Pré-Publicações do Departamento de Matemática
Universidade de Coimbra
Preprint Number 07–11

RANKING MULTIOBJECTIVE SHORTEST PATHS

ERNESTO QUEIRÓS MARTINS, JOSÉ MANUEL PAIXÃO, MÁRIO SILVA ROSA
AND JOSÉ LUIS SANTOS

ABSTRACT: This paper is concerned with the ranking of multi-objective shortest paths accordingly to an order relation verifying certain conditions such is the case, for instance, of the lexicographic order. We present a new labelling algorithm that makes use of shortest deviation paths for obtaining the set of Pareto solutions for the multi-objective shortest path problem. The computational experience reported at the end of the paper shows that the new algorithm clearly outperforms the previous approaches when one looks for the ℓ-th shortest non-dominated paths.

KEYWORDS: Multiple objective programming, combinatorial optimization, ranking algorithm, total order, non-dominated path.
AMS SUBJECT CLASSIFICATION (2000): 90B10, 90C27, 90C29, 90C35.

1. Introduction

The multiobjective shortest path problem (MSPP) is a natural extension of the classical shortest path problem when several parameters are assigned to the arcs of the underlying network.

The MSPP has been dealt with by many authors addressing either theoretical aspects, algorithmic approaches or applications. In particular, the bi-objective case has been extensively studied in the literature. For a review of the MSPP, the interested reader is referred to [8].

For the MSPP, one intends to determine a path that minimizes simultaneously all the criteria under consideration. Usually, there is a conflict among the different criteria and such an ideal solution does not exist. The resolution of the MSPP turns into finding *non-dominated paths* (ND paths), that is, paths for which there is no other path with better values for all the criteria.

Received March 29, 2007.

This work was supported in part by FCT through POCTI - Research Units Pluriannual Funding to CMUC (*Centro de Matemtica da Universidade de Coimbra*) and CIO (*Operations Research Center of the University of Lisbon*), and grant POCTI/MAT/139/2001 cofunded by the EU program FEDER.

The authors would like to thank Anders J.V. Skriver for the friendly way he made his software available.

In the worst case, as proved by Hansen, [14], one may have to deal with an exponential number of ND paths. Therefore, the computation of the entire set of non-dominated paths can be hard to accomplish.

Several strategies have been adopted for tackling the MSPP. As proposed by Martins and Santos, [17], those approaches can be classified into two groups. The first one includes the algorithms that select a single ND path as is the case of a global optimization problem where an utility function is defined ([3]) or the interactive procedures ([5]) where the user guides the searching within the criteria space. In the second group, one considers the approaches for the MSPP that find all the non-dominated paths. That is the case of the well-known labelling algorithm, [2, 14, 26, 27], the ranking paths procedures, [6, 7], and the algorithm presented by Mote [18] that computes the full set of ND paths, after solving the linear programming relaxation for the MSPP.

In this paper, we present a new labelling technique that is able to obtain the ND paths ranked accordingly to a total order relation defined on $\mathbb{R}^k$. This means that the new algorithm produces the ℓ "best" shortest ND paths with ℓ varying from one to the total number of ND paths.

The new algorithm is based on a ranking path procedure where labels are scanned by tracing the nodes of the next shortest deviation path. The computational experience reported in section 6 of the paper, shows that the new algorithm is quite competitive relatively to the previous approaches.

In the next section of the paper, we introduce the notation used through out the paper. The section 3 is dedicated to the description of the algorithm that is exemplified in the following section. The correctness of the algorithm is proved in section 5 and, as already mentioned, computational results are shown in section 6. Finally, conclusions are summarized in section 7.

2. Definitions and notation

In this section, some definitions are given and just a crucial result is presented taking into account that detailed mathematical background can be found in Martins and Santos, [17].

A network is denoted by $G = (N, A, c)$, where $N = \{1, \ldots, n\}$ is the set of nodes (or vertices) and $A \subset N \times N$ is the set of arcs. Each arc $a \in A$, $a = (i, j)$, has a tail ($tail(a) = i$) and a head ($head(a) = j$) node. The set of arcs for which i is the tail node will be denoted by $A(i) = \{(x, y) \in A : x = i\}$. Let k be the number of criteria, then the vectorial function c attributes a k

[15] E.Q. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16:236–245, 1984.
[16] E.Q. Martins, M.M. Pascoal, and J.L. Santos. Deviation algorithms for ranking shortest paths. *International Journal of Foundations of Computer Science*, 10 (3):247–261, 1999.
[17] E.Q. Martins and J.L. Santos. The labelling algorithm for the multiobjective shortest path problem. Internal Technical Report, TR 1999/005, CISUC.
[18] J. Mote, I. Murthy, and D.L. Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53:81–92, 1991.
[19] J.P. Paixão, M.S. Rosa, and J.L. Santos. Labelling methods for the general case of the multi-objective shortest path problem - a computational study. Working Paper 07-xx, CMUC and submitted for publication., 2001.
[20] U. Pape. Implementation and efficiency of moore-algorithms for the shortest route problem. *Mathematical Programming*, 7:212–222, 1974.
[21] U. Pape. Algorithm 562: Shortest paths lengths. *ACM Transactions on Mathematical Software*, 6:450–455, 1980.
[22] J.L. Santos. Computational results in networks optimization problems. (http://lirio.mat.uc.pt/redes/paper_results.htm).
[23] J.L. Santos. Multiobjective shortest path problem. (http://www.mat.uc.pt/~zeluis/INVESTIG/MSPP/mspp.htm).
[24] J.L. Santos. O problema do trajecto óptimo multiobjectivo, 1997. (Master degree dissertation; Mathematics Department; University of Coimbra).
[25] J.L. Santos. *Optimização vectorial em redes*. PhD thesis, Departamento de Matemática, Universidade de Coimbra, 2003.
[26] A.J. Skriver and K.A. Andersen. A label correcting approach for solving bicriterion shortest-path problems. *Computers & Operations Research*, 27:507–524, 2000.
[27] P. Vincke. Problèmes multicritères. *Cahiers du Centre d'Études de Recherche Opérationelle*, 16:425–439, 1974.

Ernesto Queirós Martins
Department of Mathematics, University of Coimbra

José Manuel Paixão
Operations Research Center, Department of Statistics and Operations Research, University of Lisbon
E-mail address: jpaixao@fc.ul.pt

Mário Silva Rosa
Centre for Mathematics of the University of Coimbra, Department of Mathematics, University of Coimbra
E-mail address: mrosa@mat.uc.pt

José Luis Santos
Centre for Mathematics of the University of Coimbra, Department of Mathematics, University of Coimbra
E-mail address: zeluis@mat.uc.pt
URL: http://www.mat.uc.pt/~zeluis

networks, the algorithm takes slightly longer than relatively to $\leq_{max}$ and $\leq_{sum}$. However, for the grid networks, the L&DP performs faster for the $\leq_{lex}$ order relation.

7. Conclusion

In this paper, we presented a new algorithm to solve the MSPP with the particular feature of efficiently ranking the non-dominated paths, accordingly to an order relation. We proved the correctness of the algorithm and reported computational experience using a public code as reference basis. From the computational results shown in the paper, one may conclude that the new algorithm has the lowest average working effort to obtain a non-dominated path and clearly outperforms the previously known procedures for the MSPP, when one looks for ranking the ND paths.

References

[1] R.K. Ahuja, T. L. Magnanti, and J.B. Orlin. *Network Flows – theory, algorithms, and applications*. Prentice-Hall, Inc., New Jersey, 1993.

[2] J. Brumbaugh-Smith and D. Shier. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43:216–224, 1989.

[3] R.L. Carraway, T.L. Morin, and H. Moskowitz. Generalized dynamic programming for multicriteria optimization. *European Journal of Operational Research*, 44:95–104, 1990.

[4] Dimacs Center. 9th dimacs implementation challenge - shortest paths, 2006. (http://www.dis.uniroma1.it/∼challenge9/).

[5] J.N. Clímaco, C.H. Antunes, and M.J. Alves. Interactive decision support for multiobjective transportation problems. *European Journal of Operational Research*, 65/1:58–67, 1993.

[6] J.N. Clímaco and E.Q. Martins. On the determination of the nondominated paths in a multiobjective network problem. Proceedings of V Sympösium über Operations Research, Köln, (1980), in *Methods in Operations Research*, 40, (Anton Hain, Königstein, 1981), 255–258.

