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 (IDSIA)
Galleria 2, CH-6928 Manno, Switzerland

Abstract

Many real problems can be modelled as robust shortest path problems on interval digraphs, where intervals represent uncertainty about real costs and a robust path is not too far from the shortest path for each possible configuration of the arc costs.

A branch and bound algorithm for this problem is presented.

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

1 Introduction

When transportation problems are modelled in mathematical terms, a road network is usually represented 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 one 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 reality it is not easy to estimate arc costs exactly, since they depend 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. 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 at the same time (*scenario model* - see Yu and Yang [7] and Dias and Clímaco [1]) and a model where an interval of possible values is associated with each arc (*interval data model* - see Dias and Clímaco [1] and Karaşan et al. [3]) have been studied. In this work the interval data model, which will be described in detail in Section 2, is considered.

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 [4], 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. [3].

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 [7], which proved that the robust deviation shortest path problem with scenarios is $\mathcal{NP}$-hard, conjectured that the problem with interval data is also $\mathcal{NP}$-hard. The conjecture has been recently proven to be true (Zieliński [8]).

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. [3] a mixed integer programming formulation and a preprocessing technique for the robust shortest path problem (inspired by the one described in Yaman et al. [6] for the robust spanning tree problem) are presented. This technique, which unfortunately works only for particular classes of graphs (see Section 4 for more details), is used to retrieve some arcs which
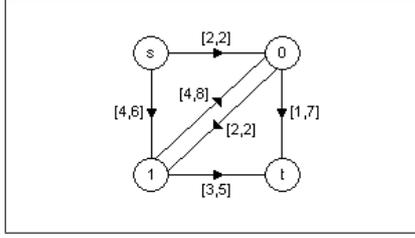
Fig. 1. Example of an interval graph.

will never be in an optimal path.

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 [5].

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.

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 $0 \leq l_{ij} \leq u_{ij}$, is associated with each arc $(i, j) \in A$. In Figure 1 an example of an interval graph is given.

According to Karaşan et al. [3], 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. each 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
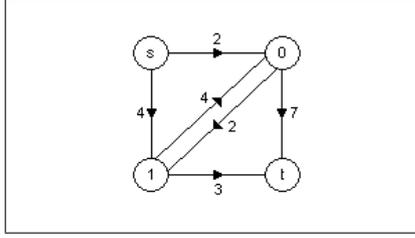
3

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

shortest path is a path which guarantees reasonably good performance under any possible arc cost configuration.

From the literature, the following result is known:

Observation 1 (Karaşan et al. [3]) *Given a path p from s to t, the scenario r which maximizes the robust deviation for p 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 [2]) for the evaluation of the robustness cost of a given path. The robustness cost of p can be calculated by subtracting 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. [3] modelled the problem with 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 in the scenario induced by the robust shortest path (defined by y variables).

4

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

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

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

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

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

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

The key-inequalities of the formulation are the (2)s, which maintain consistency between x variables and y variables by realizing an arc cost alignment. The remaining constraints are basically those of the classic shortest path problem formulation (see, for example, Karaşan et al. [3]).

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 to be used. The elements of the method are described in Sections 3.2-3.5. 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:

- path : path from s to t in G;
- scenario u: scenario such that $c_{ij}^u = u_{ij} \ \forall (i,j) \in A$;
- $T(d)$: search-tree nodes contained in the subtree rooted in search-tree node d;
- $RC(p)$: robustness cost of path p. According 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;

- $SP(B, in, out)$: let s denote the scenario where $c_{ij}^s = u_{ij}\ \forall (i,j) \in B$ and $c_{ij}^s = l_{ij}\ \forall (i,j) \in A \backslash B$, then $SP(B, in, out)$ 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. $SP(B, in, out) = +\infty$ if such a path does not exist.

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 the 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 search-tree 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 path $P(d)$ (starting from node s) 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.

The following important result can be derived.

