On a relaxed maximally disjoint path pair problem: A bicriteria approach

Marta M. B. Pascoal[(1,2)*] João C. N. Clímaco[(2)],

[(1)] CMUC, Department of Mathematics, University of Coimbra
3001-501 Coimbra, Portugal
E-mail: marta@mat.uc.pt

[(2)] Institute for Systems Engineering and Computers – Coimbra, University of Coimbra
rua Sílvio Lima, Pólo II, 3030-290 Coimbra, Portugal
E-mail: jclimaco@inescc.pt

December 2018

Abstract: In some application areas in telecommunication and transportation networks, there are problems requiring the determination of pairs of paths, aiming at minimizing the number of links, or link groups, which they share, and their total cost. In this paper it is proposed a new bicriteria algorithm to deal with this problem. The algorithm is based on ranking pairs of paths by order of the total cost, using an adaptation of a path ranking algorithm, after a suitable modification of the network topology. Non-dominated solutions are then filtered by means of a dominance test. Firstly, computational experiments are reported in order to assess the efficiency of the algorithm to calculate the whole set of non-nominated pairs of paths. Secondly, we present computational results focused on the non-dominated solutions close to the maximal disjoint pair (i.e. quasi-disjoint pairs only, for a predefined admissible relaxation value), because in some application problems, like shared risk link group pairs of paths, only those solutions have practical relevance.

Keywords: Bicriteria problems; Pairs of paths; Maximally disjoint labels; Cost.

1 Introduction

Many engineering problems, such as telecommunication and transportation network design problems, can be modeled as network shortest path problems [1], and in many situations due to reliability requirements it is advisable to choose a pair of disjoint paths (the so called "active path" and a "backup path" which can be used in case a failure occurs along the active path). It must be remarked that, in some cases, the determination of k disjoint paths (with $k > 2$) is advisable and that the required pairs can be node disjoint pairs, arc disjoint pairs, or both. Although the method proposed in this paper is dedicated to $k = 2$ and to arc disjoint paths, it can be easily extended to other cases. A review of these and other disjoint path problems can be found in [11]. In many practical situations instead of just protecting links, it is necessary to consider the so called shared risk link groups (SRLG's), such that a SRLG is a group of links sharing a common risk. Moreover, note that a link can belong to several SRLG's. In this case it is necessary to find disjoint pairs of paths regarding the SRLG's in order to avoid an interruption of service due to a single failure scenario. Of course, in some situations disjoint paths do not exist and, so, we search for maximally

*Corresponding author

disjoint pairs of paths. This type of problems is very common in optical networks, e.g. concerning Multi-Protocol Label Switching (MPLS) and Wavelength-Division Multiplexer (WDM) network problems. These type of formulations are also relevant in other related problems, as for example correlated congestion problems in transportation networks and in wireless multimedia sensor networks (WMSN) and in models involving cascading failures of power grid networks [5, 11, 12, 16, 19]. It can be noted that the arc disjoint case can be seen as a particular case of the group disjoint case, if one assigns a different group to every network arc. In [10], it is proved that the problem of computing a pair of paths, with the least number of groups in common, is NP-complete. Several heuristics and parallel processing procedures have been proposed for dealing with these problems, for instance, [15, 16, 17, 18, 19]. In many situations, such as in MPLS networks, it is also advisable to minimize a cost function (being a min-sum function, very commonly). However, for instance in cases of MPLS design problems, this is a secondary goal. So, it is justifiable the use of a lexicographic formulation regarding the two criteria, having the number of common groups as the first objective and the total cost as the second one, whenever there is a priority for the first objective in the first. For instance, in [8] two heuristics are presented for handling this type of problems and an exact approach is proposed in [6]. Besides the two above referred to functions, it is common the introduction of an upper bound to the total number of arcs of the pair of paths and, in some cases, it is justifiable aiming at minimizing this number of arcs. In the method proposed in this paper it is very simple to introduce such a constraint. Finally, in some application cases, for instance in some transportation problems, it is of practical interest to consider the whole set of non-dominated solutions for the above referred to problem, or, at least, to find all the non-dominated solutions until a fixed upper bound to the cost function.

In this paper we address the determination of pairs of paths between two network nodes, with two goals:

- the minimization of the cost of the two paths,
- the minimization of the number of groups that they both have in common,

and we also propose a method for finding the set of the non-dominated solutions of the problem.

Computational results are presented and discussed for finding the set of non-dominated solutions. Secondly, we present computational results focused on the non-dominated solutions close to the maximal disjoint pair (i.e. quasi-disjoint pairs only, for a predefined admissible relaxation value), because, in some application problems, those solutions may have special practical relevance. Thirdly, the study is repeated by replacing the minimization of the cost function by the minimization of the total number of arcs of the pair of paths.

The rest of this text is organized into four parts. In Section 2 notation and the problem definition are introduced. The following sections are dedicated to the proposed solution method and to the presentation of computational tests for assessing its performance under the scenarios described before. Finally, conclusions are drawn in Section 5.

2 The shortest - maximally label disjoint pairs of paths problem

Let us consider the following definitions. Let $G = (N, A)$ denote a network, where N is the set of n nodes and $A \subseteq N \times N$ is the set of m arcs. Let $p = \langle v_1, \ldots, v_r \rangle$, where $(v_i, v_{i+1}) \in A$ for $i = 1, \ldots, r-1$, be a path from v_1 to v_r in G. Given $s, t \in N$, the source and the terminal nodes, and P denotes the set of paths in G from s to t.

Let L be the set of network groups, also called labels, and $(i, j) \in A$ be an arc. Then, two values are associated with the arc (i, j):

- $c_{ij} \in \mathbb{R}_0^+$, which represents the cost for using the arc (i, j), and
- $l_{ij} \in L$, which represents the label of the arc (i, j).

For a path $p \in P$, its cost and the set of its arc labels are defined by

$$c(p) = \sum_{(i,j) \in p} c_{ij} \quad \text{and} \quad l(p) = \cup_{(i,j) \in p} \{l_{ij}\},$$

respectively. Such notions can be extended to pairs of paths in P. Given $(p, q) \in P \times P$,

- the pair's cost is given by $c(p, q) = c(p) + c(q)$, and
- the number of labels which are common to both paths is given by $l(p, q) = |l(p) \cap l(q)|$.

The shortest and maximally label disjoint pairs of path (SMLPP) problem can be modeled as a biobjective problem that aims at finding pairs of paths which optimize the cost and the number of common labels previously defined, that is,

$$\begin{array}{ll} \min & c(p, q) \\ \min & l(p, q) \\ \text{such that} & p, q \in P \end{array}$$

In general these two objective functions are conflituous, therefore there is no pair of paths which optimizes both simultaneously. Instead, the solutions for the SMLPP problem are non-dominated pairs of paths from s to t, defined as follows. A pair of paths (p, q) is said to be non-dominated if and only if there is no other pair of paths $(p', q') \in P \times P$ that dominates it, that is, such that

$$\begin{cases} c(p', q') \leq c(p, q) \\ l(p', q') \leq l(p, q) \end{cases}$$

with $c(p', q') \neq c(p, q)$ and/or $l(p', q') \neq l(p, q)$. Thus, the goal of the SMLPP problem is to find the set of non-dominated pairs of paths in P. A review on multicriteria path problems can be found in [4].

