An algorithm for the relative robust shortest path problem with interval data

R. Montemanni and L.M. Gambardella

IDSIA
Galleria 2
6928 Manno-Lugano
Switzerland
{roberto, luca}@idsia.ch

Technical Report IDSIA-05-02
28th February 2002

Abstract

Many real transport and telecommunications problems can be represented in mathematical terms as shortest path problems on weighted digraphs, where a fixed cost is associated with each arc. Sometimes the level of abstraction induced by this model is too high, and consequently more complex representations of reality have to be considered.

In this paper the interval data model, where an interval of costs is associated with each arc of the graph, is adopted and the concept of relative robustness is used to drive optimization.

An exact algorithm, which is able to manage large problems, is presented. This algorithm can also be used as a heuristic method, being able to find high quality solutions very quickly.

Computational results, which highlight the high performance of the approach we propose, are finally presented.

1 Introduction

The shortest path problem, where a fixed cost is associated with each arc of a graph, is a well-studied optimization problem. It can be solved in polynomial time with Dijkstra’s algorithm (see Dijkstra [2]).

In case a transport problem is considered, the road network is usually modelled through a graph by associating an arc to each road of the network. Each cost represents the travel time for the respective road (arc) and the shortest path problem is solved every time the quickest way to go from a given place of the network to another is faced. Unfortunately costs are not easy to estimate, because they depend on many factors, such as traffic conditions, the presence of accidents or traffic jams, weather conditions etc. For this reason there is the
risk to produce a very abstract model, which is not enough realistic and may not give useful information.

For this reason in the literature, some more complex models, which try to overcome the problems of the fixed costs approach have been proposed. In particular a model where a set of alternative graphs are considered in the same time (scenario model - see Yu and Yang [7] and Dias and Clímaco [1]) and a model where an interval of 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.

Having chosen this model to represent reality, we need to define the criterion which will drive optimization. The relative robustness criterion is the one we adopt. This criterion is discussed in Kouvelis and Yu [5], a book entirely devoted to robust discrete optimization, and has already been used in Karaşan et al. [3] for the shortest path problem with interval data.

In Karaşan et al. [3] a preprocessing technique (inspired by the one described in Karaşan et al. [4] for the robust spanning tree problem) is presented. This technique, which unfortunately works only for acyclic, layered graphs with a small width, permits to eliminate some arcs which will never be in an optimal path.

In this paper an exact algorithm is presented. It works for general directed graph and can be used as a heuristic method by truncating the execution before its natural end.

In Section 2 the problem we consider is formally described, while in Section 3 the algorithm we propose is discussed. In Section 4 computational results are presented. Finally conclusions can be found in Section 5.

2 Problem description

In this paper the relative robustness criterion (Kouvelis and Yu [5]) is applied to the shortest path problem 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 $u_{ij} \geq l_{ij} > 0$, is associated with each arc $(i, j) \in A$. Intervals represent ranges of possible costs. They model uncertainty about the exact value of these costs.

Our work is concerned with transport problems, and for this reason each $[l_{ij}, u_{ij}]$ is an interval of possible travel times for the road associated with arc $(i, j)$.

Accordingly to Karaşan et al. [3], we can formally describe the relative robust shortest path problem with interval data with 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 $r$.

---

1 An acyclic graph is a graph whose arcs do not form any cycle.

2 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.
Definition 3. A path \( p \) from \( s \) to \( t \) is said to be a relative robust shortest path if it has the smallest (among all paths from \( s \) to \( t \)) maximum (among all possible scenarios) robust deviation.

In the case of transport problems, a scenario can be seen as a snapshot of the network situation, while a relative robust shortest path is a path which guarantees reasonably good performance under any possible configuration of travel times over the network.

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 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. \( c_{ij}^r = u_{ij} \quad \forall (i, j) \in p \) and \( c_{kh}^r = l_{kh} \quad \forall (k, h) \notin p \).

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 \( p \).

