A branch and bound algorithm for the robust shortest path problem with interval data

R. Montemanni [*], L.M. Gambardella, A.V. Donati

Istituto Dalle Molle di Studi sull'Intelligenza Artificiale

Galleria 2, CH-6928 Manno, Switzerland

Abstract

Many real problems in transportation and telecommunications can be modelled in mathematical terms as shortest path problems on interval digraphs, where an interval cost is associated with each arc. Intervals represent uncertainty, typical of real situations, about the exact values of costs.

A robust shortest path is a path which is not too far from the best one, whatever the exact values of arc costs are. This criterion, expressed in mathematical terms, is used to drive optimization.

A branch and bound algorithm for the robust shortest path problem is described. It is based on a new lower bound and on some new reduction rules which exploit the properties of the branching strategy adopted.

Computational results are finally presented.

Key words: Branch and bound, shortest path problem, robust optimization, interval data.

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

1 Introduction

In the field of transportation problems, a road network is usually modelled in mathematical terms as a weighted digraph, where each arc is associated with a road and costs represent travel times. In this context, a shortest path problem has to be solved every time the quickest way to go from a place to another has to be calculated.

A similar problem arises in telecommunications when a packet has to be sent from a source node to a destination node on a network. Also in this case, where the network is usually modelled as a weighted digraph and costs are associated with transmission delays, a shortest path problem is faced.

Unfortunately in the reality it is not easy to estimate arc costs exactly, being them depending on many factors which are difficult to predict, such as traffic conditions, accidents, traffic jams or weather conditions for the transportation case, network congestions or hardware failures for the telecommunications case. For this reason the fixed cost model previously introduced may be inadequate, as it generates a very high abstraction level. To overcome this problem, more complex models have been presented in the literature. In particular a model where a set of alternative graphs are considered in the same time (*scenario model* - see Yu and Yang [8] and Dias and Clímaco [2]) and a model where an interval of possible values is associated with each arc (*interval data model* - see Dias and Clímaco [2] and Karaşan et al. [4]) have been studied. In this work the interval data model, which will be described in detail in Section 2, is considered.

Gambardella), `alberto@idsia.ch` (A.V. Donati).

With the interval data model, uncertainty is modelled by associating an interval of costs with each arc. Each interval represents a range of possible values for the real cost.

The *relative robustness criterion*, which will be formally defined in Section 2, has been chosen to drive optimization. This criterion is discussed in Kouvelis and Yu [5], a book entirely devoted to robust discrete optimization, and has already been used for the shortest path problem with interval data in Karaşan et al. [4].

A *relative robust shortest path* from s to t is a path from s to t which minimizes the maximum deviation from the optimal shortest path from s to t over all realizations of arc costs.

Yu and Yang [8], which proved that the relative robust shortest path problem with scenarios is $\mathcal{NP}$-hard, conjectured that also the problem with interval data is $\mathcal{NP}$-hard. Unfortunately no proof was presented, neither in Karaşan et al. [4], which reports the same conjecture.

It is interesting to observe that, as far as we are aware, there are just few problems which have been proved to be $\mathcal{NP}$-hard in their scenarios version but polynomial in their interval data version. They are the problem of selecting p elements of minimum total weight out of a set of m elements (see Averbakh [1]), the *absolute robust spanning tree problem* (see Yaman et al. [7]) and the *absolute robust shortest path problem* (see Karaşan et al. [4]).

In the remainder of this paper we will refer to the *relative robust shortest path problem* simply as the *robust shortest path problem* and to a *relative robust shortest path* simply as a *robust shortest path.*

In Karaşan et al. [4] a mixed integer programming formulation and a preprocessing technique for the robust shortest path problem (inspired by the one described in Yaman et al. [7] for the robust spanning tree problem) are presented. This last technique, which unfortunately works only for *acyclic*, *layered* graphs with a small *width*, is used to eliminate some arcs which will never be in an optimal path. We remaind the reader that an *acyclic* graph is a graph whose arcs do not form any cycle and a *layered* graph is a graph whose vertices can be partitioned into a chain of disjoint subsets, in such a way that the cardinality of each subset is limited by a given constant, called *width*, and arcs exist only from each subset to the following one in the chain.