3 Solution method

The method proposed in the following, for finding the set of non-dominated solutions for the SMLPP problem results from answering to two main questions: how to find pairs of paths from s to t, and how to handle the functions c and l from a biobjective perspective. The first of these points is addressed by adapting a modification of the graph G, proposed in [3]. The second is an adaptation of the ranking method introduced in [2] for the biobjective shortest path problem. The two procedures are now briefly outlined.

The first step of our algorithm consists of transforming the given network into another one that allows representing a pair of paths between two nodes as a single path. Then, the given network, $G = (N, A)$, is transformed into a new network, $G' = (N', A')$, such that

- each node $i \in N$ is duplicated as the node i',
- each arc (i, j) is duplicated as the arc (i', j'), and
- a new arc is added that links node t to node s', the arc (t, s').

Thus, the new sets of nodes and arcs are

$$N' = N \cup \{i' : i \in N\}$$

and

$$A' = A \cup \{(i', j') : (i, j) \in A\} \cup \{(t, s')\}.$$

The initial node of network G' is the former initial node, s, whereas the new terminal node is node t'.

The values associated with the arcs of G are extended naturally, in order to maintain the cost and the labels of the paths in the original network representation. Thus, arc costs and the labels of the former network, G, are maintained. Moreover, the costs and labels for the duplicated arcs are

$$c_{i'j'} = c_{ij} \quad \text{and} \quad l_{i'j'} = l_{ij},$$

for any $(i, j) \in A$. Furthermore, $c_{ts'} = 0$ and $l_{ts'} = x$, where x is an extra label, such that $x \notin L$. These operations are summarized in Algorithm 1.

Algorithm 1: Duplicate the network $G = (N, A)$

1 $N' \leftarrow N \cup \{i' : i \in N\}$
2 $A' \leftarrow A \cup \{(i', j') : (i, j) \in A\} \cup \{(t, s')\}$
3 $s \leftarrow s$; $t \leftarrow t'$
4 **for** $(i, j) \in A$ **do** $c_{i'j'} \leftarrow c_{ij}$; $l_{i'j'} \leftarrow l_{ij}$
5 $c_{ts'} \leftarrow 0$; $l_{ts'} \leftarrow x$, for some $x \notin L$

The new network G' has $2n$ nodes and $2m + 1$ arcs. In practice the duplication of the arcs can be avoided because their information is still stored in the original network. Thus, it is enough to store $2n$ nodes and $m + 1$ arcs only.

Assume that the symbol $\diamond$ stands for the concatenation of two paths. That is, given a path q with the same terminal node as the initial node of another path r, then $q \diamond r$ is the path that results from following the sequence r, q. The next result follows directly from the definitions above.

Lemma 1 *Any path p from s to t' in the network G' has the form: $p = q \diamond \langle t, s' \rangle \diamond r'$, with q a path from s to t and r' a path from s' to t'. Moreover,*

$$c(p) = c(q) + c(r') \quad \textit{and} \quad l(p) = |l(q) \cap l(r')|.$$

Lemma 1 shows that every path from s to t' in G' corresponds to a pair of paths from s to t in G, q and r, where r is the path obtained from r' by replacing any node i' by its original node i.

The duplication of the network G allows to represent pairs of paths from s to t in G as paths from s to t' in G', which transforms the SMLPP problem into a biobjective path problem with c and l as the objective functions.

[2] introduced an algorithm for finding the non-dominated solutions of the biobjective shortest path problem. Their method is based on ranking paths by non-decreasing order of one of the two objective functions, say c_1 or c_2. This originates a sequence of solutions from which the non-dominated ones can be selected, $\{p_i\}_{i=1,\ldots,k}$. Assuming that the solutions are ranked by order of c_1, then the subsequence of non-dominated solutions is non-decreasing in c_1 but non-increasing in c_2. That is,

$$\begin{cases} c_1(p_i) < c_1(p_{i+1}) \\ c_2(p_i) > c_2(p_{i+1}) \end{cases} \quad \text{or} \quad \begin{cases} c_1(p_i) = c_1(p_{i+1}) \\ c_2(p_i) = c_2(p_{i+1}) \end{cases} \tag{1}$$

In fact, when ranking paths according to c_1, either $c_1(p_i) < c_1(p_{i+1})$ or $c_1(p_i) = c_1(p_{i+1})$. In the first case, if $c_2(p_i) \leq c_2(p_{i+1})$ then p_i dominates p_{i+1}. In the second, p_{i+1} dominates p_i if and only if $c_2(p_i) > c_2(p_{i+1})$. Thus, the conditions (1) follow from the fact that p_i is non-dominated. The first and the last elements of the sequence are obtained by solving the shortest path problem with respect to c_1 and to c_2, respectively. In cases where there are multiple optimal solutions with respect to c_1, the first one may be dominated. Then, the ranking method computes the alternative solutions and the dominance test discards the first one, provided that one that dominates it is found. This is also not a problem for the last solution. Let c_2^* be the best value, for c_2, that a pair of paths in G can have. Then, the last solution to be considered is the shortest for c_1 that has a c_2 value of c_2^*, therefore it is not dominated. The remaining elements are calculated by means of a ranking shortest path algorithm, applied to function c_1.

Contrary to the standard biobjective shortest path problem, while the objective function c in the SMLPP problem is an additive cost function, however, the function l is more difficult to handle, as shown by [10]. Still, a ranking algorithm can be applied for finding paths in G', according to c, and the result below follows easily, using the reasoning above.

Lemma 2 *Let $\{p_i\}_{i \geq 1}$ be the sequence of non-dominated paths from s to t' in G' with respect to (c, l). Then, these paths can be arranged in a way that satisfies*

$$\begin{cases} c(p_i) < c(p_{i+1}) \\ l(p_i) > l(p_{i+1}) \end{cases} \quad \textit{or} \quad \begin{cases} c(p_i) = c(p_{i+1}) \\ l(p_i) = l(p_{i+1}) \end{cases}$$

As a consequence, paths in G', that is, pairs of paths in G, can be ranked by order of c, and the dominance test proposed by Clímaco and Martins can prune those which are dominated. The dominance test consists of comparing the objective function values of each current solution with the objective function values of the latest non-dominated solution candidate. At a given step of the method, say step k, let m_c denote the greatest cost of the last path computed in G' until step k and let M_l denote the smallest number of common labels of the pairs of paths computed until step k. Because paths are ranked according to c, their costs never decrease. Then, given a new path p in G', the algorithm proceeds as follows:

- If $c(p) = m_c$ and $l(p) = M_l$, then p is added to the set of candidates to non-dominated solutions.
- If $c(p) = m_c$ and $l(p) < M_l$, then the candidate solutions are dominated. The path p is a new candidate to non-dominated solution.
- If $c(p) = m_c$ and $l(p) > M_l$, then the path p is dominated. The set of candidates to non-dominated solutions remains unchanged.
- If $c(p) > m_c$ and $l(p) = M_l$, then the path p is dominated and the current solutions, in the set of candidates, are non-dominated.
- If $c(p) > m_c$ and $l(p) < M_l$, then the current solutions, in the set of candidates, are non-dominated. The path p is a new candidate to non-dominated solution.
- If $c(p) > m_c$ and $l(p) > M_l$, then the current solutions, in the set of candidates, are non-dominated. The path p is also dominated.