Applying Observation 1, Karaşan et al. [3] described the problem through a mixed integer programming formulation, which is presented after the following introduction to its variables:

- \( y_{ij} \): it is 1 when arc \((i, j)\) is on the active 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 active path (defined by \( y \) variables).

\[
(RRSP) \quad \text{Min } \sum_{(i,j) \in A} u_{ij} y_{ij} - x_t \quad (1)
\]

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

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

\[
x_s = 0 \quad (4)
\]

\[
y_{ij} \in \{0, 1\} \quad \forall (i, j) \in A \quad (5)
\]

\[
x_j \geq 0 \quad \forall j \in V \quad (6)
\]

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

Yu and Yang [7], which proved that the relative robust shortest path problem with scenarios is \( \mathcal{NP} \)-hard, conjectured that also the problem with interval data is \( \mathcal{NP} \)-complete. Unfortunately no proof is presented, neither in Karaşan et al. [3], which reports the same conjecture.

3 An exact algorithm

In this section we present an exact algorithm, which can also be used as a heuristic method by stopping the execution before its natural end. We first
introduce the notation adopted in the remainder of this paper, then we describe
the idea on which the method is based and some rules which should speed up
the algorithm. Finally the complete algorithm is summarized in Section 3.4.

3.1 Notation
The notation adopted in the remainder of this paper is described.
• $l$: scenario in which $c_{ij}^l = l_{ij} \forall (i,j) \in A$;
• $u$: scenario in which $c_{ij}^u = u_{ij} \forall (i,j) \in A$;
• $P$: set of all the possible paths from $s$ to $t$ in $G$;
• $C^p(q)$: cost of path $q$ in the scenario induced by path $p$;
• $SP(p)$: shortest path from $s$ to $t$ in the scenario induced by path $p$;
• $UC(p)$: cost of path $p$ in scenario $u$;
• $LC(p)$: cost of path $p$ in scenario $l$;
• $RC(p)$: relative robustness cost of path $p$;
• $p_i$: $i$-th shortest path from $s$ to $t$ in scenario $u$;
• $LSP$: shortest path from $s$ to $t$ in scenario $l$.

3.2 Starting idea
If we examine the paths from $s$ to $t$ in non-decreasing order of $UC(p)$ (i.e.
following the order of a shortest paths ranking in scenario $u$), we are able to
provide, at each iteration, a lower bound for the cost of the relative robust path
from $s$ to $t$ of the paths not yet examined. In particular the following results
arise.

Lemma 1. $UC(p_1) \geq C^p(SP(p)) \forall p \in P$

Proof. We call $r$ the scenario induced by $p$. From the definition of scenario $u$ we have:
$$c_{ij}^u \geq c_{ij}^r \forall (i,j) \in A$$

Using (7) we can conclude that:
$$UC(p_1) = \sum_{(i,j) \in p_1} c_{ij}^u \geq \sum_{(i,j) \in p_1} c_{ij}^r \geq \sum_{(i,j) \in SP(p)} c_{ij}^r = C^p(SP(p)) \forall p \in P \quad (8)$$

The result of Lemma 1 is used in the following theorem, which provides, if
the paths are examined in non-decreasing order of $UC(p)$, a lower bound for
the relative robustness cost of not paths not yet considered.

Theorem 1. $RC(p_k) \geq UC(p_i) - UC(p_1) \forall k \geq i$
Proof. Using Lemma 1 he have:

\[ RC(p_i) = UC(p_i) - C^{pi}(SP(p_i)) \geq UC(p_i) - UC(p_1) \quad \forall p_i \in P \quad (9) \]

By definition we have:

\[ UC(p_k) \geq UC(p_i) \quad \forall k \geq i \quad (10) \]

Using (9) and (10) we have:

\[ RC(p_k) \geq UC(p_k) - UC(p_i) \geq UC(p_i) - UC(p_1) \quad \forall k \geq i \quad (11) \]

The result of Theorem 1 is used in Proposition 1, which provides an exit criterion for the exact algorithm we present in this paper.

**Proposition 1.** If \( UC(p_i) - UC(p_1) \geq \min_{j \in \{1, \ldots, i\}} \{RC(p_j)\} \) then \( \arg\min_{p_j \in \{p_1, \ldots, p_i\}} \{RC(p_j)\} \) is a relative robust shortest path.