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

[8] J. Current and M. Marsh. Multiple transportation network design and routing problems: taxonomy and annotation. *European Journal of Operational Research*, 65:4–19, 1993.

[9] R. Dial. Algorithm 360. shortest path forest with topological ordering. *Communications of ACM*, 12:632–633, 1969.

[10] 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.

[11] M. Ehrgott. *Multiple Criteria Optimization – Classification and Methodology*. Shaker Verlag, Aachen, 1997.

[12] F. Guerriero and R. Musmanno. Label correcting methods to solve multicriteria shortest path problems. *Journal of Optimization Theory and Applications*, 111:589–613, 2001.

[13] W. Habenicht. Efficient routes in vector-valued graphs. In *Proceedings of the 7th conference on graphtheoretic concepts in computer science (WG 81)*, pages 349–355. Mühlbacher, 1982.

[14] P. Hansen. Bicriterion path problems. in *Multiple Criteria Decision Making: Theory and Application*, editors: G. Fandel and T. Gal, Lectures Notes in Economics and Mathematical Systems, 177, 109-127, Springer Heidelberg, 1980.

dimensional vector cost to each arc:

$$\begin{array}{rccl} c: & A & \longrightarrow & \mathbb{R}^k \\ & (i,j) & \longmapsto & c(i,j) = \mathbf{c_{i,j}} = (c^1_{i,j}, \ldots, c^k_{i,j}). \end{array}$$

A path p, from the vertex i to j, is an alternating sequence of nodes and arcs of the form $p = \langle v_0, a_1, v_1, \ldots, a_r, v_r \rangle$, where:

- $v_\ell \in N$, $\forall \ell \in \{0, \ldots, r\}$;
- $v_0 = i$ and $v_r = j$;
- $a_\ell = (v_{\ell-1}, v_\ell) \in A$, $\forall \ell \in \{1, \ldots, r\}$.

The set of all paths from i to j is denoted by $P_{i,j}$ and P_G represents the set of all paths in the network, that is, $P_G = \bigcup_{i,j \in N} P_{i,j}$. A cycle is a path with non repeated vertices except the initial and terminal ones which are coincident; that is, $v_0 = v_r$.

With no loss of generality, we consider that N has an initial node s and a terminal node t such that:

- for any arc $a \in A$, $tail(a) \neq t$ and $head(a) \neq s$;
- for any $i \in N - \{s, t\}$, $P_{s,i} \neq \emptyset$ and $P_{i,t} \neq \emptyset$.

In order to simplify the notation, P will be used instead of $P_{s,t}$.

Multiple arcs (arcs with the same pair of head and tail nodes) are not allowed. As a consequence, p can be denoted only by the sequence of its nodes, $\langle v_0, v_1, \ldots, v_r \rangle$. We denote by $sub_p(u, w)$ the subpath of p from u to w ($u, w \in N \cap p$), that is, the subsequence $\langle v_\ell, v_{\ell+1}, \ldots, v_{\ell+h} \rangle$, where $v_\ell = u$ and $v_{\ell+h} = w$.

The vectorial objective function f is defined by

$$\begin{array}{rccl} f: & P_G & \longrightarrow & \mathbb{R}^k \\ & p & \longmapsto & f(p) = (f_1(p), \ldots, f_k(p)), \end{array}$$

where $f_\ell(p) = \sum_{(i,j) \in p} c^\ell_{i,j}, \forall \ell \in \{1, \ldots, k\}$.

The concatenation operator, $\diamondsuit$, joins two paths $p = \langle v_0, \ldots, v_{r_p} \rangle$ and $q = \langle u_0, \ldots, u_{r_q} \rangle$ such that $v_{r_p} = u_0$. Then, $p \diamondsuit q = \langle v_0, \ldots, v_{r_p} = u_0, \ldots, u_{r_q} \rangle$.

Now, let us recall that, for the MSPP, one looks for the set of non-dominated paths from s to t, mathematically described as follows:

Definition 1. : *Let p and q be two paths of $P_{i,j}$. We say that $\underline{p \text{ dominates } q}$ or $\underline{q \text{ is dominated by } p}$ ($p <_D q$) if and only if*

$$f(p) \neq f(q) \text{ and } f_\ell(p) \leq f_\ell(q), \forall \ell \in \{1, \ldots, k\}.$$

Definition 2. : *Let p be a path in $P_{i,j}$, $i,j \in N$. If there is no path $q \in P_{i,j}$ such that $q <_D p$, then p is called non-dominated, efficient or Pareto optimal path. The set of non-dominated paths from i to j is denoted by $\bar{D}_{i,j}$ and $\bar{D}$ will be used for $\bar{D}_{s,t}$.*

Note that the dominance relation ($<_D$) is not a total order relation in $\mathbb{R}^k$ and, therefore, does not allow the full ranking of the paths in the network. Nevertheless, this may be achieved by considering a total order relation [11] as the ones defined as follows:

Definition 3. : *Let p and q be two paths of $P_{i,j}$.*

- $p \leq_{lex} q \Leftrightarrow f(p) = f(q)$ *or* *(lexicographic)* $\exists x \in \{1,\dots,k\} : f_x(p) < f_x(q)$ *and* $f_y(p) = f_y(q), \forall y < x$;
- $p \leq_{sum} q \Leftrightarrow \sum_{x=1}^{k} f_x(p) \leq \sum_{x=1}^{k} f_x(q)$ *(or other positive weighted sum);*
- $p \leq_{max} q \Leftrightarrow \max\limits_{1\leq x\leq k}\{f_x(p)\} \leq \max\limits_{1\leq x\leq k}\{f_x(q)\}$ *(maximum component);*

The computation of the ND paths can be accomplished in a more efficient way if the order relation $\leq_R$ verifies the following properties:

Property 1: $p <_D q \Rightarrow p \leq_R q, \forall p,q \in P_{i,j}$.

Property 2: $p \leq_R p\diamondsuit\langle j,\ell\rangle, \forall p \in P_{i,j}, \forall (j,\ell) \in A(j)$.

The following Lemma, proved in [19], establishes the sufficient conditions for that the total order above defined hold properties 1 and 2.

Lemma 1. : *If $c^{\ell}_{i,j} > 0, \forall (i,j) \in A$ and $\ell \in \{1,\dots,k\}$, then properties 1 and 2 are satisfied when $\leq_{lex}$, $\leq_{max}$ and $\leq_{sum}$ are used.*

3. The label-deviation path algorithm (L&DP)

The new algorithm for the MSPP, presented in this paper, is based on the procedure for ranking paths developed by Martins et al., [16], for the shortest path problem with a single objective. That procedure consists of sequentially determining deviation paths in order to consider a non-decreasing sequence of shortest paths from s to t. Next, we remind some basic aspects of the procedure and two results stated for the single objective case that will be used in the new label-deviation path algorithm.

Now, let us consider a path $p = \langle v_0,\dots,v_r\rangle \in P$ and let v_i be a node of p $(0 \leq i < r)$. A deviation path from p at vertex v_i through the arc $(v_i,j) \in A(v_i)$ is a path $q \in P$ which coincides with p from $s = v_0$ to v_i, then follows arc (v_i,j) and finally goes to $v_r = t$ (see Figure 1 (a)). So,