Additionally, the values of m_c and M_l need to be updated during the process, whenever $c(p) < m_c$ or $l(p) < M_l$.

The algorithm starts by computing the shortest path in G' with respect to c, say p, which is then used to initialize m_c and M_l ($m_c = c(p)$, $M_l = l(p)$). Afterwards, the algorithm continues and other paths in G' are ranked by order of c. The non-dominated solutions are filtered as paths are ranked, using the test outlined above.

The halting condition of the former algorithm is the following: the algorithm stops when during the ranking, the cost (c_1) of the current path is greater than the cost (c_1) of the path which optimizes the second function (c_2). If there are alternative optima concerning c_2, it will be considered the minimal cost (c_1) corresponding to these alternative optima. This procedure cannot be repeated in the case of the SMLPP problem, given the hardness of that particular subproblem. Instead, the paths are ranked until all solutions have been computed or an acceptable pair of paths with respect to the number of shared labels, has been found. In this respect it can be noted that in applications to telecommunications it may be considered reasonable to use solutions that are either label disjoint or have few common labels, for instance up to 2 (0 common labels means that the two paths are label disjoint).

Finally, any algorithm for ranking shortest paths can be used for generating solutions of the SMLPP problem, for instance [7, 13, 14]. In our implementation we have used the MPS algorithm, a deviation algorithm proposed by Martins, Pascoal and Santos in [13].

The proposed method is outlined in Algorithm 2. The sequence of paths generated by the ranking algorithm is denoted by $\{p_k\}$ and satisfies $c(p_k) \leq c(p_{k+1})$, for $k \geq 1$. Moreover, the algorithm uses two auxiliary sets, set P_X and set P_N. The first is used to store paths in G' that are temporary candidates to non-dominated paths. The latter is a set that stores the non-dominated paths in G', that is, pairs of paths in G.

Algorithm 2: Finding the non-dominated SMLPPs

```
1  G' ← Duplicate the network G                          // Network modification
2  p*_c ← shortest pair of paths from s to t' with respect to c in G'
3  m_c ← c(p*_c); M_l ← l(p*_c)
4  k ← 0
5  P_N ← ∅; P_X ← {p*_c}
6  while there are paths left to rank and a stopping condition is not met do
7      k ← k + 1
8      p_k ← k-th shortest path from s to t' with respect to c in G'
9      // Dominance test
10     if c(p_k) = m_c then
11         if l(p_k) = M_l then P_X ← P_X ∪ {p_k}
12         if l(p_k) < M_l then
13             M_l ← l(p_k)
14             P_X ← {p_k}
15     else
16         if l(p_k) < M_l then
17             m_c ← c(p_k); M_l ← l(p_k)
18             P_N ← P_N ∪ P_X; P_X ← {p_k}
```

The implementation of Algorithm 2 will be denoted by `SMLPP`. Three other versions of this method were considered, as described next.

Version `SMLPP1` Every two paths from s to t in G, say q and r, correspond to two pairs of paths, (q, r) and (r, q). A condition was imposed in this new version of the algorithm to prevent the calculation of the two pairs of paths, which forces them to satisfy $c(q) \leq c(r)$. Therefore, a path $q \diamond \langle t, s' \rangle \diamond r'$ in G' is discarded before being stored whenever

$$c(q) > c(r').$$

It should be remarked that this measure does not fully avoid path pair repetitions, given that cases such that $c(q) = c(r)$ can still occur. However, such a filter would be more demanding in terms of operations.

Version `SMLPP2` Assume that m_c and M_l have been updated at some step of Algorithm 2, and let $p = q \diamond \langle t, s' \rangle \diamond r'$ be a path selected after that update. By definition $c(p) \geq m_c$, therefore $l(p) > M_l$ implies that p is dominated. When using a deviation algorithm [13] the generation of a new path can be avoided as long as it contains a partial path that has more than M_l common labels.

Version `HMLPP` The proposed `SMLPP` method can be used to solve an application, where the minimization of the hops (number of arcs) between the given nodes is sought [9]. This problem will be designated as the bicriteria minimum hop-maximally label-disjoint path pairs (HMLPP) problem. This will be treated as a particular case of the previous problem, where the arc costs are $c_{ij} = 1$, for any $(i, j) \in A$.

4 Computational experiments

The methods presented earlier, `SMLPP`, `SMLPP1`, `SMLPP2` and `HMLPP`, were coded in C language and were tested on an Intel® i7-6700 Quad core, with 8Mb of cache, a 3.4 GHz processor and 16 Gb of RAM. The codes ran over openSUSE Leap 42.2

As mentioned earlier, the MPS algorithm [13] is used in order to rank the pairs of paths in G. This is a ranking deviation algorithm, which uses an auxiliary set X of candidates to the next best solution, p_k, for some $k \geq 1$. The set X is initialized with the path p_1. Afterwards, the path p_k is selected as the shortest path in X, and is scanned in order to generate new candidate paths. These candidates deviate from p_k at one of its nodes and are selected so that they have the smallest possible cost. The new candidate paths are stored in set X, and the process is repeated while necessary. A bound of 7 000 000 pairs of paths was imposed as the maximum number of candidates generated by the algorithm for the experiments reported below. This value can be increased (decreased), which results in an increase (decrease) of the run times and of the number of problems solved until the end, as explained in the following.

Three sets of experiments were performed. One for assessing and comparing the codes `SMLPP`, `SMLPP1`, `SMLPP2` for the SMLPP problem, and another for studying the behavior of the codes when the sought pairs of paths are allowed to have at most $\Delta = 2$ labels in common. The last set of tests is dedicated to the case where the number of hops replaces the cost objective function, the HMLPP problem.

The instances are randomly generated for a given number of nodes n and a given number of arcs m. The initial and terminal nodes are $s = 1$ and $t = n$, respectively. First a set of arcs linking the n nodes is created, $(1, 2), (2, 3), \ldots, (n - 1, n)$, in order to ensure the existence of at least one path from s to t. The extreme nodes of the remaining $m - n$ arcs are then randomly selected. Both the arc costs and the arc labels are uniformly obtained within intervals defined by the user. The generator used in the experiments was adapted from the code available at http://mat.uc.pt/~eqvm/cientificos/fortran/Codigos/spantree.f.Z. The new version associates two values with each arc, its cost and its label, rather than just one.

Test set I The first set of experiments is divided in two randomly generated instances of the SMLPP problem.

