

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

João Clímaco, M. Eugénia Captivo and Marta Pascoal

On the bicriterion – minimum cost/minimum label – spanning tree problem

No. 12

2008

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

On the bicriterion – minimum cost/minimum label – spanning tree problem

JOÃO C. N. CLÍMACO^(1,2), M. EUGÉNIA CAPTIVO⁽³⁾, MARTA M. B. PASCOAL^(1,4)

⁽¹⁾ Instituto de Engenharia de Sistemas e Computadores – Coimbra
Rua Antero de Quental, 199, 3000-033 Coimbra, Portugal

⁽²⁾ Faculdade de Economia da Universidade de Coimbra
Avenida Dias da Silva, 165, 3004-512 Coimbra, Portugal
E-mail: jclimaco@inescc.pt

⁽³⁾ Faculdade de Ciências, Universidade de Lisboa
Centro de Investigação Operacional
Campo Grande, Bloco C6, 1749-016 Lisboa, Portugal
E-mail: mecaptivo@fc.ul.pt

⁽⁴⁾ Departamento de Matemática da Universidade de Coimbra,
Apartado 3008, 3001-454 Coimbra, Portugal
E-mail: marta@mat.uc.pt

August 2008

Abstract: We deal with a bicriterion spanning tree problem relevant in some application fields such as telecommunication networks or electric networks. Each edge is assigned with a cost value and a label (such as a color). The first criterion intends to minimize the total cost of the spanning tree (the summation of its edge costs), while the second intends to get the solution with a minimum number of different labels. As these criteria are generally conflictual we developed an algorithm to generate the set of non-dominated spanning trees. Computational experiments are presented and results discussed.

Keywords: Spanning tree, Minimum cost, Minimum label, Multi-Objective Decision Making.

1 Introduction and motivation

The determination of spanning trees leads to several optimization problems with many applications, specially when the network connectivity is a requirement. One of the most studied of those problems is the minimum cost spanning tree problem, or simply the minimum spanning tree problem, (MCSTP). Its goal is to find a minimum cost connected subgraph of a network considering an additive objective function. This problem can be solved in polynomial time using, for instance, the algorithms proposed by Kruskal [8], in 1956, or by Prim [12], in 1957. However, other objective functions are of interest. For example, one of those cases is the minimum label spanning tree problem (MLSTP), where it is intended to determine the most uniform subgraph of a network, assuming each edge is associated with a color. This type of problem has applications involving telecommunications, as different colors can be seen as different operators, calls of different types, or different technologies. The problem was introduced in by Chang & Leu in 1997 [3], who proved it to be NP-hard, by reducing it to a minimum cover problem. Despite this problem being harder than the MCSTP Chang & Leu developed an exact exponential algorithm, as well as two heuristic approaches with time complexity of $\mathcal{O}(mn)$ and $\mathcal{O}(lmn)$, where ℓ , n and m are the number of distinct colors, vertices and edges in the network, respectively. Since then other researchers have studied this problem and presented other heuristics, for instance [2, 4, 7, 13, 14, 15].

This paper concerns the bicriterion minimum cost/minimum-label spanning tree problem (BM-CMLSTP), that is, the determination of spanning trees in a network where each edge is associated with a cost and a label, and it is intended to optimize both the spanning trees cost, given by its edge costs summation, as well as the number of distinct labels on its edges. As the two objective functions, cost and number of labels, are usually conflictual, instead of an optimal solution, our goal will be to determine a set of solutions that are not dominated by any other. First we describe an algorithm to find the whole set of non-dominated spanning trees. As it will be shown, the numerous spanning trees with exactly the same cost and number of labels lead this algorithm to perform poorly for medium size instances, therefore a second method, that computes the set of spanning trees with non-dominated objective values, is proposed.

The manuscript is organised in the following manner. Section 2 introduces concepts and notation used along the text. Section 3 presents a method for finding all non-dominated spanning trees, based on an algorithm that computes spanning trees by non-decreasing order of cost together with a dominance test, and in Section 4 an alternative process, for finding the spanning trees with non-dominated objective values is proposed. Computational experiments with these two methods are also presented. Concluding remarks are drawn in the last Section.