In this paper a branch and bound algorithm for the robust shortest path problem with interval data is presented. It is based on a lower bound and on some reduction rules which work by exploiting some properties of the particular branching strategy adopted. The algorithm can be seen as an improvement of the method described in Montemanni and Gambardella [6].

In Section 2 the robust shortest path 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 shortest path problem is defined on a directed graph $G = (V, A)$, where V is a set of vertices, A is a set of arcs. A starting vertex $s \in V$, and a destination vertex $t \in V$ are given and an interval $[l_{ij}, u_{ij}]$, with $0 \leq l_{ij} \leq u_{ij}$,

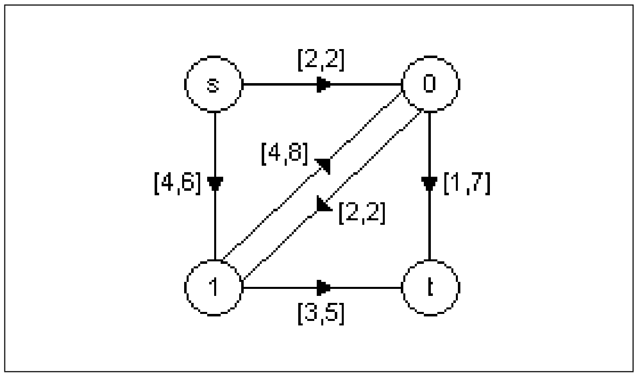

Fig. 1. Example of interval graph.

is associated with each arc $(i, j) \in A$. In Figure 1 an example of interval graph is given.

Accordingly to Karaşan et al. [4], we can formally describe the robust shortest path problem with interval data through the following definitions:

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

Definition 2 *The* robust deviation *for a path p from s to t in a scenario r is the difference between the cost of p in r and the cost of the shortest path from s to t in scenario r.*

Definition 3 *A path p from s to t is said to be a* robust shortest path *if it has the smallest (among all paths from s to t) maximum (among all possible scenarios) robust deviation.*

A scenario can be seen as a snapshot of the network situation, while a robust shortest path is a path which should guarantee reasonably good performance under any possible arc costs configuration.

From the literature, the following result is known:

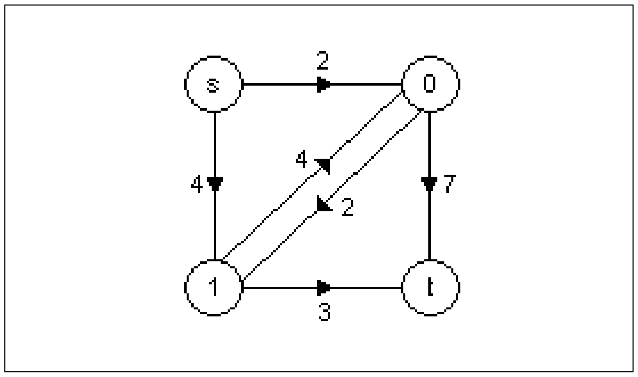

Fig. 2. Scenario induced by $p = \{s, 0, t\}$ on the graph of Figure 1.

Observation 1 (Karaşan et al. [4]) *Given a path p from s to t, the scenario r which makes the robust deviation maximum is the one where each arc (i, j) on p has cost u_{ij} and each arc (k, h) not on p has cost l_{kh}, i.e. $c^r_{ij} = u_{ij}$ $\forall(i, j) \in p$ and $c^r_{kh} = l_{kh}$ $\forall(k, h) \notin p$.*

In the remainder of this paper we will refer to the scenario r derived from path p as described in Observation 1, as the scenario *induced* by path p. We will also refer to the cost of p minus the cost of a shortest path of the scenario induced by p as the *robustness cost* of p. Figure 2 depicts an example of the scenario induced by path $p = \{s, 0, t\}$ on the graph of Figure 1. The robustness cost of p is in this case $(2 + 7) - (4 + 3) = 2$.