- The first part, Set I.A, consists of networks with $n = 1\,000, 2\,000, 5\,000$ nodes, average degree $\delta = m/n$, with $\delta = 2, 5, 10$, and maximal number of arc labels of $|L| = 3, 5, 10, 15, 20$.
- The second part, Set I.B, considers smaller instances compatible with realistic network topologies. These instances have $n = 10, 20, 30, 40, 50$ nodes, average degree $\delta = 2, 4, 6$, and the same number of arc labels as the previous set.

In both cases, for each arc $(i, j) \in A$, the cost was uniformly generated in $c_{ij} \in \{1, \ldots, 100\}$, and the label was uniformly generated in $l_{ij} \in \{1, \ldots, |L|\}$. The results presented in the following are mean values obtained for 10 different seeds, for each set of parameters.

Table 1: Problems of Set I.A solved until the end (%)

	$\delta = 5$					$\delta = 10$					$\delta = 20$				
$n\backslash\|L\|$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20
1 000	20	10	50	90	60	40	70	60	90	90	60	60	100	100	90
2 000	20	50	80	90	90	0	70	50	90	80	30	50	100	90	100
5 000	0	30	80	60	80	10	40	90	100	100	10	70	100	90	100

Table 2: Mean number of non-dominated solutions in Set I.A

	$\delta = 5$					$\delta = 10$					$\delta = 20$				
$n\backslash\|L\|$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20
1 000	2.7	4.4	4.5	2.7	3.1	3.0	3.4	4.2	4.4	4.5	4.3	5.3	5.0	5.1	4.2
2 000	3.2	3.7	5.0	5.2	3.7	3.5	5.7	5.2	5.8	4.2	4.0	5.4	5.3	6.9	4.7
5 000	2.7	2.4	3.5	4.7	3.4	2.8	4.2	4.8	5.5	5.2	4.5	4.2	4.5	5.3	3.7

The first purpose of the tests in Set I.A was to assess the empirical performance of the introduced methods, when applied to the previous set of instances. The first remark to the results is that the required run times of the three codes varied a lot for all instances – see Figure 1. In fact, part of the problems were solved very fast, whereas others could only be solved partially because of the memory limitations that have been imposed. The percentage of problems solved until the end (the same for all three codes) is summarized in Table 1. It can be noted that the denser instances and with a bigger number of labels, seem to be easier to solve than the smaller instances with fewer labels. Although not intuitive, this has to do with the algorithm's stopping condition, because in bigger networks it is easier to find pairs of fully label disjoint paths, and thus halt the method very quickly. On the other hand, Table 2 shows that the number of non-dominated solutions for these instances is fairly small. The average values are between 2.4 and 6.9 solutions.

The mean run times for the three codes are presented in Figure 1. In general they are small, at most 30 milliseconds, and increase with n and decrease with $|L|$ and δ. The general tendency

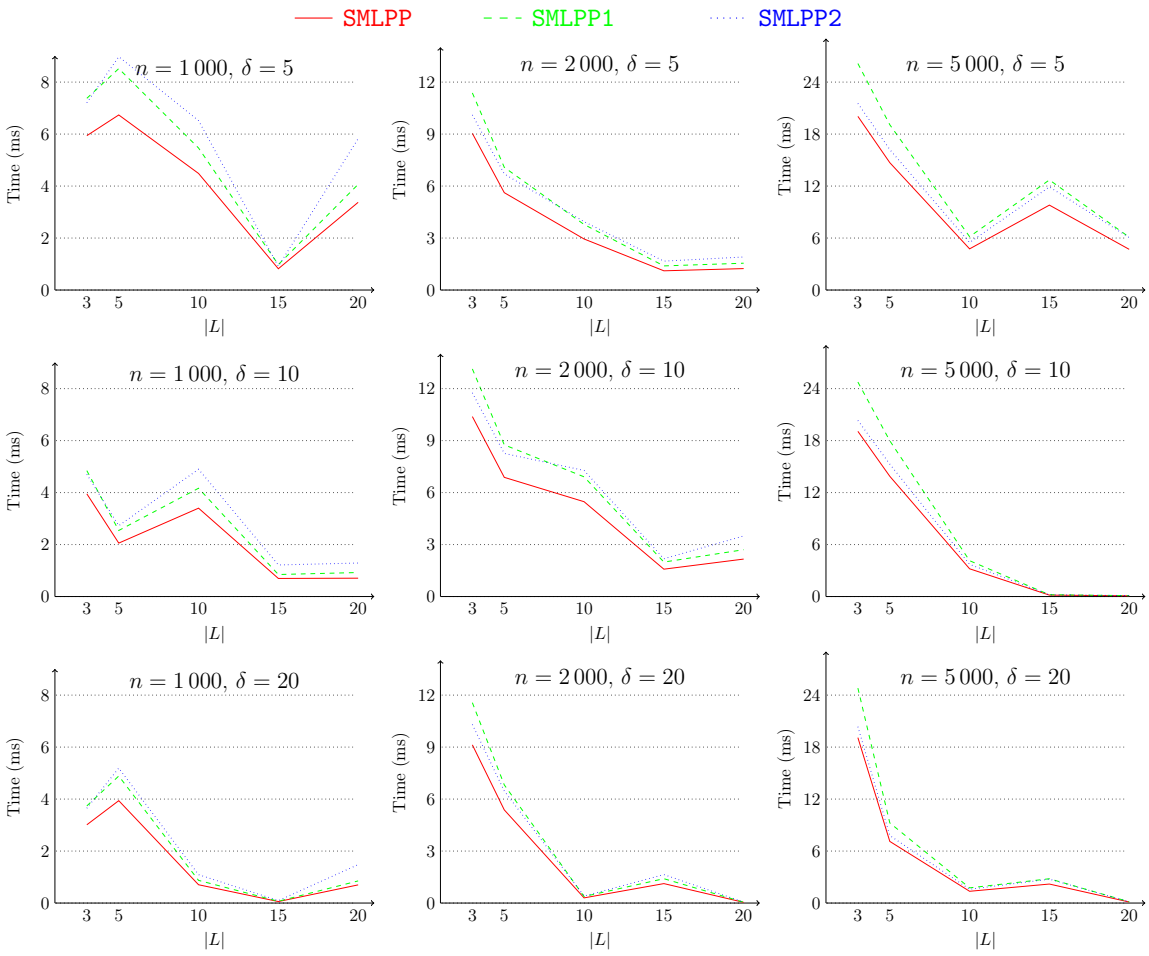

Figure 1: Mean run times for the codes `SMLPP`, `SMLPP1`, `SMLPP2` in Set I.A

is that the code `SMLPPP` runs faster than the other two codes, nevertheless, the results of all three codes behave similarly and are very close.

