

Instituto de Engenharia de Sistemas e Computadores de Coimbra
Institute of Systems Engineering and Computers
INESC – Coimbra

Marta Pascoal, M. Eugénia Captivo, João Clímaco and Ana Laranjeira

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

No. 1

2011

ISSN: 1645-2631

Instituto de Engenharia de Sistemas e Computadores de Coimbra
INESC – Coimbra
Rua Antero de Quental, 199; 3000 - 033 Coimbra; Portugal
www.inescc.pt

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

MARTA PASCOAL^(1,2), M. EUGÉNIA CAPTIVO⁽³⁾, JOÃO CLÍMACO^(2,4), ANA LARANJEIRA⁽¹⁾
marta@mat.uc.pt, mecaptivo@fc.ul.pt, jclimaco@fe.uc.pt, mat0401@mat.uc.pt

⁽¹⁾ Departamento de Matemática da Universidade de Coimbra,
Apartado 3008, 3001-454 Coimbra, Portugal

⁽²⁾ Instituto de Engenharia de Sistemas e Computadores – Coimbra
Rua Antero de Quental, 199, 3000-033 Coimbra, Portugal

⁽³⁾ Faculdade de Ciências, Universidade de Lisboa
Centro de Investigação Operacional
Campo Grande, Bloco C6, 1749-016 Lisboa, Portugal

⁽⁴⁾ Faculdade de Economia da Universidade de Coimbra
Avenida Dias da Silva, 165, 3004-512 Coimbra, Portugal

January 2011

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, Multi-Objective Decision Making.

1 Introduction and motivation

Besides the cost, probably the most common, other objective functions are relevant in network optimization problems. The number of different colors (or labels) in the feasible solution arcs is one of those functions, and was introduced by Chang and Leu [1] in the context of determining spanning trees. The goal of the minimal label spanning tree problem (MLSTP) is to find the most uniform 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 [8, 12], [1] proved the NP-hardness of the MLSTP. Following up that work other approximate algorithms have been developed for the MLSTP and related problems. Literature reviews on these methods can be found in [3].

A bicriteria version of this problem, the minimal cost–minimal number of labels spanning tree problem (MCLSTP), was studied in [2]. 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 shortest path algorithm, based on ranking paths by non-decreasing order of cost, introduced by Clímaco and Martins [9]. 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.

Given the potential applications in the aforementioned fields in this work we extend the study presented in [2] and address the determination of shortest paths with minimal number of labels. The manuscript is organized as follows. In Section 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 per each non-dominated pair of objective values. Different implementations of this method are described, and at a second stage reoptimization techniques and cost lower 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 the second using random multi-graphs that simulate a transportation network where different means of transportation are distinguished by different arc labels. The last section addresses the additional minimization of the number of transitions along the paths and presents concluding remarks.

2 The minimum 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 can be solved in polynomial time, for instance, by a labelling algorithm [5]. 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 [13] proved this problem is NP-hard.

The MCLPP consists of determining paths from s to t which are efficient solutions of the bicriterion problem:

$$\begin{aligned} \min\{c(p) : p \in \mathcal{P}\} \\ \min\{l(p) : p \in \mathcal{P}\} \end{aligned}$$

In general there is a conflict between functions c and l , and thus there is not an optimal solution for such a problem. Our goal is then to determine a set of solutions that are *efficient*, or *non-dominated*.

Given $p, p' \in \mathcal{P}$, p dominates p' (denoted $p_D p'$) if and only if

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

and at least one of the inequalities is strict. In such a case we also say that $(c(p), l(p))$ dominates $(c(p'), l(p'))$. Path p' is dominated if and only if there is another path between the same pair of nodes p such that $p_D p'$, otherwise in general p is said to be efficient and its image, $(c(p), l(p))$, is

said is said to be non-dominated. We intend to compute a set of paths such that their objective values are non-dominated, \mathcal{P}_{NDV} , for the MCLPP.

The method proposed here computes a set of solutions corresponding to non-dominated pairs of objective values 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 of the applications.

