in(d)$ is selected as described in Section 3.5.1. Edge e is used to create two new tree-search nodes.

The first new tree-search node arising from the branching is d' and has $in(d') = in(d)$ and $out(d') = out(d) \cup \{e\}$. The second tree-search node, d'', has $in(d'') = in(d) \cup \{e\}$ and $out(d'') = out(d)$. The reduction rules described in Section 3.7 are applied to d'' in order to enlarge the arc set $out(d'')$.

The values of $lb(d')$ and $lb(d'')$, which depend on $out(d')$ and $out(d'')$ respectively, are calculated as described in Section 3.6. If $lb(d') \geq ub$ then node d' is cut because the spanning trees associated with $T(d')$ cannot be better than the best one found so far. Analogously, if $lb(d'') \geq ub$ then node d'' is cut.

It is easy to see that $t(d'') = t(d)$, while $t(d')$ has to be recomputed. This calculation is carried out as described in Section 3.5.1.

In case $RC(t(d')) < ub$, ub is updated and the not yet examined nodes with a lower bound greater then or equal to $RC(t(d'))$ are cut.

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

The selection of the branching edge e and the calculation of $t(d')$, the spanning tree associated with the new tree-search node d', are strongly correlated in the algorithm we propose.

In particular Proposition 15, which is a direct consequence of Theorem 14, will define a very quick way to calculate $t(d')$ once e is known. We will use this result to embed the calculation of $t(d')$ within the process of selection of edge e.

Theorem 14 is a generalization of a result presented in Gabow et al. [3].

Theorem 14 *Let $H = \{V_H, E_H\}$ be a given graph and let r be a scenario. Let t be the spanning tree with minimum cost in scenario r among those which include the edges of a given edge set B. Let e be an edge in t. If we define $F = \{g \in E_H \backslash \{e\} \mid t \backslash \{e\} \cup \{g\}$ is a spanning tree$\}$ and $f = argmin_{g \in F} \left\{c_g^r\right\}$, then $t \backslash \{e\} \cup \{f\}$ is a spanning tree of graph $K = \{V_H, E_H \backslash \{e\}\}$ and it has the minimum cost in scenario r among the spanning trees of K which include the edges on the edge set B.*

PROOF. The proof is a straightforward adaptation of the proof given in Gabow [3] for the original theorem.

See Section A.1 for a complete proof. □

Theorem 14 can be adapted to our context, and the following result, useful to derive $t(d')$ when $t(d)$ and e are known, can be derived:

Proposition 15 *Given a tree-search node d and selected $e \in t(d)\backslash in(d)$, $t(d') = t(d)\backslash\{e\} \cup \{f\}$, with f defined as in Theorem 14.*

PROOF. We observe that the edges contained in $out(d)$ cannot be contained in any spanning tree associated with the tree-search nodes in $T(d)$. Then we can define, in the context of Theorem 14, $H = \{V, E\backslash out(d)\}$ and scenario r as scenario u. Consequently $t = t(d)$. Now, given $e \in t(d)\backslash in(d)$, Theorem 14 states that $t' = t(d)\backslash\{e\}\cup\{f\}$, with f defined as in Theorem 14, is a minimum spanning tree in scenario u for the graph $K = \{V, E\backslash out(d')\}$. We can then conclude that $t(d') = t'$. □

The strategy we adopt to select edge e is strongly based on the result presented in Proposition 15. In practice we select the edge e that guarantees the highest possible cost of $t(d')$ in scenario u. We select e as follows:

$$e := \operatorname*{argmax}_{g \in t(d)\backslash in(d)} \min_{\substack{h \,\in\, E\backslash out(d)\backslash\{g\} \\ \text{s.t. } t\backslash\{g\}\cup\{h\} \\ \text{is a spanning tree of } G}} \{u_h - u_g\} \tag{23}$$

The criterion given by equation (23) has been chosen mainly because it permits to embed the calculation of $t(d')$ into the edge selection process, improving the algorithm performance. A second reason is that, as observed above, it produces the highest possible value for $t(d')$, and this should favor the cutting of node d'.

