Computers & Operations Research ▮▮▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

An effective algorithm for obtaining the minimal cost pair of disjoint paths with dual arc costs☆

Teresa Gomes[a,b,*], José Craveirinha[a,b], Luísa Jorge[b,c]

[a]Department of Electrical and Computer Engineering, University of Coimbra, Pólo II, 3030-290 Coimbra, Portugal
[b]INESC-Coimbra, Rua Antero de Quental 199, 3000-033 Coimbra, Portugal
[c]Polytechnic Institute of Bragança, Campus de Sta Apolónia, 5301-857 Bragança, Portugal

ARTICLE INFO

Keywords:
OR in telecommunications
Paths with minimal cost sum
Dual arc costs
Disjoint paths

ABSTRACT

Routing optimisation in some types of networks requires the calculation of the minimal cost pair of disjoint paths such that the cost functions associated with the arcs in the two paths are different. An exact algorithm for solving this NP-complete problem is proposed, based on a condition which guarantees that the optimal path pair cost has been obtained. This optimality condition is based on the calculation of upper and lower bounds on the optimal cost. A formal proof of the correctness of the algorithm is described. Extensive experimentation is presented to show the effectiveness of the algorithm, including a comparison with an integer linear programming formulation.

1. Introduction

1.1. Background and motivation

In today's telecommunications networks it is necessary, for reliability reasons, to use protection schemes involving the calculation of two (or more) disjoint paths for each node-to-node connection, especially when large amounts of traffic have to be routed in the network. This concern is particularly relevant in optical networks, namely WDM (wavelength division multiplexing) networks due to the very high rates supported by lightpaths, and in the Internet using MPLS (multiprotocol label switching). In this context the problem of obtaining (arc or node) disjoint paths, for increasing network reliability while minimising bandwidth consumption, is extremely important. In telecommunication networks, when diverse routing is used, the path that carries traffic under normal conditions is called the active path (AP), and the path that carries traffic when some failure affects the AP is called the backup path (BP).

In a network, with certain type of route protection schemes, the minimisation of bandwidth usage can be achieved by solving the min-sum problem: finding k disjoint paths between two (distinct) nodes s and t such that the sum of the cost of the routes is minimised. A polynomial-time algorithm for solving this problem was proposed in [1] and a more efficient version for $k=2$ was presented in [2]. In this type of problem the arc costs are uniform, that is, they have the same value regardless of the considered path.

In order to reduce bandwidth usage, it is desirable to allow bandwidth sharing among BPs of disjoint APs. This is a condition that leads to networks with non-uniform arc costs, that is, networks with different arc costs in the APs and BPs. In fact the reserved bandwidth in each arc of a BP associated with a given AP is less than or equal to the bandwidth that would be required in that same arc by the AP. Hence the problem of finding a pair of disjoint paths (the AP and the BP) leads to the min-sum problem with ordered dual costs, or MSOD problem. This problem was shown to be NP-complete for undirected and directed networks in [3] and [4], respectively.

The problem of finding k disjoint paths from s to t (two distinct nodes), in a network with k different costs on every arc such that the total cost of the paths is minimised, was studied in [5]. The paths may be arc or node disjoint and the networks may be directed or undirected. Firstly it is proved that this problem is strongly NP-complete even for $k=2$, when the relationship between the k arc costs (in the same arc) is arbitrary. Two polynomial-time heuristics, for the problem of finding disjoint paths with min-sum objective function when the cost structure is not uniform, were then proposed. Worst-case bounds were obtained for both heuristics.

The problem addressed in this paper, relevant in the context of survivable routing, is the calculation of a pair of arc-disjoint paths from a source to a destination in a network where two different

☆ Work partially supported by programme POSC of the EC programme cosponsored by national funds.

* Corresponding author at: Department of Electrical and Computer Engineering, Pólo II of Coimbra University, 3030-290 Coimbra, Portugal. Tel.: +351 239 796261; fax: +351 239 796247.
E-mail addresses: teresa@deec.uc.pt (T. Gomes), jcrav@deec.uc.pt (J. Craveirinha), ljorge@inescc.pt (L. Jorge).

0305-0548/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2008.04.002

and unordered costs are assigned to the arcs, a situation usually designated as dual arc costs [3].

Several heuristics for obtaining an asymmetrically weighted pair of disjoint paths with minimal cost, a particular type of the MSOD problem, are described in [4]. The proposed heuristics are based on k-shortest path searching, Suurballe's algorithm [2], integer linear programming (ILP), linear relaxation and minimum cost network flow (MCNF) techniques.

In [6] the problem of dynamic routing of restorable bandwidth-guaranteed LSPs (label switched paths) in MPLS networks was addressed, and algorithms for setting up such LSPs were proposed. This problem has, as a sub-problem, the determination of a minimal cost pair of disjoint paths with different path costs. In [6] it was considered that the worst-case guarantee in [5] was not good enough for this purpose, so a different approach to the problem of finding a pair of disjoint paths with different arc costs, was presented. This was based on a mathematical programming formulation of the problem. In order to obtain lower bounds on the optimal solution value of this problem, a relaxation of the integrity constraints was considered, as well as the dual of this linear programming problem. A heuristic that calculates a disjoint pair of paths, as well as upper and lower bounds for the optimal disjoint path pair cost, was then described.

The approach of Ho et al. [7], to the problem of survivable routing, requires solving the min-sum problem concerning the cost of the active and backup disjoint path pair, where the protection path depends on the chosen AP (this a problem similar to the one addressed in [6]). In [7] this problem is formulated as an ILP process. Since the problem is NP-complete two heuristics are also proposed: the iterative two-step-approach (ITSA) and the maximum likehood relaxation (MLR). The MLR is a modified Dijkstra's algorithm which has polynomial-time complexity. The ITSA was already outlined in [4] where it was designated as an enhancement to the TSA. The TSA uses a shortest path algorithm for obtaining the AP, then removes the arcs of the AP from the network; finally the shortest path algorithm is used again for obtaining the BP. ITSA iteratively inspects k-shortest paths as APs, in an ascending order of cost, from source to destination. The TSA is used in every iteration of the ITSA until the optimal path pair is obtained or a stopping criterion is satisfied. The efficiency of the ITSA is determined by the efficiency of the k-shortest path algorithm. However, according to [8], the ITSA can work extremely well in solving the diverse routing problem with shared protection.

1.2. Contributions of the paper

In this paper we propose an exact algorithm for finding an arc-disjoint path pair, with minimal cost in a network with dual arc costs, as formalised in the next sub-section. We also analyse its effectiveness in a significant number of randomly generated networks.