Table 3: Mean number of generated solutions for the problems solved until the end in Set I.A

	$\delta = 5$					$\delta = 10$					$\delta = 20$				
$n\backslash\|L\|$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20
	17072	1829455	2966760	3743	4027	1777135	1051208	2284922	8445	91021	888435	2324251	768438	56238	10522
1 000	27959	1801220	1253720	3631	3920	467798	201285	704680	8441	90798	888297	2324246	768438	56238	10522
	27957	**1800714**	**1234061**	**3470**	**3840**	**467772**	**201202**	**658315**	**8068**	**84251**	**888001**	**2320932**	**741748**	**52564**	**10133**
	627503	753032	520080	21354	3716	*	3176545	25373	256351	29107	2709106	259781	215395	51233	21870
2 000	461136	655567	516273	20802	3626	*	3174804	25250	256337	29107	2631655	259780	215395	51233	21870
	461027	**655460**	**503936**	**20233**	**3568**	*	**3173848**	**25111**	**243633**	**27961**	**2630179**	**259571**	**211170**	**49825**	**20696**
	*	1	192601	730310	41714	284	1307067	684601	54781	27681	202374	538472	521776	36858	27778
5 000	*	1	185794	724523	41687	284	1306730	684597	54765	27680	202374	538202	521751	36858	27778
	*	1	**173249**	**711521**	**41064**	284	**1306231**	**668449**	**53614**	**26953**	**202252**	**537771**	**514607**	**34524**	**26304**

*: All problems were solved until the end

Table 3 presents the number of pairs of paths that had to be generated for solving the given instances, considering only those that were solved until the end. The reason is to prevent the mean values from being too biased by the maximum allowed number of generated solutions. The values that correspond to the code(s) that produced the least number of pairs of paths are shown in bold on Table 3. As mentioned earlier, some instances required the computation of many of these solutions, while others were solved in very few iterations, which results in a visible variation of the average values on Table 3. The values vary from one code to another but no code prevails over the others with respect to the number of computed solutions. Nevertheless, the code `SMLPP2` seemed to be successful in generating less pairs of paths than `SMLPP` and `SMLPP1`, although this is not reflected in the run times.

As shown in Figure 5, the run times for solving these instances were slightly bigger than for the former tests, but were still small and did not exceed 30 milliseconds, in average. Moreover, there seems to be a tendency of the code `SMLPP` to outperform the others for the sparser graphs, and of the code `SMLPP2` for the denser graphs. Still, all the run times were fairly close to each other and none of the codes always dominated the other two. It is also worth noticing that often an increase in the maximum number of labels, $|L|$, corresponds to instances that are faster to solve, which matches the fact that most of these problems were solved until the end.

Table 4: Problems of Set I.B solved until the end (%)

	$\delta = 2$					$\delta = 4$					$\delta = 6$				
$n \backslash \|L\|$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20
10	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
20	100	100	100	100	100	60	60	70	70	80	100	90	100	90	80
30	100	100	100	100	100	60	70	80	70	100	60	80	100	80	100
40	100	100	100	100	100	50	50	70	50	50	70	60	80	80	90
50	100	100	100	100	100	10	40	60	60	80	10	90	90	80	80

Because on the first part of the experiments the bigger problems were easier to solve than the smaller ones, the second part was focused on much smaller instances (trying to test realistic sizes in applications). The percentage of problems solved until the end is shown in Table 4, whereas the mean run times of the codes `SMLPP`, `SMLPP1` and `SMLPP2` are depicted in Figures 2 to 4. More of these instances are solved until the end than for the Set I.A. Additionally, most of the instances in Set I.B are solved until the end. In particular, all the pairs of paths have been determined for the networks with $n = 10$ nodes and with average node degree $\delta = 2$. Moreover, the run times for Set I.B are smaller than for the previous cases, which stems from the fact that the topologies are easier to handle due to the drastic decrease of the sizes. In this case none of the three codes dominates the others with respect to the run time.

Test set II As mentioned earlier, in many applications full label disjointness of the two paths is wished but not mandatory. It may also be the case that no such paths exist, but a solution is needed nevertheless. In both cases, pairs of paths with few common labels can still be useful.

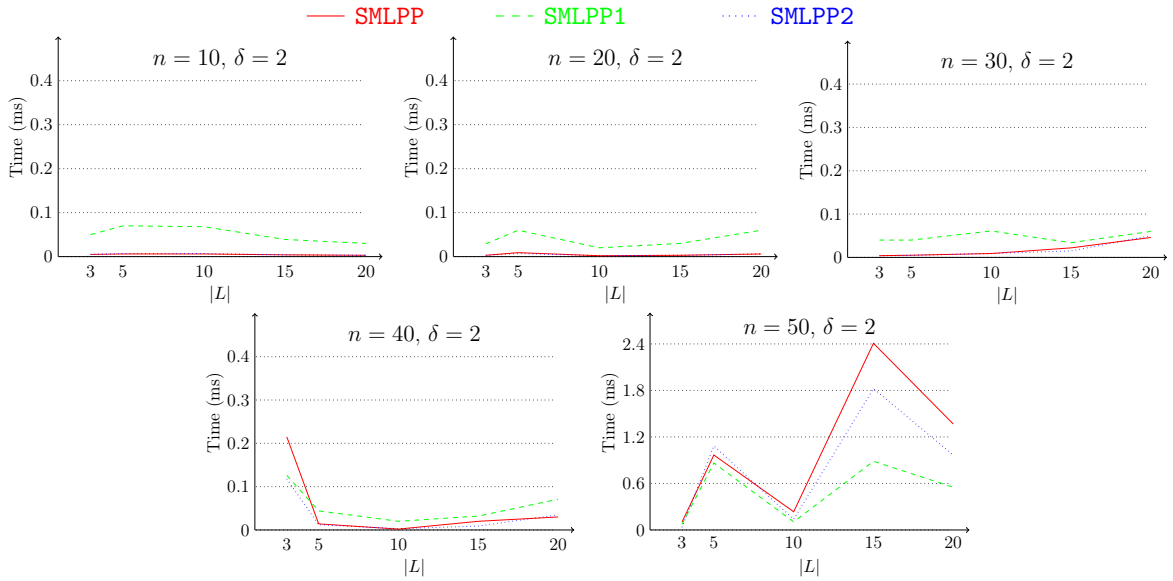

Figure 2: Mean run times for the codes `SMLPP`, `SMLPP1`, `SMLPP2` on Set I.B, $\delta = 2$