3.6 Lower bound

Given a tree-search node d, with the respective edge sets $in(d)$ and $out(d)$, the calculation of the lower bound $lb(d)$ for the robustness cost of the spanning trees associated with $T(d)$ is described in this section.

Some theoretical results justify the lower bound. They are described in the following paragraphs.

Lemma 16 $MST(E, in(d), out(d)) \leq MST(E, in(p), out(p)) \ \ \forall p \in T(d)$

PROOF. Direct consequences of the branching rule described in Section 3.7 are:

$$in(d) \subseteq in(p) \ \ \forall p \in T(d) \tag{24}$$

$$out(d) \subseteq out(p) \ \ \forall p \in T(d) \tag{25}$$

(24) and (25) show that there are less freedom degrees in the calculation of $MST(E, in(p), out(p))$ than in the calculation of $MST(E, in(d), out(d))$, sharing them the same edge set E. For this reason we can conclude:

$$MST(E, in(d), out(d)) \leq MST(E, in(p), out(p)) \ \ \forall p \in T(d) \tag{26}$$

$\square$

As $MST(E, in(d), out(d))$ is the cost of $t(d)$ in scenario u, Lemma 16 states that, in scenario u, $t(d)$, the cost of the spanning tree associated with node d, is less than or equal to the costs of the spanning trees associated with $T(d)$.

Lemma 17 $MST(E \backslash out(d), \emptyset, \emptyset) \geq MST(t(p), \emptyset, \emptyset) \ \ \forall p \in T(d)$

PROOF. $MST(t(p), \emptyset, \emptyset)$ is the cost of the minimum spanning tree in the scenario induced by tree $t(p)$.

A consequence of (25) is:

$$E \backslash out(p) \subseteq E \backslash out(d) \ \ \forall p \in T(d) \tag{27}$$

By definition of $out(p)$ we also have:

$$t(p) \subseteq E \backslash out(p) \ \ \forall p \in T(d) \tag{28}$$

Using (27) and (28) we have:

$$t(p) \subseteq E \backslash out(p) \subseteq E \backslash out(d) \ \ \forall p \in T(d) \tag{29}$$

If we consider (29) together with the definition of function MST, we can conclude:

$$MST(E \backslash out(d), \emptyset, \emptyset) \geq MST(t(p), \emptyset, \emptyset) \ \ \forall p \in T(d) \tag{30}$$

□

Lemma 17 states that $MST(E \backslash out(d), \emptyset, \emptyset)$ is an upper bound for the costs of the minimum spanning trees in the scenarios induced by the spanning trees associated with $T(d)$.

The results of Lemma 16 and Lemma 17 are used in the following theorem.

Theorem 18 $MST(E, in(d), out(d)) - MST(E \backslash out(d), \emptyset, \emptyset) \leq RC(t(p)) \ \ \forall p \in T(d)$

PROOF. By definition of function RC we have:

$$RC(t(p)) = MST(E, in(p), out(p)) - MST(t(p), \emptyset, \emptyset) \tag{31}$$

Using (31) together with Lemma 16 and Lemma 17 we can conclude:

$$MST(E, in(d), out(d)) - MST(E \backslash out(d), \emptyset, \emptyset) \leq$$

$$MST(E, in(p), out(p)) - MST(t(p), \emptyset, \emptyset) = RC(t(p)) \ \ \forall p \in T(d) \tag{32}$$

□

Theorem 18 provides a lower bound for the robustness cost of the spanning trees associated with $T(d)$. We can consequently give the following definition:

$$lb(d) := MST(E, in(d), out(d)) - MST(E \backslash out(d), \emptyset, \emptyset)$$

In practice $lb(d)$ is defined as the cost in scenario u of $t(d)$, the spanning tree associated with the tree-search node d, minus the cost of the minimum 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.

3.7 Reduction rules

In this section some reduction rules, which are to be used every time a new tree-search node d is created in order to increase the dimension of $out(d)$, are described.

