4OR-Q J Oper Res (2013) 11:275–294
DOI 10.1007/s10288-013-0229-0

RESEARCH PAPER

Bicriteria path problem minimizing the cost and minimizing the number of labels

Marta Pascoal · M. Eugénia Captivo · João Clímaco · Ana Laranjeira

Received: 5 April 2012 / Revised: 29 December 2012 / Published online: 9 March 2013

Abstract We address a bicriterion path problem where each arc is assigned with a cost value and a label (such as a color). The first criterion intends to minimize the total cost of the path (the summation of its arc costs), while the second intends to get the solution with a minimal number of different labels. Since these criteria, in general, are conflicting criteria we develop an algorithm to generate the set of non-dominated paths. Computational experiments are presented and results are discussed.

Keywords Minimal cost · Minimal number of labels · Bicriteria · Shortest path

Mathematics Subject Classification (2000) 05C85 · 90C27 · 90C29

M. Pascoal (✉) · A. Laranjeira
Departamento de Matemática da FCTUC, Apartado 3008, EC Santa Cruz,
3001-501 Coimbra, Portugal
e-mail: marta@mat.uc.pt

A. Laranjeira
e-mail: mat0401@mat.uc.pt

M. Pascoal · J. Clímaco
Instituto de Engenharia de Sistemas e Computadores—Coimbra, Rua Antero de Quental,
199, 3000-033 Coimbra, Portugal

M. E. Captivo
Centro de Investigação Operacional, Faculdade de Ciências, Universidade de Lisboa,
Campo Grande, Bloco C6, 1749-016 Lisbon, Portugal
e-mail: mecaptivo@fc.ul.pt

J. Clímaco
Faculdade de Economia, Universidade de Coimbra, Avenida Dias da Silva,
165, 3004-512 Coimbra, Portugal
e-mail: jclimaco@fe.uc.pt

1 Introduction and motivation

The weight or cost is, probably, the most common objective function in network optimization problems but, besides that, other objective functions are relevant. The number of different colors (or labels) associated with the arcs in a feasible solution is one of those functions, and was introduced by Chang and Leu (1997) in the context of determining spanning trees. The goal of the minimal label spanning tree problem (MLSTP) is to find the most uniformly connected spanning subgraph of a network, assuming each edge is associated with a label. Contrarily to the minimal spanning tree problem (MSTP), for which several polynomial algorithms are known (Kruskal 1956; Prim 1957), Chang and Leu (1997) proved the NP-hardness of the MLSTP and introduced two heuristic methods, as well as an exponential time algorithm to solve it. Following up that work other approximate algorithms have been developed for the MLSTP and related problems. A literature review on these methods can be found in Consoli et al. (2006). Applications of this type of problem include telecommunications, if different labels represent different technologies or operators, and transportation, for instance if referring to different modes of transport (Van-Nes 2002).

A bicriteria version of this problem, the minimal cost–minimal number of labels spanning tree problem (MCLSTP), was studied in Clímaco et al. (2010). As a first approach to the MCLSTP the computation of all the efficient spanning trees was proposed, by means of an adaptation of the bicriteria algorithm introduced by Clímaco and Martins (1982). This algorithm aims at solving the bicriteria shortest path problem and it is based on the ranking of paths by non-decreasing order of cost. In Clímaco et al. (2010) a similar method was developed that ranks spanning trees and thus computes the efficient spanning trees with respect to cost and number of labels. However, when applied to spanning trees this becomes computationally costly for instances with more than ten colors, even for medium size networks. Therefore, a second approach consists in calculating just one efficient spanning tree for any non-dominated pair of objective values.

In this work we extend the study presented in Clímaco et al. (2010) and address the determination of shortest paths with minimal number of labels. The remainder of the paper is organized as follows. In Sect. 2 we define the minimal cost—minimal number of labels path problem (MCLPP) and propose different algorithmic approaches. The method that is first proposed generates a set of paths one for each non-dominated pair of objective values. Different implementations of this method are described, and at a second stage reoptimization techniques and cost upper bounds are used in order to enhance the initial method. Section 3 is devoted to the discussion of computational experiments split into two parts, the first involving generic random graphs and random square grids, and the second using random multi-graphs that simulate a transportation network where different means of transportation are distinguished by different arc labels, as well as on a network inspired by the transportation network of Coimbra. The last section presents concluding remarks and future research directions.

2 The minimal cost and minimal number of labels path problem

Let $(\mathcal{N}, \mathcal{A})$ be a directed network, where $\mathcal{N}$ denotes the set of n nodes, $\mathcal{A}$ denotes the set of m arcs, and where arc (i, j) is associated with the cost for using it, $c_{ij} \in \mathbb{R}$, and with one of the ℓ possible labels, l_{ij}. For convenience each label is represented by an integer in $\{1, 2, \dots, \ell\}$. Let also c and l be two functions such that $c(p) = \sum_{(i,j)\in p} c_{ij}$ denotes the cost of path p, and $l(p)$ is the number of different labels in the arcs of path p. Given an initial node, s, and a terminal node, t, $\mathcal{P}$ denotes the set of paths from s to t in $(\mathcal{N}, \mathcal{A})$.

The goal of the shortest path problem is to determine a minimal cost path between nodes s and t in $(\mathcal{N}, \mathcal{A})$. This problem has been widely studied and, when the network $(\mathcal{N}, \mathcal{A})$ does not contain cycles with a negative cost, it can be solved in polynomial time, for instance, by a labelling algorithm (Ahuja et al. 1993). Because the shortest path problem will be seen as a subproblem of the MCLPP, we assume that the networks satisfy this condition. If the subproblem is to be solved by means of Dijkstra's algorithm then the stronger assumption that the costs are non-negative should be made. When considering l as the objective function we intend to obtain the minimal number of labels path problem, the goal of which is to find a most homogeneous path between two given nodes of $(\mathcal{N}, \mathcal{A})$. Wirth (2001) proved this problem is NP-hard.

The MCLPP consists of determining solutions for the bicriterion problem:

$$\min\{c(p) : p \in \mathcal{P}\}$$
$$\min\{l(p) : p \in \mathcal{P}\},$$

that is, paths from s to t which are *good* solutions with respect to c and l. In general there is a conflict between these functions, and thus there is not a solution that is simultaneously optimal with respect to both. Instead, the literature (Steuer 1986) distinguishes two kinds of solutions, *efficient*, in the domain of the decision variables, and *non-dominated*, in the domain of the objective value vectors, defined in the following. Our goal will be to determine the set of all *non-dominated* solutions.

Since the works by Vincke (1974) and Hansen (1980) several multicriteria path problems, and in particular bicriteria path problems, have been studied. We mention a few (Bornstein et al. 2012; Clímaco and Martins 1982; Iori et al. 2010), but for a more complete survey on this subject the reader is referred to Clímaco and Pascoal (2012) and Raith and Ehrgott (2009).

A path $p \in \mathcal{P}$ is said to be efficient iff there is no $p' \in \mathcal{P}$ such that

$$c(p') \leq c(p), \quad l(p') \leq l(p)$$

and at least one of the inequalities is strict. When there is $p' \in \mathcal{P}$ such that $c(p') \leq c(p)$, $l(p') \leq l(p)$ and at least one of these inequalities is strict we say that $(c(p'), l(p'))$ dominates $(c(p), l(p))$, for some $p \in \mathcal{P}$. We intend to compute a minimal complete set of paths for the MCLPP, which is a minimal set of efficient paths the images of which cover all non-dominated images. In other words, we look for a set of efficient paths $\bar{\mathcal{P}}$ such that for any two $p, p' \in \bar{\mathcal{P}}$, $p \neq p'$, have different images, and for any non-dominated pair $(\bar{c}, \bar{\ell})$ there exists $p \in \bar{\mathcal{P}}$ so that $(c(p), l(p)) = (\bar{c}, \bar{\ell})$.