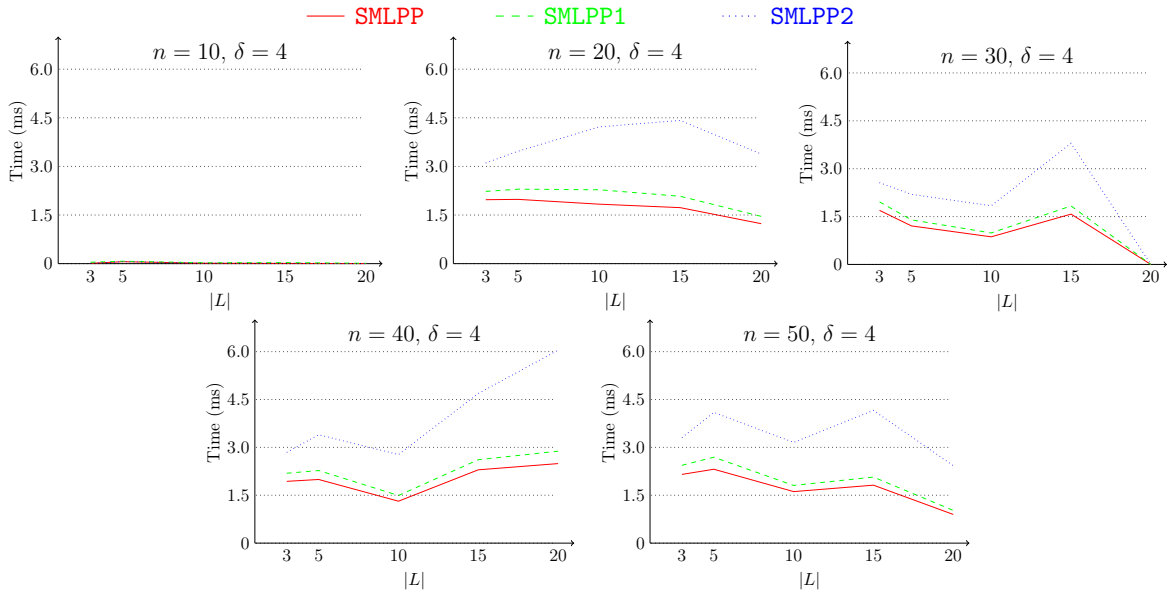

Figure 3: Mean run times for the codes `SMLPP`, `SMLPP1`, `SMLPP2` on Set I.B, $\delta = 4$

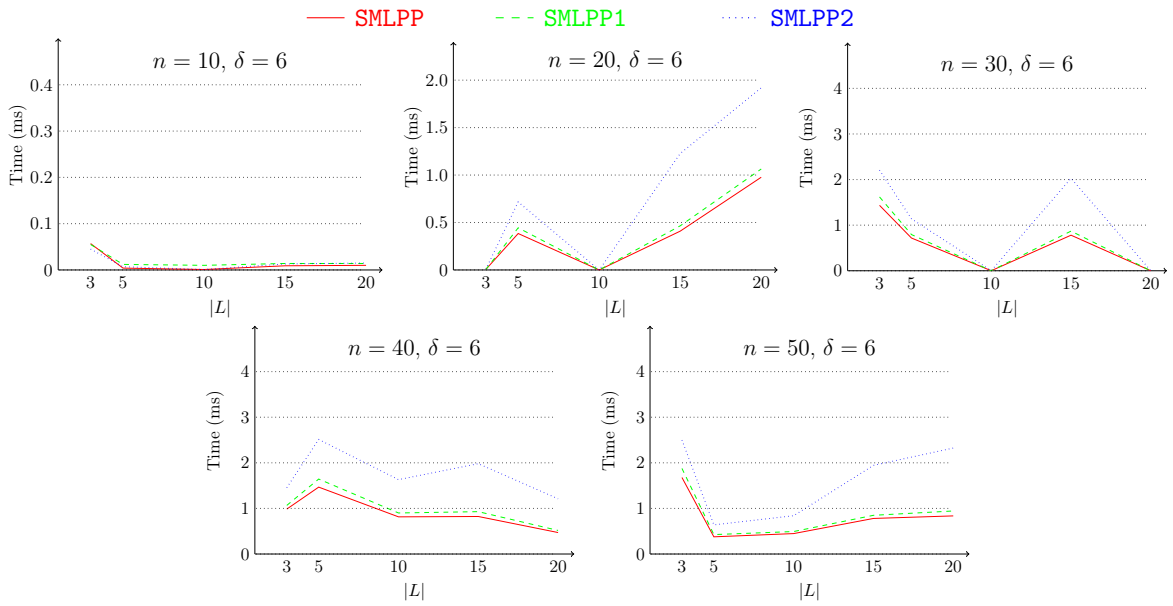

Figure 4: Mean run times for the codes `SMLPP`, `SMLPP1`, `SMLPP2` on Set I.B, $\delta = 6$

Therefore, the next results focus on the ability of the algorithms to find pairs of paths with at most $\Delta = 1, 2$ labels in common for the same instances, in particular for those that could not be solved until the end. Table 5 presents the least number of common labels of the solutions obtained by code `SMLPP` for those instances. The average value for these least numbers of common labels is either 1, for the cases where all the last pairs of paths that were found have one label in common, or below 1.5. Besides, for these tests the number of path common labels was 2 for 5% of the instances with $n = 1\,000$, and 7% of the instances with $n = 5\,000$. For the remaining cases pairs of paths with a single label in common were found. Table 6 summarizes the run times to compute pairs of paths with 1 or 2 labels in common, again for the instances that could not be solved until the end. It is also worth noting that some of the problems with solutions with 1 common label may also have solutions with 2 common labels. The average run times in Table 6 when $\Delta = 2$ include these cases, as well as those instances for which the last solution has 2 common labels, which are significantly harder to solve than the former. According to Table 6, solutions with $\Delta = 1, 2$ common labels could be found in less than 24 milliseconds. These times are compatible with practical applications of the method.

Table 5: Mean least number of common labels using `SMLPP` for problems that were not solved until the end in Set I.A

	$\delta = 5$					$\delta = 10$					$\delta = 20$				
$n \backslash \|L\|$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20
1 000	1.1	1.2	1.5	1.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	*	*	1.0
2 000	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	*	1.0	*
5 000	1.2	1.5	1.5	1.3	1.0	1.0	1.5	1.0	*	*	1.2	1.0	*	1.0	*

*: All problems were solved until the end

Table 6: Mean run times (in milliseconds) of SMLPP for finding solutions with 1 or 2 common labels for problems that were not solved until the end

	$\delta = 5$					$\delta = 10$					$\delta = 20$					
$n \backslash \lvert L \rvert$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20	Δ
1 000	6.766	6.870	7.771	—	8.991	0.002	0.002	0.002	0.007	0.003	0.005	0.007	*	*	0.008	2
	7.506	7.467	7.436	8.144	7.828	6.416	6.481	7.339	6.917	6.305	6.251	6.618	*	*	6.878	1
2 000	0.003	0.004	0.003	0.068	0.002	0.008	0.008	0.007	0.006	0.007	0.013	0.013	*	0.017	*	2
	11.083	10.027	11.632	10.783	12.287	10.246	11.174	10.499	12.405	10.646	10.452	10.825	*	10.522	*	1
5 000	18.598	21.803	21.670	0.008	23.536	0.020	0.020	0.026	*	*	18.886	0.044	*	0.046	*	2
	20.385	20.260	21.643	21.841	22.083	21.165	20.604	13.710	*	*	22.471	19.866	*	20.681	*	1

—: No such solutions were found

*: All problems were solved until the end

Still having in mind the calculation of pairs of paths with a limited number of labels in common, on a second part of the empirical experiments two variants of the code `SMLPP2` were implemented:

- The code `SMLPPFix`, which prevents the calculation of pairs of paths that can only generate solutions with more than Δ common labels. These solutions are not interesting for this problem. As the MPS algorithm uses deviation paths, a subpath starting in s in the duplicated network that has more than Δ common labels surely leads to a pair of paths with more than Δ common labels as well.