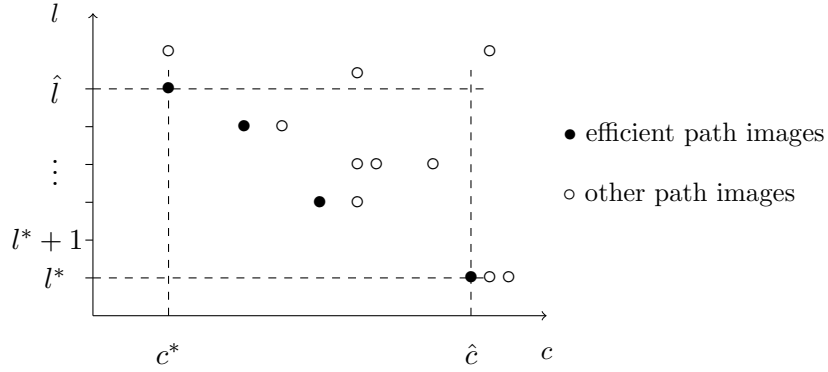


Figure 1: MCLPP solutions

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 space of the non-dominated objective function images, that lies in $[c^*, \hat{c}] \times \{l^*, \dots, \hat{l}\}$, Figure 1. In fact, there are no solutions such that $c(p) < c^*$ or $l(p) < l^*$. Besides,

$$c(p) \geq \hat{c} \text{ and } l(p) \geq l^* \text{ (with } (c(p), l(p)) \neq (\hat{c}, l^*)) \Rightarrow (\hat{c}, l^*) \text{ dominates } (c(p), l(p)),$$

and, similarly

$$c(p) \geq c^* \text{ and } l(p) \geq \hat{l} \text{ (with } (c(p), l(p)) \neq (c^*, \hat{l})) \Rightarrow (c^*, \hat{l}) \text{ dominates } (c(p), l(p)).$$

It is easier to manipulate the network in order to fix its labels than to fix path costs. Besides, the range of labels is usually narrower than the range of path costs. For this reason we look for the 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 paths in a network is not necessarily a complete sequence. A counter-example is given in Figure 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.

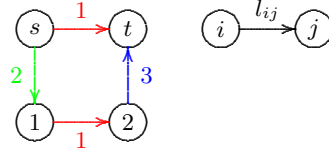


Figure 2: Network with labeled arcs

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

- 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.

Proof. Let k be a certain number of labels between l^* and \hat{l} . There is nothing to prove if there is no path with k labels. Otherwise, let $p^* \in \mathcal{P}$ be the shortest path with exactly k labels, that is, such that $l(p^*) = k$ and $c(p^*) \leq c(q)$ for any $q \in \mathcal{P}$ with $l(q) = k$. Then, given any $p' \in \mathcal{P} - \{p^*\}$ three situations may occur,

- $l(p') > k$, which means that p' does not dominate p^* ,
- $l(p') = k$, then the same conclusion holds because, by assumption, $c(p^*) \leq c(p')$,
- $l(p') < k$, then
 - either $c(p') > c(p^*)$, and p' does not dominate p^* ,
 - or $c(p') \leq c(p^*)$, and p' dominates p^* and $c(p') \leq c(p^*) \leq c(q)$ for any $q \in \mathcal{P}$ such that $l(q) = k > l(p')$, thus p' dominates q as well. The proof is complete if p' is efficient, otherwise the reasoning can be repeated and because there is a finite number of values in $\{l^*, \dots, \hat{l}\}$ there exists an efficient path which dominates p^* and all the other paths with k labels.