2 Notation and definitions

In the following we consider an undirected network $(\mathcal{N}, \mathcal{A})$, where \mathcal{N} denotes the set of n vertices and \mathcal{A} the set of m edges. With each edge $\{i, j\}$ a real value c_{ij} , called cost, and a label, or color, l_{ij} , are associated. The cost of a given spanning tree T is $c(T) = \sum_{\{i,j\} \in T} c_{ij}$, while $l(T)$ represents the number of distinct labels/colors in T . We look for spanning trees T that simultaneously minimize $c(T)$ and $l(T)$ in the set of all spanning trees of the network.

As mentioned earlier, we intend to compute spanning trees that minimise c and l , however if the two objective functions are conflictual there is no solution that minimises both simultaneously. Optimality is substituted by the concept of non-dominance. One solution is a non-dominated solution if there is no other feasible solution which improves one objective function without worsening the other.

Given two spanning trees T and T' it is said that T dominates T' , or that T' is dominated by T , $T_D T'$, if and only if $c(T) \leq c(T')$, $l(T) \leq l(T')$ and at least one of the inequalities is strict. T' is said to be dominated if and only if there is another spanning tree T such that $T_D T'$.

The set of dominated spanning trees is denoted by \mathcal{T}_D , while the others form the set of non-dominated spanning trees, denoted by \mathcal{T}_N .

3 Solving the BMCLSTP by ranking spanning trees

One of the classes of algorithms to find the set of non-dominated solutions of bicriterion problems was proposed in [10], and is based on ranking algorithms. The work [10] focuses multicriteria optimal path problems, and was later specialised by the same authors for the bicriterion shortest path problem [11], but it can also be adapted for other multicriteria problems, like the BMCMLSTP. With this method solutions are ranked by non-decreasing order of one of the objective functions, while a dominance test is added in order to compute the set \mathcal{T}_N by partitioning it into several subsets. Spanning trees can be determined by order of cost in polynomial time - see, for instance, [5, 6] - thus we will use a ranking algorithm to obtain the BMCMLSTP non-dominated solutions.

To figure out about the dominance of the determined solutions we note that if the spanning trees are listed by non-decreasing order of cost, then the non-dominated ones should have a non-increasing number of colors. The potential non-dominated spanning trees are stored in a set, \mathcal{T}_X

and, when a new tree T is scanned $c(T)$ and $l(T)$ are compared to M_c and m_l , respectively, where M_c denotes the greatest cost of the spanning trees that have been determined so far, while m_l represents the least number of distinct colors of those spanning trees. As spanning trees are listed two situations might arise: $c(T) = M_c$ or $c(T) > M_c$. In the first case we can conclude T is dominated if $l(T) > m_l$, otherwise T is candidate to be a non-dominated spanning tree, therefore it is stored in \mathcal{T}_X . In the second case the determination of a new subset of solutions begins, the set \mathcal{T}_X is reset to include only the tree T .

Clímaco and Martins established a stopping condition for ranking in the bicriterion shortest path problem, based on the best value of the second objective function a non-dominated solution can have (and therefore on the worst value the first objective function can present). As the determination of the minimum number of labels is an NP-hard problem, some polynomial time heuristics are known in order to find approximate values. We also refer to the exact, but with exponential time, algorithm proposed by [3], as well as the mixed integer linear formulation of the minimum label spanning tree problem introduced by [1], that allowed to solve problems with 1000 nodes and 10 labels or 50 nodes and labels in less than 44 seconds and 10 seconds, respectively. One can also expect these values to be defined by the user, even though taking the risk of not finding the whole set \mathcal{T}_N . In the following an adaptation of the algorithm proposed by Clímaco and Martins aiming to determine the non-dominated solutions of the BMCMLSTP is presented.