The method proposed here computes a minimal complete set of solutions by calculating the shortest path corresponding to each possible number of labels and checking its dominance. Checking for alternative efficient solutions is still possible, but it is also computationally costly, and usually the added information is not very valuable, thus computing one "representative" efficient solution for each non-dominated pair of objective values is sufficient for most applications.

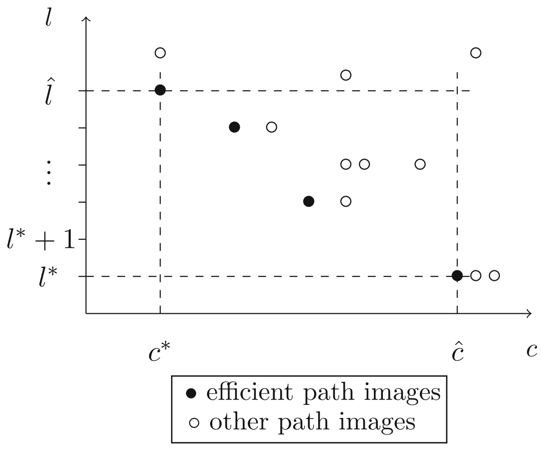


Fig. 1 MCLPP solutions

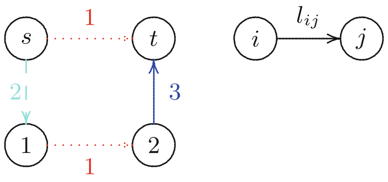


Fig. 2 Network with labeled arcs

Let l^* (c^*) be the minimal number of labels (cost value) of a path, and $\hat{l}$ ($\hat{c}$) be the maximal number of labels (cost value) of an efficient path. These values define the region of the non-dominated objective function images, that lies in $[c^*, \hat{c}] \times \{l^*, \ldots, \hat{l}\}$, see Fig. 1. In fact, there are no solutions such that $c(p) < c^*$ or $l(p) < l^*$. Besides, if $c(p) \geq \hat{c}$ and $l(p) \geq l^*$ with at least one strict inequality, then $(\hat{c}, l^*)$ dominates $(c(p), l(p))$, and, similarly if $c(p) \geq c^*$ and $l(p) \geq \hat{l}$ with at least one strict inequality, then $(c^*, \hat{l})$ dominates $(c(p), l(p))$.

As mentioned above, we calculate a shortest path corresponding to each number of labels. In fact it is easier to manipulate the network in order to fix the labels a path can contain than to add constraints to fix the path cost. Besides, the range of labels is usually smaller than the range of path costs. For this reason we look for an efficient path with each possible number of labels, as stated in Proposition 1. Before presenting this result we show that the number of labels of all paths from s to t in a network is not necessarily a complete sequence. A counter-example is given in Fig. 2, where there are only two paths from s to t in that network,

- $p = \langle s, t \rangle$, with $l(p) = 1$, and
- $q = \langle s, 1, 2, t \rangle$, with $l(q) = 3$,

however none of them has exactly two labels.

Proposition 1 *For any $k \in \{l^*, \ldots, \hat{l}\}$:*

- *either there is no path with k labels,*
- *or else there is at least a path p from s to t such that $l(p) = k$ and:*
  - *either p is efficient,*
  - *or there is an efficient path p' from s to t such that $l(p') < k$, which dominates all the paths with k labels.*

For a fixed number of labels the shortest path dominates other paths with a greater cost. Thus, by Proposition 1 it can be concluded that the set of efficient paths contains the shortest path with k labels, for some of the possible number of labels $k = l^*, \dots, \hat{l}$. Moreover, the application of a single-objective shortest path algorithm with respect to c to $(\mathcal{N}, \mathcal{A})$ provides path p^*, and thus the value c^*, the cost of p^*, and an upper bound on $\hat{l}$, given by the number of labels $l(p^*)$. However, since the minimum label path problem is NP-hard (Wirth 2001), l^* cannot be known in polynomial time. Thus, a sequence of constrained shortest path problems in networks with a specified number of labels, ranging between 1 and $l(p^*)$, are solved.

Some notation is now introduced. Let $l(E) = \{l_{ij} : (i, j) \in E\}$, for any $E \subseteq \mathcal{A}$. Then, for a fixed $k \in \{1, 2, \dots, \ell\}$, $(\mathcal{N}, \mathcal{A})_k$ defines the set of all the networks $(\mathcal{N}, A_k)$, with $A_k \subseteq \mathcal{A}$ and $|l(A_k)| = k$. We say that p is a path in $(\mathcal{N}, \mathcal{A})_k$ if p is a path in at least one of the networks in $(\mathcal{N}, \mathcal{A})_k$. The number of labels in every set $(\mathcal{N}, \mathcal{A})_k$ is bounded, therefore all its paths p satisfy $l(p) \leq k$. Denoting by p_k the shortest path from s to t in $(\mathcal{N}, \mathcal{A})_k$, as mentioned before the set $\{(c(p_k), l(p_k)) : k = l^*, \dots, \hat{l}\}$ contains all the non-dominated objective values of the MCLPP. Each path p_k can be computed by examining every subnetwork of $(\mathcal{N}, \mathcal{A})$ in $(\mathcal{N}, \mathcal{A})_k$ and, because in general l^* is unknown, doing this for $k = 1, \dots, l(p^*)$ provides a set of efficient paths with all the non-dominated objective values. Moreover, Proposition 2 can be used to reduce the number of this type of operations.

Proposition 2 *If $l(p_k) = k^* < k$, then p_k dominates every path p such that $k^* < l(p) \leq k$.*

Proof Given a path $p \in \mathcal{P}$ such that $l(p_k) = k^* < l(p) \leq k$, both p and p_k are paths in $(\mathcal{N}, \mathcal{A})_k$ so, by definition of p_k, $c(p_k) \leq c(p)$, which means that p_k dominates p. □

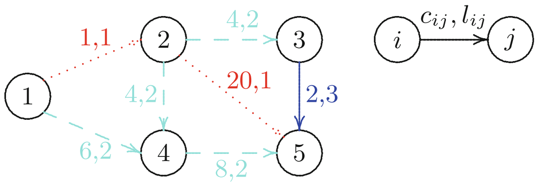
This result allows to skip the shortest path determination for some of the sets $(\mathcal{N}, \mathcal{A})_k$, whenever $l(p_k) < k$, namely for the sets $(\mathcal{N}, \mathcal{A})_r$, $l(p_k) \leq r \leq k$. This is particularly useful if k is taken by decreasing order, considering the sequence $\{(\mathcal{N}, \mathcal{A})_k\}_{k=l(p^*)}^{1}$. Algorithm 1, together with Procedure 1, summarize this method for computing a set of paths with all the non-dominated objective function values.

Algorithm 1. *Finding a minimal complete set for the MCLPP*

```
// Input: network $(\mathcal{N}, \mathcal{A})$ with arc costs and labels;
           nodes $s$ and $t$
// Output: $\bar{P}$, a minimal complete set of paths from $s$ to $t$ with respect to $c$ and $l$
// $P$ is a set that contains possible efficient paths
01 $p^* \leftarrow$ shortest path from $s$ to $t$ in $(\mathcal{N}, \mathcal{A})$
02 $P \leftarrow \{p^*\}$
03 $k \leftarrow l(p^*) - 1$
04 While $k \geq 1$ Do
05     $p_k \leftarrow$ Finding the shortest path from $s$ to $t$ in $(\mathcal{N}, \mathcal{A})_k$, as in Procedure 1
06     If $p_k$ exists Then $P \leftarrow P \cup \{p_k\}$
07     $k \leftarrow \min\{l(p_k), k\} - 1$
08 EndWhile
09 $\bar{P} \leftarrow P$ after removal of the dominated paths
```

Note that $(\mathcal{N}, \mathcal{A})$ is the only network in set $(\mathcal{N}, \mathcal{A})_\ell$, however the number of labels of p_ℓ, i.e. p^*, is not necessarily $\hat{l}$, but rather an upper bound on that value. The value $\hat{l}$