□

If the number of labels is fixed, the shortest path dominates other paths with a greater cost. Thus, by Proposition 1 and its proof, 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 shortest path algorithm to $(\mathcal{N}, \mathcal{A})$ provides the value c^* and an upper bound on \hat{l} . However, since the MLPP is NP-hard, 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 $l(p^*)$ (if p^* is a shortest path from s to t in $(\mathcal{N}, \mathcal{A})$), 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 $\{(c(p_k), l(p_k)) : k = l^*, \dots, \hat{\ell}\}$ 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 . \square

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 minimal cost / number of labels paths*

```

//  $P_{NDV}$  is a set that contains possible efficient paths
01  $p^* \leftarrow$  shortest path from  $s$  to  $t$  in  $(\mathcal{N}, \mathcal{A})$ 
02  $P_{NDV} \leftarrow \{p^*\}$ 
03  $k \leftarrow l(p^*) - 1$ 
04 While  $k \geq 1$  Do
05    $p_k \leftarrow$  shortest path from  $s$  to  $t$  in  $(\mathcal{N}, \mathcal{A})_k$ 
06   If  $p_k$  is defined Then  $P_{NDV} \leftarrow P_{NDV} \cup \{p_k\}$ 
07    $k \leftarrow \min\{l(p_k), k\} - 1$ 
08 Remove the dominated paths from  $P_{NDV}$ 

```

Note that $(\mathcal{N}, \mathcal{A})$ is the only network in set $(\mathcal{N}, \mathcal{A})_\ell$, however the number of labels of p_k is not necessarily $\hat{\ell}$, but rather an upper bound on that value. Nevertheless $\hat{\ell}$ is obtained after a path with a cost greater than c^* is computed or when the algorithm exits the **While** loop. Moreover, instead of filtering the dominated paths that are stored in P_{NDV} only at the end of the algorithm, these paths can be filtered within the loop. The constrained problem 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 $(\mathcal{N}, \mathcal{A})_k$*

```

01  $BestCost \leftarrow +\infty$ 
02 For any  $(\mathcal{N}, A_k)$  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      $p_k \leftarrow p$ 

```

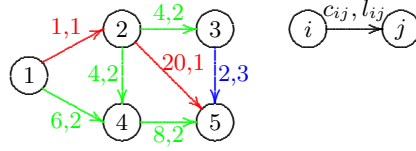


Figure 3: Network $(\mathcal{N}, \mathcal{A})$

As an illustration of Algorithm 1 we consider its application to the network represented in Figure 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 $k = 3, 2, 1$. The following table shows a scheme of the paths that are calculated throughout the procedure.

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

In Procedure 1 it is assumed that for each number of labels k all the subnetworks of $(\mathcal{N}, \mathcal{A})$ with its set of arcs restricted to contain exactly k labels are considered. One 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 remaining node contains a new label that is added to its antecessors' labels. Figure 4 shows an example of such a tree, if the arc labels are 1, 2 and 3.

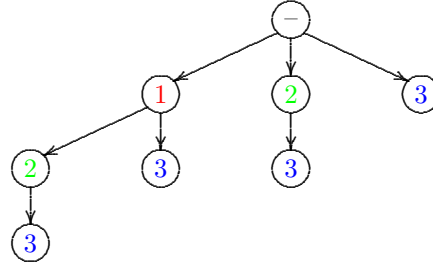


Figure 4: Search tree of a network with the labels 1, 2 and 3.

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 $\{1, \dots, \ell\}$, as depicted in Figure 4.

Algorithm 2. *Finding minimal cost / number of labels paths with a BFS tree*

```

01  $p \leftarrow$  shortest path in  $(\mathcal{N}, \mathcal{A})$ 
02  $\hat{l} \leftarrow l(p)$ 
03  $X \leftarrow \{\emptyset\}$ 
04  $P_{NDV} \leftarrow \emptyset$ 
05  $x \leftarrow 0$ 
07 While  $x \leq \hat{l}$  Do
08    $C \leftarrow$  first element in  $X$ 
09    $X \leftarrow X - \{C\}$ 
10    $x \leftarrow$  number of labels in  $C$ 
11   For  $k \in \{1, \dots, \ell\} - C$  Do
12     Insert  $C \cup \{k\}$  at the end of  $X$ 
13      $A' \leftarrow$  {arcs in  $\mathcal{A}$  with a label in  $C \cup \{k\}$ }
14      $p \leftarrow$  shortest path in  $(\mathcal{N}, A')$ 
15     If  $p$  is defined Then  $P_{NDV} \leftarrow P_{NDV} \cup \{p\}$ 
16 Remove dominated solutions from  $P_{NDV}$ 