**Proof.** Using Theorem 1 and the hypothesis we have:

\[ RC(p_k) \geq UC(p_i) - UC(p_1) \geq UC(p_i) - UC(p_1) \quad \forall k \geq i \quad (12) \]

From (12) we can conclude that \( \arg\min_{p_j \in \{p_1, \ldots, p_i\}} \{RC(p_j)\} \) is a relative robust shortest path.

Accordingly to Martins and dos Santos [6], and notwithstanding a theoretical computational complexity of \( O(k|A|) \) for their algorithm, ranking the first \( K \) shortest paths of a fixed scenario is, in practice, an easy task.

Another consideration, which follows from Observation 1, is that the evaluation of the relative robustness cost of a given path can be done by solving a shortest path problem in the scenario induced by it. This operation has a computational complexity of \( O(|V|^2) \) (see Dijkstra [2]).

The algorithm which follows from the theoretical results and considerations above works in the following way: a procedure to rank the paths of scenario \( u \) in non-decreasing order of \( UC(p) \) is run. For each path retrieved, the respective relative robustness cost is calculated by solving a shortest path problem in the scenario induced by it. The algorithm stops when the condition described in Proposition 1 is matched or when a given maximum number of paths (\( K \)) has been examined.

A more formal description of the algorithm will be given in Section 3.4, where some improvements to the basic idea described above (which will be presented in Section 3.3) are also included.

### 3.3 Improvements

Some theoretical results, which should speed up the algorithm briefly sketched near the end of Section 3.2, are presented in this section.
3.3.1 Rule A

The result described in this section will be used to calculate, in particular circumstances, the relative robustness cost of the path \( p \) under investigation without solving a shortest path problem in the scenario induced by \( p \), and consequently saving computation time.

**Theorem 2.** If \( p \) is a path from \( s \) to \( t \) and \( p \cap LSP = \emptyset \) then \( SP(p) = LSP \).

**Proof.** By definition we have:

\[
LC(LSP) = \sum_{(i,j) \in LSP} l_{ij} \leq \sum_{(i,j) \in q} l_{ij} \leq \sum_{(i,j) \in q \cap p} u_{ij} + \sum_{(i,j) \in q \setminus p} l_{ij} = \sum_{(i,j) \in q} l_{ij} + \sum_{(i,j) \in q \setminus p} l_{ij} = C(p) \quad \forall q \in P
\]  

Using (13) and the hypothesis we have:

\[
SP(p) \leq C(p) \leq SP(p)
\]

From (14) we can conclude that:

\[
SP(p) = LSP
\]

Theorem 2 states that if there is no overlapping between a path \( p \) from \( s \) to \( t \) and \( LSP \), then we can calculate the relative robustness of \( p \) without solving a shortest path problem on the graph induced by \( p \), because we already know that \( SP(p) = LSP \).

3.3.2 Rule B

The results presented in this section will be used to skip, in particular circumstances, the calculation of the relative robustness cost of the path \( p \) under investigation, saving computation time.

**Lemma 2.** If \( i > j \) and \( p_i \cap SP(p_j) \subseteq p_j \cap SP(p_j) \) then \( C^{p_i}(SP(p_i)) \leq C^{p_j}(SP(p_j)) \).

**Proof.** Using the hypothesis we have:

\[
C^{p_i}(SP(p_j)) = \sum_{(i,j) \in SP(p_j) \cap p_i} u_{ij} + \sum_{(i,j) \in SP(p_j) \setminus p_i} l_{ij} = \sum_{(i,j) \in SP(p_j) \cap p_j} u_{ij} + \sum_{(i,j) \in SP(p_j) \setminus p_j} l_{ij} + \sum_{(i,j) \in \{SP(p_j) \setminus p_j\}} (l_{ij} - u_{ij})
\]
By definition we have:
\[
\sum_{(i,j) \in \{SP(p_i) \setminus p_i\} \setminus \{SP(p_j) \setminus p_j\}} (l_{ij} - u_{ij}) \leq 0
\] (17)

From (16) and (17) we have that:
\[
C_{p_i}(SP(p_j)) \leq C_{p_j}(SP(p_j))
\] (18)

By definition we also have:
\[
C_{p_i}(SP(p_i)) \leq C_{p_i}(SP(p_j))
\] (19)

From (18) and (19) we can conclude that:
\[
C_{p_i}(SP(p_i)) \leq C_{p_j}(SP(p_j))
\] (20)

Theorem 3 uses the result of Lemma 2 to provide an important criterion, which permits to identify whether a path is dominated by another or not.