Fig. 3 Network $(\mathcal{N}, \mathcal{A})$



coincides with the number of labels of the lexicographically minimal path, therefore either $\hat{l} = l(p_\ell)$, and thus there are no paths with cost c^* and less than $\hat{l}$ labels, or there is another path $\bar{p}$ such that $l(\bar{p}) = \hat{l} < l(p_\ell)$, and thus $c(\bar{p}) = c^*$ (otherwise p_ℓ dominates $\bar{p}$). Nevertheless $\hat{l}$ is obtained after a path with a cost greater than c^* is computed or when the algorithm exits the `While` loop, which means that there are no more paths to be found. Moreover, instead of filtering the dominated paths that are stored in P only at the end of the algorithm, these paths can be filtered within the loop. The constrained problem at line 05 is solved by computing the shortest path problem from s to t in every subnetwork of $(\mathcal{N}, \mathcal{A})_k$, as outlined in Procedure 1.

```
Procedure 1. Finding the shortest path from $s$ to $t$ in $(\mathcal{N}, \mathcal{A})_k$
    // Input: network $(\mathcal{N}, \mathcal{A})_k$ with arc costs and labels;
              nodes $s$ and $t$
    // Output: $p_k$, a shortest path from $s$ to $t$ in the set $(\mathcal{N}, \mathcal{A})_k$
    01 $BestCost \leftarrow +\infty$
    02 For every $(\mathcal{N}, A_k)$ subnetwork of $(\mathcal{N}, \mathcal{A})_k$ Do
    03     $p \leftarrow$ shortest path from $s$ to $t$ in $(\mathcal{N}, A_k)$
    04     If $c(p) < BestCost$ Then
    05         $BestCost \leftarrow c(p)$
    06         $p_k \leftarrow p$
    07     EndIf
    08 EndFor
```

As an illustration of Algorithm 1 we consider its application to the network represented in Fig. 3 with $s = 1$ and $t = 5$, which is split into three steps corresponding to the sets of networks $(\mathcal{N}, \mathcal{A})_k$, with $k = 3, 2, 1$. Table 1 shows a scheme of the paths that are calculated throughout the procedure.

In Procedure 1 it is assumed that for each number of labels k all the subnetworks of $(\mathcal{N}, \mathcal{A})$ with their set of arcs restricted to contain exactly k labels are considered. One

Table 1 Paths calculated by Procedure 1 applied to the network in Fig. 3

Step	$(\mathcal{N}, \mathcal{A})_k$	$l(\mathcal{A}_k)$	p	$(c(p), l(p))$	p_k	P
1	$(\mathcal{N}, \mathcal{A})_3$	$\{1, 2, 3\}$	$\{\langle 1, 2, 3, 5\rangle\}$	(7,3)	$\{\langle 1, 2, 3, 5\rangle\}$	$\{\langle 1, 2, 3, 5\rangle\}$
2	$(\mathcal{N}, \mathcal{A})_2$	$\{1, 2\}$	$\langle 1, 2, 4, 5\rangle$	(13,2)		
		$\{1, 3\}$	$\langle 1, 2, 5\rangle$	(21,1)		
		$\{2, 3\}$	$\langle 1, 4, 5\rangle$	(14,1)	$\langle 1, 2, 4, 5\rangle$	$\{\langle 1, 2, 3, 5\rangle, \langle 1, 2, 4, 5\rangle\}$
3	$(\mathcal{N}, \mathcal{A})_1$	$\{1\}$	$\langle 1, 2, 5\rangle$	(21,1)		
		$\{2\}$	$\langle 1, 4, 5\rangle$	(14,1)	$\langle 1, 4, 5\rangle$	
		$\{3\}$	–	–		$\{\langle 1, 2, 3, 5\rangle, \langle 1, 2, 4, 5\rangle, \langle 1, 4, 5\rangle\}$

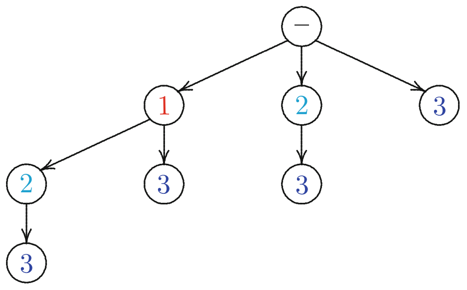


Fig. 4 Search tree of a network with the labels 1, 2 and 3

way to implement this is to enumerate all the k elements combinations of the ℓ labels, and mark all the arcs labeled with other values as non-available. If done by increasing order of the number of labels, the construction of these combinations can be aided by a search tree, with a root with no labels associated, and such that each child node contains a new label that is added to its ancestors' labels. Figure 4 shows an example of such a tree, if the arc labels are 1, 2 and 3.

Given a total number of ℓ labels, retrieving all the paths from the root to a level k of the search tree provides all the combinations of those labels with k elements. If the tree is constructed following a breadth first search (BFS) policy, each of its levels, for instance k, can support the shortest path determination in $(\mathcal{N}, \mathcal{A})_k$. Algorithm 2 shows the pseudo-code of an alternative to Algorithm 1 that considers an increasing sequence of the number of labels provided by a BFS tree, the nodes of which result from adding the arcs with a certain label to the shortest path associated with its father. In order to implement the BFS, the set X manages the tree nodes by being manipulated in a first in first out (FIFO) manner. The nodes in each level correspond to all combinations of k elements of $\{1, \ldots, \ell\}$, as depicted in Fig. 4.

Algorithm 2. *Finding a minimal complete set for the MCLPP with a BFS tree*

```
// Input: network $(\mathcal{N}, \mathcal{A})$ with arc costs and labels;
          nodes $s$ and $t$
// Output: $\bar{P}$, a minimal complete set of paths from $s$ to $t$ with respect to $c$ and $l$
// $P$ is a set that contains possible efficient paths
// $C$ is an auxiliary set manipulated in order to obtain all the label combinations
01 $p^* \leftarrow$ shortest path in $(\mathcal{N}, \mathcal{A})$
02 $X \leftarrow \{\emptyset\}$
03 $P \leftarrow \{p^*\}$
04 $x \leftarrow 0$
05 While $x \leq l(p^*)$ and $X \neq \emptyset$ Do
06    $C \leftarrow$ first element in $X$
07    $X \leftarrow X - \{C\}$
08    $x \leftarrow |C|$
09    $y \leftarrow$ greatest label in $C$ (or 0 if $C = \emptyset$)
10    For $k = y + 1, \ldots, \ell$ Do
11       Insert $C \cup \{k\}$ at the end of $X$
12       $A' \leftarrow$ {arcs in $\mathcal{A}$ with a label in $C \cup \{k\}$}
13       $p \leftarrow$ shortest path in $(\mathcal{N}, A')$
14       If $p$ exists Then $P \leftarrow P \cup \{p\}$
15    EndFor
16 EndWhile
17 $\bar{P} \leftarrow P$ after removal of the dominated paths
```

The previous methods compute the shortest path in every subnetwork of $(\mathcal{N}, \mathcal{A})$ obtained by making combinations of the original ℓ labels, which means that the number of operations performed by these methods is proportional to $\sum_{k=1}^{\ell} \binom{\ell}{k}$, and thus it increases quickly with ℓ. However, two aspects can be taken into account when aiming to reduce that number.

Reoptimization of paths Many of the shortest path problems that have to be solved are similar, only the set of arcs may change by including or excluding arcs with a certain label at a time. The use of this information depends on the algorithm implementation. Although this is not clear for Algorithm 1, it is easy to see that, when using Algorithm 2, the problem, and the path, associated with a node in the search tree differs from its ancestor because the arcs with a new label can be used and become part of the solution. This allows to replace many of the shortest path problems with the reoptimization of a shortest path after the inclusion of a set of new arcs in the network. The resulting method is summarized in Algorithm 3.