It is important to observe that an increasing in the dimension of $out(d)$ is crucial for algorithm RST, because the lower bound $lb(d)$ is tighter when $out(d)$ is larger (see Section 3.6) and a tighter value for $lb(d)$ increases the probability to cut the tree-search node d.

The reduction rules used within RST are described in the following subsections.

3.7.1 Rule R1

Reduction rule $R1$, described in Proposition 20, is based on the following theoretical result.

Theorem 19 *Give a node d of the tree-search, if $\{i,j\},\{j,k\} \in in(d)$ then*

$\forall p \in T(d) \quad \{i,k\} \notin t(p)$.

PROOF.

As $t(p)$ is a tree, we know that it cannot contain cycles. But $\{i,j\},\{j,k\} \in t(p) \quad \forall p \in T(d)$ by definition of $in(d)$. Consequently $\forall p \in T(d) \quad \{i,k\} \notin t(p)$, otherwise a cycle would appear.

□

Theorem 19 implies the reduction rule presented in Proposition 20.

Proposition 20 (Rule *R1*) *Given a node d of the tree-search, if $\{i,j\},\{j,k\} \in in(d)$ then $\{i,k\}$ can be inserted into $out(d)$.*

PROOF. Follows directly from Theorem 19. □

3.7.2 Rule R2

Reduction rule $R2$ is based on a generalization of the preprocessing rule $P1$, presented in Section 3.3.1.

In order to described rule $R2$, we first need the following definitions, which generalize the concepts of weak-tree (see Definition 5) and weak-edge (see Definition 6) respectively.

Definition 21 *A spanning tree is a B-*weak tree *if it has, for some scenario, minimum cost among the spanning trees which include the edges of the edge set B.*

Definition 22 *Edge $\{i, j\}$ is a B-*weak edge *if it lies on some B-weak tree.*

We can now give the following characterization for B-weak edges.

Theorem 23 *Given $B \subseteq E$ such that B does not contain any cycle, edge e is a B-weak edge if and only if there exists a spanning tree of minimum cost, among those containing the edges of B, 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. The proof is a straightforward adaptation of the proof given in Yaman et al. [10] for Theorem 7.

See Section A.2 for a complete proof. □

Theorem 11 implies a practical criterion to identify (non-)B-weak edges in polynomial time. This criterion is based on a modification of Kruskal's algorithm [6] for the classic minimum spanning tree problem similar to the one described in Section 3.3.1 for the identification of (non-)weak edges.

The results above imply the following reduction rule.

Proposition 24 (Rule *R2*) *Give a node d of the tree-search, if $\{i, j\}$ is a non-in(d)-weak edge, then $\{i, j\}$ can be inserted into out(d).*

PROOF. Each robust spanning tree is a weak tree (Theorem 8). Each weak tree including the edges of $in(d)$ (if it exists) must be a $in(d)$-weak tree (definition of $in(d)$-weak tree). Then only $in(d)$-weak trees could eventually 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. □

3.8 Pseudo-code

The branch and bound algorithm whose elements have been described in the previous sections, is summarized in Figure 1, where a pseudo-code is presented.

The algorithm starts by applying preprocessing rules $P1$ and $P2$, described in Section 3.3.1 and Section 3.3.2 respectively. Then the structures to represent r, the root of the tree search are initialized, and r is inserted into S, the set of tree-search nodes to be examined. Variables ub and $ubTree$ are also initialized to $RC(t(r))$ and $t(r)$ respectively.

An iterative statement is then repeated until the tree-search has been completely examined. At each iteration, d, the node in S with the smallest value of $lb(d)$, is selected and extracted from S. If $in(d)$ is not a spanning tree of graph G, edge $e \in t(d) \backslash in(d)$ is selected accordingly to equation (23). A new node d' is then derived from d by inserting edge e into $out(d')$. $RC(t(d'))$ is calculated and, in case of improvement of the best solution found so far, ub and $ubTree$ are updated and each tree-search node p with $lb(p) \geq ub$ is deleted from S. $lb(d')$ is calculated as described in Section 3.6 and if $lb(d') < ub$ then d' is inserted into S. A second new node, d'', with e in $in(d'')$, is created from d. Reduction rules $R1$ and $R2$, described in Section 3.7.1 and Section 3.7.2 respectively, are applied to d'' and $lb(d'')$ is calculated as described in Section 3.6. If $lb(d'') < ub$ then d'' is inserted into S.