**Theorem 3.** If \( i > j \) and \( p_i \cap SP(p_j) \subseteq p_j \cap SP(p_j) \) then \( RC(p_i) \geq RC(p_j) \).

**Proof.** From Lemma 2 we have:
\[
C_{p_i}(SP(p_i)) \leq C_{p_j}(SP(p_j))
\] (21)

By definition we have:
\[
UC(p_i) \geq UC(p_j)
\] (22)

From (21) and (22) we can conclude that:
\[
RC(p_i) = UC(p_i) - C_{p_i}(SP(p_i)) \geq UC(p_j) - C_{p_j}(SP(p_j)) = RC(p_j)
\] (23)

The result of Theorem 3 is used in Proposition 2 to give a formal criterion to decide whether is possible to skip the calculation of the relative robustness cost of the path under investigation or not.

**Proposition 2.** If \( \exists p \in \{p_1, p_2, \ldots, p_{i-1}\} \) such that \( p_i \cap SP(p) \subseteq p \cap SP(p) \) then \( RC(p_i) \geq \min_{j \in \{1, \ldots, i-1\}} \{RC(p_j)\} \).

**Proof.** From Theorem 3 we have:
\[
RC(p_i) \geq RC(p)
\] (24)

By definition we have:
\[
RC(p) \geq \min_{j \in \{1, \ldots, i-1\}} \{RC(p_j)\}
\] (25)

From (24) and (25) we can conclude that:
\[
RC(p_i) \geq \min_{j \in \{1, \ldots, i-1\}} \{RC(p_j)\}
\] (26)
Proposition 2 states that if we know that $p_i$, the $i$-th shortest path in scenario $u$, is dominated (accordingly to Theorem 3) by a path $p_j$ with $j < i$, then we do not need to calculate $RC(p_i)$ (i.e., solving a shortest path problem in the scenario induced by $p_i$), because we already know that it will not improve the best result already available.