Observation 1 is very important because it suggests a procedure of complexity $O(n^2)$ (see Dijkstra [3]) for the evaluation of the robustness cost of a given path. In practice it is enough to subtract the cost of the shortest path in the scenario induced by path p from the cost, in the same scenario, of path p.

Applying Observation 1, Karaşan et al. [4] modelled the problem through a mixed integer programming formulation, which is presented after a brief description of its variables:

- y_{ij}: it is 1 when arc (i, j) is on the robust shortest path from s to t; 0

otherwise;

- x_i: it contains the cost of the shortest path from s to i on the scenario induced by the robust shortest path (defined by y variables).

$$(MIP) \quad \text{Min} \sum_{(i,j)\in A} u_{ij} y_{ij} - x_t \tag{1}$$

$$\text{s.t. } x_j \le x_i + l_{ij} + (u_{ij} - l_{ij})\, y_{ij} \qquad \forall (i,j) \in A \tag{2}$$

$$\sum_{(j,k)\in A} y_{jk} - \sum_{(i,j)\in A} y_{ij} = \begin{cases} 1 & \text{if } j = s \\ -1 & \text{if } j = t \\ 0 & \text{otherwise} \end{cases} \qquad \forall j \in V \tag{3}$$

$$x_s = 0 \tag{4}$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{5}$$

$$x_j \ge 0 \qquad \forall j \in V \tag{6}$$

The key-inequalities of the formulation are the (2)s, which maintain consistency between x variables and y variables by realizing an arc costs alignment. The remaining constraints are basically those of the classic shortest path problem formulation (see, for example, Karaşan et al. [4]).

3 The branch and bound algorithm RSP

A branch and bound algorithm for the robust shortest path problem with interval data, which constructs and visits a search-tree, is presented in this section. We will refer to this method as *algorithm RST* (Robust Shortest Path).

Section 3.1 is dedicated to the introduction of the notation used. The elements of the method are described in Section 3.2, Section 3.3, Section 3.4 and Section 3.5. In Section 3.6 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:

- path : path from s to t in G;
- scenario u: scenario such that $c^u_{ij} = u_{ij}\ \forall (i,j) \in A$;
- $T(d)$: search-tree nodes contained in the subtree rooted in search-tree node d;
- $RC(p)$: robustness cost of path p. Accordingly to Observation 1, it is obtained by subtracting the cost of the shortest path in the scenario induced by path p from the cost of path p in scenario u;
- $SP(B, in, out)$: defined scenario s as the scenario where $c^s_{ij} = u_{ij}\ \forall (i,j) \in B$ and $c^s_{ij} = l_{ij}\ \forall (i,j) \in A \backslash B$, $SP(B, in, out)$ is the cost of p, the path with the minimum cost in scenario s among those which include all the arcs of the arc set in and do not contain any arc from the arc set out.

3.2 Structure of the search-tree node

Each node d of the search-tree constructed by the algorithm is identified by the following structures:

- $in(d)$: list of arcs. The arcs contained in $in(d)$ must appear in all of the paths associated with the nodes of $T(d)$;

- $out(d)$: list of arcs. The arcs contained in $out(d)$ are forbidden for all of the paths associated with the nodes of $T(d)$;
- $P(d)$: path associated with the search-tree node d. $P(d)$ is the path with minimum cost in scenario u which respects the limitations imposed by arc sets $in(d)$ and $out(d)$, i.e. $P(d)$ must contain the arcs in $in(d)$ and cannot include the arcs in $out(d)$;
- $lb(d)$: lower bound for the robustness cost of the paths associated with the search-tree nodes of $T(d)$. The calculation of the lower bound is described in Section 3.4.

3.3 Branching strategy

The root r of the tree search constructed by the algorithm is the node with $in(r) = \emptyset$, $out(r) = \emptyset$ and consequently (see Section 3.4) $lb(r) = 0$. Initially r is the only node of the set of the nodes to be examined.