Algorithm 1 *Algorithm to compute all non-dominated solutions of the BMCMLSTP*

```

 $T_c \leftarrow$  least cost spanning tree,  $M_c \leftarrow c(T_c)$ ,  $m_l \leftarrow l(T_c)$ 
 $LB_l \leftarrow$  lower bound for the number of distinct colors of a spanning tree
 $\mathcal{T}_X \leftarrow \emptyset$ ,  $\mathcal{T}_N \leftarrow \emptyset$ , continue  $\leftarrow$  True,  $k \leftarrow 0$ 
While (continue) Do
     $k \leftarrow k + 1$ 
     $T_k \leftarrow$   $k$ -th shortest tree
    If ( $c(T_k) = M_c$ ) Then /* Dominance test */
        If ( $l(T_k) = m_l$ ) Then  $\mathcal{T}_X \leftarrow \mathcal{T}_X \cup \{T_k\}$ 
        Else
            If ( $l(T_k) < m_l$ ) Then  $\mathcal{T}_X \leftarrow \{T_k\}$ ,  $m_l \leftarrow l(T_k)$ 
    Else
        If ( $l(T_k) < m_l$ ) Then
             $\mathcal{T}_N \leftarrow \mathcal{T}_N \cup \mathcal{T}_X$ ,  $\mathcal{T}_X \leftarrow \{T_k\}$ ,  $M_c \leftarrow c(T_k)$ ,  $m_l \leftarrow l(T_k)$ 
            If ( $l(T_k) = LB_l$ ) Then continue  $\leftarrow$  False

```

Methods like the algorithms by Kruskal [8] or Prim [12], are well-known to compute the minimum cost spanning tree. On the other hand polynomial algorithms have been presented with the goal of determining the K best spanning trees by order of cost, T_1, \dots, T_K , for a given K . Those algorithms were introduced by Gabow [5] and by Katoh, Ibaraki and Mine [6], and work in a similar way, finding the best spanning tree and then obtaining the next solution by means of computing the exchange between an in-tree and non-tree edge which gives the minimum possible increase of the tree cost. They differ on the details about how to maintain the partition (introduced by Lawler [9] for obtaining the K best combinatorial optimization problem solutions) and how to select the edge exchanges. Gabow's method has time complexity of $\mathcal{O}(K m \alpha(m, n) + m \log m)$, while the one proposed by Katoh, Ibaraki and Mine runs in time of $\mathcal{O}(K m)$, both have $\mathcal{O}(K + m)$ space complexity.

3.1 Computational experiments

Some preliminary tests have been made in order to evaluate the method above for computing the set of non-dominated spanning trees minimising the cost and the number of labels. The instances consisted of:

- random networks, with $n = 10$, $m = 45$, or $n = 15, 20$, $m = 50, 100$, and $\ell = 5, 10$, or $n = 25, 30$, $a = 2, 4$ (where $m = n \times a$), and $\ell = 5, 10, 20$;
- grid networks, with $n = 20$ and forms 2×10 , 4×5 , or $n = 36$ and forms 2×18 , 3×12 , 6×6 , and $\ell = 5, 10, 20$.

The edge costs were integer values uniformly generated between 1 and 100.

The set of non-dominated spanning trees on the instances above was computed by a C language implementation of Algorithm 1 on a Dual Core AMD Opteron at 1 GHz, with 1 Mb of RAM.

ℓ	$n = 15$				$n = 20$			
	$m = 50$		$m = 100$		$m = 50$		$m = 100$	
	$ \mathcal{T}_N $	Time	$ \mathcal{T}_N $	Time	$ \mathcal{T}_N $	Time	$ \mathcal{T}_N $	Time
5	4.4	338.33638	4.4	33.32670	3.4	302.88853	5.1	41.61100
10	5.5	291.93943	5.6	31.94850	4.2	392.78989	6.4	32.71180
20	6.5	253.12881	7.1	32.82930	6.4	469.49116	8.4	38.32590

Table 1: Number of non-dominated solutions and running times (in seconds) on random networks with $n = 15, 20$

It was only possible to determine the entire set of non-dominated solutions for small size instances. The number of spanning trees with exactly the same cost increased very fast with the size of instances or the maximum number of colors, and many problems could not run until the end due to memory overflow. Table 1 presents some of the partial average results obtained in 10 problems of the smaller random instances mentioned above. The reason why both $|\mathcal{T}_N|$ and the CPU times decrease for some bigger network cases is that only the easiest problems ran until the end and the remaining ones have not been considered for the average values computation.

4 An alternative method for the BMCLSTP

4.1 Theoretical results and their consequences

Let c^* and l^* denote, respectively, the minimum cost and the minimum number of labels of any spanning tree. Let \hat{c} be the minimum cost of a spanning tree with l^* labels, which corresponds to the maximum cost associated to a non-dominated spanning tree. Let \hat{l} be the minimum number of labels of a spanning tree with cost c^* , which corresponds to the maximum number of labels of a non-dominated spanning tree.