Theorem 1 *The branching rule adopted leads to an exhaustive search of the space of the paths from s to t in G.*

PROOF. In this proof we consider the search-tree resulting from the application of the branching strategy described above, without using the lower bound and the reduction rules described in Sections 3.4 and 3.5.

Let p be an arbitrary path from s to t. Suppose p has m arcs, where $m \geq 1$. It suffices to show there exists a leaf l in the search tree such that $in(l) = p$. We do this by identifying a sequence of nodes $r_1, \ldots, r_m$ in the search tree, where m is the number of arcs in p and $in(r_i)$ contains the first i arcs in p (hence $r_m = l$). To begin, set d_1 to be r. Consider the examination of d_1 in the branching strategy. If the first arc a on $P(d_1)$ is also the first arc on p, then $in(d'')$ is set to $\{a\}$; in addition, set r_1 to be d''. If this is not the case, then set d_2 to be d'. Consider the eventual examination of d_2 by the strategy. If the first arc a on $P(d_2)$ is also the first arc on p, then $in(d'')$ for d_2 is set to $\{a\}$; in addition, set r_1 to be d'' for d_2. If this is not the case, then set d_3 to be d' for d_2. Eventually we must consider all arcs out of s (due to the structure of the $out()$ sets for the d_i sets), hence, eventually, $in(r_1) = \{a\}$. If $m = 1$, then $r_1 = l$ and we are done. Otherwise, repeat the same procedure starting from r_1 to find a search tree node r_2 such that $in(r_2)$ contains the first two arcs on p. Continuing this procedure we obtain $r_m = l$, which concludes the proof. □

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 1 $SP(A, in(d), out(d)) \leq SP(A, in(f), out(f)) \quad \forall f \in T(d).$

PROOF. Direct consequences of the branching rule described in Section 3.3 are:

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

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

Because of (7) and (8), there are fewer degrees of freedom in the calculation of $SP(A, in(f), out(f))$ than in the calculation of $SP(A, in(d), out(d))$, although they share the same arc set A. The result follows. $\square$

$SP(A, in(d), out(d))$ is the cost of $P(d)$, the path associated with the search-tree node d, in scenario u. Consequently Lemma 1 states that the cost of $P(d)$ in scenario u is less than or equal to the costs of the paths associated with any search-tree node in $T(d)$ in the same scenario.

Lemma 2 $SP(A \backslash out(d), \emptyset, \emptyset) \geq SP(P(f), \emptyset, \emptyset) \quad \forall f \in T(d)$.

PROOF. A consequence of (8) is:

$$A \backslash out(f) \subseteq A \backslash out(d) \quad \forall f \in T(d) \tag{9}$$

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

$$P(f) \subseteq A \backslash out(f) \quad \forall f \in T(d) \tag{10}$$

Using (9) and (10) we have:

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

Using (11) we can conclude:

$$SP(A \backslash out(d), \emptyset, \emptyset) \geq SP(P(f), \emptyset, \emptyset) \quad \forall f \in T(d) \tag{12}$$

$\square$

Since $SP(P(f), \emptyset, \emptyset)$ is the cost of the shortest path in the scenario induced by path $P(f)$, Lemma 2 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 2 $SP(A, in(d), out(d)) - SP(A \backslash out(d), \emptyset, \emptyset) \leq RC(P(f)) \quad \forall f \in T(d)$.

PROOF. By definition $RC(P(f))$ can be expressed through the following equation:

$$RC(P(f)) = SP(A, in(f), out(f)) - SP(P(f), \emptyset, \emptyset) \tag{13}$$

Using (13) together with Lemma 1 and Lemma 2 we can conclude:

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

$$SP(A, in(f), out(f)) - SP(P(f), \emptyset, \emptyset) = RC(P(f)) \quad \forall f \in T(d) \qquad (14)$$

□

Theorem 2 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, according to $in(d)$ and $out(d)$, from z to t and to add to it the cost of the arcs contained in $in(d)$.