Class			$\leq_{lex}$	$\leq_{max}$				$\leq_{sum}$			
	%ND	#ND	LDP	LDP	LS	LC	Fifo	LDP	LS	LC	Fifo
RandN	20	16.54	0.02	0.00	6.11	(12.38)	(23.18)	0.00	7.92	(14.87)	(22.31)
	40	33.08	0.51	0.00	15.12	(22.82)	(28.90)	0.00	19.10	(22.73)	(29.66)
	60	49.62	3.42	0.01	25.98	(31.54)	(36.58)	0.00	29.19	(32.52)	(37.83)
	80	66.16	12.52	0.08	38.12	(37.08)	(41.45)	0.01	40.76	(37.73)	(42.22)
	100	84.80	64.78	15.79	53.24	(51.90)	(52.75)	11.77	55.80	(52.16)	(52.91)
	final	—	75.71	57.51	54.15	52.69	53.63	59.65	56.71	52.98	53.63
RandD	20	30.24	0.65	0.00	8.47	(16.99)	(31.78)	0.00	12.92	(21.52)	(29.44)
	40	60.48	5.12	0.00	21.70	(30.74)	(39.36)	0.00	27.69	(30.58)	(40.68)
	60	90.72	16.68	0.05	36.70	(42.69)	(53.28)	0.01	42.60	(43.29)	(51.69)
	80	120.96	38.39	1.09	53.31	(53.98)	(58.01)	0.12	57.77	(54.19)	(59.09)
	100	153.04	94.55	35.34	72.69	(69.37)	(71.06)	39.11	76.99	(69.88)	(71.11)
	final	—	100.86	74.62	74.01	71.30	73.00	77.32	78.13	72.23	73.00
RandK	20	37.90	0.03	0.00	9.06	(17.81)	(33.46)	0.00	12.91	(21.41)	(31.11)
	40	75.80	0.70	0.00	22.97	(35.72)	(47.90)	0.00	27.44	(37.42)	(45.89)
	60	113.70	3.90	0.01	38.45	(42.46)	(50.71)	0.01	42.61	(46.85)	(53.25)
	80	151.60	15.17	0.16	55.27	(56.49)	(61.62)	0.04	57.76	(56.38)	(63.00)
	100	191.52	82.14	37.12	74.54	(71.25)	(72.25)	32.30	77.25	(71.94)	(72.47)
	final	—	94.16	77.10	75.75	74.03	74.85	79.49	78.36	74.50	74.85
CompN	20	189.96	3.05	0.02	8.74	(24.20)	(45.48)	0.00	14.78	(28.50)	(47.67)
	40	379.92	17.11	0.41	24.57	(35.99)	(55.87)	0.04	32.20	(40.99)	(56.19)
	60	569.88	41.90	2.96	45.42	(54.85)	(65.84)	0.72	52.28	(55.22)	(66.45)
	80	759.84	77.82	18.19	70.05	(66.01)	(74.74)	11.19	75.77	(64.34)	(73.53)
	100	951.82	121.92	78.19	98.48	(80.80)	(83.11)	90.16	103.49	(81.34)	(83.16)
	final	—	122.19	92.84	98.73	81.01	83.29	93.10	103.76	81.53	83.29
CompK	20	357.66	13.79	0.04	36.79	(101.70)	(166.58)	0.01	57.54	(97.42)	(164.25)
	40	715.32	59.95	0.52	92.44	(133.65)	(184.78)	0.09	116.47	(142.84)	(193.77)
	60	1072.98	137.96	8.01	158.47	(181.32)	(217.72)	1.49	180.19	(195.30)	(225.01)
	80	1430.64	237.78	48.15	231.14	(217.04)	(241.21)	23.69	248.19	(221.05)	(242.38)
	100	1790.24	352.80	252.72	312.09	(261.93)	(267.69)	280.11	328.85	(265.95)	(267.97)
	final	—	353.29	273.32	312.54	262.58	268.15	284.17	329.43	266.38	268.15
GridN	20	2340.16	1.93	3.66	35.23	(42.00)	(31.48)	4.09	52.76	(38.00)	(31.48)
	40	4680.32	12.48	17.43	51.08	(53.97)	(31.50)	21.57	58.05	(40.25)	(31.48)
	60	7020.48	33.58	39.51	65.21	(63.32)	(31.47)	48.80	62.41	(41.52)	(31.48)
	80	9360.64	63.45	67.29	77.99	(64.34)	(31.46)	80.32	66.71	(42.23)	(31.47)
	100	11702.9	99.70	97.88	89.06	(74.55)	(31.82)	111.68	71.40	(45.30)	(31.71)
	final	—	99.73	128.30	89.06	74.60	31.86	134.73	71.40	45.33	31.86
GridK	20	4158.98	4.02	8.31	51.33	(74.15)	(69.12)	9.40	81.48	(64.32)	(69.05)
	40	8317.96	20.96	36.32	77.49	(85.76)	(69.19)	41.97	95.23	(67.55)	(69.05)
	60	12476.9	49.61	77.73	102.37	(99.93)	(68.97)	89.34	106.13	(70.46)	(68.86)
	80	16635.9	88.09	127.66	127.42	(105.17)	(69.65)	142.17	117.61	(71.95)	(69.61)
	100	20797.2	135.66	184.90	152.38	(125.88)	(72.04)	195.12	131.54	(81.07)	(72.31)
	final	—	135.68	203.51	152.38	126.18	72.32	203.04	131.54	81.08	72.32

%ND: percentage of ND s-t paths; final: CPU time to finishes the algorithm;
#ND: number of ND s-t paths;
LDP: heap implementation for L&DP algorithm; LS: Dial implentation for label setting algorithm
LC: deque implementation for label correcting algorithm; Fifo: FIFO rule implementation for label correcting algorithm

TABLE 7. **CPU time (sec) for ranking ND paths on the biggest problem for each class.**

first ND s-t paths, making it quite competitive in the cases when it is not necessary to find the full set of ND paths. This is clearly illustrated by Table 7 where we show the average computational times taken by the algorithms for determining a fixed number, say ℓ, of non-dominated paths. Note that the Label Correcting algorithms (LC-Deque and FIFO) can not produce such result before finding all of the non-dominated paths. On the contrary, the L&DP and Label Setting algorithms compute those paths accordingly to the order relation that is considered. That is, for an order relation verifying properties 1 and 2 of section 2, the L&DP and the LS algorithm obtain the ℓ-shortest non-dominated paths.

In Table 7, we present the outcome of the computational experience carried out, using the largest size test problems for each class of instances, when fixing ℓ to a value approximately equal to a percentage of the total number of non-dominated paths. For example, for the RandN test instances, the L&DP algorithm with $\leq_{max}$ only requires 0.01 seconds for finding the ℓ-shortest paths for ℓ equal, in average, to 60% of the non-dominated paths. That is clearly much less than the corresponding time (25.98 seconds) for the LS algorithm and, let us remind that the Label Correcting algorithms can not produce such result before finding all the non-dominated paths requiring in average 52.69 and 53.63 seconds, respectively, for the LC-Deque and FIFO. For these algorithms, we show within brackets the time instant when the label of the corresponding path is scanned, in the particular case, 31.54 and 36.58 seconds.

From Table 7, we conclude that the L&DP is quite efficient in finding the ℓ-shortest non-dominated paths even when ℓ is close to the total number of non-dominated paths. The exception to this occurs for the grid networks where the FIFO algorithm proves to be faster when ℓ corresponds to more than 40% percent of the total number of non-dominated paths.

Let us remark that, for ranking 100% of the non-dominated paths, the L&DP algorithm requires, in general, a small share of the time needed for guaranteeing that the full set of non-dominated paths has been found (shown in Table 7 by the row named *final*). This is particularly evident for the classes RandN, RandD and RandK, where the "full ranking" accounts for less than 50% of the "final time".

Finally, let us mention that we present in Table 7 the results obtained for the L&DP algorithm with $\leq_{lex}$ since this is usually used for ranking multi-objective solutions. In this case, for the random and the complete

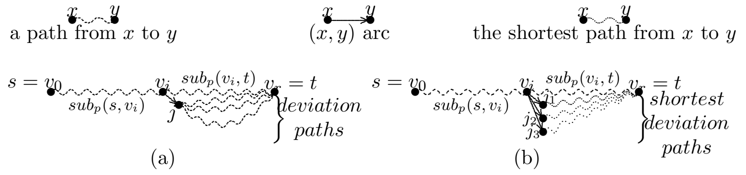


FIGURE 1. **Deviation paths of $p = \langle v_0, v_1, \ldots, v_r\rangle$ from vertex v_i.**
(a) Deviation paths of p through the arc (v_i, j);
(b) Shortest deviation paths of p at node v_i.

$q = sub_p(s, v_i)\diamond\langle v_i, j\rangle\diamond w$, where w is a path of $P_{j,t}$. Note that, (v_i, j) is the first arc for which q diverges from p. This arc is called the *deviation arc of q relatively to p*.

For the single objective case, if the least cost path from j to t is denoted by $T^\star(j)$ then $q^p_{v_i,j} = \langle v_0, \ldots, v_i\rangle\diamond\langle v_i, j\rangle\diamond T^\star(j)$ is the shortest deviation path from p through the arc (v_i, j).

Hence, if one considers the vertex $v_i \in p$, the shortest deviation path from p at the node v_i is the one corresponding to $\min\limits_{(v_i,j)\in A(v_i)} f(q^p_{v_i,j})$, (see Figure 1 (b)). Therefore, the shortest deviation path from p will be determined by $\min\limits_{v_i\in p}\left(\min\limits_{(v_i,j)\in A(v_i)} f(q^p_{v_i,j})\right)$.