The exactness of the algorithm results from the fact that it enables the calculation of optimal solution(s) and guarantees its (their) optimality. Furthermore the algorithm is founded on the calculation of upper and lower bounds on the optimal cost. The algorithm is based on the resolution of two k-shortest path problems in an articulate manner, until an optimal stopping condition is verified.

An ILP formulation of the problem is also presented for performance comparison with the proposed resolution approach.

Experimental results with randomly generated networks, having up to 3200 nodes, show that the algorithm solves the problem exactly in practically all the cases in directed networks, the cases of failure being due to memory exhaustion alone. In the small percentage of cases in which an optimal solution is not attained, the solution provided by the algorithm is sub-optimal, and lower and upper bounds for the optimal cost can be calculated. The performance comparison with the ILP approach was carried out using ILOG CPLEX. The experimental results showed that the proposed algorithm enables optimal solutions to be obtained in much shorter CPU times than CPLEX. Moreover the algorithm is capable of solving larger problems than the ILP approach using CPLEX.

The paper is organised in the following manner. In Section 1.3 the problem is formalised, the notation is introduced as well as an ILP formulation of the problem. The main steps of the algorithm are presented together with two statements concerning the lower and upper bounds of the optimal cost, and the detection of the optimal path pair cost; finally the exactness of the algorithm is proved. Extensive experimental results with randomly generated directed networks having different cost ranges are presented and analysed in Section 3, including a comparison with the results obtained with the ILP formulation, using CPLEX. Finally, some conclusions are drawn in Section 4.

1.3. Problem formulation

Let $G=(V,E)$ be a directed network with node set $V=\{v_1,v_2,\ldots,v_n\}$ and arc set $E=\{e_1,e_2,\ldots,e_m\}$ (where n and m designate the number of nodes and arcs in G, respectively), where two different non-negative cost functions (or metrics) in the arcs, are defined:

$$\eta^{(j)}:E\to\mathbb{N}_0 \quad (j=1,2) \tag{1}$$

$$\eta^{(j)}((v_a,v_b))=c^{(j)}_{v_av_b} \quad (v_a,v_b)\in E \tag{2}$$

The cost $C^{(j)}$ of a (loopless) path p in G with respect to metric $\eta^{(j)}$, is

$$C^{(j)}(p)=\sum_{(v_a,v_b)\in p} c^{(j)}_{v_av_b} \quad (j=1,2) \tag{3}$$

Let path p, $p=\langle v_1,e_1,v_2,\ldots,v_{i-1},e_{i-1},v_i\rangle$, be given as an alternate sequence of nodes and arcs from G, such that the tail of e_k is v_k and the head of e_k is v_{k+1}, for $k=1,2,\ldots,i-1$ (all the v_i in p are different). Let the set of nodes in p be $V^*(p)$ and the set of arcs in p be $E^*(p)$. Two paths $p=\langle v_1,e_1,v_2,\ldots,v_{i-1},e_{i-1},v_i\rangle$ and q are arc-disjoint if $E^*(p)\cap E^*(q)=\emptyset$. Two paths p and q are disjoint if $V^*(p)\cap V^*(q)=\emptyset$, and are internally disjoint [9] if $\{v_2,\ldots,v_{i-1}\}\cap V^*(q)=\emptyset$. We will say that two paths are node disjoint if they are internally disjoint.

The addressed problem is to find a pair (p,q) of arc-disjoint paths which minimises the total cost of the pair, defined by

$$C[(p,q)]=C^{(1)}(p)+C^{(2)}(q) \tag{4}$$

where p and q have the same source and sink node. We recall that, in [5], this problem was proved to be NP-complete.

The interest in developing an exact and effective solution to the problem of obtaining the minimal cost pair of disjoint paths with arbitrary dual arc costs, having in mind possible applications (namely in telecommunication networks), provided the major motivation for this paper.

The problem of finding a pair (p,q) of arc-disjoint paths, which minimises the total cost of the pair (see Eq. (4)) can also be formulated as an ILP problem, following a procedure similar to [10].

The flow conservation indicator, defined for all nodes $v_k\in V$:

$$f(v_k)=\begin{cases}1 & \text{if } v_k \text{ is the source of } p \text{ and } q\\ -1 & \text{if } v_k \text{ is the sink of } p \text{ and } q\\ 0 & \text{otherwise}\end{cases} \tag{5}$$

Objective:

$$\min\sum_{j=1,2}\sum_{(v_a,v_b)\in E} x^{(j)}_{v_av_b}c^{(j)}_{v_av_b} \tag{6}$$

Constraints:

$$\sum_{v_i:(v_k,v_i)\in E} x^{(j)}_{v_kv_i}-\sum_{v_h:(v_h,v_k)\in E} x^{(j)}_{v_hv_k}=f(v_k),\quad \forall v_k\in V,\ \ j=1,2 \tag{7}$$

$$x^{(1)}_{v_k v_i} + x^{(2)}_{v_k v_i} \leqslant 1, \quad \forall (v_k, v_i) \in E \tag{8}$$

$$x^{(j)}_{v_k v_i} \in \{0, 1\}, \quad \forall (v_k, v_i) \in E, \ j = 1, 2 \tag{9}$$

where

$$x^{(1)}_{v_a v_b} = \begin{cases} 1 & \text{if } (v_a, v_b) \in p \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$x^{(2)}_{v_a v_b} = \begin{cases} 1 & \text{if } (v_a, v_b) \in q \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

with $(v_a, v_b) \in E$.

This formulation will only be used for evaluating the algorithm performance with respect to a benchmarking standard tool which might be applicable to the problem under analysis.

2. Proposed approach

Let $p^{(1)}_h$ be the h-shortest (loopless) path from s to t with respect to metric $\eta^{(1)}$ (obtained by MPS in its loopless version [11], for example). Let $q(p^{(1)}_h)^{(2)}$ be the shortest path from s to t with respect to $\eta^{(2)}$ (obtained using the Dijkstra algorithm) in the graph $G^{(1)}$ obtained from G by removing (temporarily) the arcs in $p^{(1)}_h$.

Let $p^{(2)}_k$ be the k-shortest (loopless) path for s to t with respect to $\eta^{(2)}$. Let $q(p^{(2)}_k)^{(1)}$ be the shortest path from s to t with respect to $\eta^{(1)}$ in the graph $G^{(2)}$, obtained from G by removing (temporarily) the arcs in $p^{(2)}_k$.

Let $A_h = (p^{(1)}_h, q(p^{(1)}_h)^{(2)})$, $B_k = (q(p^{(2)}_k)^{(1)}, p^{(2)}_k)$, $C(A_h) = C^{(1)}(p^{(1)}_h) + C^{(2)}[q(p^{(1)}_h)^{(2)}]$, and $C(B_k) = C^{(1)}[q(p^{(2)}_k)^{(1)}] + C^{(2)}(p^{(2)}_k)$.