When the algorithm exits from the iterative statement, $ubTree$, which contains a robust spanning tree of G, is returned.

4 Computational results

In this section some computational results are presented in order to evaluate the performance of the branch and bound algorithm RST, described in Section 3.

We test our method on the same families of problems used in Yaman et al.

Algorithm RST

W := set of non-weak edges (rule *P1*);
$E := E \backslash W$;
R := set of strong edges (rule *P2*);
$in(r) := R$;
$out(r) := \emptyset$;
Calculate $lb(r)$;
Calculate $t(r)$;
$S := \{r\}$;
$ub := RC(t(r))$;
$ubTree := t(r)$;
While $(S \neq \emptyset)$
    $d := \text{argmin}_{d \in S} \{lb(d)\}$;
    $S := S \backslash \{d\}$;
    If $(in(d) \neq t(d))$
        Select edge e (equation (23));
        $in(d') := in(d)$;
        $out(d') := out(d) \cup \{e\}$;
        Calculate $RC(t(d'))$;
        If $(RC(t(d')) < ub)$
            $ub := RC(t(d'))$;
            $ubTree := t(d')$;
            $\forall p \in S$ such that $lb(p) \geq ub$
                $S := S \backslash \{p\}$;
        Calculate $lb(d')$;
        If $(lb(d') < ub)$
            $S := S \cup \{d'\}$;
        $in(d'') := in(d) \cup \{e\}$;
        $out(d'') := out(d)$;
        $t(d'') := t(d)$;
        Apply rules *R1* and *R2* to d'';
        Calculate $lb(d'')$;
        If $(lb(d'') < ub)$
            $S := S \cup \{d''\}$;
Return $ubTree$;

Fig. 1. A pseudo-code for the branch and bound algorithm RST.

[10] to evaluate the performance of their preprocessing techniques. 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. For each edge $\{i, j\}$, l_{ij} is uniformly distributed in l_k and

Table 1
Computational results.

$\lvert V\rvert$	Yaman et al. [10]	Yaman et al. new	RST
10	1.96	0.85	0.04
15	33.09	286.47	1.84
20	693.88	>2016.28	56.46
25	2027.60	>3600.00	484.24

u_{ij} is uniformly distributed in u_k. The sets adopted are the following ones:

- set 1: $l_1 = [0, 10)$ and $u_1 = (l_{ij}, 10] \quad \forall \{i, j\} \in E$
- set 2: $l_2 = [0, 15)$ and $u_2 = (l_{ij}, 15] \quad \forall \{i, j\} \in E$
- set 3: $l_3 = [0, 20)$ and $u_3 = (l_{ij}, 20] \quad \forall \{i, j\} \in E$
- set 4: $l_4 = [0, 10)$ and $u_4 = (l_{ij}, 20] \quad \forall \{i, j\} \in E$
- set 5: $l_5 = [0, 15)$ and $u_5 = (l_{ij}, 30] \quad \forall \{i, j\} \in E$
- set 6: $l_6 = [0, 20)$ and $u_6 = (l_{ij}, 40] \quad \forall \{i, j\} \in E$

For the case $|V| = 25$ only five test problems from the first set defined above are considered, because of the long computation times.

We compare our results with those obtained by solving the mixed integer program MIP, simplified by the application of the preprocessing rules described in Yaman et al. [10].