- The code `SMLPPFixRank`, which starts by ranking pairs of paths without testing their dominance until the first one with at most Δ common labels, and then resumes as `SMLPPFix`. In this case the first pair of paths that can be non-dominated has at most Δ common labels, therefore the dominance test is not needed until such a solution is found.

Tests were run for the previous instances and considering $\Delta = 2$.

The percentage of problems that were solved until the end under the previous memory constraints, was the same as shown in Table 1. Although different numbers of pairs of paths are generated by the three codes, the attempt to speed up the original code was not fully well succeeded, as all the codes presented very similar results in terms of the run time. For this reason the run times are now shown here.

Test set III A third set of experiments was dedicated to the HMLPP problem. By definition the arc costs are $c_{ij} = 1$, for any $(i, j) \in A$, and one of the goals of the problem is to minimize the number of path hops. The used networks were those in Set I.A. Additionally, like for the previous instances, each arc label was uniformly generated in $l_{ij} \in \{1, \ldots, |L|\}$, for any $(i, j) \in A$, with $|L| = 3, 5, 10, 15, 20$.

Table 7: Problems solved until the end (%) for the HMLPP problem

	$\delta = 5$					$\delta = 10$					$\delta = 20$				
$n \backslash \|L\|$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20
1 000	40	30	60	90	60	90	90	80	100	100	100	100	100	100	100
2 000	40	80	80	90	90	90	100	60	100	80	100	100	100	100	100
5 000	20	80	80	70	80	90	100	90	100	100	100	100	100	100	100

Table 8: Mean number of non-dominated solutions for the HMLPP problem

	$\delta = 5$					$\delta = 10$					$\delta = 20$				
$n \backslash \|L\|$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20
1 000	23.7	20.0	14.3	3.2	15.6	16.5	10.8	43.8	50.0	49.4	41.6	14.5	56.0	84.8	55.9
2 000	4.5	23.8	11.0	11.2	14.9	13.3	18.2	17.4	34.3	43.5	23.6	48.3	39.4	9.6	45.5
5 000	19.6	8.3	12.8	7.0	22.7	33.9	11.1	8.0	25.0	20.3	83.5	11.9	130.9	35.1	162.3

Table 9: Mean least number of common labels using `SMLPP` for problems that were not solved until the end

	$\delta = 5$					$\delta = 10$					$\delta = 20$				
$n \backslash \lvert L \rvert$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20
1 000	1.2	1.1	1.3	1.0	1.5	1.2	1.1	1.0	*	*	*	*	*	*	*
2 000	1.0	0.5	1.0	1.0	1.0	1.0	*	1.0	*	1.0	*	*	*	*	*
5 000	1.0	1.5	1.0	1.0	1.5	1.0	*	1.0	*	*	*	*	*	*	*

*: All problems were solved until the end

According to Table 7 more instances of the HMLPP problem could be solved until the end than for the general SMLPP problem, although Table 8 shows that the mean number of non-dominated pairs of paths is bigger for the HMLPP problem than for the SMLPP problem. Also, this is more likely to happen for the denser instances with more different labels.

As shown in Figure 5, the run times for solving these instances were sometimes slightly bigger and more unstable than for the former tests, but were still small and did not exceed 30 milliseconds, in average. Moreover, there seems to be a tendency of the `SMLPP` code to outperform the others, specially for the sparser graphs. Still, all the run times were fairly close to each other. It is also worth noticing that, although there are some exceptions, often an increase in the maximal number of labels, $|L|$, corresponds to instances that are faster to solve. This matches the fact that most of these problems were solved until the end. The differences are probably due to the fact that in the HMLPP problem the arc costs are all equal and, thus, the influence of the network topology can prevail.

An analysis of the least number of labels in common for the pairs of paths that could be computed for the problems interrupted due to memory constraints is given in Tables 9 and 10. According to Table 9, the average number of labels in common for the last pair of paths in problems that were not solved until the end is either 1.0 or below 1.5, given that 1 was the number obtained for most of these instances. Among them there were 4% of the instances with $n = 1\,000$ with a last solution with 2 common labels. This percentage was of 1% of the instances when $n = 5\,000$.

In terms of the average run time required to output these solutions, Table 10 shows that pairs of paths with 1 label in common were found in less than 14 milliseconds, whereas pairs of paths with 2 common labels were found in less than 13 milliseconds.

Table 10: Mean run times (in milliseconds) solved until the end of SMLPP for finding solutions with 1 or 2 common labels for problems that were not solved until the end

	$\delta = 5$					$\delta = 10$					$\delta = 20$					
$n \backslash \lvert L \rvert$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20	Δ
1 000	2.771	2.545	3.128	—	2.830	—	0.001	—	*	*	*	*	*	*	*	2
	2.686	2.705	2.395	2.834	2.767	3.269	3.215	3.096	*	*	*	*	*	*	*	1
2 000	0.000	0.001	—	—	—	—	*	0.001	*	—	*	*	*	*	*	2
	4.624	4.408	4.854	4.792	5.297	6.056	*	5.612	*	5.723	*	*	*	*	*	1
5 000	0.001	11.055	—	0.001	12.598	—	*	0.002	*	*	*	*	*	*	*	2
	9.305	8.664	10.123	10.028	12.082	13.034	*	10.261	*	*	*	*	*	*	*	1

—: No such solutions were found

*: All problems were solved until the end

5 Conclusions

This work addressed the problem of finding pairs of paths that are group disjoint as much as possible, while minimizing the cost. A bicriteria approach to this problem (the SMLPP problem) was introduced. The approach consists of a modification of the network topology, followed by the application of an algorithm that computes the non-dominated pairs of paths with respect to the two objective functions. The algorithm ranks pairs of paths according to the cost function, while discarding the dominated solutions. The method also handles well a common objective function, the number of hops.

Some different variants of this method were implemented aiming at skipping dominated solutions. The proposed variants were tested for randomly generated instances. In general, the instances with small densities and small values of $|L|$ were harder to solve than denser instances with a wider range of groups. Some of the instances of the SMLPP were solved very quickly, whereas solving others depended on the allocated memory space. Nevertheless, in average the non-dominated solutions for the problem were found in short times. Solutions of practical interest, in particular, pairs of paths with 2 or 1 labels in common, were found in average in less than 31 milliseconds. These times seem to be reasonable in practical terms.

The second variant of the problem, the HMLPP problem, includes the number of hops in the paths as the min-sum objective function. The same algorithms were tested for experiments similar to the previous ones, but considering all costs equal to 1. The results showed that these instances were easier to solve than the first ones, despite having more non-dominated solutions. For this case, the average run times for finding the non-dominated solutions did not exceed 30 milliseconds.

Acknowledgments

This work was partially supported by the Portuguese Foundation for Science and Technology (FCT) under project grants UID/MAT/00324/2013 and UID/MULTI/00308/2019. The work was also partially financially supported by the MobiWise project: From mobile sensing to mobility advising, P2020 SAICTPAC/0011/2015, co-financed by COMPETE 2020, Portugal 2020 - Operational Program for Competitiveness and Internationalization (POCI), European Union's European Regional Development Fund, and FCT, and by FEDER Funds and National Funds under project CENTRO-01-0145-FEDER-029312.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] J. Clímaco and E. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404, 1982.