2.1. Main steps of the algorithm

A k-shortest path enumeration algorithm, using arc costs $c^{(1)}_{ij}$, obtains the paths $p^{(1)}_h$, $h = 1, 2, \ldots, W$, where W is the total number of paths from s to t in the network. The arcs of each path $p^{(1)}_h$ are temporarily removed from the network graph and the shortest path with respect to metric $\eta^{(2)}$, $q(p^{(1)}_h)^{(2)}$ (disjoint with $p^{(1)}_h$) is obtained in the resulting graph (using for example the Dijkstra algorithm) by using arc costs $c^{(2)}_{ij}$ (with the cost of the removed arcs equal to ∞). In this manner, (*procedure* $\mathscr{A}$) it is possible to keep track of the current best candidate pair of paths found so far, the one with cost $\min_h C(A_h)$.

Now let the arc costs $c^{(2)}_{ij}$ be used for obtaining the k-shortest paths $p^{(2)}_k$, $k = 1, 2, \ldots, W$. For each $p^{(2)}_k$ all its arcs are temporarily removed from the network graph and the shortest path with respect to metric $\eta^{(1)}$, disjoint with $p^{(2)}_k$, is obtained. In this manner (*procedure* $\mathscr{B}$) it is also possible to keep track of the current best candidate pair of paths found so far, the one with cost $\min_k C(B_k)$. Procedure $\mathscr{B}$ can be considered as 'symmetrical' to procedure $\mathscr{A}$.

Given the best current paths in procedures $\mathscr{A}$ and $\mathscr{B}$, upper and lower bounds for the minimal value of $C[(p, q)]$ (4), c_{opt}, can be obtained and the width of the interval containing c_{opt} can be calculated.

Lemma 2.1 (*Lower and upper bounds for* c_{opt}). *Let* $A_h = (p^{(1)}_h, q(p^{(1)}_h)^{(2)})$ *and* $B = (q(p^{(2)}_k)^{(1)}, p^{(2)}_k)$ *be the current minimal cost path pairs obtained using procedures* $\mathscr{A}$ *and* $\mathscr{B}$, *respectively.*

Procedures $\mathscr{A}$ *and* $\mathscr{B}$ *ensure that* A_h *and* B_k *are path pairs such that*

$$C(A_h) = \min_i C[(p^{(1)}_i, q(p^{(1)}_i)^{(2)})], \quad i = 1, \ldots, u \tag{12}$$

$$C(B_k) = \min_j C[(q(p^{(2)}_j)^{(1)}, p^{(2)}_j)], \quad j = 1, \ldots, v \tag{13}$$

with $1 \leqslant h \leqslant u \leqslant W$ *and* $1 \leqslant k \leqslant v \leqslant W$.

Let

$$c_{\text{L}} = C^{(1)}(p^{(1)}_h) + C^{(2)}(p^{(2)}_k) \tag{14}$$

$$c_{\text{U}} = \min[C(A_h), C(B_k)] \tag{15}$$

$$r = c_{\text{U}} - c_{\text{L}} \tag{16}$$

then

$$c_{\text{L}} \leqslant c_{\text{opt}} \leqslant c_{\text{U}} \quad \text{or} \quad C(A_h) = C(B_k) = c_{\text{opt}} \tag{17}$$

Proof. The j-th element ($j = 1, 2$) of A_h or B_k is the path which was obtained by using metric $\eta^{(j)}$. Whenever a path $p^{(1)}_h$ ($p^{(2)}_k$) does not have a disjoint path $q(p^{(1)}_h)^{(2)}$ ($q(p^{(2)}_k)^{(1)}$) the cost of A_h (B_k) is ∞.

By construction, paths $p^{(1)}_i$ ($p^{(2)}_j$) are obtained by non-decreasing order of their cost, with respect to metric $\eta^{(1)}$ ($\eta^{(2)}$).

By definition $c_{\text{opt}} = \min_i C(A_i) = \min_j C(B_j)$, $1 \leqslant i, j \leqslant W$, therefore $c_{\text{opt}} \leqslant \min[C(A), C(B)]$. This proves (15) is an upper bound for c_{opt}.

First case: assume that $C(B_k) > c_{\text{opt}}$ and $C(A_h) > c_{\text{opt}}$. This implies that a path pair B_j ($j > k$), can be found (using procedure $\mathscr{B}$) such that $C(B_j) = c_{\text{opt}}$ and

$$C^{(2)}(p^{(2)}_j) \geqslant C^{(2)}(p^{(2)}_k) \tag{18}$$

$$C^{(1)}[q(p^{(2)}_j)^{(1)}] \geqslant C^{(1)}(p^{(1)}_h) \tag{19}$$

Procedure $\mathscr{B}$ ensures Eq. (18) is true. Eq. (19) is verified otherwise we necessarily have $C(A) = c_{\text{opt}}$.

By adding (18) and (19) we have $c_{\text{L}} \leqslant c_{\text{opt}}$ and $r > 0$ (because $c_{\text{L}} \leqslant c_{\text{opt}} < c_{\text{U}}$).

Second case: assume $C(B_k) > C(A_h)$ and $C(A_h) = c_{\text{opt}}$. This implies that a path pair B_j ($j > k$) can be found (using procedure $\mathscr{B}$) such that $C(B_j) = c_{\text{opt}}$ and $p^{(2)}_j = q(p^{(1)}_h)^{(2)}$. Similarly to (18) and (19) we have

$$C^{(2)}(p^{(2)}_j) \geqslant C^{(2)}(p^{(2)}_k) \tag{20}$$

$$C^{(1)}[q(p^{(2)}_j)^{(1)}] = C^{(1)}(p^{(1)}_h) \tag{21}$$

By adding (20) and (21) we have $c_{\text{L}} \leqslant c_{\text{opt}}$.

Third case: consider $C(A_h) > C(B_k)$ and $C(B_k) = c_{\text{opt}}$. A proof similar to the second case ($C(B_k) > C(A_h)$ and $C(A_h) = c_{\text{opt}}$) can be made and is omitted.

Fourth case: $C(A_h) = C(B_k) = c_{\text{opt}}$. This completes the proof. □

Note that $r > 0$ in case 1, $r \geqslant 0$ in cases 2 and 3, and $r \leqslant 0$ in case 4.

Corollary 2.1 (*Optimal cost detection*). *Let* $A_h = (p^{(1)}_h, q(p^{(1)}_h)^{(2)})$ *and* $B = (q(p^{(2)}_k)^{(1)}, p^{(2)}_k)$ *be the current minimal cost path pairs obtained using procedures* $\mathscr{A}$ *and* $\mathscr{B}$, *respectively*, $1 \leqslant h \leqslant u \leqslant W$ *and* $1 \leqslant k \leqslant v \leqslant W$, *where* u *and* v *are the order of the last paths generated by each procedure.*

If $r \leqslant 0$, c_{opt} *has been reached by procedure* $\mathscr{A}$ *or* $\mathscr{B}$.