3.4 Algorithm

In this section the results described in Section 3.2 and Section 3.3 are summarized into an algorithm, whose pseudo-code is presented in Figure 1.

```plaintext
RelRobShortestPath(K)  // K = number of paths to be considered
Calculate $LSP$;
i := 0;
stop := false;
$UB := \infty$;
While($i \leq k$ and stop = false)
    Retrieve the $i$-th shortest path from $s$ to $t$ in scenario $u$;
    If($\exists j < i$, such that $p_i \cap SP(p_j) \subseteq p_j \cap SP(p_j)$) // Rule B
        If($p_i \cap LSP = \emptyset$) // Rule A
            $SP(p_i) := LSP$;
        Else // Rule A
            $SP(p_i) :=$ shortest path from $s$ to $t$ in the scenario induced by $p_i$;
        $RC(p_i) := AC(p_i) - C^{p_i}(SP(p_i))$;
        If($RC(p_i) < UB$)
            $UB := RC(p_i)$;
            $PathUB := p_i$;
        If($UB \leq AC(p_i) - AC(p_1)$)
            stop := true;
    Return $PathUB$;

Figure 1: Algorithm for the relative robust shortest path problem with interval data.

The algorithm starts by solving a shortest path problem (the algorithm described in Dijkstra [2] is used) in scenario $l$ and by initialising some variables. An iterative statement is then entered. At each iteration $i$, the $i$-th shortest path from $s$ to $t$ in scenario $u$ is retrieved (the algorithm described in Martins and dos Santos [6] is used). A test is carried out to verify whether this path can eventually improve the best relative robust path currently available. If the result of the test is positive then another condition is checked. It permits to understand whether it is necessary to solve a shortest path problem in the scenario induced by $p_i$ or not (if the problem has to be solved then the algorithm described in Dijkstra [2] is used). At this point we are able to evaluate the relative robustness cost of $p_i$, and eventually to update the best relative robust path currently available. Finally a test is carried out to see whether the exit condition is verified. Once the algorithm exits from the iterative statement, the best shortest path in terms of relative robustness, among those considered, is returned.
It is interesting to notice that if the variable stop is false when the algorithm terminates, then the path returned is a heuristic solution to the problem, otherwise it is an exact solution.

4 Computational results

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

The algorithm described in Figure 1 has been implemented in C++ and all the tests have been carried out on a computer equipped with a Pentium 4 1.5GHz processor and 256MB of memory.

In Section 4.1 the benchmarks we adopt in this paper are described. In Section 4.2 the results obtained by the algorithm are discussed.

4.1 Description of the benchmarks

The weighted graphs on which the benchmarks adopted in this paper are based are described in this section. These graphs can be divided in three families, which are presented separately in the following subsections.

4.1.1 Graph Sottoceneri

This graph models the main roads of the Sottoceneri region, which is the southern part of Canton Ticino (Switzerland). Interval costs have been chosen in order to cover the road conditions of all the different times of a typical day.

4.1.2 Karaşan graphs

The structure of these graphs is the same as that of the randomly generated benchmarks adopted in Karaşan et al. [3]. As observed in Section 1, these graphs are acyclic and layered.

Names of the graphs give some indications about their characteristics, i.e. a graph of type $K-n-c-d-w$ has $n$ vertices, all its costs are less than $c$, $d$ is, accordingly to Karaşan et al. [3], a measure of the maximum wideness of each interval cost and $w$ is the width of the graph, again accordingly to Karaşan et al. [3].

4.1.3 Random graphs

This family is composed of random graphs we have generated by setting up edges between random pairs of vertices of a given set $V$ and by assigning random interval costs to them.

Also in this case an indication of the structure of the graphs is provided by their names. A graph of type $R-n-c-\delta$ has $n$ vertices, its costs are less than $c$ and $\delta$ is an approximation for its arc density.

4.2 Results

In this section we present the results obtained on different families of benchmarks by the algorithm we propose.
Table 1: Results. Averages over 50 runs.

<table>
<thead>
<tr>
<th></th>
<th>Sotto K-90-20-0.9-2</th>
<th>K-180-20-0.9-3</th>
<th>R-7000-100-0.001</th>
<th>R-7000-100-0.0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact solutions (%)</td>
<td>82.00</td>
<td>8.00</td>
<td>0.00</td>
<td>84.00</td>
</tr>
<tr>
<td>Gap (%)</td>
<td>6.85</td>
<td>58.81</td>
<td>88.41</td>
<td>2.04</td>
</tr>
<tr>
<td>Execution time</td>
<td>17.22188</td>
<td>34.67444</td>
<td>73.64907</td>
<td>103.67913</td>
</tr>
<tr>
<td>Time last</td>
<td>0.00284</td>
<td>0.00934</td>
<td>0.12596</td>
<td>0.42028</td>
</tr>
<tr>
<td>Number of</td>
<td>23236.84</td>
<td>293654.78</td>
<td>200000.00</td>
<td>1511.12</td>
</tr>
<tr>
<td>iterations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iteration last</td>
<td>2.56</td>
<td>80.86</td>
<td>349.64</td>
<td>1.10</td>
</tr>
<tr>
<td>improvement</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For the problems based on Karaşan graphs, the starting vertex is always 0 and the destination vertex is always the last one (by definition, see Karaşan et al. [3]). For the problems based on Random graphs and on Sottoceneri the starting and destination points are selected randomly.