Lemma 1 *Let T, T' be two spanning trees such that $T' = T - \{\{x, y\}\} \cup \{\{x', y'\}\}$, being $\{x, y\}$ a leaving edge and $\{x', y'\}$ an entering edge. Then $l(T') = l(T)$ or $l(T') = l(T) \pm 1$.*

Proof. It is trivial. In fact, $T - \{x, y\}$ includes $l(T)$ or $l(T) - 1$ labels. So, adding $\{x', y'\}$ to obtain T' we have the following possibilities:

1. $T - \{x, y\}$ includes $l(T)$ labels.

Adding $\{x', y'\}$, of course $l(T')$ is equal to $l(T)$ or $l(T) + 1$.

2. $T - \{x, y\}$ includes $l(T) - 1$ labels.

Adding $\{x', y'\}$, of course $l(T')$ is equal to $l(T) - 1$ or $l(T)$.

So, $l(T')$ is equal to $l(T)$ or $l(T) \pm 1$. □

Proposition 1 *There is at least a non-dominated tree for any l such that $l \in [l^*, \hat{l}]$, except for those $l_1 \in [l^*, \hat{l}]$ for which there exists at least a spanning tree with $l_2 < l_1$ dominating all the spanning trees with l_1 labels. In this case, the best of those spanning trees (with l_2 labels) is non-dominated (or are, in case of alternative optima).*

Proof. Suppose we consider a spanning tree with minimum cost c^* and among those with this cost, the solution with minimum number of labels, i.e. \hat{l} .

Starting from this non-dominated spanning tree (c^*, \hat{l}) and considering the spanning trees ranked according to the cost (as in Algorithm 1) and Lemma 1, it is easy to conclude that the first obtained spanning tree, decreasing \hat{l} , has exactly $\hat{l} - 1$ labels and a cost c_1 greater than c^* .

Taking into account the definition of non-dominated spanning tree it is easy to see that there are two possibilities, either the obtained spanning tree is non-dominated or there is some other tree with the same cost (c_1) and better l . Of course, continuing the ranking according to the cost, as soon as the cost is greater than c_1 , considering the whole set of solutions with cost c_1 and selecting the one with best l (say l_2 , being $l_2 \leq \hat{l} - 1$), we obtain the second non-dominated spanning tree. (Note that it is possible the existence of alternative optima, and so several non-dominated spanning trees with the same cost and number of labels.)

The process would continue until a spanning tree with l^* labels is determined. □

From this proposition it is possible to propose a new approach to calculate non-dominated spanning trees such that $k \in [l^*, \hat{l}]$. Of course, it is enough to calculate the minimum cost spanning tree corresponding to each $k \in [l^*, \hat{l}]$ (for technical details see next paragraph) and check whether some of obtained solutions are dominated among them. These solutions have to be eliminated.

It is not very interesting to check systematically whether several spanning trees are alternative non-dominated solutions with the same cost and number of labels, specially because the computational cost is high and the added information is not very valuable in most of the cases. However it is possible in practical applications to look for some of these solutions in special interesting cases.

4.2 The algorithm

According to the latter section, there is at least one spanning tree with k colors, for any $k \in [l^*, \hat{l}]$. As the computational experiments have shown already, many of them have exactly the same objective values, therefore we now propose a method for computing only one spanning tree for each non-dominated pair of objective values.

If there is a non-dominated spanning tree with k labels, $k \in [l^*, \hat{l}]$, then it must be a minimum cost spanning tree on some subnetwork of the original network where the set of edges is restricted to have k distinct labels, otherwise it would have the same number of labels and worse cost. In order to find the non-dominated solutions for each number of labels all the combinations with k out of the ℓ colors in the network are considered. Then any algorithm to find the minimum cost spanning tree can be applied on the subnetwork of $(\mathcal{N}, \mathcal{A})$ containing only the edges with those k labels, as described below. Again this is an NP-hard problem itself, however for a not very large number of distinct network labels ℓ this procedure ran with reasonable execution times, as we shall see in the next section.