From the above, one may depict a very simply procedure for generating shortest paths from s to t, ordered by non-decreasing costs. The first path (p_1^*) is the shortest path from s to t and the next path in the sequence (p_2^*) will be the least cost path in $D(p_1^*) = \{q^{p_1^*}_{v_i,j} : v_i \in p_1^*, (v_i, j) \in A\}$, the set of deviation paths from p_1^*. Then, p_3^* will be obtained considering all the deviation paths from p_1^* and p_2^*, excluding p_2^*, and the algorithm proceeds successively in this way, until all the shortest paths are generated.

Since p_2^* is itself a deviation path from p_1^* through an arc (v_i, j) with $v_i \in p$, that we denote by $\theta(p_2^*)$, we only need to consider the deviation paths from p_2^* related to the vertices in p_2^*, subsequent to the tail of $\theta(p_2^*)$, inclusive, since the remaining deviation paths have been obtained from p_1^*. Therefore, p_3^* is obtained from $D(p_1^*)\backslash\{p_2^*\}$ or $D(p_2^*) = \{q^{p_2^*}_{v_i,j} : v_i \in sub_{p_2^*}(tail(\theta(p_2^*)), t), (v_i, j) \in A\}$. Now, $\theta(p_3^*)$ will be the deviation arc through which p_3^* is generated and, consequently, defines $D(p_3^*)$ the set of deviation paths from p_3^* that must be

considered, together with $(D(p_1^*) \cup D(p_2^*)) \backslash \{p_2^*, p_3^*\}$, for determining the next shortest path. The procedure follows like that until no new deviation path can be found for scanning.

Next, we show how to reduce the computational effort for the calculation of $\min_{(v_i,j)\in A(v_i)} f(q^p_{v_i,j})$ required for each vertex v_i in the successive $D(p^*_\ell), \ell \geq 1$.

Let us recall that $T^\star(i)$ is the shortest path from i to t. Those paths can be chosen in such a way that $T^\star = \bigcup_{i\in N} T^\star(i)$ forms the shortest tree rooted at t. Then, for each arc $(i,j) \in A$, $\bar{c}_{i,j} = c_{i,j} + f(T^\star(j)) - f(T^\star(i))$ is the reduced cost of (i,j) relatively to $T^\star$ and $\bar{f}(p) = \sum_{(i,j)\in p} \bar{c}_{i,j}$ is the corresponding reduced cost for a path p. Now, let us recall the following result already presented in ([16]):

Lemma 2. : *Let $T^\star$ be a shortest tree rooted at t and $\bar{c}$ the reduced cost computed for $T^\star$. Then:*

(1) $0 \leq \bar{c}_{i,j}$, $\forall (i,j) \in A$.
(2) $\bar{c}_{i,j} = 0$, $\forall (i,j) \in A \cap T^\star$.
(3) $\bar{f}(T^\star(i)) = 0$, $\forall i \in N$.
(4) $\bar{f}(p) = f(p) + f(T^\star(j)) - f(T^\star(i)), \forall p \in P_{i,j}$.

Now, let us remind that we denote by $\theta(p)$ the deviation arc for the path p relatively to another path previously obtained by the procedure described above. Then, when looking for the deviation paths from p we only need to consider the vertices subsequent to the $tail(\theta(p))$, say v_k, since the remaining ones have been already scanned. Therefore, for $q^p_{v_i,j}$, the shortest deviation path from p, v_i is subsequent to v_k and one can state the following:

Lemma 3. : $\bar{f}(q^p_{v_i,j}) = \bar{f}(p) + \bar{c}_{v_i,j}$.

Proof: Note that $q^p_{v_i,j}$ is of the form $q^p_{v_i,j} = sub_p(s,v_i)\diamondsuit\langle v_i, j\rangle\diamondsuit T^\star(j)$, where $v_i \in sub_p(v_k,t)) = T^\star(v_k)$. Hence,

$$\bar{f}(q^p_{v_i,j}) = \underbrace{\bar{f}(sub_p(s,v_i))}_{=\bar{f}(p)} + \underbrace{\bar{f}(\langle v_i,j\rangle)}_{=\bar{c}_{v_i,j}} + \underbrace{\bar{f}(T^\star(j))}_{=0} = \bar{f}(p) + \bar{c}_{v_i,j}. \qquad \square$$

Corollary 1. : $\bar{f}(p) \leq \bar{f}(q^p_{v_i,j})$.

From the previous results, it is clear that the deviation paths obtained from p at node v_i can be easily sorted out if $A(v_i) = \{(x,y) \in A : x = v_i\}$ is rearranged by no decreasing order of $\bar{c}$. Consequently, one needs to consider

Class problem	Alg. set X order	L&DP Dial $\leq_{lex}$	L&DP Dial $\leq_{max}$	L&DP Dial $\leq_{sum}$	L&DP heap $\leq_{lex}$	L&DP heap $\leq_{max}$	L&DP heap $\leq_{sum}$	Label Setting Dial $\leq_{max}$	Label Setting Dial $\leq_{sum}$	Label Correcting Deque $\leq_{max}$	Label Correcting Deque $\leq_{sum}$	Label Correcting Fifo
RandN	time	1.355	1.115	1.081	1.266	0.987	1.021	0.907	0.951	0.866	0.872	0.872
	nIter	0.810	1.146	1.131	0.810	0.830	0.812	0.997	0.997	1.001	0.999	1.000
	percD	5.890	21.66	21.92	5.890	2.661	1.375	0.377	0.013	1.090	0.668	1.000
	memo	1.258	1.940	1.951	1.257	1.264	1.235	1.134	1.128	1.075	1.067	1.000
RandD	time	1.297	1.190	1.184	1.342	1.078	1.101	0.961	0.991	0.892	0.891	0.882
	nIter	0.760	1.070	1.067	0.760	0.778	0.761	0.998	0.998	1.001	0.999	1.000
	percD	6.610	32.51	32.95	6.610	3.445	2.078	0.374	0.012	1.179	0.663	1.000
	memo	1.251	1.911	1.941	1.249	1.264	1.237	1.133	1.129	1.074	1.067	1.000
RandK	time	1.437	1.431	1.413	1.460	1.134	1.176	0.974	1.020	0.933	0.938	0.946
	nIter	0.785	1.159	1.150	0.785	0.810	0.788	0.973	0.973	0.989	0.984	1.000
	percD	13.33	52.96	52.70	13.33	4.264	2.132	0.441	0.060	1.159	0.673	1.000
	memo	1.221	1.930	1.948	1.220	1.248	1.218	1.102	1.093	1.061	1.053	1.000
CompN	time	1.210	1.104	1.103	1.234	0.987	0.985	1.014	1.064	0.885	0.881	0.884
	nIter	0.927	1.033	0.961	0.927	0.970	0.936	0.999	0.999	1.000	0.999	1.000
	percD	5.855	5.456	2.975	5.855	2.498	1.232	0.398	0.104	0.815	0.704	1.000
	memo	1.389	1.448	1.511	1.355	1.278	1.220	1.145	1.258	1.058	1.055	1.000
CompK	time	1.162	1.003	1.039	1.168	0.941	0.918	0.928	0.974	0.860	0.866	0.888
	nIter	0.912	0.979	0.938	0.912	0.985	0.928	0.961	0.961	0.994	0.987	1.000
	percD	4.241	2.459	1.068	4.241	1.788	0.830	0.493	0.258	0.891	0.808	1.000
	memo	1.282	1.279	1.220	1.276	1.271	1.196	1.075	1.064	1.049	1.044	1.000
GridN	time	1.615	1.816	1.556	1.655	2.124	2.122	1.542	1.310	1.212	0.976	0.850
	nIter	0.405	0.371	0.353	0.405	0.343	0.337	0.999	0.999	1.023	1.006	1.000
	percD	1.024	2.301	1.355	1.024	1.542	1.115	0.186	0.012	0.761	0.507	1.000
	memo	1.198	1.402	1.556	1.130	1.268	1.228	1.133	1.276	1.062	1.047	1.000
GridK	time	1.762	2.016	1.972	1.783	2.386	2.609	1.745	1.389	1.473	1.058	0.930
	nIter	0.402	0.396	0.377	0.402	0.355	0.350	0.988	0.988	1.026	1.004	1.000
	percD	0.588	1.905	1.050	0.588	1.315	0.951	0.204	0.056	0.814	0.582	1.000
	memo	1.145	1.390	1.329	1.113	1.284	1.242	1.091	1.100	1.063	1.037	1.000

time = CPU time (sec); nIter = number of iterations; memo = memory space (Mb);
percD = 100(number of labels created-rotND)/(number of labels created);