Proof. We will consider two cases: $r = 0$ and $r < 0$.

First case: $r = 0$.

Since $r = 0$, we are necessarily in cases 2, 3 or 4 of Lemma 2.1. In each one of these cases, at least one of the two path pairs A_h and B_k is optimal.

Second case: $r < 0$.

By Lemma 2.1, r can only be negative if $C(A_h) = C(B_k) = c_{\text{opt}}$. □

Note that Corollary 2.1 does not state that if $r > 0$ then $c_{\text{opt}} \neq \min[C(A), C(B)]$, but only that $r \leqslant 0$ implies $c_{\text{opt}} = \min[C(A), C(B)]$.

Now we will show that if procedures $\mathscr{A}$ and $\mathscr{B}$ are executed in an articulate manner, an exact algorithm for calculating a pair of arc-disjoint paths can be obtained.

2.2. Detailed description of the algorithm

An algorithm which solves the problem of obtaining the arc-disjoint path pair with minimal $C[(p,q)]$ (see Eq. (4)) by using a k-shortest path enumeration algorithm (such as MPS [12] in its loopless version [11]), and an algorithm for finding the shortest path between a pair of nodes (for example the Dijkstra algorithm) will be presented. Note that although Yen's algorithm has the lowest worst-case complexity among k-shortest path ranking algorithms [13,14], we prefer to use MPS because, in [11], experimental results show that in practical situations this algorithm is more efficient than Yen's, in terms of CPU time and RAM space.

In the algorithm the current best path pair is stored in $A=(p^{(1)},q(p^{(1)})^{(2)})$ and $B=(q(p^{(2)})^{(1)},p^{(2)})$ in procedures $\mathscr{A}$ and $\mathscr{B}$, respectively.

Algorithm DP2LC

```
(1)   Comment: Find the first pair of disjoint paths, by using MPS in the graph
      with costs $c_{ij}^{(1)}$, and using Dijkstra in the pruned network, with costs $c_{ij}^{(2)}$.
      (a)   $u \leftarrow 0$
      (b)   Do
                (i) $u \leftarrow u+1$;
                (ii) MPS (loopless) generates $p_u^{(1)}$, and Dijkstra finds $q(p_u^{(1)})^{(2)}$.
            While $C(A_u)=\infty$
(2)   Comment: Find the first pair of disjoint paths, by using MPS in the graph
      with costs $c_{ij}^{(2)}$, and using Dijkstra in the pruned network, with costs $c_{ij}^{(1)}$.
      (a)   $v \leftarrow 0$
      (b)   Do
                (i) $v \leftarrow v+1$;
                (ii) MPS (loopless) generates $p_v^{(2)}$, and Dijkstra finds $q(p_v^{(2)})^{(1)}$.
            While $C(B_v)=\infty$
(3)   $A \leftarrow A_u$, $B \leftarrow B_v$
(4)   $r \leftarrow \min[C(A),C(B)]-[C^{(1)}(p^{(1)})+C^{(2)}(p^{(2)})]$
(5)   Comment: Identify (and generate) an optimal pair of disjoint paths.
      While $r>0$ Do
      (a)   Procedure $\mathscr{A}$:
            While $r>0 \wedge C(B)\leqslant C(A)$ Do
                (i) $u \leftarrow u+1$
                (ii) MPS (loopless) generates $p_u^{(1)}$, and Dijkstra generates $q(p_u^{(1)})^{(2)}$
                (iii) If $C(A_u)<C(A) \vee [C(A_u)=C(A) \wedge C^{(1)}(p_u^{(1)})>C^{(1)}(p^{(1)})]$ Then
                          (A) $A \leftarrow A_u$
                          (B) $r \leftarrow \min[C(A),C(B)]-[C^{(1)}(p^{(1)})+C^{(2)}(p^{(2)})]$
                      End If
            End While
      (b)   Procedure $\mathscr{B}$:
            While $r>0 \wedge C(A)\leqslant C(B)$ Do
                (i) $v \leftarrow v+1$
                (ii) MPS (loopless) generates $p_v^{(2)}$, and Dijkstra generates $q(p_v^{(2)})^{(1)}$
                (iii) If $C(B_v)<C(B) \vee [C(B_v)=C(B) \wedge C^{(2)}(p_v^{(2)})>C^{(2)}(p^{(2)})]$ Then
                          (A) $B \leftarrow B_v$
                          (B) $r \leftarrow \min[C(A),C(B)]-[C^{(1)}(p^{(1)})+C^{(2)}(p^{(2)})]$
                      End If
            End While
      End While
```

The first two steps of the algorithm consist of obtaining the first pair of disjoint paths, A and B, in each procedure. Then the main cycle (step 5) seeks the improvement of A and B, which are updated in steps 5a and 5b, respectively, as soon as the algorithm can improve the bounds c_L or c_U. When $r\leqslant 0$ the optimal path cost was found.

The output of the algorithm is composed of a pair, the one with minimal cost, or two pairs (A and B if $C(A)=C(B)$) which either are equal or have equal cost. In both cases the optimal path cost was obtained, and at least a minimal cost path pair was identified.

2.3. Proof of the correctness of the algorithm

The proof takes into account the control and stopping conditions of the algorithm and Corollary 2.1. The proof of the algorithm correctness also assumes that $A=(p^{(1)},q(p^{(1)})^{(2)})$ $(B=(q(p^{(2)})^{(1)},p^{(2)}))$, obtained in step 5a (5b), corresponding to procedure $\mathscr{A}$ ($\mathscr{B}$), when u (v) paths have been generated by the k-shortest path enumeration algorithm, using metric $\eta^{(1)}$ ($\eta^{(2)}$), such that $C(A)=\min_i C[(p_i^{(1)}),q^{(2)}(p_i^{(1)})]$, $i=1,\ldots,u$ $(C(B)=\min_j C[(q(p_j^{(2)})^{(1)},p_j^{(2)})]$, $j=1,\ldots,v)$.

Note that the first pair of disjoint paths was obtained, for procedures $\mathscr{A}$ and $\mathscr{B}$, in steps 1 and 2.

Proposition 2.1 (*Correctness of DP2LC*). *Algorithm DP2LC obtains c_{opt}, assuming at least a link disjoint path pair exists in the network.*

Proof. If, when entering step 5 of DP2LC, $r \leqslant 0$, then from Corollary 2.1 c_{opt} has been found and $c_{\text{opt}} = \min[C(A_u), C(B_v)]$—and zero iterations of cycle 5 were carried out.