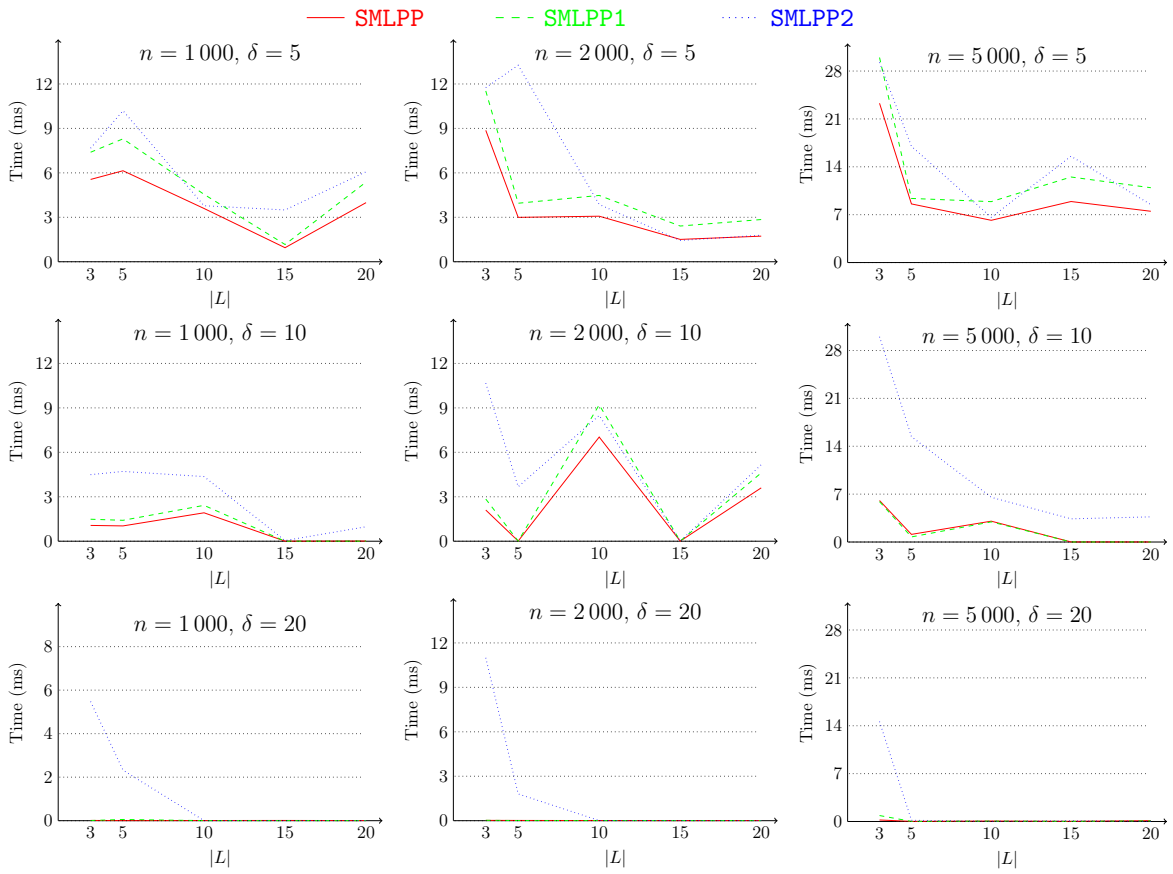

Figure 5: Mean run times for the codes `SMLPP`, `SMLPP1`, `SMLPP2` applied to the HMLPP problem

[3] J. Clímaco and M. Pascoal. Finding non-dominated shortest pairs of disjoint simple paths. *Computers & Operations Research*, 36:325–336, 2009.

[4] J. Clímaco and M. Pascoal. Multicriteria path and tree problems: discussion on exact algorithms and applications. *International Transactions in Operational Research*, 19:63–98, 2012.

[5] D. Coudert, P. Datta, S. Pérennes, H. Rivano, and M.-E. Voge. Shared risk resource group complexity and approximability issues. *Parallel Processing Letters*, 17(02):169–184, 2007.

[6] J. Craveirinha, M. Pascoal, J. Clímaco, and T. Gomes. An exact approach for a bicriteria maximal SRLG-disjoint/minimal cost path pair problem in telecommunication networks. In *XI Meeting Grupo Espanol Decision Multicriterio*. Malaga, June 2017.

[7] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28:652–673, 1998.

[8] T. Gomes, L. Jorge, P. Melo, and R. Girão-Silva. Maximally node and SRLG-disjoint path pair of min-sum cost in GMPLS networks: a lexicographic approach. *Photonic Network Communications*, 31(1):11–22, 2016.

[9] L. Gouveia, P. Patrício, and A. de Sousa. Models for optimal survivable routing with a minimum number of hops: comparing disaggregated with aggregated models. *International Transactions in Operational Research*, 18:335–358, 2011.

[10] J. Q. Hu. Diverse routing in optical mesh networks. *IEEE Transactions on Communications*, 51(3):489–494, 2003.

[11] F. Iqbal and F. Kuipers. *Disjoint paths in networks*, pages 1–11. Wiley Online Library, 2015.

[12] M. Maimour. Maximally radio-disjoint multipath routing for wireless multimedia sensor networks. In *Proceedings of the 4th ACM workshop on Wireless multimedia networking and performance modeling*, pages 26–31, New York, NY, USA, 2008. ACM.

[13] E. Martins, M. Pascoal, and J. Santos. Deviation algorithms for ranking shortest paths. *The International Journal of Foundations of Computer Science*, 10:247–263, 1999.

[14] E. Martins, M. Pascoal, and J. Santos. A new improvement for a K shortest paths algorithm. *Investigação Operacional*, 21:47–60, 2001.

[15] B. Martín, A. Sánchez, C. Beltran-Royo, and A. Duarte. Solving the edge-disjoint paths problem using a two-stage method. *International Transactions in Operational Research*, 0, 2018.

[16] V. Miletić, T. Šubić, and B. Mikac. Optimizing maximum shared risk link group disjoint path algorithm using NVIDIA CUDA heterogeneous parallel programming platform. In *X International Symposium on Telecommunications*, pages 1–6. IEEE, 2014.

[17] X. Pan and G. Xiao. Heuristics for diverse routing in wavelength-routed networks with shared risk link groups. *Photonic Network Communications*, 11(1):29–38, 2006.

[18] M. Rostami, A. Zarandi, and S. Hoseininasab. MSDP with ACO: A maximal SRLG disjoint routing algorithm based on ant colony optimization. *Journal of Network and Computer Applications*, 35(1):394–402, 2012.

[19] G. Xiao and X. Pan. Heuristic for the maximum disjoint paths problem in wavelength-routed networks with shared-risk link groups. *Journal of Optical Networking*, 3(1):38–49, 2004.