```

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 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 minimal cost / number of labels paths using a BFS tree with path reoptimization*

```

01  $p \leftarrow$  shortest path in  $(\mathcal{N}, \mathcal{A})$ 
02  $\hat{l} \leftarrow l(p)$ 
03  $X \leftarrow \{\emptyset\}$ 
04  $P_{NDV} \leftarrow \emptyset$ 
05  $x \leftarrow 0$ 
06 While  $x \leq \hat{l}$  Do
07    $C \leftarrow$  first element in  $X$ 
08    $X \leftarrow X - \{C\}$ ;
09    $x \leftarrow$  number of labels in  $C$ 
10    $A' \leftarrow$  {arcs in  $\mathcal{A}$  with a label in  $C$ }
11    $q \leftarrow$  shortest path in  $(\mathcal{N}, A')$ 

```

```

12   For  $k \in \{1, \dots, \ell\} - C$  Do
13     Insert  $C \cup \{k\}$  at the end of  $X$ 
14      $p \leftarrow$  shortest path obtained from inserting in  $q$  the arcs in  $\mathcal{A}$  with the label  $k$ 
15     If  $p$  is defined Then  $P_{NDV} \leftarrow P_{NDV} \cup \{p\}$ 
16 Remove dominated solutions from  $P_{NDV}$ 

```

Using upper bounds 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 optimum 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.

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. \square

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

Proof. Let \bar{p} be an efficient path. Then \bar{p} is a shortest path in $(\mathcal{N}, \mathcal{A})_{l(\bar{p})}$. Assume there is a path p with at most $l(\bar{p})$ labels and such that $c(p) < c(\bar{p})$. Because \bar{p} is a shortest path in $(\mathcal{N}, \mathcal{A})_{l(\bar{p})}$ then $l(p) \neq l(\bar{p})$ must hold, that is $l(p) < l(\bar{p})$. Thus we have $l(p) < l(\bar{p})$ and $c(p) < c(\bar{p})$, which means that \bar{p} is not efficient, as it was assumed. \square

These results can be implemented 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, 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 at an iteration k of the algorithms, $CostUB$ is updated as

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

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

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 following steps.

Step	$CostUB$	$(\mathcal{N}, \mathcal{A})_k$	$l(\mathcal{A}_k)$	$p : (c(p), l(p))$	p_k	P_{NDV}
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	$[14, 13, 7]$	$(\mathcal{N}, \mathcal{A})_1$	$\{1\}$ $\{2\}$	$\langle 1, 2, 5 \rangle : (21, 1)$ $\langle 1, 4, 5 \rangle : (14, 1)$	$\langle 1, 4, 5 \rangle$	$\{\langle 1, 2, 3, 5 \rangle, \langle 1, 2, 4, 5 \rangle, \langle 1, 4, 5 \rangle\}$

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 on 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 only be efficient if $l(q) < l(p^*)$, and thus q is still defined 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

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

is satisfied. These conditions can be made stronger, namely they can be replaced by $\pi_i + c_{ij} < \pi_j$ and $\pi_i + c_{ij} < CostUB_k$, if a path p that is calculated is stored whenever $c(p) < CostUB_{l(p)}$, thus depending on the method's implementation.

Tightening upper bounds The above upper bound for the paths cost 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 is obtained at an early stage of the algorithm (which can be done by means of a labelling method). 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 when

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

on some network in $(\mathcal{N}, \mathcal{A})_k$, $k = 1, \dots, \hat{l}$.

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, were implemented:.