If that is not the case, then the algorithm enters cycle 5. If necessary the algorithm can enumerate all paths $p_u^{(1)}$ ($p_v^{(2)}$). The first optimal path pair found will be enumerated either in step 5a or step in 5b. Suppose the algorithm quits cycle 5a with and optimal path pair P (A or B). Therefore we will necessarily have $r \leqslant 0$ or, $r > 0$ and $C(P = A) < C(B)$. If $r \leqslant 0$, by Corollary 2.1 $c_{\text{opt}} = c_{\text{U}} = \min[C(A), C(B)]$. If $r > 0$ then necessarily $C(P = A) < C(B)$ and the algorithm remains in 5b until $r \leqslant 0$ (that is until $C(B) = C(A) = c_{\text{opt}}$ and $r \leqslant 0$, or $c_{\text{opt}} = C(A)$ and $r = 0$). □

2.4. Length constraint

Note that the previous approach can also be used for obtaining the minimal cost disjoint path pair with constraints on the maximum number of arcs allowed per path, a problem of interest in various applications, namely in telecommunication networks. If an algorithm (such as KD in [15]) for enumerating the k-shortest paths with length constraints was used instead of a k-shortest path enumeration algorithm (MPS loopless [11]) and if the Bellman-Ford (see [16]) algorithm was used instead of the Dijkstra algorithm, the obtained pair would be the optimal disjoint path pair, satisfying the desired length constraint.

2.5. Pair of node-disjoint paths

If an optimal pair of node-disjoint paths is desired, then, in procedure $\mathscr{A}$, all arcs incident and all arcs emergent from the nodes belonging to $p_h^{(1)}$ (with the exception of s and t) are removed from the graph and a node disjoint path, $q(p_h^{(1)})^{(2)}$, is sought by using the Dijkstra algorithm. Similarly for the pair $p_k^{(2)}$ and $q(p_k^{(2)})^{(1)}$, in procedure $\mathscr{B}$. Therefore the previous algorithm (with this simple adaptation) would also solve the problem of obtaining the node-disjoint path pair of minimal cost. However, the performance of the algorithm in this case would have to be evaluated.

2.6. Directed and undirected networks

The proposed algorithm works for both directed and undirected networks. In fact the MPS algorithm can be used in undirected networks, if each edge is replaced by two arcs in opposite directions, with equal costs.

The Dijkstra algorithm has to receive a pruned graph where all the arcs in a path p (selected by a k-shortest path algorithm) have been temporarily removed. If the network is directed, only the directed arcs in p are temporarily removed from the network graph. If the network is undirected, for each arc (i, j) in p two directed arcs, (i, j) and (j, i), are temporarily removed in the corresponding network directed graph, before executing the Dijkstra algorithm.

3. Experimental results

In this section two sets of experiments will be presented. In the first set of experiments, using low density networks with up to 800 nodes, an exhaustive approach was made because we feel that if a sub-set of node pairs was randomly selected in each tested network the results could be misleading if the sub-set did not include any of the relatively few node pairs for which only sub-optimal solutions were obtained. In the second set of experiments, with the purpose of observing the behaviour of DP2LC with more dense and larger networks and for comparing with the ILP approach using ILOG CPLEX 10.110, we used 100 node pairs (randomly selected) in each tested

Table 1
Defining symbols to refer to the range of costs

	$\mathscr{E}$	$\bar{\mathscr{E}}$
$\mathscr{Z}$	$([0, 100], [0, 100])$ $([0, 10\,000], [0, 10\,000])$	$([0, 10], [0, 10\,000])$ $([0, 100], [0, 10\,000])$
$\bar{\mathscr{Z}}$	$([1, 100], [1, 100])$ $([1, 10\,000], [1, 10\,000])$	$([1, 10], [1, 10\,000])$ $([1, 100], [1, 10\,000])$

network. This enabled the required results to be obtained in a reasonable amount of time.

It will be shown that the algorithm DP2LC performs quite well and obtains the minimal cost pair of disjoint paths for almost every node pair, and uses significantly less CPU time than CPLEX.

3.1. Low density networks

Extensive and systematic tests were carried out with the algorithm in a significant number of networks with different topologies, consistent with typical ranges of telecommunications transport networks, and using various cost ranges.

Results are presented for directed networks. The number of arcs m is equal to $3n$, $4n$ and $6n$, where n is the number of nodes in the network. For each number of nodes ($n = 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 600, 800$) 10 different networks, with a given number of arcs, were randomly generated.[1] The used network generator creates an Hamiltonian cycle and the remaining arcs are generated, with the additional constraint that arc-connectivity is always greater than one. There are no multiple arcs between a pair of nodes. For each of these networks the costs of the arcs were randomly generated in different ranges.

The symbols introduced in Table 1 will be used to refer to ranges of cost values. The first (second) range in each pair refers to the costs $c^{(1)}(c^{(2)})$.

Identical (different) ranges for the two arc costs will be designated by $\mathscr{E}$ ($\bar{\mathscr{E}}$). Ranges with lower bound equal to 0 (1) will be designated by $\mathscr{Z}$ ($\bar{\mathscr{Z}}$). Finally $\mathscr{Z}\mathscr{E}$, $\mathscr{Z}\bar{\mathscr{E}}$, $\bar{\mathscr{Z}}\mathscr{E}$ and $\bar{\mathscr{Z}}\bar{\mathscr{E}}$, will identify the four groups (each group with two ranges) of ranges in the table.

In this set of experiments DP2LC was tested for all end-to-end node pairs ($n(n-1)$) in every network.

In all tested cases the algorithm obtained an optimal solution in almost all cases. Sub-optimal solutions were obtained in a few, quite rare cases, in which the optimal stopping condition cannot be verified because memory is exhausted in one of the procedures. In these cases the algorithm has the capability of calculating the relative error of a sub-optimal solution, X, r_{e}, defined by

$$r_{\text{e}} = \frac{C(X) - c_{\text{L}}}{c_{\text{L}}} \tag{22}$$

The possibility of calculating this error and the bounds ($c_{\text{U}} = C(X)$, c_{L} (14)) on the optimal cost is another advantage of the proposed approach.

In Fig. 1 the average relative error of the sub-optimal solutions, for networks where sub-optimal solutions occurred, for ranges $\mathscr{Z}\bar{\mathscr{E}}$, is shown. It can be seen that the relative error is quite small.

DP2LC only failed in finding optimal solutions for all node pairs in the ranges $\bar{\mathscr{Z}}$, in the case of range $([1, 100], [1, 10\,000])$ and m=4n, 6n. Even so those were quite rare situations, corresponding to a total number of sub-optimal solutions (total number of node pairs for which a solution was obtained, the optimality of which could not be

[1] The program used for network generation was kindly borrowed from José Luis Santos of the Department of Mathematics of Coimbra University.