At each iteration the not yet examined node d with the smallest value of $lb(d)$ is selected and, if it exists, the first arc a on $P(d)$ which is not contained in $in(d)$ is identified. If $P(d) = in(d)$, node d is a leaf of the search-tree and consequently a does not exist. Otherwise two new search-tree nodes are created. The first new node, d', has $in(d') = in(d)$ and $out(d') = out(d) \cup \{a\}$, while d'', the second one, has $in(d'') = in(d) \cup \{a\}$ and $out(d'') = out(d)$. In case d' and d'' are not proved to be dominated, they are inserted into the set of search-tree nodes to be examined.

There is an important property connected with the use of the branching rule described above. Each set $in(d)$ contains a chain of connected arcs which form a sub-path starting from s. This property is important because it favors a massive application of the reduction rules described in Section 3.5.

3.4 Lower bound

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

The lower bound is based on the following theoretical results.

Lemma 4 $SP(A, in(d), out(d)) \leq SP(A, in(p), out(p)) \quad \forall p \in T(d)$

PROOF. Direct consequences of the branching rule described in Section 3.5 are:

$$in(d) \subseteq in(p) \quad \forall p \in T(d) \tag{7}$$

$$out(d) \subseteq out(p) \quad \forall p \in T(d) \tag{8}$$

Because of (7) and (8), there are less freedom degrees in the calculation of $SP(A, in(p), out(p))$ than in the calculation of $SP(A, in(d), out(d))$, sharing them the same arc set A. For this reason we can conclude:

$$SP(A, in(d), out(d)) \leq SP(A, in(p), out(p)) \quad \forall p \in T(d) \tag{9}$$

$SP(A, in(d), out(d))$ is the cost of $P(d)$, the path associated with the search-tree node d, in scenario u. Consequently Lemma 4 states then that the cost of $P(d)$ in scenario u is less than or equal to the costs of the paths associated with $T(d)$ in the same scenario.

Lemma 5 $SP(A \backslash out(d), \emptyset, \emptyset) \geq SP(P(p), \emptyset, \emptyset) \;\; \forall p \in T(d)$

PROOF. A consequence of (8) is:

$$A \backslash out(p) \subseteq A \backslash out(d) \;\; \forall p \in T(d) \tag{10}$$

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

$$P(p) \subseteq A \backslash out(p) \;\; \forall p \in T(d) \tag{11}$$

Using (10) and (11) we have:

$$P(p) \subseteq A \backslash out(p) \subseteq A \backslash out(d) \;\; \forall p \in T(d) \tag{12}$$

Using (12) we can conclude:

$$SP(A \backslash out(d), \emptyset, \emptyset) \geq SP(P(p), \emptyset, \emptyset) \;\; \forall p \in T(d) \tag{13}$$

Observing that $SP(P(p), \emptyset, \emptyset)$ is the cost of the shortest path in the scenario induced by path $P(p)$, Lemma 5 states that $SP(A \backslash out(d), \emptyset, \emptyset)$ is an upper bound for the cost of the shortest paths in the scenarios induced by the paths associated with $T(d)$.

Theorem 6 $SP(A, in(d), out(d)) - SP(A \backslash out(d), \emptyset, \emptyset) \leq RC(P(p)) \;\; \forall p \in T(d)$

PROOF. By definition we have:

$$RC(P(p)) = SP(A, in(p), out(p)) - SP(P(p), \emptyset, \emptyset) \tag{14}$$

Using (14) together with Lemma 4 and Lemma 5 we can conclude:

$$SP(A, in(d), out(d)) - SP(A \backslash out(d), \emptyset, \emptyset) \leq$$

$$SP(A, in(p), out(p)) - SP(P(p), \emptyset, \emptyset) = RC(P(p)) \quad \forall p \in T(d) \tag{15}$$

Theorem 6 suggests a lower bound for the robustness costs of the paths associated with $T(d)$. We can then give the following definition:

$$lb(d) := SP(A, in(d), out(d)) - SP(A \backslash out(d), \emptyset, \emptyset)$$

3.5 Reduction rules

As observed in Section 3.3, the branching rule adopted in our algorithm generates at each node d a set $in(d)$ whose arcs form a path from s to z, with $z \in V$. This path must be contained in all the paths associated with $T(d)$ by definition.