- V1: a direct version of Algorithm 1,

- V2: an improved version of V1 enhanced with the upper bounds on the path costs described in the paragraphs above, and
- BFS: a version of Algorithm 3, using a BFS tree and reoptimizing shortest paths in every level of the search tree.

Random graphs In a first set of tests we considered thirty randomly generated graph instances of each dimension, with $n = 100, 200, \dots, 1000$, $m = 5n, 10n, 20n$, and $\ell = 5, 10, 20$. The arc costs were uniformly generated in $\{1, 2, \dots, 100\}$.

m	$n \times 5$			$n \times 10$			$n \times 20$			
	ℓ	5	10	20	5	10	20	5	10	20
$n = 100$		2.600	2.800	3.400	2.667	3.000	3.067	3.000	3.400	3.467
$n = 200$		2.200	2.333	2.533	3.200	3.267	3.867	3.200	3.667	3.733
$n = 300$		2.800	3.067	3.267	3.400	3.533	3.733	3.333	3.733	4.133
$n = 400$		2.200	2.400	2.533	3.600	3.667	4.267	3.667	4.733	5.200
$n = 500$		2.800	2.933	2.933	3.333	3.600	3.867	3.133	3.733	4.400
$n = 600$		2.933	3.933	4.200	3.667	3.867	4.733	3.400	3.667	4.200
$n = 700$		2.333	2.933	3.200	4.067	4.067	5.000	3.667	4.467	4.200
$n = 800$		4.800	3.133	2.533	3.267	4.333	4.000	3.733	4.133	4.667
$n = 900$		3.067	3.133	3.267	3.400	3.133	3.933	3.800	4.667	5.200
$n = 1000$		2.867	3.267	3.267	3.800	3.600	4.867	3.400	4.200	4.267

Table 1: Mean number of efficient paths

Table 1 summarizes the mean number of efficient solutions obtained for those instances. That number depends on the number of arc labels, ℓ , which is the maximum number of listed efficient paths. According to Table 1 the mean number of determined efficient paths ranged between 2.2 and 5.2, moreover this number usually grows with the average node degree of the graphs.