However, the main advantages connected with the use of the branching strategy described in Section 3.3 arise from the following theoretical results, which define reduction rules $R1$ and $R2$.

Theorem 3 *Given a node d of the search-tree, if $(i,j) \in in(d)$ then $\forall (i,k) \in A \backslash \{(i,j)\}$, $\forall f \in T(d)$ $(i,k) \notin P(f)$.*

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

Theorem 3 implies the result presented in Proposition 1, which can be used, given a search-tree node d, to increase the dimension of the arc set $out(d)$.

Proposition 1 (Rule $R1$) *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) can be inserted into $out(d)$.*

PROOF. Follows directly from Theorem 3. □

Theorem 4 and Proposition 2 are very similar to Theorem 3 and Proposition 1. For this reason no proof is presented for them.

Theorem 4 *Given a node d of the search-tree, if $(i, j) \in in(d)$ then $\forall (k, j) \in A \backslash \{(i, j)\}$, $\forall f \in T(d)$ $(k, j) \notin P(f)$.*

Theorem 4 implies the result presented in Proposition 2, which can be used, given a search-tree node d, to increase the dimension of the arc set $out(d)$.

Proposition 2 (Rule *R2*) *Given 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 $out(d)$.*

The results presented in Proposition 1 and Proposition 2 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 search-tree, 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 f with $lb(f) \geq ub$ 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.

10

Algorithm RSP

$in(r) := \emptyset; \ out(r) := \emptyset; \ lb(r) := 0;$
$S := \{r\}; \ ub := RC(P(r)); \ ubPath := P(r);$
While $(S \neq \emptyset)$
    $d := \mathrm{argmin}_{f \in S} \{lb(f)\}; \ S := S\backslash\{d\};$
    If $(in(d) \neq P(d))$
        $a := $ the first arc in $P(d)$ not contained in $in(d);$
        $in(d') := in(d); \ out(d') := out(d) \cup \{a\};$
        Calculate $RC(P(d'));$
        If $(RC(P(d')) < ub)$
            $ub := RC(P(d')); \ ubPath := P(d');$
            $\forall f \in S$ such that $lb(f) \geq ub$
                $S := S\backslash\{f\};$
        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 and analyzed.

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 into three families. These families are the following.

***Random* graphs.** This family is composed of random graphs we have generated. A graph of type R-n-c-δ has n vertices and an approximate arc density of δ (i.e. $|A| \sim \delta n(n-1)$). Arcs are set up between random pairs of vertices and

interval costs are generated randomly in such a way that $u_{ij} \leq c \;\; \forall (i,j) \in A$ and $0 \leq l_{ij} \leq u_{ij} \;\; \forall (i,j) \in A$.

***Karaşan* graphs.** The structure of these randomly generated graphs is the same as that of the benchmarks adopted in Karaşan et al. [3]. As stated in [3], they should simulate telecommunication networks. These graphs are *acyclic*, *layered*, and have a small *width*. We remind 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 the *width*, and arcs exist only from each subset to the following one in the chain. These graphs are also *complete*, 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. [3] for a more exhaustive description of this family of graphs.

***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++. The algorithm described in Dijkstra [2] has been used to solve the classic shortest path problems faced in algorithm RSP.

All the tests presented have been carried out on a computer equipped with a Pentium II 400MHz processor. The commercial solver *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. [3]). For *random* and *real* graphs s and t are selected randomly.

For each graph considered, 10 problems have been created and solved. The

Table 1
Computational results.