This property can be used to speed up the evaluation of $SP(A, in(d), out(d))$. It is enough to calculate the shortest path, accordingly to $in(d)$ and $out(d)$, from z to t and to add to it the cost of the arcs contained in $in(d)$.

The main advantages connected with the use of the branching strategy described in Section 3.3, are however those arising from the following theoretical results, which define reduction rules $R1$ and $R2$.

Theorem 7 *Given a node d of the search-tree, if $(i,j) \in in(d)$ then $\forall (i,k) \in A \backslash \{(i,j)\}$, $(i,k) \notin P(p) \quad \forall p \in T(d)$.*

PROOF. As $P(p)$ is a path, we know that $\forall i \in V$ no more than one arc of type (i,k) can be in $P(p)$. But $(i,j) \in P(p) \ \ \forall p \in T(d)$ by definition of $in(d)$ and because of the branching rule adopted. Consequently, $\forall (i,k) \in A \backslash \{(i,j)\}$, $(i,k) \notin P(p) \ \ \forall p \in T(d)$.

Theorem 7 implies the result presented in Proposition 8, which can be used, given a search-tree node d, to increase the dimension of the arc set $out(d)$.

Proposition 8 (Rule *R1*) *Give a node d of the search-tree, if $(i,j) \in in(d)$ then $\forall (i,k) \in A \backslash \{(i,j)\}$, (i,k) can be inserted into $out(d)$.*

PROOF. Follows directly from Theorem 7.

Theorem 9 *Give a node d of the search-tree, if $(i,j) \in in(d)$ then $\forall (k,j) \in A \backslash \{(i,j)\}$, $(k,j) \notin P(p) \ \ \forall p \in T(d)$.*

PROOF. As $P(p)$ is a path, we know that $\forall j \in V$ no more than one arc of type (k,j) can be contained in $P(p)$. But $(i,j) \in P(p) \ \ \forall p \in T(d)$ by definition of $in(d)$ and because of the branching rule adopted. Consequently, $\forall (k,j) \in A \backslash \{(i,j)\}$, $(k,j) \notin P(p) \ \ \forall p \in T(d)$.

Theorem 9 implies the result presented in Proposition 10, which can be used, given a search-tree node d, to increase the dimension of the arc set $out(d)$.

Proposition 10 (Rule *R2*) *Give a node d of the search-tree, if $(i,j) \in in(d)$ then $\forall (k,j) \in A \backslash \{(i,j)\}$, (k,j) can be inserted into $out(d)$.*

PROOF. Follows directly from Theorem 9.

It is interesting to observe that the results presented in Proposition 8 and Proposition 10 are very important for algorithm RSP. For a given search-tree node d, a larger dimension of $out(d)$ implies a tighter lower bound $lb(d)$ (see Section 3.4).

3.6 Pseudo-code

The branch and bound algorithm whose elements have been described in the previous sections is summarized in Figure 3, where a pseudo-code is presented.

The algorithm starts by initializing the structures of r, the root of the tree search, which is then inserted into S, the set of nodes to be examined. Variables ub and $ubPath$ (which contain the cost and the arcs of the best robust path encountered so far) are also initialized to $RC(P(r))$ and $P(r)$, respectively.

An iterative statement is then repeated until the search-tree has been completely examined. d, the node in S with the smallest value of $lb(d)$ is selected and extracted from S. If $in(d)$ is not a complete path from s to t then a, the first arc on $P(d)$ which is not contained in $in(d)$ is selected. A new node d' is then derived from d by inserting a into $out(d')$. $RC(P(d'))$ is calculated and, in case of improvement of the best solution found so far, ub and $ubPath$ are updated and each node p with $lb(p) \geq ub$ is deleted from S. If $lb(d') < ub$ then d' is inserted into S. A second new node, d'', with a inserted into $in(d'')$, is created from d. Reduction rules $R1$ and $R2$ are applied to d'' and $lb(d'')$ is calculated. If $lb(d'') < ub$ then d'' is inserted into S.