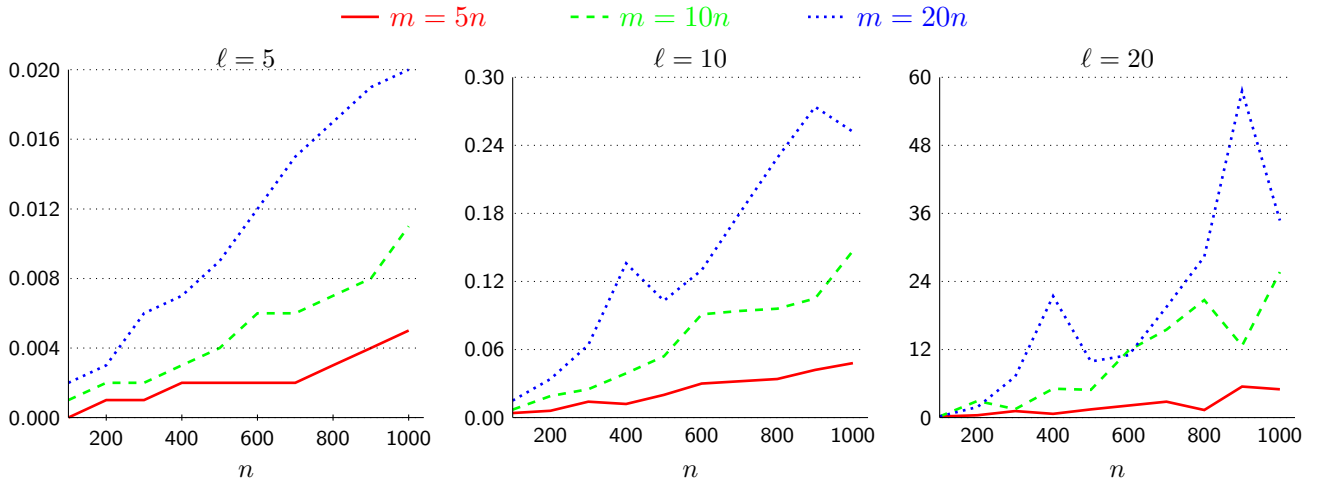


Figure 5: Mean CPU times (secs) of V1

Figure 5 presents the running times, in seconds, obtained by code V1 on this set of tests, whereas Figure 6 shows the equivalent but with respect to code V2. 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 program also becomes slower when the number of nodes or the average node degree is larger, although this performance is not always stable. The inclusion of upper bounds to limit the number of labels produced during the shortest path computation has a big impact on the program running times, and V2 CPU times varied from some milliseconds only to 1.25 seconds approximately. Similarly to what was observed for V1, the performance of V2 is still highly dependent on the number of labels and, in general, it is better for sparser than for denser graphs. However, V2 presented a performance less steady than V1's.

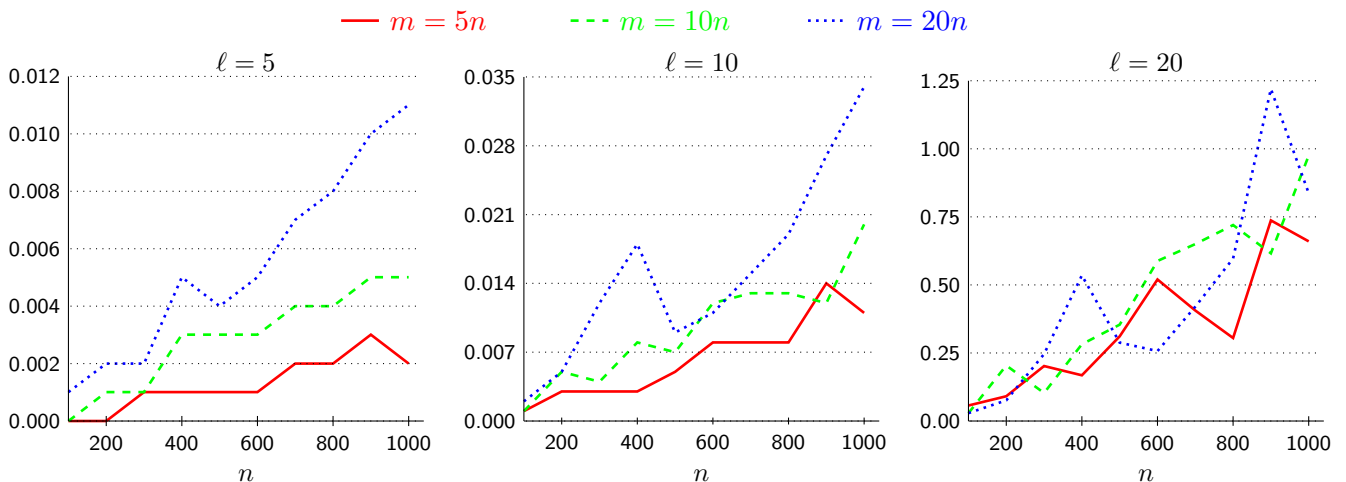


Figure 6: Mean CPU times (secs) of V2

Finally, Figure 7 compares the versions with and without the search tree, namely it presents the values of the ratio between the original versions BFS and V1, both without using any information on upper bounds. The graphics show that, on the one hand for graphs with $5n$ and $10n$ arcs the version using the search tree is always worse than V1, and that the difference is bigger when ℓ is bigger too. On the other hand, in general, for denser graphs the ratio becomes closer to 1 when ℓ increases, and BFS slightly outperforms V1 for small instances with twenty labels.