We generated the families of benchmarks with the same specifics used in [10], and we repeated exactly the same tests presented there. Some points must anyway be clarified. The problems are not necessary the same used in [10], notwithstanding that they have the same characteristics, because they are randomly generated. The computer on which tests are run is different (we used a Pentium II PC 400 MHz instead of the Pentium II PC 450 MHz mentioned in [10]) and the version of ILOG CPLEX[1] used to solve the mixed integer programs is different (we used version 6.0 instead of version 6.5.1). For these reasons, for each family of problems we present the results reported in [10] (*Yaman et al. [10]* in Table 1), the results obtained by rerunning CPLEX on the problems we have generated (*Yaman et al. new*) and the results obtained by our branch and bound algorithm on these problems (RST). The maximum computation time for each run has been fixed at 3600 seconds. After this time the computation is stopped also in case it is not terminated. This happened for some tests summarized in the last two rows of the third column of Table 1 (entries containing ">"). In these cases a contribute of 3600 seconds has been counted in order to obtain the averages reported in the table.

[1] www.cplex.com.

Table 1 shows that, as expected, using the new approach described in this paper is much more convenient than solving MIP directly.

It must also be observed that RST obtains extremely shorter computation times than the method suggested in [10] also when the results of RST are compared with those reported in [10], which are obtained, as pointed out before, in a more favorable environment.

In Aron and Van Hentenryck [1] an alternative procedure to retrieve weak (and B-weak) edges is proposed. This technique is more efficient than the one we adopt, and consequently if we had used this technique, the results presented in the last column of Table 1 would have been even better.

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 embeds some results previously presented in the literature and some new elements, such as a new lower bound which works by exploiting some properties connected with the branching rule adopted. Some new reduction rules, again based on the exploitation of some peculiarities of the branching rule adopted, contribute to speed up the method.

Some computational tests have been carried out and the results prove that the algorithm we propose is very competitive. It strongly improves the results obtained by a method 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] D.P. Bertsekas and R. Gallagher. *Data Networks.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

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

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

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

[6] 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.

[7] T.L. Magnanti and L. Wolsey. Optimal trees. In *Handbook in Operations Research and Management Science* (M.O. Ball et al. eds.), volume 7, pages 503–615. North Holland, Amsterdam, 1995.

[8] R. Montemanni, L.M. Gambardella, A.E. Rizzoli, and A.V. Donati. An branch and bound algorithm for the robust shortest path problem with interval data. Technical Report IDSIA-09-02, Istituto Dalle Molle di Studi sull'Intelligenza

Artificiale, May 2002. (ftp://ftp.idsia.ch/pub/techrep/IDSIA-09-02.ps.gz).

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

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

A Appendix

A.1 Proof of Theorem 14

Let $t' = t \backslash \{e\} \cup \{f\}$. Suppose t' does not have minimum cost in scenario r among trees on H which include the edges in B. There exist then $g \in t' \backslash B$ and $h \in E_H$ such that $t'' = t' \backslash \{g\} \cup \{h\}$ is a spanning tree of H, include the edges of B and has a smaller cost than t' in scenario r, i.e. $c_h^r < c_g^r$. 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 $c_h^r < c_g^r$, the tree $t \backslash \{g\} \cup \{h\}$ would be a spanning tree and would cost less than t in scenario r. 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:

$$c_f^r \leq c_h^r < c_g^r \tag{A.1}$$

So $g \neq f$ and $g \in s \backslash \{f\} = t_a \cup t_b$.

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 (A.1) implies that both of these spanning trees have a cost less than t in scenario r, a contradiction to the hypothesis. Thus the original assumption is false and t' has minimum cost in scenario r. $\square$

A.2 Proof of Theorem 23

In this proof we will call *B-spanning tree* a spanning tree including the edges of the edge set B.

If there exists a minimum B-spanning tree that uses edge e for the scenario defined in the hypothesis, then edge e is a B-weak edge by definition.

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

Consider Kruskal's algorithm [6] 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 $\mathcal{L} = B$, forcing the edges of B 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 B-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 B-spanning tree for scenario s, then it cannot be on any minimum B-spanning tree under any scenario, and so it must be that edge e is not a B-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 B-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 $(C\backslash D) \cup (\bigcup_{e' \in D} C_{e'}\backslash\{e'\})$ 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 B-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 B-spanning tree under any scenario s', and hence e cannot be a B-weak edge. $\square$