The maximum number of paths considered (parameter $K$ in the pseudo-code of Figure 1) has been fixed at values which push the memory requirements of the algorithm near the limits of the computer used. In particular $K$ has been fixed at 100000 for the tests on Sottoceneri, at 300000 for those on K-90-20-0.9-2, at 200000 for those on K-180-20-0.9-3 and at 5000 for those on R-7000-100-0.001 and R-7000-100-0.0001.

For each family of problems 50 runs have been considered, and the results are grouped in Table 1, where rows have the following meaning:

- **Exact solutions (%):** percentage of solutions whose optimality is confirmed by the algorithm;
- **Gap (%):** average gap between the lower bounds and the upper bounds retrieved by the algorithm;
- **Execution time:** average total execution time (in seconds);
- **Time last improvement:** average time (in seconds) of the last improvement in the heuristic solution;
- **Number of iterations:** average total number of iterations;
- **Iteration last improvement:** average iteration number of the last improvement in the heuristic solution.

Table 1 shows how the performance of the method changes when different families of problems are considered. The percentage of solutions whose optimality is confirmed is very high for problems based on Sottoceneri and Random graphs, but it is low for problems based on Karaşan graphs. The average gap between lower bounds and upper bounds remains wide for the problems based on Karaşan graphs, while it is closed for the other problems considered.
Observing the average execution times in Table 1 we can observe how our method is always extremely fast. In particular all the best solutions found are retrieved within half a second.

Another observation is about the average number of iterations required to retrieve the best heuristic solutions. It is always very small, and this suggests the existence of a correlation between the cost of a path in scenario \( u \) and its relative robustness cost.

In this sense it is interesting to report we have been able to solve to optimality, using \textit{LP Solve}\textsuperscript{3} 2.0, the mixed-integer program \textit{RRSP} for the problems based on \textit{Sottoceneri} and \textit{K-90-20-0.9-2} (the other problems have not been considered because it was impossible to solve them in less than 1 hour).

The results obtained are very encouraging because optimality of the solution found by our algorithm has been confirmed for all of the problems considered, i.e. also for the 18% of the problems based on \textit{Sottoceneri} and for the 92% of the problems based on \textit{K-90-20-0.9-2} for which the algorithm we propose was not able to confirm optimality autonomously. This suggests that the quality of the heuristic solutions provided by our method is extremely high and that, on the other hand, the lower bound described in Theorem 1 is not tight enough for the problems based on \textit{Kara\c san} graphs.

The average time required by \textit{LP Solve} to solve \textit{RRSP} has been 1.08092 seconds for the problems based on \textit{Sottoceneri} and 16.07023 seconds for those based on \textit{K-90-20-0.9-2}. This means that for these two families of problems, solving \textit{RRSP} is more convenient than running the algorithm we propose. For the other families of problems our method is however the only possible choice, because mixed-integer program solvers are not able to manage them in reasonable time.

It must finally be observed that using the graph reduction techniques described in Kara\c san et al. \cite{3}, our results may improve for problems based on \textit{Kara\c san} graphs. We have not implemented these techniques because they work for acyclic, layered graphs only, and we consider more general graphs.

The fact that our method obtains not very good results on problems based on \textit{Kara\c san} graphs is however not a big limitation, because we are mainly interested in transport problems, and the characteristics of the benchmarks adopted in Kara\c san et al. \cite{3} are extremely different from those of a road networks, having a very regular structure (they have been created to simulate telecommunication networks).

We can conclude that the method we propose is competitive as an exact algorithm for large problems, for which it is not possible to solve the mixed-integer programming formulation in reasonable time. Our approach is however very competitive as a heuristic techniques, being able to find good solutions in very short times. This characteristic, which is connected with a possible correlation between the cost of each path in scenario \( u \) and the respective relative robustness cost, suggests an alternative use of the method we propose, i.e. to run it with small values of \( K \) (see pseudo-code of Figure 1) as a heuristic algorithm.

\textsuperscript{3}\textit{LP Solve} 2.0 is available at ftp://ftp.es.ele.tue.nl/pub/lp_solve.
5 Conclusion

The relative robust shortest path problem with interval data has been studied in this paper.

After a formal description of the problem, an exact algorithm, which can be used as a heuristic method by truncating the execution before its natural end, has been described.

Computational results confirm the effectiveness of the approach we propose. It is able to guarantee optimal solutions for most of the problems considered. It also retrieves very quickly high quality heuristic solutions for the remaining benchmarks.

Acknowledgements

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

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