Algorithm 3. *Finding a minimal complete set for the MCLPP with a BFS tree and path reoptimization*

```
// Input: network $(\mathcal{N},\mathcal{A})$ with arc costs and labels;
          nodes $s$ and $t$
// Output: $\bar{P}$, a minimal complete set of paths from $s$ to $t$ with respect to $c$ and $l$
// $P$ is a set that contains possible efficient paths
// $C$ is an auxiliary set manipulated in order to obtain all the label combinations
01 $p^* \leftarrow$ shortest path in $(\mathcal{N},\mathcal{A})$
02 $X \leftarrow \{\emptyset\}$
03 $P \leftarrow \{p^*\}$
04 $x \leftarrow 0$
05 While $x \leq l(p^*)$ and $X \neq \emptyset$ Do
06     $C \leftarrow$ first element in $X$
07     $X \leftarrow X - \{C\}$
08     $x \leftarrow |C|$
09     $y \leftarrow$ greatest label in $C$ (or 0 if $C = \emptyset$)
10     $A' \leftarrow$ {arcs in $\mathcal{A}$ with a label in $C$}
11     $q \leftarrow$ shortest path in $(\mathcal{N}, A')$
12     For $k = y+1,\dots,\ell$ Do
13         Insert $C \cup \{k\}$ at the end of $X$
14         $p \leftarrow$ shortest path obtained from inserting in $q$ the arcs in $A'$ with the label $k$ // Procedure 2
15         If $p$ exists Then $P \leftarrow P \cup \{p\}$
16     EndFor
17 EndWhile
18 $\bar{P} \leftarrow P$ after removal of the dominated paths
```

Algorithm 3 uses Procedure 2 as the reoptimization method. This is based on a label correcting approach—see Ahuja et al. (1993)—and assumes that a shortest path tree rooted at s is already known as the solution to a previous subproblem. The procedure then works in two phases. First the arcs labeled with the new inserted color are analyzed, and if they allow the improvement of the label of a node this node is inserted back to the list of temporary node labels. Second the labelling procedure proceeds as usual, trying to improve the current paths and using the new color as well as the previous till all node labels are definite.

Procedure 2. *Shortest path obtained from the insertion of a set of arcs with label k in a path p*

```
// Input: network $(\mathcal{N}, A')$;
        set of arcs $\mathcal{A}$;
        shortest path from $s$ to any node $i \in \mathcal{N}$, defined by $\pi_i$, its cost, and $\xi_i$,
        the node that precedes $i$ in that path;
        new arc label $k$
// Output: shortest path after the insertion of the arcs with label $k$ in $p$
01 $L \leftarrow \emptyset$
02 For any $i \in \mathcal{N}$ such that $l_{ij} = k$ Do
03     For every $(i,j) \in \mathcal{A}$ such that $l_{ij} = k$ Do
04         If $\pi_i + c_{ij} < \pi_j$ Then
05             $\pi_j \leftarrow \pi_i + c_{ij}$
06             $\xi_j \leftarrow i$
07             $L \leftarrow L \cup \{j\}$
08         EndIf
09     EndFor
10 EndFor
11 While $L \neq \emptyset$ Do
12     $i \leftarrow$ node in $L$
13     $L \leftarrow L - \{i\}$
14     For every $(i,j) \in \mathcal{A}$ such that $(i,j) \in A'$ or $l_{ij} = k$ Do
15         If $\pi_i + c_{ij} < \pi_j$ Then
16             $\pi_j \leftarrow \pi_i + c_{ij}$
17             $\xi_j \leftarrow i$
18             $L \leftarrow L \cup \{j\}$
19         EndIf
20     EndFor
21 EndWhile
```

2.1 Enhancing paths computation

In the shortest path problems that are solved, many repeated paths may be obtained. For instance, in the example above the path $\langle 1, 4, 5\rangle$ is calculated twice, first as the shortest path in one of the networks in $(\mathcal{N}, \mathcal{A})_2$, and second in one of the networks in $(\mathcal{N}, \mathcal{A})_1$. However, because that path is not optimal for $(\mathcal{N}, \mathcal{A})_2$ it is only stored the second time it is computed. Proposition 3 is a simple result which shows that this information can be managed in order to limit the labels produced in intermediate shortest path problems. According to this result the cost of any path computed at a certain step of the algorithm can be used as a cost upper bound of forthcoming permanent labels. In the case above, in step 2 the objective values of $\langle 1, 4, 5\rangle$ are calculated $(14, 1)$, therefore in step 3 the path $\langle 1, 2, 5\rangle$, with image $(21, 1)$, does not have to be computed till the end.

Proposition 3 *Let $p \in \mathcal{P}$, then $c(p)$ is an upper bound on the cost of the efficient paths with k labels, for any $k \geq l(p)$.*

Proof Let q be an efficient path with $l(q) = k \geq l(p)$ and assume, by contradiction, that $c(q) > c(p)$. Then p dominates q, and q cannot be efficient. □

Corollary 1 *Let p^* denote the shortest path in $(\mathcal{N}, \mathcal{A})$, and p_k denote the shortest path in $(\mathcal{N}, \mathcal{A})_k$, that is, a path such that $c(p_k) \leq c(p)$ for any p which satisfies $l(p) \leq k$, $k = l(p^*), \ldots, 1$. Then $\{(c(p_k), l(p_k)) : k = l(p^*), \ldots, 1\}$ contains all the non-dominated pairs of objective values.*

Table 2 Paths calculated by Algorithm 1 using the array $CostUB$ applied to the network in Fig. 3

Step	$CostUB$	$(\mathcal{N},\mathcal{A})_k$	$l(\mathcal{A}_k)$	p	$(c(p),l(p))$	p_k	P
1	$[\infty,\infty,7]$	$(\mathcal{N},\mathcal{A})_3$	$\{1,2,3\}$	$\langle 1,2,3,5\rangle$	(7,3)	$\langle 1,2,3,5\rangle$	$\{\langle 1,2,3,5\rangle\}$
2	$[\infty,13,7]$	$(\mathcal{N},\mathcal{A})_2$	$\{1,2\}$	$\langle 1,2,4,5\rangle$	(13,2)	$\langle 1,2,4,5\rangle$	$\{\langle 1,2,3,5\rangle,\langle 1,2,4,5\rangle\}$
3	[21,13,7]	$(\mathcal{N},\mathcal{A})_1$	$\{1\}$	$\langle 1,2,5\rangle$	(21,1)		
	[14,13,7]	$(\mathcal{N},\mathcal{A})_2$	$\{2\}$	$\langle 1,4,5\rangle$	(14,1)	$\langle 1,4,5\rangle$	
			$\{3\}$	–	–		$\{\langle 1,2,3,5\rangle,\langle 1,2,4,5\rangle,\langle 1,4,5\rangle\}$

Proof Let $\bar{p}$ be an efficient path and assume $(l(\bar{p}),c(\bar{p}))\notin\{(c(p_k),l(p_k)) : k = l(p^*),\dots,1\}$. Two cases should be considered,

1. $l(\bar{p}) = l(p_k)$ for some $k = l(p^*),\dots,1$,
2. $l(\bar{p}) \neq l(p_k)$ for any $k = l(p^*),\dots,1$.

In case 1. $c(\bar{p}) \geq c(p_k)$, because by definition p_k is the shortest path in $(\mathcal{N},\mathcal{A})_k$, and $l(\bar{p}) = l(p_k)$. This means that p_k dominates $\bar{p}$ and thus $\bar{p}$ is not efficient.