Algorithm 2 *Algorithm to compute the minimum cost spanning tree with k labels*

$BestCost \leftarrow +\infty$
For every subset C of $\{1, \dots, \ell\}$ with k elements **Do**
 $\mathcal{A}' \leftarrow$ subset of \mathcal{A} with all the edges with labels in C
 $T \leftarrow$ minimum cost spanning tree in $(\mathcal{N}, \mathcal{A}')$
 If $c(T) < BestCost$ **and** $l(T) = k$ **Then** $BestCost \leftarrow c(T)$; $BestT \leftarrow T$

Let T_{ND} be a set that contains candidates to non-dominated spanning trees in $(\mathcal{N}, \mathcal{A})$. The hardness of the MLSTP makes the value of l^* to be unknown in advance, however, as it is easy to obtain \hat{l} , we propose the minimum cost spanning tree to be computed for every number of labels combination, starting from \hat{l} . If for a given k no spanning tree is found when all k label subnetworks are examined, that means the optimum value of l has been found, $l^* = k + 1$, and the procedure can be halted, as no more spanning trees will be found.

Algorithm 3 *Algorithm to compute non-dominated spanning trees – minimum cost/minimum number of labels*

$T \leftarrow$ minimum cost spanning tree
 $k \leftarrow l(T)$; $T_{ND} \leftarrow \emptyset$; $continue \leftarrow \text{True}$
While $k \geq 1$ **and** $continue$ **Do**
 $T \leftarrow$ minimum cost spanning tree with k labels
 If T is defined and is not dominated **Then** $T_{ND} \leftarrow T_{ND} \cup \{T\}$
 Else If no spanning tree was found **Then** $continue \leftarrow \text{False}$
 $k \leftarrow k - 1$

It should be noticed that minimum cost spanning trees in a network with k labels might not use all these labels. Then, in a case where there exists an optimum of the cost in a network, for instance with six labels, using just four labels, we can avoid the search for trees with five labels. This enables a potential simplification of the search.

A slightly different version of Algorithm 3 can be implemented in case the best spanning tree number of labels, l^* , is known, as the computation can halt when the l^* label subnetworks have been considered. This variant is given in Algorithm 4.

Algorithm 4 *Algorithm to compute non-dominated spanning trees – minimum cost/minimum number of labels using l^**

$T \leftarrow$ minimum cost spanning tree
 $k \leftarrow l(T)$; $T_{ND} \leftarrow \emptyset$
While $k \geq l^*$ **Do**
 $T \leftarrow$ minimum cost spanning tree with k labels
 If T is defined and is not dominated **Then** $T_{ND} \leftarrow T_{ND} \cup \{T\}$
 $k \leftarrow k - 1$

In [1] Captivo, Clímaco and Pascoal introduced mixed integer linear programming formulations for the MLSTP, that have been able to solve this problem with reasonable processing times. That work reports it took about 137 seconds or 7 seconds to solve the MLSTP in 500 node and 20 label instances or 50 node and label instances, respectively, using CPLEX 11.0 on a PC Intel CoreTM2, 2.4 GHz with 2GB of RAM.

It should be noticed that both Algorithms 3 and 4 use Algorithm 2 as the underlying method for obtaining the minimum cost spanning trees with a given number of color. An alternative procedure would consist of considering an increasing sequence of the number of labels until \hat{l} , although some minimum cost spanning tree problems might have no solution. Starting from single

label subnetworks allows to access all combinations of k elements by using an auxiliary breadth first search tree with the chosen labels. Each node of the search tree should contain a new label and be associated with its predecessor. The nodes in each level of this search tree correspond to all combinations of k elements of $\{1, \dots, \hat{\ell}\}$, for some k . This process is described in Algorithm 5.

Algorithm 5 *Algorithm to compute non-dominated spanning trees – minimum cost/minimum number of labels using a search tree*

```

 $T \leftarrow$  minimum cost spanning tree in  $(\mathcal{N}, \mathcal{A})$ ;  $\hat{\ell} \leftarrow l(T)$ 
 $X \leftarrow \{\emptyset\}$ ;  $T_{ND} \leftarrow \emptyset$ ;  $x \leftarrow 0$ 
While  $x \leq \hat{\ell}$  Do
   $C \leftarrow$  element in  $X$ ;  $X \leftarrow X - \{C\}$ ;  $x \leftarrow$  number of labels in  $C$ 
  For  $k \in \{1, \dots, \ell\} - C$  Do
     $X \leftarrow X \cup \{C \cup \{k\}\}$ 
     $\mathcal{A}' \leftarrow \{\text{arcs in } \mathcal{A} \text{ with the labels in } C \cup \{k\}\}$ 
     $T \leftarrow$  minimum cost spanning tree in  $(\mathcal{N}, \mathcal{A}')$ 
    If  $T$  is defined and is not dominated Then  $T_{ND} \leftarrow T_{ND} \cup \{T\}$ 