TABLE 6. **Average ratios obtained between each code and publicMOSP.**

The values reported in Table 6 lead us to the general conclusion that the new algorithm is outperformed by the other procedures. However, there is a particular feature of the algorithm, related to the effort taken to find the

Class						
RandN	n	3000	6000	9000	12000	15000
	time	4.58	15.09	27.50	43.59	58.95
	nIter	193688.3	446963.5	716094.4	1022522.6	1312879.5
	percD	1.74	1.86	1.88	1.85	1.96
	memo	82.60	189.34	302.46	430.41	552.43
RandD	d	2	4	6	8	10
	time	0.11	2.67	12.65	38.35	88.15
	nIter	46749.3	179493.7	352591.9	566341.2	817052.4
	percD	0.36	1.15	1.79	2.22	2.52
	memo	21.34	77.09	149.70	239.21	344.03
RandK	k	2	4	6	8	10
	time	0.13	2.55	12.70	39.17	90.18
	nIter	38139.1	154656.6	352591.9	627810.1	969430.6
	percD	30.67	6.55	1.79	0.48	0.15
	memo	16.56	60.35	149.7	298.14	514.94
CompN	n	40	60	80	100	120
	time	0.26	2.59	16.45	59.60	172.92
	nIter	7624.1	20312.1	41500.2	69319.4	107447.4
	percD	3.18	3.98	4.63	4.94	5.70
	memo	3.51	9.25	18.78	31.29	48.56
CompK	k	2	4	6	8	10
	time	0.03	1.92	50.30	406.34	—
	nIter	1566.8	17135.7	69319.4	183173.7	—
	percD	51.31	16.07	4.94	1.63	—
	memo	1.94	8.54	31.29	88.24	—
GridN	n	64	81	100	121	144
	time	0.06	0.34	1.52	8.24	54.78
	nIter	6155.0	14052.2	32181.3	68787.6	141617.3
	percD	5.03	7.01	8.09	8.47	9.21
	memo	2.63	6.06	13.96	29.88	61.85
GridK	k	2	4	6	8	10
	time	0.00	0.06	1.54	21.69	113.87
	nIter	761.5	7814.4	32181.3	80915.8	141569.4
	percD	21.48	10.82	8.09	5.59	3.81
	memo	0.30	2.99	13.96	39.26	76.41

n = number of nodes; d = number of arcs$/n$; k = number of criteria
time = CPU time (sec); nIter = number of iterations;
percD = 100(number of labels created-rotND)/(number of labels created);
memo = memory space (Mb)

TABLE 5. **Computational results for publicMOSP code.**

produces a strong effect on the percentage of dominated labels created by the algorithm and, consequently, on its performance. In general, the worst results were obtained when using the $\leq_{lex}$ operator.

one shortest deviation path per node of p from $tail(\theta(p))$ to t, reducing the number of elements in $D(p)$. Then, the shortest deviation path from p at the node v_i will be obtained through the first arc in $A(v_i)$ that has not been used as a deviation arc, called the active arc of $A(v_i)$.

Note that, in order to assure that all the arcs of $A(v_i)$ are considered, the first arc of $A(v_i)$ (for all $i \in N$) will be an arc of $T^\star$. At last, since $T^\star(s)$ is not obtained from a previous path, $\theta(T^\star(s))$ is fixed as the first arc of $T^\star(s)$.

The algorithm presented in Martins et al., [16], can be easily extended for the case where a k-uple cost is assigned to each arc of the network. That is attained by considering a total order relation in $\mathbb{R}^k$ such as the ones given in Definition 3. Therefore, $T^\star$ will be formed by $\leq_R$-shortest paths from j to t (the shortest path in the network when the $\leq_R$ is considered).

From the Lemma 2, we conclude that, for any $p \in P_{s,\ell}$, $f(p) - \bar{f}(p)$ is a constant value and equal to $f(T^\star(s)) - f(T^\star(\ell))$. In consequence, as stated in the next result, the relation between p and q with the operators $<_D$, $\leq_{lex}$, $\leq_{max}$ and $\leq_{sum}$ are not affected when $\mathbf{c_{i,j}}$ is replaced by $\bar{\mathbf{c}}_{\mathbf{i,j}}$.

Lemma 4. : *Let p and q be two paths of $P_{s,i}$. Then:*

(1) $f(p) \leq_R f(q) \Leftrightarrow \bar{f}(p) \leq_R \bar{f}(q)$, $R \in \{lex, max, sum\}$;
(2) $f(p) <_D f(q) \Leftrightarrow \bar{f}(p) <_D \bar{f}(q)$.

Now, note that, for a vertex v_i, the arcs of $A(v_i)$ are sorted by the $\leq_R$ order and let $q^p_{v_i,h}$ be a deviation path from p at v_i using the active arc in $A(v_i)$. If $sub_{q^p_{v_i,h}}(s,h)$ is dominated by some subpath from s to h previously determined, then any path generated from $q^p_{v_i,h}$ at vertex $x \in T^\star(h)$ will be also dominated. So, $q^p_{v_i,h}$ can be ignored and a new deviation path from p at node v_i must be generated using the next arc in $A(v_i)$.

The algorithm outlined on the previous page computes the set of non-dominated paths on a network G with k-uple costs assigned to the arcs using an order relation ($\leq_R$) which holds the properties 1 and 2 stated in section 2. The algorithm consists of an initialization step and an iterative cycle (step 2) for searching the elements in X, the set of candidates for the next non-dominated path. In this searching process, a path in X is selected in order to create new deviation paths by using an internal cycle (step 2.2). Let us mention that in step 2.2.1, the logical function $DT(w, \Pi_h)$ indicates when the sub-path w is dominated by some path from the set Π_h. Finally, note that Π_h is the set of the temporary ND paths from s to h and, at the end of

L&DP algorithm

{X: set of candidates for the next non-dominated path}
{Π_i: set of temporary non-dominated paths from s to $i, i \in N$}
{$\theta(p)$: deviation arc of p}
{$DT(p,\Pi)$: dominance test applied to p over Π; $DT(p,\Pi) = true \Leftrightarrow \exists q \in \Pi : q <_D p$}
{$\leq_R$: total order relation verifying properties 1 and 2 of section 2}

step 1: {initialization}
  $T^\star \leftarrow$ tree of the $\leq_R$-shortest paths from i to t, $\forall i \in N$
  $\bar{\mathbf{c}}_{\mathbf{i,j}} \leftarrow \mathbf{c}_{\mathbf{i,j}} + f(T^\star(j)) - f(T^\star(i)), \forall (i,j) \in A$
  Sort $A(i)$ by non-decreasing order of $\bar{\mathbf{c}}_{\mathbf{i,j}}$ using $\leq_R$
  $p \leftarrow T^\star(s)$ {Assume that $p = \langle v_0, \ldots, v_r \rangle$}
  $\theta(p) \leftarrow (v_0, v_1)$
  $X \leftarrow \{p\}$
  $\Pi_i \leftarrow \emptyset, \forall i \in N$