In case 2. take $k = l(\bar{p})$. Then p_k is such that $c(p_k) \leq c(\bar{p})$ and $l(p_k) \leq l(\bar{p})$, with $(c(p_k),l(p_k)) \neq (c(\bar{p}),l(\bar{p}))$, therefore p_k dominates $\bar{p}$ and, again, $\bar{p}$ is not efficient. □

These results can be incorporated in the algorithm by storing the best cost of the computed paths with k labels, $k = l(p^*),\dots,1$. If the shortest path problems are solved by means of a labelling algorithm (Ahuja et al. 1993), the stored upper bounds can be used to prevent the creation of labels with a cost that exceeds the upper bound values. In order to make use of this result, the previous algorithms can store an upper bound of the cost for any number of labels by including an $l(p^*)$-components array, $CostUB$, initialized with $CostUB_k = +\infty$, $k = 1,\dots,l(p^*)$. Whenever a new path, p, is computed in a set of networks with k labels, $CostUB$ is updated as

$$CostUB_r \leftarrow \min\{CostUB_r, c(p)\},\ r = l(p),\dots,k.$$

Such values can be used to restrict the node labels when computing the shortest path in a network $(\mathcal{N}, A_k)$, $k \in \{1,\dots,\hat{l}\}$.

Reducing the number of path labels in the original approach reduces the number of calculated paths as well. In particular, considering a fixed number of labels it is less likely that the computation of high cost shortest paths is completed. For instance, the application of Algorithm 1 modified by including the array $CostUB$ as described results in the steps in Table 2. After obtaining the path $p = \langle 1,2,4,5\rangle$ such that $c(p) = 13$ and $l(p) = 2$ for $(\mathcal{N},\mathcal{A})_2$, the paths $\langle 1,2,5\rangle$ and $\langle 1,4,5\rangle$ are no longer generated in step 2. However, because they have only one label they are still generated along step 3. Generally speaking, after a path p^* is obtained in $(\mathcal{N},\mathcal{A})_k$ no other path q in the same set of networks and such that $c(q) \geq c(p^*)$ is considered, regardless of its number of labels. However, such a path q can be efficient if $l(q) < l(p^*)$, and thus q is still eligible for some set of networks $(\mathcal{N},\mathcal{A})_r$, with $l(q) \leq r < l(p^*)$.

In terms of implementation let us assume a labelling algorithm is used to solve each shortest path problem, and let π_i denote the cost of a path from s to i throughout that algorithm, for any $i \in \mathcal{N}$. In a regular run of a labelling algorithm the label of a node j is updated if there is a node i such that $(i, j) \in \mathcal{A}$ and $\pi_i + c_{ij} < \pi_j$. When considering a network in $(\mathcal{N}, \mathcal{A})_k$, $k = 1, \dots, \hat{l}$, and taking the upper bound on the paths cost into account the value π_j is only updated if

$$\pi_i + c_{ij} < \pi_j \text{ and } \pi_i + c_{ij} \leq CostUB_k$$

is satisfied. When applying Procedure 1 several paths p are determined until p_k is known, so their costs can be used to update $CostUB_{l(p)}$ if $c(p) < CostUB_{l(p)}$. In that case the path p should be stored, besides paths p_k.

Tightening upper bounds The above path costs can still be strengthened if combined with information about the cost of the best paths from intermediate nodes to t. Let us assume that the shortest path from any node $i \in \mathcal{N}$ to t in $(\mathcal{N}, \mathcal{A})$ is obtained at an early stage of the algorithm [which can be done by means of a labelling method (Ahuja et al. 1993)]. Let $\mathcal{T}_t$ denote the tree, rooted at t, containing all those paths, and π_i^t denote the cost of the path from i to t in that tree. $\mathcal{T}_t$ provides lower bounds for the cost of paths, which can be used to prevent useless node labels by restricting some node j's labelling to the cases which satisfy

$$\pi_i + c_{ij} < \pi_j \text{ and } \pi_i + c_{ij} + \pi_j^t \leq CostUB_k,$$

on some network in $(\mathcal{N}, \mathcal{A})_k$, $k = 1, \dots, \hat{l}$. In order to be able to use this cost lower bounds and to discard some node labels without penalizing too much the number of performed operations, we propose $\mathcal{T}_t$ is calculated only once, prior to the `While-Do` loop in Algorithm 3, with respect to the original network $(\mathcal{N}, \mathcal{A})$.

3 Computational experiments

Empirical experiments were made in order to evaluate some implementations of the presented methods. The tests ran on a Dual Core AMD Opteron at 2 GHz, with 4 Gb of RAM. The following codes, written in C language and compiled with the option `-O3`, were implemented:

- `V1`: a version of Algorithm 1. `V1` constructs iteratively all the combinations with k elements of the ℓ labels, for given values of ℓ and k;
- `V2`: an improved version of `V1` enhanced with comparing the computed path costs to the previously determined;
- `V3`: an improved version of `V2` enhanced with the information on the minimum cost of a path from any node to t, computed once at the beginning of the algorithm;
- `BFS1`: a version of Algorithm 2, using a BFS tree. The combinations with k elements of ℓ labels are obtained by adding one label at a time to a previous combination with less elements;

Table 3 Mean value of $|\bar{P}|$ in random graphs

m	$n \times 5$			$n \times 10$			$n \times 20$		
ℓ	5	10	20	5	10	20	5	10	20
$n = 100$	2.667	2.933	3.000	2.733	2.867	2.800	3.067	3.600	3.667
$n = 200$	2.333	2.667	2.467	3.200	3.533	3.933	3.200	3.667	3.333
$n = 300$	2.800	3.133	2.733	3.400	3.600	3.600	3.267	4.067	3.733
$n = 400$	2.267	2.600	2.533	3.600	4.000	4.200	3.733	4.933	4.667
$n = 500$	2.733	2.867	2.867	3.467	3.667	4.067	3.133	3.867	4.467
$n = 600$	2.933	3.933	3.933	3.667	4.467	4.933	3.400	4.333	4.133
$n = 700$	2.333	3.133	2.867	4.067	4.400	4.933	3.667	4.867	4.667
$n = 800$	2.733	3.000	2.467	3.267	3.533	3.733	3.733	4.733	4.867
$n = 900$	3.067	3.400	3.133	3.400	3.467	4.067	3.800	5.200	5.333
$n = 1{,}000$	2.867	3.467	3.067	3.800	3.867	4.200	3.400	4.400	4.467

- `BFS2`: a variant of `BFS1`, reoptimizing shortest paths in every level of the search tree, as in Algorithm 3.

The single-objective shortest path problems were solved by a label correcting algorithm with the set of temporary labels managed as a FIFO list.

Random graphs In a first set of tests we considered thirty randomly generated graph instances of each dimension, with $n = 100, 200, \ldots, 1{,}000$, $m = 5n, 10n, 20n$, and $\ell = 5, 10, 20$. The source and the destination nodes, s and t, were chosen randomly, the arc costs and arc labels were uniformly generated in $\{1, 2, \ldots, 100\}$ and in $\{1, \ldots, \ell\}$, respectively. It was assumed that there are no parallel arcs nor self-loops.

Let $|\bar{P}|$ denote the cardinality of the minimal complete set of paths for the MCLPP. Table 3 summarizes the mean values of $|\bar{P}|$ obtained for those instances. That number depends on the number of arc labels, ℓ, which is the maximum value for the cardinality of the minimal complete set for the MCLPP. According to Table 3 the mean cardinality of the minimal complete set for the MCLPP ranged between 2.2 and 5.3, moreover this number usually grows with the average node degree of the graphs (Fig. 8).

Table 4 presents running times, in seconds, on this set of tests, whereas Fig. 5 shows the ratios with respect to code `V1`. As expected, the highest increase in the execution times is due to the increase on the number of different labels in the graphs, the times vary from some milliseconds for instances with five labels to tens of seconds for instances with twenty labels. The programs also become slower when the number of nodes or the average node degree is larger, although this performance is not always stable.

The graphics show that, almost always the versions using the search tree (`BFS1` and `BFS2`) are worse than `V2` and `V3`, and that the gap is bigger when ℓ is bigger too. Moreover, the worst of the codes based on the search tree, `BFS1`, is only better than `V1` for small and sparse instances with $m = 5n$, $10n$ and $\ell = 5$. `BFS2` behaves better than `BFS1`, it always outperforms `V1` and it also outperforms `V2` and `V3` for small problems with few nodes and $\ell = 5$.