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 (namely by means of the forward star form, [4]), 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 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 taxis network, which is generated as a connected network, $(\mathcal{N}, \mathcal{A})$, with $n = 100, \dots, 1000$

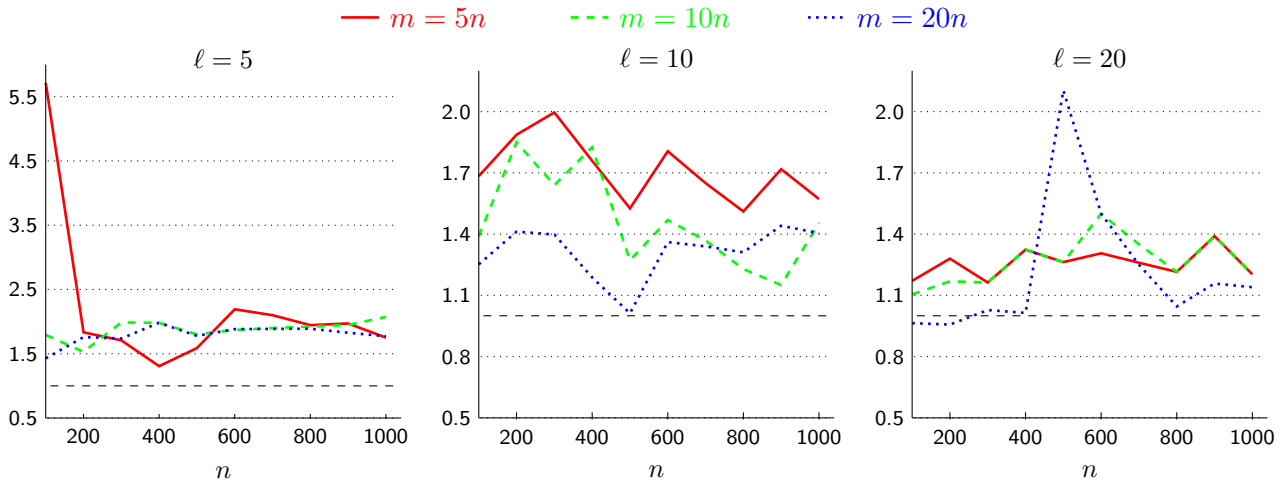


Figure 7: Mean CPU times ratio: BFS/V1

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

- The buses network is composed of several circuits, 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 but with a different number of nodes, and costs in $\{1, 2, \dots, 20\}$.
- A minibus, which is similar to a linear bus circuit but with a different number of nodes.

The results of these tests are summarized in Tables 2 and 3 for the set of instances with $\ell = 3$ and $\ell = 5$, respectively. The first part of these tables groups mean values of the running times and the number of computed solutions for each number of graph 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 $\ell = 5$. Because in these tests only the taxis network has a fixed number of arcs, the size of the whole network is not always the same. Thus, the second part of the tables groups data related with instances of approximately the same number of arcs, and which presented similar times, as shown by the small standard deviations. As for the mean times, these increase with \bar{m} and are smaller for the cases of less means of transportation. Still the problems could be solved in less than 16 milliseconds and less than 29 milliseconds, for instances with 3 and 5 means of transportation, respectively.

n	100	200	300	400	500	600	700	800	900	1000		
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}	8681	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

Legend. μ : Mean CPU times; \bar{nd} : Mean number of efficient paths; \bar{m} : Mean number of arcs; σ : Standard deviation of the CPU times.

Table 2: Mean number of efficient paths and CPU time (secs) of $V1$, $\ell = 3$

n	100	200	300	400	500	600	700	800	900	1000		
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}	8649	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

Legend. μ : Mean CPU times; \bar{nd} : Mean number of efficient paths; \bar{m} : Mean number of arcs; σ : Standard deviation of the CPU times.

Table 3: Mean number of efficient paths and CPU time (secs) of $V1$, $\ell = 5$

4 Dealing with the number of transitions

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 Figure 8, then path p , with a single transshipment, is more user convenient than path q , with two.



Figure 8: Two paths with the same number of labels

Let $r(p)$ denote the number of transitions of any path p . Mathematically, including the minimization of r as another objective results in a tricriteria path problem. Solutions for this problem can be obtained with some modifications of the methods previously described. Two approaches can be used for finding problem solutions. One consists of replacing c with an aggregated function that penalizes consecutive arcs with different labels, thus defining a problem that is similar to the MCLPP. Tests with the implementation of such approach ran slightly slower than the former (although with the same magnitude of time), but with a similar relative behaviour.