step 2: {searching for a new candidate to the next ND path}
  while $(X \neq \emptyset)$ do
2.1: Select $p \in X$ such that $\bar{f}(p) \leq_R \bar{f}(q), \forall q \in X$ {Suppose $p = \langle v_0, \ldots, v_r \rangle$}
  Assume that $\theta(p)$ is the arc (v_j, v_{j+1}) of p
  $X \leftarrow X \backslash \{p\}$
{search a node of p for generating a shortest deviation path}
2.2: for $i \leftarrow j$ to r do
2.2.1: $w \leftarrow sub_p(s, v_i)$
  if $(DT(w, \Pi_{v_i}) = \text{true})$
  then goto step 2 {ignore the remaining nodes of p}
  else $\Pi_{v_i} \leftarrow \{w\} \cup \{u \in \Pi_{v_i} : DT(u, \{w\}) = false\}$;
  Suppose (v_i, v_{i+1}) is the ℓ-th arc of $A(v_i)$, i.e., $a_\ell = (v_i, v_{i+1})$
2.2.2: $arcFound \leftarrow false$ {find the next active arc in $A(v_i)$}
  for $x \leftarrow \ell$ to $|A(v_i)|$ do
  if $(DT(w \diamondsuit \langle v_i, head(a_x) \rangle, \Pi_{head(a_x)}) = false)$
  then $arcFound \leftarrow true$ and goto step 2.2.3
  endfor {for $x \leftarrow \ell$ to $|A(v_i)|$ do}
2.2.3: {computation of a new candidate for the next ND path}
  if $(arcFound = true)$
  then $h \leftarrow head(a_x)$ {$a_x = (v_i, h)$}
  $q \leftarrow w \diamondsuit \langle v_i, h \rangle \diamondsuit T^\star(h)$ {$q^p_{v_i,h}$}
  $\theta(q) \leftarrow a_x$
  $X \leftarrow X \cup \{q\}$
  $\Pi_h \leftarrow \{w \diamondsuit \langle v_i, h \rangle\} \cup \{u \in \Pi_h : DT(u, \{w \diamondsuit \langle v_i, h \rangle\}) = false\}$
  endfor {for $i \leftarrow j$ to r do}
  endwhile {while $(X \neq \emptyset)$ do}

step 3: {all the non-dominated paths have been determined}
$\bar{D} \leftarrow \Pi_t$

Class						
RandN	n	3000	6000	9000	12000	15000
	ND	64.28	74.00	84.18	90.40	84.80
	rotND	193260.8	445764.8	714064.4	1019683.8	1308722.9
	rotND/n	64.42	74.29	79.34	84.97	87.25
RandD	d	2	4	6	8	10
	ND	8.32	38.62	71.42	115.84	153.04
	rotND	46686.2	179151.0	351698.0	564780.4	814661.5
	rotND/n	9.34	35.83	70.34	112.96	162.93
RandK	k	2	4	6	8	10
	ND	6.10	31.12	71.42	123.00	191.52
	rotND	31755.1	151904.5	351698.0	627617.5	969384.9
	rotND/n	6.35	30.38	70.34	125.52	193.88
CompN	n	40	60	80	100	120
	ND	187.12	312.52	509.86	706.58	951.82
	rotND	7616.8	20282.5	41422.1	69178.7	107138.9
	rotND/n	190.42	338.04	517.78	691.79	892.82
CompK	k	2	4	6	8	10
	ND	12.64	169.58	706.58	1790.24	—
	rotND	1276.9	16791.8	69178.7	183137.6	—
	rotND/n	12.77	167.92	691.79	1831.38	—
GridN	n	64	81	100	121	144
	ND	752.30	1545.72	3190.46	6183.24	11702.90
	rotND	6152.3	14043.5	32148.6	68713.8	141357.8
	rotND/n	96.13	173.38	321.49	567.88	981.65
GridK	k	2	4	6	8	10
	ND	17.40	472.64	3190.46	10423.80	20797.18
	rotND	704.7	7751.3	32148.6	80863.9	141548.5
	rotND/n	7.05	77.51	321.49	808.64	1415.48

n = number of nodes; d = number of arcs/n; k = number of criteria

TABLE 4. **Number of ND s-t paths (ND), number of ND labels (rotND) and average number of ND labels per node (rotND/n).**

structure (a queue) and higher effectiveness (only a small percentage of dominated labels were determined).

(3) The new algorithm requires, in general, the least number of iterations but due to the large percentage of dominated labels, shows larger computational times comparatively to the other procedures. Additionally, note that the best performance for the LD&P is produced when using the binary heap structure.

(4) For all the procedures, the $\leq_R$ order relation ($R \in \{lex, max, sum\}$), has a significant influence on the pivot label selection process. This

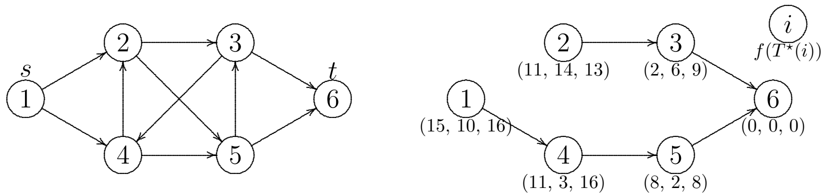
of ND s-t paths (ND), number of ND labels (rotND) and the average number of ND labels per node (rotND/n) - for only a subset of the groups that illustrate reasonably well the size and characteristics of the instances used for the computational tests. In that table, one can see that for each class of test instances either ND or rotND rapidly grow with the size of the network. The only exception appears to happen for the class RandN where a linear relation seems to occur between those parameters and number of nodes. An interesting aspect is that for the grids (GridN and GridK), ND is much larger than rotND/n, contrary to the other classes where those parameters show values quite similar to each other. This means that in general the number of ND paths from s to a particular node does not depend on the choice of the ending vertex.

Since *publicMOSP* will be used as a basic reference for comparing the performance of the other codes, we show, in Table 5, its behaviour for the same groups considered for Table 4. Note that, in spite of being a non sophisticated code, the *publicMOSP* is very effective (generating mostly ND labels) and, in consequence, running reasonably fast for all the groups, even for some of the largest size ones. In fact, for most of the cases, it does not require, in average, more than one minute for finding the solution. Also, let us refer that the results presented in the Table 5 clearly show that the complete networks are the hardest problems to solve.

In Table 6, we present the average ratios obtained for each code relatively to *publicMOSP*. Note that a value less than 1 means that the corresponding code outperforms the public code; if it is greater than 1, then *publicMOSP* performed better. For instance, for the test group RandN, the Label Correcting algorithm (Deque, $\leq_{max}$}) tooks, in average, less time and slightly the same number of iterations than *publicMOSP*. The full information provided at [22] shows that the size of instances does not have a significant influence on the ratios.

From Table 6, we can draw the following conclusions:

(1) The label setting algorithm is the most effective one. However, this is achieved by keeping the set X ordered accordingly to $\leq_R$ where $R \in \{lex, max, sum\}$, and, consequently, increasing the computational effort in terms of CPU time and required memory.
(2) The label correcting algorithm shows the best performance in terms of CPU time and used memory. This is a consequence of its simple



A graph
(a)

Lexicographic shortest tree $T^\star$
(b)

FIGURE 2. Network for the example.

(i,j)	(1,2)	(1,4)	(2,3)	(2,5)	(3,4)	(3,6)	(4,2)	(4,5)	(5,3)	(5,6)
$c(i,j)$	(8,4,1)	(4,7,0)	(9,8,4)	(8,0,7)	(1,5,9)	(2,6,9)	(7,4,1)	(3,1,8)	(10,8,2)	(8,2,8)

TABLE 1. Arc costs for the example of Figure 2(a).

the algorithm, Π_t will correspond to $\bar{D}$, the full set of non-dominated paths on the network from s to t.

4. Example

The network depicted in Figure 2(a) is used to illustrate the L&DP algorithm performance when it is considered the lexicographic order ($\leq_{lex}$). The costs for the arcs are given in Table 1 and the iterative results obtained for this example are shown in Figure 3. In the graph that figure, the successive generated deviation paths are represented with a dashed line identifying those ones not scanned so far.

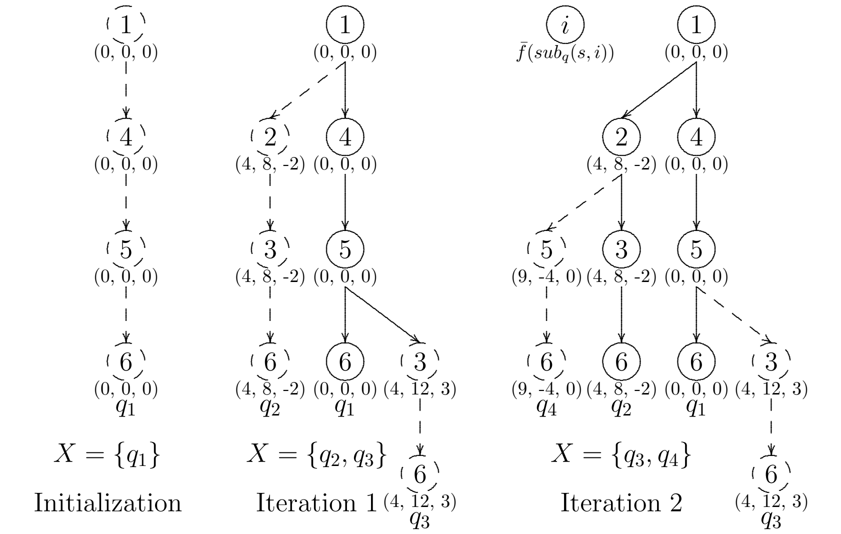
The lexicographic shortest path from 1 to 6 in the network, $T^\star(s) = \langle 1,4,5,6\rangle$, is the first deviation path (q_1) to be included in X. Then, q_1 is selected for scanning and, from that, two shortest deviation paths (q_2 and q_3) are generated. Note that $q_3 \in X$ in spite of being dominated by q_2, because ND deviation paths could be obtained from $sub_{q_3}(s,5) = \langle 1,4,5\rangle$. We would like to stress that the deviation path from q_1 at the node 4, $\langle 1,4,2\rangle \diamondsuit T^\star(2)$, is discarded since $\langle 1,4,2\rangle$ is dominated by $\langle 1,2\rangle$, already in Π_2.