Table 4 Mean CPU times (s) in random graphs

m	$n \times 5$			$n \times 10$			$n \times 20$		
ℓ	5	10	20	5	10	20	5	10	20
V1									
$n = 100$	0.000	0.001	0.175	0.000	0.003	0.148	0.000	0.006	0.174
$n = 200$	0.000	0.002	0.264	0.001	0.010	2.280	0.002	0.019	1.267
$n = 300$	0.000	0.005	0.583	0.002	0.016	1.485	0.003	0.044	4.226
$n = 400$	0.001	0.008	0.524	0.002	0.036	3.420	0.005	0.083	11.366
$n = 500$	0.001	0.015	1.228	0.003	0.049	3.897	0.006	0.069	8.006
$n = 600$	0.002	0.023	2.208	0.004	0.070	9.434	0.007	0.101	7.441
$n = 700$	0.002	0.016	1.805	0.005	0.080	16.313	0.009	0.136	19.176
$n = 800$	0.002	0.024	1.076	0.005	0.063	8.856	0.009	0.167	25.548
$n = 900$	0.002	0.028	4.371	0.006	0.068	12.575	0.011	0.220	40.890
$n = 1{,}000$	0.002	0.039	3.300	0.006	0.087	14.646	0.010	0.189	28.727
V2									
$n = 100$	0.000	0.000	0.039	0.000	0.000	0.021	0.000	0.001	0.007
$n = 200$	0.000	0.001	0.051	0.000	0.002	0.221	0.001	0.002	0.038
$n = 300$	0.000	0.001	0.146	0.007	0.002	0.045	0.001	0.004	0.443
$n = 400$	0.001	0.001	0.053	0.001	0.003	0.169	0.001	0.009	0.802
$n = 500$	0.001	0.002	0.118	0.001	0.003	0.169	0.002	0.003	0.177
$n = 600$	0.001	0.003	0.283	0.001	0.006	0.304	0.002	0.007	0.129
$n = 700$	0.001	0.002	0.188	0.001	0.009	0.479	0.002	0.016	0.711
$n = 800$	0.001	0.003	0.101	0.001	0.007	0.994	0.003	0.013	0.422
$n = 900$	0.001	0.005	0.312	0.002	0.005	0.380	0.003	0.016	1.100
$n = 1{,}000$	0.001	0.006	0.360	0.002	0.012	0.770	0.003	0.015	0.427
V3									
$n = 100$	0.000	0.000	0.018	0.000	0.000	0.004	0.000	0.000	0.005
$n = 200$	0.000	0.000	0.023	0.000	0.001	0.073	0.001	0.001	0.018
$n = 300$	0.000	0.000	0.050	0.001	0.001	0.023	0.001	0.002	0.067
$n = 400$	0.000	0.001	0.035	0.001	0.001	0.072	0.001	0.004	0.149
$n = 500$	0.001	0.001	0.069	0.001	0.001	0.086	0.001	0.002	0.075
$n = 600$	0.000	0.001	0.119	0.001	0.002	0.137	0.002	0.002	0.061
$n = 700$	0.001	0.001	0.093	0.001	0.003	0.193	0.002	0.008	0.262
$n = 800$	0.001	0.001	0.064	0.001	0.003	0.197	0.002	0.004	0.142
$n = 900$	0.001	0.003	0.176	0.002	0.003	0.162	0.003	0.007	0.331
$n = 1{,}000$	0.001	0.002	0.147	0.002	0.004	0.235	0.003	0.008	0.220
BFS1									
$n = 100$	0.000	0.002	0.304	0.000	0.005	0.212	0.001	0.009	0.214
$n = 200$	0.000	0.006	0.566	0.001	0.023	3.518	0.002	0.039	1.953
$n = 300$	0.000	0.013	1.486	0.002	0.032	1.710	0.004	0.074	7.473
$n = 400$	0.001	0.013	0.893	0.003	0.061	6.010	0.007	0.144	22.661

Table 4 continued

m	$n \times 5$			$n \times 10$			$n \times 20$		
ℓ	5	10	20	5	10	20	5	10	20
$n = 500$	0.001	0.021	1.890	0.004	0.061	5.613	0.008	0.094	10.678
$n = 600$	0.003	0.044	5.128	0.006	0.121	14.209	0.012	0.171	11.224
$n = 700$	0.003	0.031	3.799	0.008	0.150	21.520	0.017	0.280	39.780
$n = 800$	0.003	0.039	1.711	0.007	0.108	26.397	0.018	0.288	28.694
$n = 900$	0.003	0.062	8.240	0.008	0.111	15.311	0.021	0.391	70.836
$n = 1{,}000$	0.003	0.070	7.093	0.012	0.213	30.315	0.022	0.342	39.229
`BFS2`									
$n = 100$	0.000	0.000	0.061	0.000	0.000	0.030	0.000	0.001	0.017
$n = 200$	0.000	0.001	0.097	0.000	0.004	0.386	0.000	0.003	0.065
$n = 300$	0.000	0.002	0.241	0.001	0.003	0.076	0.001	0.008	0.693
$n = 400$	0.000	0.002	0.078	0.001	0.005	0.262	0.002	0.014	1.307
$n = 500$	0.000	0.002	0.162	0.001	0.005	0.291	0.002	0.006	0.398
$n = 600$	0.001	0.004	0.461	0.001	0.010	0.550	0.003	0.010	0.224
$n = 700$	0.001	0.003	0.296	0.002	0.014	0.694	0.004	0.022	1.505
$n = 800$	0.001	0.004	0.137	0.002	0.010	1.797	0.005	0.021	0.773
$n = 900$	0.001	0.009	0.465	0.003	0.009	0.992	0.006	0.027	1.564
$n = 1{,}000$	0.001	0.009	0.720	0.003	0.018	1.389	0.006	0.021	0.522

Among the variants of Algorithm 1, `V1`, `V2` and `V3`, the latter is generally the most efficient and the first the least efficient. The gap is bigger when ℓ is bigger too. Both the comparison of the path costs to previous solution costs and the inclusion of upper bounds to limit the number of labels produced during the shortest path computation have a big impact on the program running times, and `V2` and `V3` CPU times varied from some milliseconds only to 1.100 and 0.331 s, respectively. Similarly to what was observed for `V1`, the performance of `V2` and `V3` is highly dependent on the number of labels and, in general, it is better for sparser than for denser graphs. However, these two variants presented a performance less steady than `V1`'s.

Grid graphs The second set of test instances contains square grid graphs of the form $[1, q] \times [1, q]$, where $q = 10, 20, 30, 40, 50$. The nodes in the graph correspond to the intersection between horizontal and vertical lines along the grid. The horizontal and vertical lines, together with the diagonal, represent arcs that link each node to all its neighbours in the grid. Such a grid, depicted in Fig. 6, has $n = q^2$ nodes and $m = 4(2q - 1)(q - 1)$ arcs. The source node is the bottom border vertex of the grid, whereas the top border is the destination. Like in the case of random instances, arc costs were uniformly generated in $\{1, 2, \ldots, 100\}$ and arc labels in $\{1, \ldots, \ell\}$. The results of these tests are summarized in Tables 5 and 6, and in Fig. 7.