When the algorithm exits from the iterative statement, $ubPath$, which contains the robust shortest path from s to t, is returned.

```
Algorithm RSP

$in(r) := \emptyset$;
$out(r) := \emptyset$;
$lb(r) := 0$;
$S := \{r\}$;
$ub := RC(P(r))$;
$ubPath := P(r)$;
While ($S \neq \emptyset$)
    $d := \text{argmin}_{p \in S} \{lb(p)\}$;
    $S := S \backslash \{d\}$;
    If ($in(d) \neq P(d)$)
        $in(d') := in(d)$;
        $a :=$ the first arc in $P(d)$ not contained in $in(d)$;
        $out(d') := out(d) \cup \{a\}$;
        Calculate $RC(P(d'))$;
        If ($RC(P(d')) < ub$)
            $ub := RC(P(d'))$;
            $ubPath := P(d')$;
            $\forall p \in S$ such that $lb(p) \geq ub$
                $S := S \backslash \{p\}$;
        Calculate $lb(d')$ as described in Section 3.4;
        If ($lb(d') < ub$)
            $S := S \cup \{d'\}$;
        $in(d'') := in(d) \cup \{a\}$;
        $out(d'') := out(d)$;
        Apply the reduction rules described in Section 3.5 to $d''$;
        Calculate $lb(d'')$ as described in Section 3.4;
        If ($lb(d'') < ub$)
            $S := S \cup \{d''\}$;
Return $ubPath$;
```

Fig. 3. A pseudo-code for the branch and bound algorithm.

4 Computational results

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

In Section 4.1 the benchmarks adopted are described. In Section 4.2 the results obtained by the branch and bound algorithm are presented.

4.1 Description of the benchmarks

The method described in Section 3 has been tested on problems based on graphs with different characteristics, and which can be divided in three families. These families are described in the following subsections.

4.1.1 Random *graphs*

This family is composed of random graphs we have generated.

A graph of type R-n-c-δ has n vertices, an approximated arc density of δ (i.e. $|A| \sim \delta n(n-1)$) and $u_{ij} \leq c \ \ \forall (i,j) \in A$. Arcs are set up between random pairs of vertices and interval costs are generated randomly.

4.1.2 Karaşan *graphs*

The structure of these random generated graphs is the same as that of the benchmarks adopted in Karaşan et al. [4]. These graphs should simulate telecommunication networks. As observed in Section 1, these graphs are *acyclic* and *layered*. They are also *completed*, i.e. each node of a layer of the graph is directly connected to every node of the following layer.

A graph of type K-n-c-d-w (where $0 < d < 1$) has n vertices; each interval cost $[l_{ij}, u_{ij}]$ is obtained by generating a random number $c_{ij} \in [1, c]$ and by randomly selecting l_{ij} in $[(1-d)c_{ij}, (1+d)c_{ij}]$ and u_{ij} in $[l_{ij}, (1+d)c_{ij}]$; w is finally the *width* of the graph. See Karaşan et al. [4] for a more exhaustive description of this family of graphs.

4.1.3 Real *graphs*

The graphs of this family represent real road networks, and the interval costs associated are realistic.

The following two graphs have been analyzed:

- *Sottoceneri*: this graph models the main roads of the Sottoceneri region, which is the southern part of Canton Ticino (Switzerland). The graph has 387 vertices and 1038 arcs;
- *Stuttgart*: this graph models the road network of the Stuttgart area (Germany). The graph has 2490 vertices and 16153 arcs.

4.2 Results

The algorithm described in Section 3 has been implemented in C++ and all the tests presented have been carried out on a computer equipped with a Pentium II 400MHz processor. *ILOG CPLEX 6.0* (http://www.cplex.com) has been used to solve mixed integer programs.