```

4.3 Computational experiments

Algorithms 3 (A3), 4 (A4) and 5 (A5), presented in the previous subsection, have been coded in C language and been tested on a Dual Core AMD Opteron with a 1 GHz processor, 1 Mb of cache and 4 Gb of RAM, running over SUSE Linux 9.3. The data sets considered were random undirected networks with:

- $n = 100, 200, 300, 400, 500$ vertices and average degree $a = 2, 5, 20$, considering colors obtained randomly for $\ell \in \{5, 10, 20\}$;
- $n = \ell = 20, 30$ vertices and density $d = 0.2, 0.5, 0.8$ (where $m = dn(n - 1)/2$).

The edge costs were integer values uniformly generated between 1 and 100. Tables 2 and 3 present average values obtained in 30 instances for each problem dimension.

In terms of running time the version A5 using a breadth search tree to define color combinations, and increasing the number of colors, outperformed another iterative version A3 (as well as a recursive one) to determine those combinations. However, that same version exceeded the allocated memory for larger values of ℓ , therefore the results in Tables 2 and 3 were obtained with A5, while Table 4 refers to A3.

a	2			5			20		
ℓ	5	10	20	5	10	20	5	10	20
$n = 100$	1.8	1.7	9.4	3.9	7.1	11.7	4.9	9.2	18.8
$n = 500$	1.0	2.0	4.9	2.7	5.5	12.0	4.6	9.0	18.0

Table 2: Average number of non-dominated spanning trees obtained with Algorithm 5

Table 2 reports the mean number of non-dominated solutions computed for some of the problems. That number depends not only on the number of labels in the network but also on the ratio between the number of arcs and the number of vertices in the network, that is, the average degree, a . This is the reason why the number of determined solutions when $n = 500$ is smaller than when $n = 100$. However, on instances with a larger average degree it was observed that there is almost one solution per number of labels. Also the running times to find one non-dominated spanning tree for each number of labels, reported in Table 3, depended on a .

a	2			5			20		
ℓ	5	10	20	5	10	20	5	10	20
$n = 100$	0.0001	0.0065	7.4840	0.0004	0.0123	13.4028	0.0009	0.0157	16.9403
$n = 200$	0.0008	0.0177	18.5082	0.0011	0.0312	33.6627	0.0019	0.0437	43.9046
$n = 300$	0.0012	0.0324	34.3452	0.0032	0.0560	61.6145	0.0029	0.0795	80.3074
$n = 400$	0.0016	0.0509	53.0818	0.0033	0.0904	97.0057	0.0048	0.1256	130.3784
$n = 500$	0.0025	0.0840	78.2186	0.0048	0.1303	139.5706	0.0071	0.1776	180.9731

Table 3: Average running times (in seconds) for finding one non-dominated spanning tree for each number of labels, obtained with Algorithm 5

Table 4 shows mean values, again for 30 instances, considering the data sets where the number of colors is equal to the number of vertices. On these instances the search tree used for defining all label combinations grew too much, so the results presented were obtained with the iterative version for finding combinations with k elements of ℓ labels.

d	0.2	0.5	0.8
$n = \ell = 20$	0.4381	3.3230	3.7969
$n = \ell = 30$	2180.5752	6361.8350	6389.5503

Table 4: Average running times (in seconds) for finding one non-dominated spanning tree for each number of labels, obtained with Algorithm 3

Once again, although the CPU times to find the non-dominated spanning trees increased with the density of the network, the number of labels in the network edges was the determinant factor for those times. It is worth noticing that even in the 30 vertices instances it always took more than 2000 seconds to find those non-dominated solutions.