The other approach consists of dealing with function r as an additional constraint over the paths, rather than an objective function. This means imposing a threshold, $K \in \mathbb{N}$, for the maximum

number of transitions a path is allowed to have. The determination of solutions for this problem can still be done by adapting the previous algorithms, in order to include a test on the paths' feasibility. Under this assumption the goal of Procedure 1 is to find the shortest feasible path from s to t in $(\mathcal{N}, \mathcal{A})_k$. An approximation for the solutions set can be found if line 4 of Procedure 1 is replaced with

04 If $c(p) < BestCost$ and $r(p) \leq K$ Then

However, for some of the subnetworks the shortest paths might not be feasible, leading to solutions that are not the best possible or to cases where no solution is found. In order to calculate all the possible feasible solutions, instead of computing the shortest path the shortest feasible path should be determined, by means of a ranking algorithm [10, 11, 14]. Although more correct, this approach is also computationally more costly than the first one.

5 Concluding remarks

Two approaches were proposed for the MCLPP:

- One that relies on the calculation of the shortest path on networks with $k = \hat{l}, \dots, 2, 1$, together with checking the solutions dominance.
- Another that uses BFS to find the shortest path on networks with $k = 1, 2, \dots, \hat{l}$, 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 $n = 1000$ and $\ell = 20$ were solved in less than 1.5 secs. In multi-graphs modelling networks with $\ell = 3, 5$ transportation modes the efficient paths were computed in less than 0.03 secs.

Acknowledgments This work was partially supported by the FCT Portuguese Foundation of Science and Technology (Fundação para a Ciência e a Tecnologia) under projects POSC/U308/1.3/NRE/04, POCTI/ISFL-1/152 and PTDC/EEA-TEL/101884/2008.

References

- [1] R. Chang and S.-J. Leu. The minimum labeling spanning trees. *Information Processing Letters*, 63(5):277–282, 1997.
- [2] J. Clímaco, M. E. Captivo, and M. Pascoal. On the bicriterion – minimal cost/minimal label – spanning tree problem. *European Journal of Operational Research*, 204:199–205, 2010.
- [3] S. Consoli, J. A. Moreno, N. Mladenović, and K. Darby-Dowman. Constructive heuristics for the minimum labelling spanning tree problem: a preliminary comparison. Technical Report DEIOC-4, Universidad de La Laguna, La Laguna, September 2006. (<http://hdl.handle.net/2438/504>).

- [4] R. Dial, G. Glover, D. Karney, and D. Klingman. A computational analysis of alternative algorithms and labelling techniques for finding shortest path trees. *Networks*, 9:215–348, 1979.
- [5] E. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:269–271, 1959.
- [6] H. Gabow. Two algorithms for generating weighted spanning trees in order. *SIAM Journal on Computing*, 6:139–150, March 1977.
- [7] N. Katoh, T. Ibaraki, and H. Mine. An algorithm for finding K minimum spanning trees. *SIAM Journal on Computing*, 10(2):211–247, 1981.
- [8] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.
- [9] J. Clímaco and E. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404, 1982.
- [10] E. Q. V. Martins, M. M. B. Pascoal, and J. L. E. Santos. Deviation algorithms for ranking shortest paths. *The International Journal of Foundations of Computer Science*, 10(3):247–263, 1999.
- [11] M. Pascoal. Implementations and empirical comparison for K shortest loopless path algorithms. In *Online Proc. of The Ninth DIMACS Implementation Challenge: The Shortest Path Problem*. DIMACS, USA, November 2006.
- [12] R. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
- [13] H.-C. Wirth. *Multicriteria Approximation of Network Design and Network Upgrade Problems*. PhD thesis, University of Wurtzburg, 2001.
- [14] J. Y. Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.