For *Karaşan* graphs $s = 0$ and $t = |V| - 1$ by definition (see Karaşan et al. [4]). For *random* and *real* graphs s and t are selected randomly.

For each graph considered, 10 problems have been created and solved. The results obtained are summarized in Table 1, where columns have the following meaning:

- Graph: name of each graph considered, accordingly to Section 4.1;
- MIP: average computation time in seconds required by CPLEX to solve the

Table 1
Computational results.

Graph	MIP	RSP
R-500-100-0.01	4.633	1.481
R-500-100-0.001	0.578	0.599
R-500-100-0.1	14.365	2.388
R-100-100-0.01	0.079	0.046
R-900-100-0.01	25.161	5.244
R-500-10-0.01	5.033	0.553
R-500-1000-0.01	5.863	3.069
K-30-20-0.9-2	0.016	0.025
K-60-20-0.9-2	0.088	7.085
Sottoceneri	0.246	0.115
Stuttgart	24.611	6.285

mixed integer program MIP;

- RSP: average computation time in seconds required by the branch and bound algorithm RSP.

From Table 1 we can notice how RSP is extremely effective on problems based on *random* graphs. In particular we can observe how it is substantially better than CPLEX, except for problem *R-500-100-0.001*, where the difference in the computation times is however very small. We can also notice that our method is less sensible than CPLEX to changes in the arc density and in the number of vertices, while it appears more sensible than CPLEX to changes in the maximum cost c. In particular our method seems to be more effective when c is small.

Tests on *Karaşan* graphs are not very encouraging for RSP. Our method is slower than CPLEX and the degradation in performance when the problem dimensions increase is much higher for RSP than for CPLEX. This bad performance of our algorithm can be justified by observing that *Karaşan* graphs, having a lot of alternative paths from s to t with similar costs, present the worst possible situation for RSP.

It must be pointed out that, for those problems, our method should receive a great benefit, probably even more than CPLEX, from the application of the preprocessing rules described in Karaşan et al. [4], because these rules would drastically reduce the number of paths from s to t. However we have not implemented these rules because, as stated in Section 1, they can be used only for *acyclic*, *layered* graphs with a small *width*, and realistic transportation problems, on which we are mainly interested, do not present these peculiarities.

On realistic transportation problems RSP is faster than CPLEX, especially for problems based on the *Stuttgart* graph, which is quite large.

5 Conclusion

The robust shortest path problem, which can be used to model in mathematical terms some very common transportation and telecommunication problems, has been studied in this paper.

A branch and bound algorithm for this problem has been presented. It is based on a lower bound which works by exploiting some properties connected with the branching rule adopted. Some reduction rules, again based on the branching rule adopted, also contribute to speed up the method.

The algorithm is extremely easy to implement, and computational results prove that it is in general very efficient.

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.

The graph theoretical representation of the road network of the Stuttgart area has been provided by *PTV AG* (http://www.ptv.de).

The authors would like to thanks Andrea Rizzoli for the useful discussions.

References

[1] I. Averbakh. On the complexity of a class of combinatorial optimization problems with uncertainty. *Mathematical Programming*, 90(2):263–272, 2001.

[2] L.C. Dias and J.N. Clímaco. Shortest path problems with partial information: models and algorithms for detecting dominance. *European Journal of Operational Research*, 121:16–31, 2000.

[3] E.W. Dijkstra. Note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[4] O.E. Karaşan, M.Ç. Pinar, and H. Yaman. The robust shortest path problem with interval data. Submitted for publication, August 2001. (http://ie.bilkent.edu.tr/~mustafap/pubs/rsp.ps).

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

[6] R. Montemanni and L.M. Gambardella. An algorithm for the relative robust shortest path problem with interval data. Technical Report IDSIA-05-02, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, February 2002. (ftp://ftp.idsia.ch/pub/techrep/IDSIA-05-02.ps.gz).

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

[8] G. Yu and J. Yang. On the shortest path problem. *Computers and Operations Research*, 25(6):457–468, 1998.