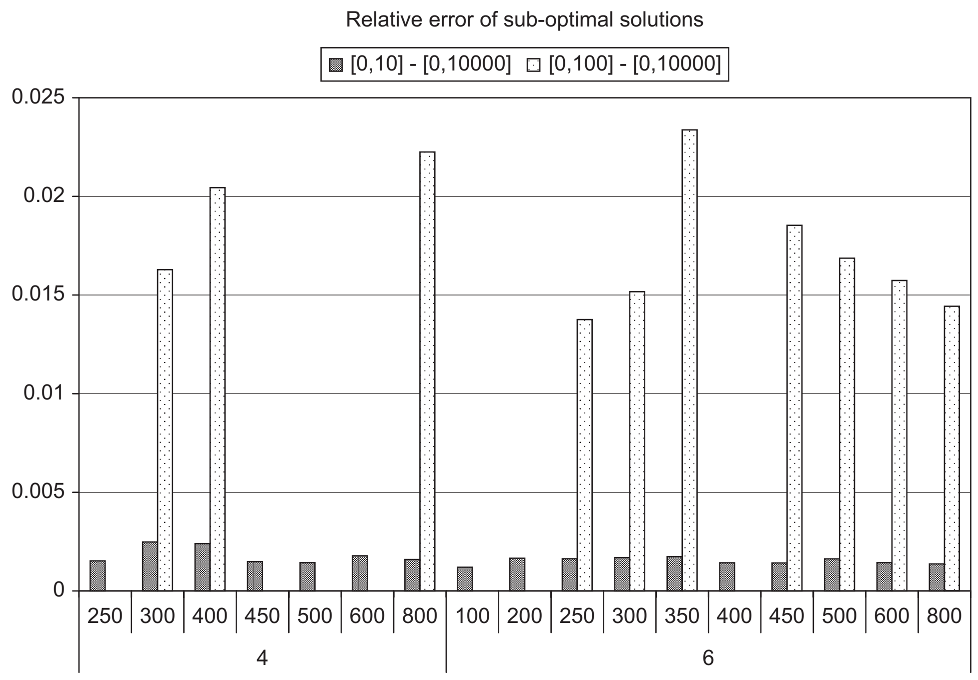

Fig. 1. Average of the relative error of sub-optimal solutions for ranges $\mathscr{L}\bar{\mathscr{E}}$.

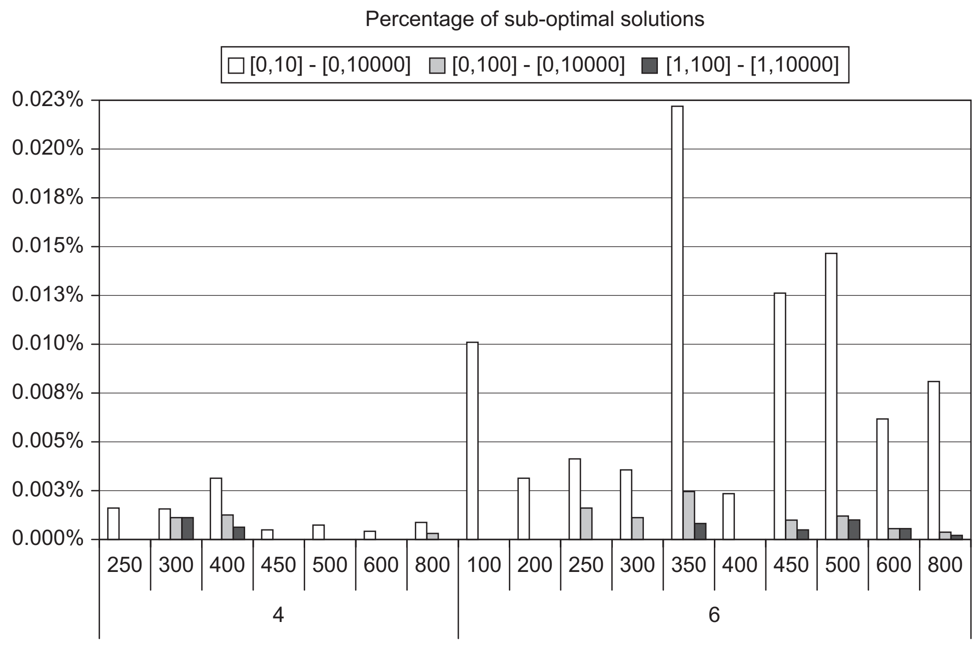

Fig. 2. Percentage of sub-optimal solutions, for networks with sub-optimal solutions for $m = 4n, 6n$ and ranges $\bar{\mathscr{E}}$.

checked) equal to 15, in approximately 235×10^6 node pairs considered in these tests (for all networks with ranges $\bar{\mathscr{L}}$). The frequency of sub-optimal solutions for range ([1, 100], [1, 10 000]) is shown in Fig. 2. Note that in Fig. 2 only non-null values are shown and the average values were calculated considering only networks where sub-optimal solutions were detected. For example, for $n = 300, 400$ and $m = 4n$ each value in Fig. 2 (ranges $\bar{\mathscr{L}}\bar{\mathscr{E}}$) corresponds to a single node pair (in a single network). In the case of ranges $\mathscr{L}\bar{\mathscr{E}}$, for $m = 4n$ and $6n$ the number of sub-optimal solutions is higher (a total of 1695 node pairs in approximately 235×10^6) which represents a very small percentage of the tested node pairs (for all networks with ranges $\mathscr{L}$). Fig. 2 shows the percentage of sub-optimal solutions, for the networks where sub-optimal solutions were detected. The worst average value is 0.022% (the highest value in all tested cases was 0.1555% for $n = 350$ and $m = 6n$). Sub-optimal solutions occur more often for the costs in the range ([0, 10], [0, 10 000]).

We have made an estimate of RAM used for storing the data required by the algorithm, namely the space required to store the paths that can be generated by MPS. This results in an (under-)estimate for the allowed maximum number of paths, which will ensure than only RAM memory is used and that no swapping takes place. CPU times were obtained in a Pentium IV at 3.2 GHz with 2 GB of RAM.