For these instances the mean value of $|\bar{P}|$ is greater than for the random. The mean value varies between 2.1 and 15.7. Therefore, the CPU times are greater for grid graphs too, between a few milliseconds and 2,430.503 s with the slowest code, `BFS1`, and

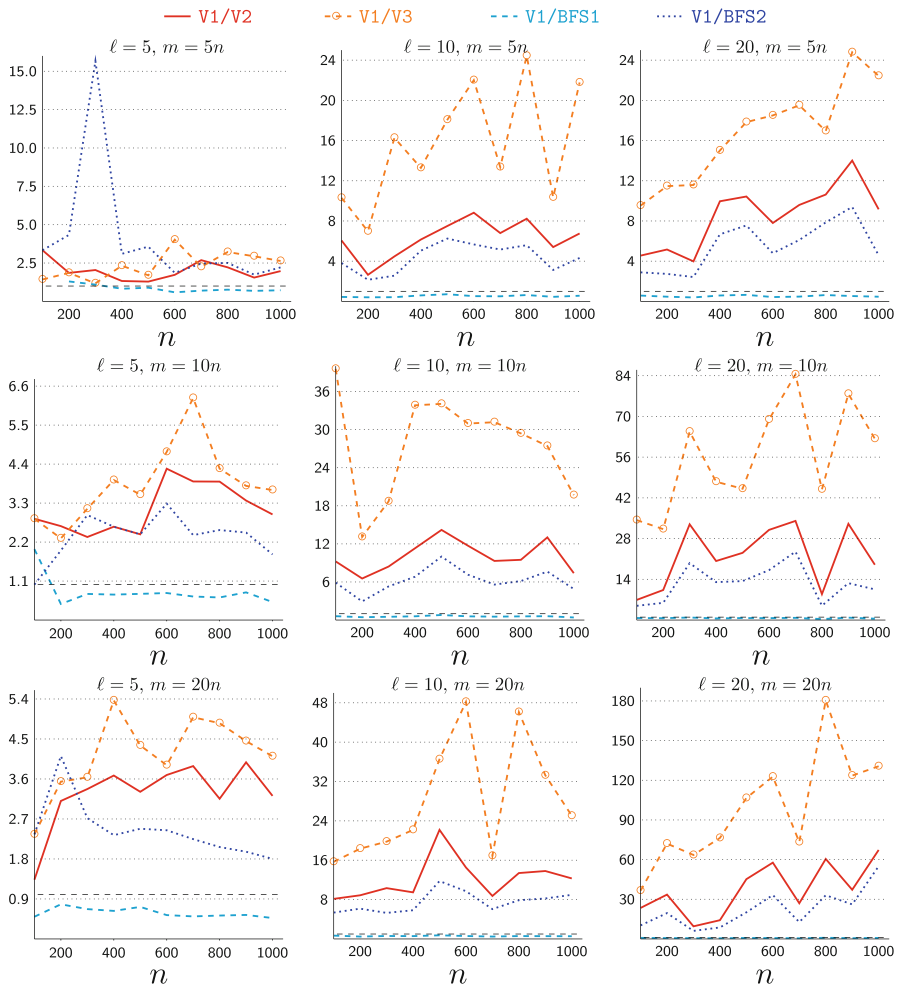


Fig. 5 Mean CPU times ratio for random graphs with respect to `V1`

between a few milliseconds and 172.201 s with the fastest, `V3`. For this set of tests the performance of `V2`, `V3`, `BFS1` and `BFS2` compared to `V1` was worse than for the random set of graphs. Although the relative behavior of the codes is similar, now the versions based on a search tree are worse or equivalent to `V1`, whereas `V2` is slightly faster. The advantage of `V3` is always clear, although it seems to become smaller when ℓ increases.

Multi-modal graphs Another set of tests ran on thirty random instances with multi-graphs simulating networks with several means of transportation. The parallel arcs in the graph are identified with different indexes, whereas the transportation modes are distinguished by different labels. The multi-graphs result from overlapping graphs, each corresponding to a mean of transportation. First, instances with three types of

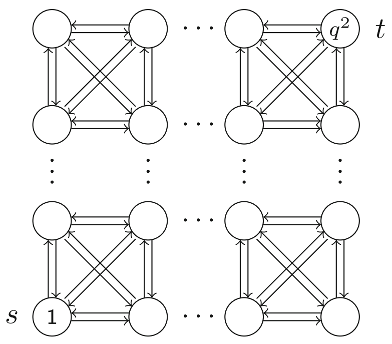


Fig. 6 Grid graph

Table 5 Mean value of $|\bar{P}|$ in grid graphs

ℓ	5	10	20
10 × 10	2.100	4.633	5.400
20 × 20	2.133	6.400	6.233
30 × 30	3.000	7.933	10.367
40 × 40	3.000	8.000	13.200
50 × 50	4.000	8.000	15.700

Table 6 Mean CPU times (s) in grid graphs

ℓ	V1			V2			V3		
	5	10	20	5	10	20	5	10	20
10 × 10	0.000	0.007	5.255	0.000	0.006	2.670	0.000	0.000	0.277
20 × 20	0.002	0.078	29.655	0.003	0.052	36.257	0.001	0.010	3.009
30 × 30	0.008	0.269	170.995	0.007	0.220	164.273	0.004	0.053	17.537
40 × 40	0.020	0.647	550.173	0.020	0.571	451.215	0.011	0.161	59.621
50 × 50	0.040	1.338	1,312.755	0.041	1.223	1,029.820	0.024	0.400	172.201
	BFS1			BFS2					
10 × 10	0.000	0.019	10.451	0.000	0.008	3.346			
20 × 20	0.003	0.147	135.872	0.003	0.069	40.493			
30 × 30	0.016	0.530	480.474	0.012	0.293	179.125			
40 × 40	0.042	1.312	1,163.854	0.033	0.807	486.944			
50 × 50	0.084	2.686	2,430.503	0.064	1.781	1, 106.517			

transportation ($\ell = 3$) were considered, and second that number was increased to five ($\ell = 5$).

In the case $\ell = 3$ it is intended to simulate the following different subnetworks,

- The taxi network, generated as a connected network, $(\mathcal{N}, \mathcal{A})$, with $n = 100, \dots, 1{,}000$ and $m = 5n, 10n, 20n$. The costs are integers uniformly obtained in $\{1, 2, \dots, 120\}$. This structure is the most complete of the three and the one that

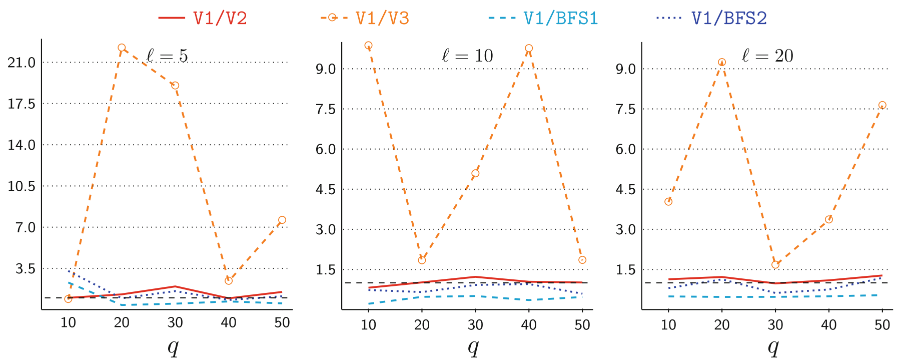


Fig. 7 Mean CPU times ratio for grid networks $q \times q$ with respect to V1

reaches more locations in the network, since it is assumed that taxis are the most flexible transportation.

- The bus network is composed of several routes, of linear or circular format, that is designed by selecting a random number of nodes in $\mathcal{N}$. The number of circuits of each type was twenty, sixty and ninety, whereas the arc costs were obtained as before.
- The subway network is a subtree of $(\mathcal{N}, \mathcal{A})$, containing a random number of nodes. The costs were uniformly generated in $\{1, 2, \dots, 50\}$.

Besides the transportation modes already mentioned for the case $\ell = 3$, the case $\ell = 5$ also included,

- A cycle-way network, similar to the subway network but with a different number of nodes, and costs in $\{1, 2, \dots, 20\}$.
- A minibus, similar to a linear bus route but with a different number of nodes.

The source and the destination nodes are randomly selected.

The results of these tests are summarized in Tables 7 and 8 for the set of instances with $\ell = 3$ and $\ell = 5$, respectively. The first part of these tables lists mean values of the running times and of the number of computed solutions for each number of nodes. The values are in accordance with those observed above for instances with 5 labels, the times are very close and the number of solutions is slightly smaller for $\ell = 3$ than for $\ell = 5$.