At iteration 2, q_2 is scanned and q_4 is the only shortest deviation path to be added up to X. Then, q_3 and q_4 are analysed, respectively, at iteration 3 and 4, producing no new shortest deviation paths.

(i,j)	(1,4)	(1,2)	(2,3)	(2,5)	(3,6)	(3,4)	(4,5)	(4,2)	(5,6)	(5,3)
$\bar{c}(i,j)$	(0,0,0)	(4,8,-2)	(0,0,0)	(5,-12,2)	(0,0,0)	(10,2,16)	(0,0,0)	(7,15,-2)	(0,0,0)	(4,12,3)

TABLE 2. Reduced cost associated to $T^\star$ (Figure 2(b)) where $A(i)$ is lexicographically sorted by $\bar{c}_{i,j}$.

	X	Π_1	Π_2	Π_3	Π_4	Π_5	Π_6
Initialization	$\{q_1\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
Iteration 1	$\{q_2, q_3\}$	$\{\langle 1\rangle\}$	$\{\langle 1,2\rangle\}$	$\{\langle 1,4,5,3\rangle\}$	$\{\langle 1,4\rangle\}$	$\{\langle 1,4,5\rangle\}$	$\{q_1\}$
Iteration 2	$\{q_3, q_4\}$	$\{\langle 1\rangle\}$	$\{\langle 1,2\rangle\}$	$\{\langle 1,4,5,3\rangle, \langle 1,2,3\rangle\}$	$\{\langle 1,4\rangle\}$	$\{\langle 1,4,5\rangle, \langle 1,2,5\rangle\}$	$\{q_1, q_2\}$
Iteration 3	$\{q_4\}$	$\{\langle 1\rangle\}$	$\{\langle 1,2\rangle\}$	$\{\langle 1,4,5,3\rangle, \langle 1,2,3\rangle\}$	$\{\langle 1,4\rangle\}$	$\{\langle 1,4,5\rangle, \langle 1,2,5\rangle\}$	$\{q_1, q_2\}$
Iteration 4	$\emptyset$	$\{\langle 1\rangle\}$	$\{\langle 1,2\rangle\}$	$\{\langle 1,4,5,3\rangle, \langle 1,2,3\rangle\}$	$\{\langle 1,4\rangle\}$	$\{\langle 1,4,5\rangle, \langle 1,2,5\rangle\}$	$\{q_1, q_2, q_4\}$



(a) (b) (c)

FIGURE 3. Simulation of the new algorithm for the example of Figure 2(a).

Class	Network	Description	Groups
RandN	Random	$n \in \{i*1000 : i \in \mathbb{N}, 1 \le i \le 15\}$; $d = 6$; $k = 6$	15
RandD	Random	$n = 5000$; $d \in \{i : i \in \mathbb{N}, 2 \le i \le 10\}$; $k = 6$	9
RandK	Random	$n = 5000$; $d = 6$; $k \in \{i : i \in \mathbb{N}, 2 \le i \le 10\}$	9
CompN	Complete	$n \in \{i*10 : i \in \mathbb{N}, 1 \le i \le 12\}$; $d = n-1$; $k = 6$	12
CompK	Complete	$n = 100$; $d = n-1$; $k \in \{i : i \in \mathbb{N}, 2 \le i \le 10\}$	9
GridN	Square Grid	$n \in \{i^2 : i \in \mathbb{N}, 5 \le i \le 12\}$; $d \approx 4$; $k = 6$	8
GridK	Square Grid	$n = 100$; $d \approx 4$; $k =\in \{i : i \in \mathbb{N}, 2 \le i \le 10\}$	9

n = number of nodes; d = number of arcs/n; k = number of criteria

TABLE 3. **Set of test instances with the arc costs randomly generated in** $[1, 1000]$**, using an uniform distribution.**

Hence, aiming for a fair comparison between the algorithms, we followed the suggestion made at the "9th DIMACS Implementation Challenge - Shortest Paths" [4], carrying out computational experience with the very same set of instances and using a public code as a benchmarking reference. For this purpose, we produced a non optimized code for the label correcting approach using a label selection policy and a FIFO rule. Both, that code (*publicMOSP*) and the set of test instances in our computational experience, have been made public in the internet [23].

Let us say that we looked for other available codes but we only be succeeded with the ones proposed by Skriver and Andersen, [26]. However, those codes were developed only for the bicriteria case and proved to be much slower than our *publicMOSP* code (see [22]).

Concerning to the L&DP algorithm, we tested 6 versions resulting from the combination of the binary heap ([1]) and the Dial's structure for keeping the set candidates, with the $\le_{lex}$, $\le_{sum}$ and $\le_{max}$ order relations.

The computational experiments were carried out on a 3.00GHz Intel(R) Pentium(R) 4 processor with 1024 KB cache size and 384MB of RAM, running over Linux operating system (Suse 9.3 version), at the Laboratory for Computational Mathematics of Centre for Mathematics of the University of Coimbra. All the codes were written in the C language and the computational tests were made up using 50 instances for each one of the groups within the 7 classes of problems described in the Table 3.

A full description of the set of test problems is provided at [23], including the data relative to the 3550 instances considered in our computational experience. Hence, we opted for showing, in Table 4, some parameters - number

Corollary 3. : *Let $\leq_R$ be an order relation verifying properties 1 and 2 and assume that $f_\ell(C) > 0$ holds for every cycle C in the network and $\ell \in \{1, \ldots, k\}$. Then, the L&DP algorithm ranks the ND paths in G.*

We would like to emphasize that the previous result means that one can stop the L&DP algorithm before X becomes an empty set. In this case, the (temporary) ND paths already scanned by this procedure are the best ND paths following the order relation $\leq_R$.

6. Computational results

In this section, we report computational experience carried out for confronting the new algorithm with the two classical versions known for the labelling algorithm where, in short words, the nodes in the network are scanned accordingly to the respective labels. In the case of the label correcting approach, the node selected for scanning is the one associated with the oldest label, while for the label setting version, is the node with the minimum label for a specific order relation.

An extensive computational study on both of the above mentioned versions for the labelling algorithm is presented in [19]. There, we conclude that the best performance is achieved by the label correcting implementation following a particular label selection policy and a FIFO rule. Also, the *"double-end-queue"* (DEQUE) procedure proposed by Pape [20, 21], produced very good results when using the $\leq_{sum}$ or $\leq_{max}$ operators for positioning the elements in the queue.

Concerning to the label setting version, the experience reported in [19] shows that the best computational results are obtained when using a Dial structure [9, 10], sorted by the $\leq_{sum}$ or $\leq_{max}$ operators, for keeping the set of non-scanned labels. However, the label correcting approach consistently outperforms the label setting version mainly because, in this case, one has to find all of the ND labels. That is, the set of all non-dominated paths from s to every note $i \in N$.

Now, let us refer that, in the literature, there are several papers reporting computational experiments for the labelling algorithm [2, 6, 7, 12, 13, 14, 18, 19, 24, 25, 26, 27]. Nevertheless, it is not easy to use those results as a basis for a fair benchmarking since they are produced by codes made by different people and ran on different machines. Also, the set of test instances differs significantly from work to work.