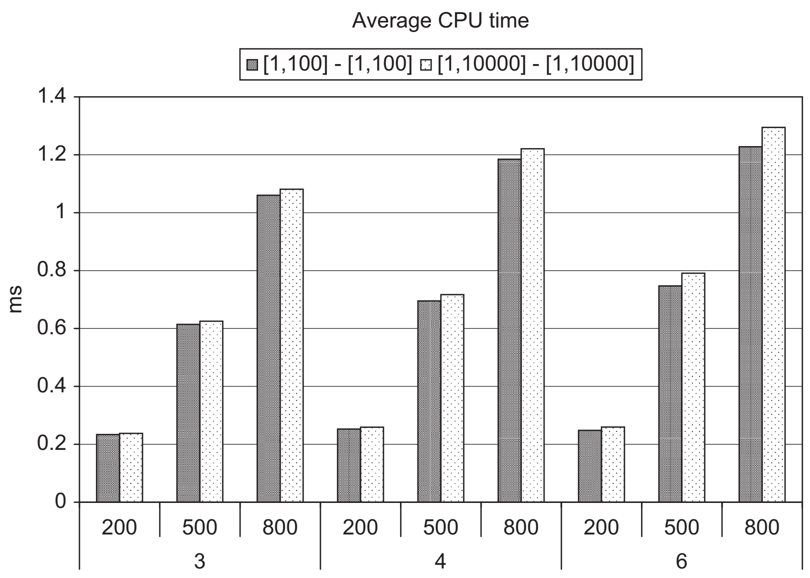

Fig. 3. Average CPU time per node pair (with optimal or sub-optimal solutions) for $n = 200, 500, 800$, $m = 3n, 4n, 6n$ and ranges $\bar{\mathscr{I}}\mathscr{E}$.

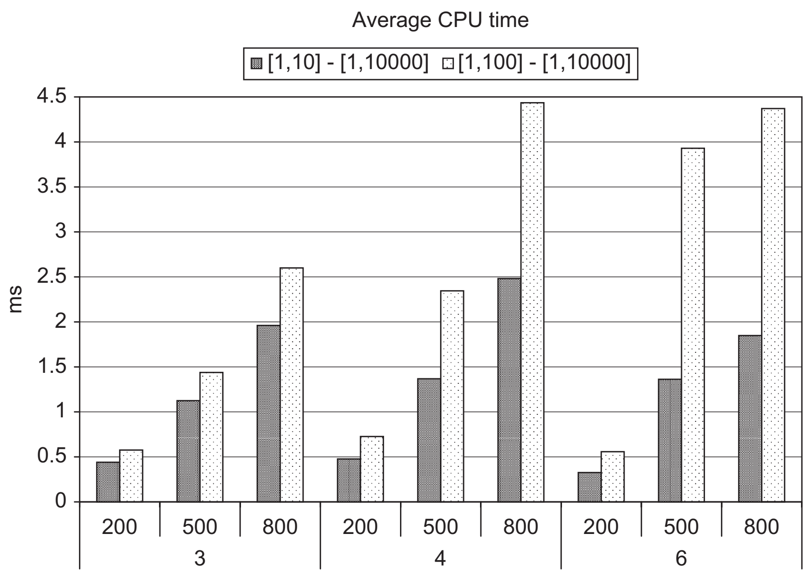

Fig. 4. Average CPU time per node pair (with optimal or sub-optimal solutions) for $n = 200, 500, 800$, $m = 3n, 4n, 6n$ and ranges $\bar{\mathscr{I}}\bar{\mathscr{E}}$.

The average CPU time was obtained per pair of disjoint paths for each node pair in the set of all s–t pairs with fixed t (for every node t). This allows the MPS algorithm to re-use the tree of shortest paths from all nodes to t and the ordered set of the network arcs.

In Figs. 3 and 4 average CPU times per node pair (with optimal or possibly sub-optimal solutions) are presented for $n = 200, 500, 800$ and $m = 3n, 4n, 6n$ for costs ranges $\bar{\mathscr{I}}\mathscr{E}$ and $\bar{\mathscr{I}}\bar{\mathscr{E}}$. Two separate figures are presented because in Fig. 4 the maximal CPU time is less than 4.5 ms and in Fig. 3 is less than 1.4 ms. Here it can be seen that the used CPU time is relatively higher in the case of different cost ranges. A complete set of results for all tested networks can be found in [17]. In Figs. 5 and 6 the average CPU times per node pair (with optimal or sub-optimal solutions) for $n = 200, 500, 800$ and $m = 3n, 4n, 6n$ for ranges $\mathscr{I}\mathscr{E}$ and $\mathscr{I}\bar{\mathscr{E}}$ are presented.

Recalling that DP2LC, for networks with arc costs in ranges $\mathscr{E}$, always obtained optimal solutions for all node pairs, Figs. 3 and 5 present the average CPU time for obtaining an optimal solution, per node pair.

From Figs. 3 and 5 we conclude that the average CPU time for obtaining an optimal solution is similar in ranges $\mathscr{E}$ and approximately independent of the lower bound of the cost ranges. In Fig. 4 a higher CPU time associated with different ranges for the arc costs $\bar{\mathscr{I}}\bar{\mathscr{E}}$ is shown. In the case of $\mathscr{I}\bar{\mathscr{E}}$ this CPU time increase is also visible in Fig. 6. Results for ranges $([0, 100], [0, 10\,000])$ (in Fig. 6) and $([1, 100], [1, 10\,000])$ (in Fig. 4) are similar but a strong increase in used CPU time was detected for networks with arc costs in the intervals $([0, 10], [0, 10\,000])$ and $m = 4n, 6n$.

In Figs. 7 and 8 the average used CPU time, per node pair, for obtaining an optimal solution for all considered networks with arc costs in ranges $\bar{\mathscr{E}}$, is presented (for networks with arc costs in ranges $\mathscr{E}$ all solutions were optimal). The results in Figs. 4 and 7 are practically identical because of the negligible number of sub-optimal solutions in the ranges $\bar{\mathscr{I}}$. However, Figs. 6 and 8 show some differences. In Fig. 8 a visible decrease in CPU time can be seen when compared with 6, for $m = 6n$ and range $([0, 10], [0, 10\,000])$.

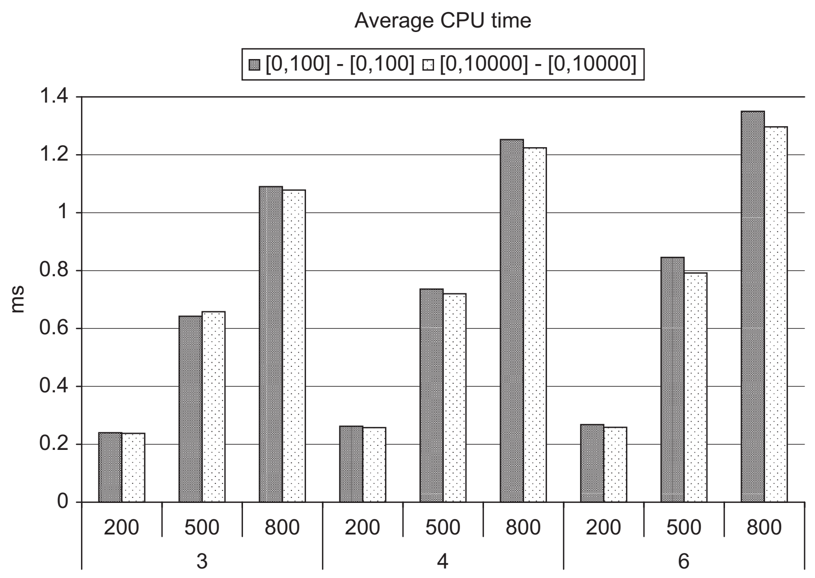

Fig. 5. Average CPU time per node pair (with optimal or sub-optimal solutions) for $n = 200, 500, 800$, $m = 3n, 4n, 6n$ and ranges $\mathscr{X}\mathscr{E}$.