Graph	MIP	RSP		
	Sec	Sec	Iter	Tree size
R-500-100-0.01	4.633	1.481	37.2	5.3
R-500-100-0.001	0.578	0.599	27.8	3.0
R-500-100-0.1	14.365	2.388	44.5	3.4
R-100-100-0.01	0.079	0.046	23.5	1.8
R-900-100-0.01	25.161	5.244	148.0	4.2
R-500-10-0.01	5.033	0.553	16.8	3.7
R-500-1000-0.01	5.863	3.069	49.2	6.6
K-30-20-0.9-2	0.016	0.025	186.8	16.1
K-60-20-0.9-2	0.088	7.085	14938.8	945.4
Sottoceneri	0.246	0.115	43.6	3.7
Stuttgart	24.611	6.285	25.7	4.0

results obtained are summarized in Table 1, where columns have the following meaning:

- Graph: name of each graph considered, according to Section 4.1;
- MIP: average computation time in seconds required by CPLEX to solve the mixed integer program MIP;
- RSP: three values are reported for algorithm RSP: the average computation time in seconds (Sec), the average number of iterations ($Iter$) and the average maximum dimension of the search-tree, expressed in number of nodes ($Tree\ size$).

From Table 1 we observe that RSP is extremely effective (in terms of computation times) 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 between the methods is however very small. Another good feature of our method is that it is less sensitive than CPLEX to changes in the arc density and in the number of vertices. On the other hand, our approach appears more sensitive than CPLEX to changes in the maximum cost c. However our method remains faster.

Tests on *Karaşan* suggest that on these problems RSP is not as fast as CPLEX, and that the degradation in performance when the problem dimensions increase, is higher for RSP than for CPLEX. The not very good performance of our algorithm can be justified by observing that *Karaşan* graphs are very peculiar. They have very long paths from s to t (each path visits a number of vertices equal to the number of layers of the graph) and there is a huge number of alternative paths from s to t, since between two consecutive layers there are

$width^2$ alternative arcs. This is the worst possible situation for RSP, since the lower bound tends to be weak. Let us observe that, for those problems, our method could benefit, perhaps even more than CPLEX, from the application of the preprocessing rules described in Karaşan et al. [3], because these rules would drastically reduce the number of paths from s to t. However we have not implemented these rules because they can be used only for *acyclic*, *layered* graphs with a small *width*, and realistic transportation problems, in which we are mainly interested, do not present these peculiarities.

On graphs *Sottoceneri* and *Stuttgart* (i.e. realistic transportation problems) RSP is faster than CPLEX, especially for problems based on the *Stuttgart* graph, which is quite a large graph.

Table 1 also suggests that the number of iterations required by our algorithm is always very small, with the exception of graph *K-60-20-0.9-2*, for which the large number of iterations justifies the relatively long computation time. It is also interesting to notice how computation times depend both on the number of iterations and on problem dimensions. This happens because each iteration require more time when a problem is larger.

Finally me make some considerations about the memory requirements of RSP. In our implementation the structure representing a search-tree node d contains $lb(d)$, a few other single values and the two lists $in(d)$ and $out'(d)$, which is a subset of $out(d)$. $out'(d)$ contains the arcs of $out(d)$ with starting vertex equal to the destination vertex of the last arc in $in(d)$. The other arcs of $out(d)$ are dynamically calculated at run time, as suggested by reduction rules *R1* and *R2*. It is easy to see that $|in(d)| \leq |V|$ and $|out'(d)| \leq |V|$. This, together with the values reported in the last column of Table 1, demonstrate that the memory requirement of our algorithm is always small (with the exception of problems *K-60-20-0.9-2*).

References

[1] 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.

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

[3] O.E. Karaşan, M.Ç. Pinar, and H. Yaman. The robust shortest path problem with interval data. *Computers and Operations Research*, 2002.

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

[5] R. Montemanni and L.M. Gambardella. An algorithm for the relative robust shortest path problem with interval data. Technical Report IDSIA-05-02, IDSIA, February 2002. (ftp://ftp.idsia.ch/pub/techrep/IDSIA-05-02.pdf.gz).

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

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

[8] P. Zieliński. The computational complexity of the relative robust shortest path problem with interval data. *European Journal of Operational Research*, to appear.