5. Correctness of the algorithm

In this section, we will prove that, considering an order relation $\leq_R$ holding the properties 1 and 2 (see section 2), the L&DP algorithm computes the full set of non-dominated paths after a finite number of iterations. Actually, as will be shown, the algorithm finds the full set of non-dominated from s to any other vertex on the network. First of all, we recall the following basic result which proof can be seen in [15]:

Theorem 1. : *Assume that for each cycle C of the network $f_\ell(C) \geq 0, \forall \ell \in \{1, \ldots, k\}$. Then, the MSPP holds the Optimality Principle stating that each non-dominated path is formed by non-dominated subpaths.*

Now, let us remind that $P_{s,i}$ is the set of paths from s to i and $\bar{D}_{s,i} \subseteq P_{s,i}$ corresponds to the subset of non-dominated paths between those nodes. In the algorithm, the set Π_i $(i \in N)$ is used to store any path w, from s to i, generated in **step 2** and not discarded by the logical function $DT(w, \Pi_i)$, that returns the value *"true"* if and only if there is a path $q \in \Pi_i$ such that $q <_D w$. Whenever a new path w joins the set Π_i this is updated by removing all the paths u such that $DT(u, \{w\}) = true$. So, we may say that, until the end of the procedure, the paths in any Π_i $(i \in N)$ are temporarily non-dominated. Now, the following result becomes quite obvious:

Lemma 5. : *Let p be a path of $\bar{D}_{s,i}$ that has been included in Π_i. Then, p will never be removed from Π_i.*

Let us recall that X is the set of candidates for the next $\leq_R$-shortest ND path and that the algorithm stops when X turns out to be an empty set. At an intermediate step, a path p is selected from X and the arcs of $A(tail(\theta(p))$ are scanned in order to find new shortest deviation paths. Remember that $\theta(p)$ is the deviation arc for p and that $A(v_i)$, the set of arcs having i as tail node, is ordered by non-decreasing reduced cost. Next, we show that the algorithm checks all the arcs in the network that may lead to ND-paths.

Lemma 6. : *Let $z = \langle v_0, \ldots, v_i \rangle$ be a ND path from s to v_i and suppose that $z \diamondsuit T^\star(v_i) \in X$. Then, all arcs of $A(v_i)$ are scanned by the algorithm.*

<u>Proof</u>: The first arc of $A(v_i)$ is in $T^\star$ and, necessarily, is considered by the algorithm, since $z \diamondsuit T^\star(v_i) \in X$. On the other hand, we know that this path will be selected at some iteration, on step 2, with $\theta(p) = (v_j, v_{j+1})$ for some $j \in \{0, 1, \ldots, i\}$. When that happens, every vertex subsequent to v_j in p,

will be analyzed having in mind the generation of shortest deviation paths. In particular, v_i will be "visited" looking for a new active arc (v_i, h) of $A(v_i)$ leading, even temporarily, to a ND path. If such an arc is found, $q^p_{v_i,h}$ is a shortest deviation path from p included in X. Later, the vertex v_i will be visited again when $q^p_{v_i,h}$ is picked up from X and a new active arc of $A(v_i)$ will be searched. At some point, one may not find an arc in $A(v_i)$ that leads to a temporary ND path. This means that all the arcs in $A(v_i)$ have been scanned by the algorithm. □

Now, in order to proof the correctness of the algorithm, we show first that all ND paths from s to t are detected and included in Π_t. Actually, this is true for any ND path from s to any other node $i \in N$ which will be an element of Π_i, at the end of the procedure. To assure the correctness of the algorithm, we proof that a finite number of iterations is required for computing $\bar{D}$.

Theorem 2. : *Let Π_t be the set of temporary non-dominated paths and assume that the Optimality Principle is verified. Then, at the end of the L&DP algorithm,*
$\Pi_t = \bar{D}$.

Proof: Firstly, let us see that $\bar{D} \subseteq \Pi_t$. If $\langle v_0 = s, \ldots, v_\ell = t\rangle$ is a path of $\bar{D}_{s,t}$ then, from the Optimality Principle, all subpaths of $\langle v_0, \ldots, v_\ell\rangle$ are ND. It is easy to show, by induction, that $\langle v_0, \ldots, v_i\rangle \in \Pi_{v_i}$, for all $i \in \{0, \ldots, \ell\}$. In fact, in the first iteration of the algorithm, $\langle v_0\rangle = \langle s\rangle$ is included in Π_s. Furthermore, if $\langle v_0, \ldots, v_i\rangle \in \Pi_{v_i}$ (for some $i \in \{0, \ldots, \ell - 1\}$) then all arcs $A(v_i)$ will be scanned (Lemma 6) and so, at some stage of the algorithm, $\langle v_0, \ldots, v_i, v_{i+1}\rangle$ will join the elements of $\Pi_{v_{i+1}}$. Since $\langle v_0, \ldots, v_i, v_{i+1}\rangle$ is a ND sub-path, from the Lemma 5, it will be not further removed from $\Pi_{v_{i+1}}$. Therefore, at the end of the algorithm, $\langle v_0 = s, \ldots, v_\ell = t\rangle$ will be an element of Π_t.

Now, let us show that $\Pi_t \subseteq \bar{D}$. Suppose that $p \in \Pi_t$ and $p \notin \bar{D}$. Hence, there is a path $q \in \bar{D}$ that $q <_D p$ and, as seen above, q will be added to Π_t at some iteration of the algorithm. This could not happen before p joined Π_t since the dominance test, performed at the step 2.2.2 of the L&DP algorithm, would prevent the possibility of including p in Π_t. Therefore, $p \in \Pi_t$ when q is determined by the algorithm. Then, in this case, either at step 2.2.1 or step 2.2.3, the dominance test will remove p from Π_t. □

Corollary 2. : *If the Optimality Principle is verified then, at the end of the L&DP algorithm,* $\Pi_i = \bar{D}_{s,i}, \forall i \in N$.

The Theorem 2 guarantees that the L&DP algorithm determines all the ND paths from a source node to a sink node in a network. Next, we show that this is achieved in a finite number of iterations.

Lemma 7. : *If $0 <_D f(C)$, for any cycle C in the network, then every temporary non-dominated path has no cycles.*

Proof: Let $p = w_1 \diamondsuit C \diamondsuit w_2$ be a non simple path from s to a node v_ℓ where $C = \langle u_0 = v_i, \ldots, u_j = v_i\rangle$ is a cycle, $w_1 = \langle v_0 = s, \ldots, v_i\rangle$ and $w_2 = \langle v_i, \ldots, v_\ell\rangle$. Then, suppose that p is produced, at some stage, by the algorithm. This means that p is a deviation path from another one previously found, say q with $\theta(p) = (x, y)$. Hence, $p = sub_q(s, x) \diamondsuit \langle x, y\rangle \diamondsuit T^\star(y)$ and, since $T^\star(y)$ is a simple path, there exists a node $j \in sub_q(s, x) \cap T^\star(y)$ such that $C = sub_q(j, x) \diamondsuit \langle x, y\rangle \diamondsuit sub_{T^\star(y)}(y, j)$. Consequently, $sub_q(s, j) \diamondsuit sub_{T^\star(y)}(j, t) = w_1 \diamondsuit w_2$ is also a deviation path from q, previously found by the algorithm and, under the assumption, dominating p. □

Theorem 3. : *Assuming that $0 <_D f(C)$ holds for every cycle C in the network, the L&DP algorithm computes $\bar{D}$ in a finite number of iterations.*

Proof: Under the assumption, the Optimality Principle is verified (Theorem 1). From Lemma 7, we know that the algorithm only generates elementary paths as possible elements for the sets Π_i. Therefore, only a finite number of iterations is required for, as stated in Theorem 2, computing the full set $\bar{D}$. □

At last, we prove that the L&DP algorithm finds the ND paths in the $\leq_R$ order (Corollary 3) using the next result.

Theorem 4. : *Let $\leq_R$ be an order relation holding properties 1 and 2 and p be the path selected from the set X, at step 2.1 of L&DP algorithm. A sub-path $w = sub_p(s, v_i)$, with v_i in p such that "$DT(w, \Pi_{v_i})$ = false", is a ND-path.*

Proof: In fact, property 2 assures that L&DP algorithm computed paths in G by non-decreasing order of its labels. On the other hand, if $q <_D w = sub_p(s, v_i)$, then $q \leq_R w$ (property 1). Hence, q is selected by the algorithm before w and, consequently, "$DT(w, \Pi_{v_i})$ = true". □