The tests mentioned above were repeated taking into account the optimal solution of the ML-STP, l^* , obtained using the formulation introduced in [1]. That information was used together with A4 and the resulting average running times over the same test bed and under the same conditions are presented in Tables 5 and 6. In every case a considerable improvement of the processing times was observed.

a	2			5			20		
ℓ	5	10	20	5	10	20	5	10	20
$n = 100$	0.0000	0.0020	2.2942	0.0000	0.0084	9.8767	0.0011	0.0117	12.340
$n = 200$	0.0000	0.0015	1.5664	0.0003	0.0173	22.5390	0.0000	0.0300	31.6215
$n = 300$	0.0004	0.0021	1.3256	0.0001	0.0253	37.1936	0.0033	0.0553	56.9374
$n = 400$	0.0003	0.0016	0.8768	0.0008	0.0380	53.5925	0.0035	0.0899	95.0802
$n = 500$	0.0004	0.0043	1.5850	0.0027	0.0671	85.5417	0.0051	0.1221	125.4363

Table 5: Average running times (in seconds) for finding one non-dominated spanning tree for each number of labels, obtained with Algorithm 3 and using l^*

d	0.2	0.5	0.8
$n = \ell = 20$	0.1836	1.2680	1.4472
$n = \ell = 30$	850.4263	2335.52173	2408.23999

Table 6: Average running times (in seconds) for finding the non-dominated spanning tree for each number of labels, obtained with Algorithm 3 and using l^*

5 Conclusions

In this paper we proposed two approaches for the calculation of the non-dominated solutions set of the bicriterion – minimum cost/minimum label – spanning tree problem. The first approach is based on a ranking algorithm according to the traditional minimum cost spanning tree model and a non-dominance test. Unfortunately the computational performance of this approach was poor. It was only possible to determine the entire set of non-dominated solutions for small size instances. So, we developed a second approach based on theoretical results presented in this paper. It is enough to calculate the minimum cost spanning tree for each l belonging to $[l^*, \hat{l}]$ (where l^* corresponds to the minimum label spanning tree and \hat{l} is the minimum number of labels of any minimum cost spanning tree(s)) and check whether some obtained solutions are dominated among them, eliminating in this case those solutions. It must be remarked that this approach can be used avoiding to check systematically whether several spanning trees are alternative non-dominated solutions with the same cost and number of labels. This is advisable because otherwise the computational cost would be high and the added information would not be very valuable in most of the cases. The computational results presented in the paper, using this version of the second procedure, are much better than those obtained with the first procedure. Problems with 500 nodes and 10 different labels are solved in less than 0.2 seconds. The computational burden increases for 20 label problems, but, even so, those problems are solved in reasonable time for off-line applications.

References

- [1] M. E. Captivo, J. Clímaco, and M. Pascoal. A mixed integer linear formulation for the minimum label spanning tree problem. Technical Report 11, INESC-Coimbra, Coimbra, August 2008. (http://www.inescc.pt/documentos/RR2008_11.pdf).
- [2] R. Cerulli, A. Fink, Gentili M, and S. Voss. Metaheuristics comparison for the minimum labelling spanning tree problem. In B. L. Golden, S. Raghavan, and E. Wasil, editors, *The Next Wave on Computing, Optimization, and Decision Technologies*, volume 29, pages 93–106. Springer US, New York, 2005.
- [3] R. Chang and S.-J. Leu. The minimum labeling spanning trees. *Information Processing Letters*, 63(5):277–282, 1997.
- [4] 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>).
- [5] H. Gabow. Two algorithms for generating weighted spanning trees in order. *SIAM Journal on Computing*, 6:139–150, March 1977.

- [6] N. Katoh, T. Ibaraki, and H. Mine. An algorithm for finding K minimum spanning trees. *SIAM Journal on Computing*, 10(2):211–247, 1981.
- [7] S. Krumke and H. Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66(2):81–85, 1998.
- [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] E. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405, 1972.
- [10] J. Clímaco and E. Martins. On the determination of the nondominated paths in a multiobjective network problem. In *Proc. of the V Symposium über Operations Research, Köln, 1980, in Methods in Operations Research*, volume 40, pages 255–258. Anton Hain, Königstein, 1981.
- [11] J. Clímaco and E. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404, 1982.
- [12] R. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
- [13] Y. Wan, G. Chen, and Y. Xu. A note on the minimum label spanning tree. *Information Processing Letters*, 84(2):99–101, 2002.
- [14] Y. Xiong. *The minimum labeling spanning tree problem and some variants*. PhD thesis, Graduate School of the Univ. of Maryland, USA, 2005.
- [15] Y. Xiong, B. Golden, and E. Wasil. Worst-case behavior of the MVCA heuristic for the minimum labeling spanning tree problem. *Operations Research Letters*, 33(1):77–80, 2005.