In this set of tests the number of nodes, n, and the number of arcs, m, in the graphs simulating the taxis network are fixed. However, the sizes of the bus and of the subway networks, as well as the cycle-way and minibus networks (when $\ell = 5$), vary, so the results are organized into two ways. In the first set of results on Tables 7 and 8 the values obtained are grouped for instances with the same n. Then, because the results in terms of CPU times were similar for instances with approximately the same total number of arcs, such values are presented in the second group. The figures reported for the first case are means for the CPU times, μ, and for the number of efficient output solutions, $\bar{nd}$. For the second case the mean running times are calculated over instances with similar numbers of arcs. The standard deviation of the running times in each sample, σ, is also provided. These standard deviations values are negligible, which confirms

Table 7 Mean value of $|\bar{P}|$ and CPU time (s) of `V1`, $\ell = 3$

n	100	200	300	400	500	600	700	800	900	1,000		
μ	0.001	0.002	0.003	0.004	0.005	0.007	0.008	0.009	0.011	0.013		
$\bar{nd}$	1.437	1.415	1.427	1.442	1.428	1.467	1.486	1.457	1.474	1.454		
$\bar{m}$	8,681	14,610	21,807	28,968	32,571	40,291	48,829	54,248	59,120	66,304	75,178	91,046
μ	0.001	0.002	0.003	0.004	0.004	0.006	0.007	0.010	0.011	0.012	0.014	0.016
σ	0.000	0.000	0.001	0.000	0.001	0.001	0.001	0.002	0.002	0.002	0.002	0.003

μ, mean CPU times; $\bar{nd}$, mean value of $|\bar{P}|$; $\bar{m}$ mean number of arcs; σ, standard deviation of the CPU times

Table 8 Mean value of $|\bar{P}|$ and CPU time (s) of `V1`, $\ell = 5$

n	100	200	300	400	500	600	700	800	900	1,000		
μ	0.001	0.003	0.004	0.006	0.008	0.007	0.008	0.009	0.011	0.013		
$\bar{nd}$	1.437	1.585	1.600	1.667	1.709	1.726	1.684	1.728	1.751	1.751		
$\bar{m}$	8,649	12,006	19,308	26,285	30,152	38,212	42,221	56,520	61,641	68,813	77,873	93,842
μ	0.001	0.003	0.004	0.006	0.007	0.009	0.010	0.010	0.018	0.019	0.023	0.029
σ	0.000	0.001	0.001	0.001	0.002	0.002	0.002	0.004	0.004	0.005	0.005	0.004

μ, mean CPU times; $\bar{nd}$, mean value of $|\bar{P}|$; $\bar{m}$, mean number of arcs; σ, standard deviation of the CPU times

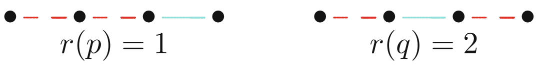
the similarity between the running times in instances with approximately the same number of arcs. The mean times increase with $\bar{m}$ and are smaller for the instances with less means of transportation. According to such classification, on average the problems could be solved in <16 and <29 ms, for instances with 3 and 5 means of transportation, respectively.

The proposed method was also evaluated by tests over data based on part of the city of Coimbra, kindly provided by the INESC-Coimbra research group on urban transportation. This network has 1,686 nodes and 8,128 directed arcs, and comprises the following $\ell = 4$ transportation types: pedestrian, car (taxi), light rail and buses. A travel time is associated with each arc, given by the product of its length and a standard urban speed value for each transportation type. Namely, 4 km/h for walking, 15 km/h for buses, 17 km/h for the light rail, and 30 km/h for cars. Applied to this instance the algorithm minimizes the travel time and the number of transportation means, and was tested for ten different origin-destination pairs. The average running time obtained for these problems was 0.002 s, to compute an average of 1.100 efficient paths between each origin-destination pair.

4 Concluding remarks

Two approaches were proposed for the MCLPP:

Fig. 8 Two paths with the same number of labels



- One that relies on the calculation of the shortest path on the set of networks $(\mathcal{N}, \mathcal{A})_k$ with $k = \hat{l}, \ldots, 2, 1$, together with checking the solutions' dominance.
- Another that uses BFS to find the shortest path on networks with $k = 1, 2, \ldots, \hat{l}$, together with reoptimization to derive new paths.

The number of subproblems and operations was reduced by using the data from intermediate subproblems and the pre-computation of $\mathcal{T}_t$.

In general the first approach outperformed the one using BFS. The best variant included the use of intermediate computations. Instances with $n = 1{,}000$ and $\ell = 20$ were solved in <0.4 s. In multi-graphs modelling networks with $\ell = 3, 5$ transportation modes the efficient paths were computed in <0.03 s. In 50×50 grid networks the best version CPU times did not exceed 172.3 s for $\ell = 20$.

Within certain application problems it is desirable to have the least possible number of transitions, namely if switching labels carries some sort of penalization of the solution. For instance, if a passenger wants to travel from a source to a destination and has two options with the same cost and the same number of labels, as depicted in Fig. 8, then path p, with a single transshipment, is more user convenient than path q, with two. This would imply an additional function to minimize the number of transitions. In the near future we intend to study this case.

Acknowledgments This work was partially supported by the FCT Portuguese Foundation of Science and Technology (Fundação para a Ciência e a Tecnologia) under projects PEst-C/EEI/UI0308/2011, PEst-OE/MAT/UI0152 and PTDC/EEA-TEL/101884/2008. The authors deeply acknowledge the INESC-Coimbra research group on urban transportation, led by João Coutinho Rodrigues, for providing data about the metropolitan area of Coimbra.

References

Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows: theory, algorithms and applications. Prentice Hall, Englewood Cliffs

Bornstein C, Maculan N, Pascoal M, Pinto L (2012) Multiobjective combinatorial optimization problems with a cost and several bottleneck objective functions: an algorithm with reoptimization. Comput Oper Res 39:1969–1976

Chang R, Leu S-J (1997) The minimum labeling spanning trees. Inf Process Lett 63(5):277–282

Clímaco J, Martins E (1982) A bicriterion shortest path algorithm. Eur J Oper Res 11:399–404

Clímaco J, Pascoal M (2012) Multicriteria path and tree problems: discussion on exact algorithms and applications. Int Trans Oper Res 19:63–98

Clímaco J, Captivo ME, Pascoal M (2010) On the bicriterion-minimal cost/minimal label-spanning tree problem. Eur J Oper Res 204:199–205

Consoli S, Moreno JA, Mladenović N, Darby-Dowman K (2006) Constructive heuristics for the minimum labelling spanning tree problem: a preliminary comparison. Technical Report DEIOC-4, Universidad de La Laguna, La Laguna, September http://hdl.handle.net/2438/504

Hansen P (1980) Bicriterion path problems. In: Fandel G, Gal T (eds) Multiple criteria decision making: theory and applications. Lectures notes in economics and mathematical systems

Iori M, Martello S, Pretolani D (2010) An aggregate label setting policy for the multi-objective shortest path problem. Eur J Oper Res 207:1489–1496

Kruskal J (1956) On the shortest spanning subtree of a graph and the traveling salesman problem. Proc Am Math Soc 7:48–50

Prim R (1957) Shortest connection networks and some generalizations. Bell Syst Tech J 36:1389–1401

Raith A, Ehrgott M (2009) A comparison of solution strategies for biobjective shortest path problems. Comput Oper Res 36:1299–1331

Steuer R (1986) Multiple criteria optimization theory, computation and application. Wiley, New York

Van-Nes R (2002) Design of multimodal transport networks: a hierarchical approach. Delft University Press, Delft

Vincke P (1974) Problèmes multicritères. Cahiers du Centre d'Études de Recherche Opérationelle 16:425–439

Wirth H-C (2001) Multicriteria approximation of network design and network upgrade problems. PhD thesis, University of Würzburg