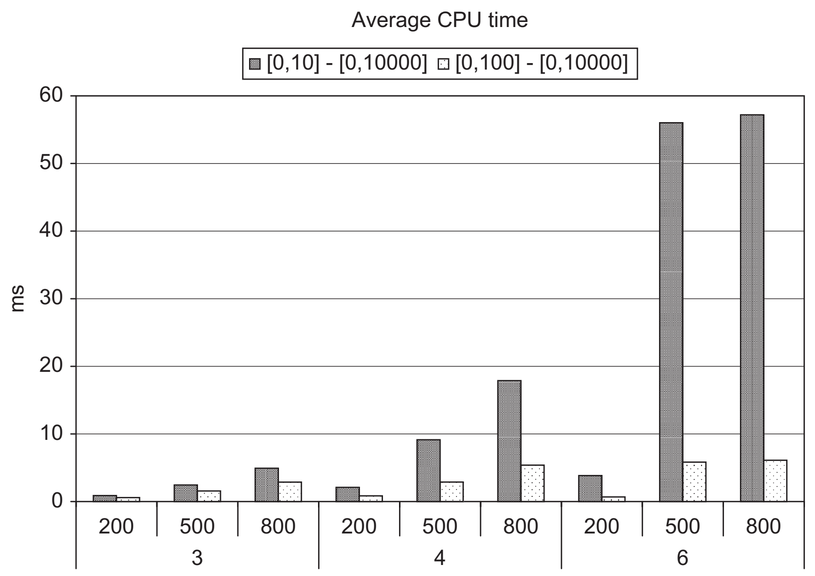

Fig. 6. Average CPU time per node pair (with optimal or sub-optimal solutions) for $n = 200, 500, 800$ and $m = 3n, 4n, 6n$ and ranges $\mathscr{X}\bar{\mathscr{E}}$.

It was verified that when longer CPU time was required to find an optimal solution (usually for ranges $\bar{\mathscr{E}}$, and in particular for range $([0, 10], [0, 10\,000])$) one of the procedures had quickly found a very low cost path pair (procedure $\mathscr{B}$) which required the other procedure (procedure $\mathscr{A}$) to generate a large number of path pairs, in the search for a better path pair. This results from the fact that if $C(B) = C[q(p^{(2)})^{(1)}, p^{(2)}] = c_{\text{opt}}$, c_{opt} will be mostly determined by the cost of $p^{(2)}$, and frequently $C^{(1)}[q(p^{(2)})^{(1)}] \gg p_1^{(1)}$, hence a large number of path pairs will have to be generated by using procedure $\mathscr{A}$ (which includes a call to Dijkstra's algorithm for obtaining $q(p_u^{(1)})^{(v)}$ for each $p_u^{(1)}$). Also, for the same m/n relation, networks with more arcs, being more sparse, lead to longer paths and more processing.

The number of sub-optimal solutions is rather small for $\mathscr{X}\bar{\mathscr{E}}$ and negligible for ranges $\bar{\mathscr{X}}\bar{\mathscr{E}}$ (as shown in Fig. 2). In these rare cases DP2LC takes a very long time to exhaust the allowed memory usage (because no CPU time limit per node pair was implemented). These high CPU times combined with a relative higher frequency (on average $\leqslant 0.0221\%$ as compared with 0.0025%, the maximal average frequency of range $([0, 100], [0, 10\,000])$) of occurrence of sub-optimal solutions in the case of range $([0, 10], [0, 10\,000])$ have some impact on the average CPU time in Fig. 6, for $n = 6m$.

A study was also made regarding the frequency of occurrence of the optimal pair among the first pair identified by procedure $\mathscr{A}$ or by procedure $\mathscr{B}$. The results showed that this frequency was greater than 99.46% for all networks with ranges $\bar{\mathscr{E}}$, regardless of network density or size, which is an indication of the rapid convergence of one of the procedures of the algorithm. This result confirms the previous statement regarding the performance of the algorithm for different cost ranges: procedure $\mathscr{B}$ which enumerates the paths according to $\eta^{(2)}$ (the larger cost range in the test networks) generated very few paths because it obtains $B = B_1$, such that $C(B_1) = c_{\text{opt}}$ quite

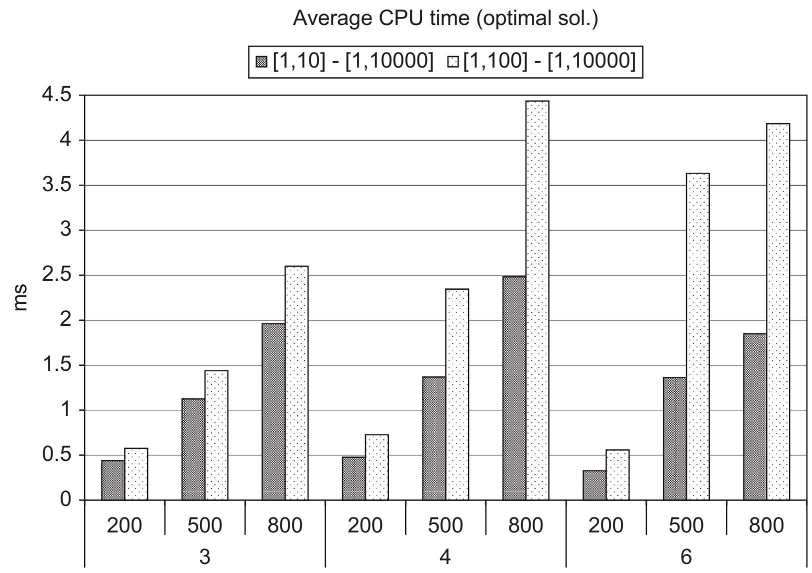

Fig. 7. Average CPU time per node pair with an optimal solution for $n = 200, 500, 800$, $m = 3n, 4n, 6n$ and ranges $\bar{\mathscr{L}}\bar{\mathscr{E}}$.

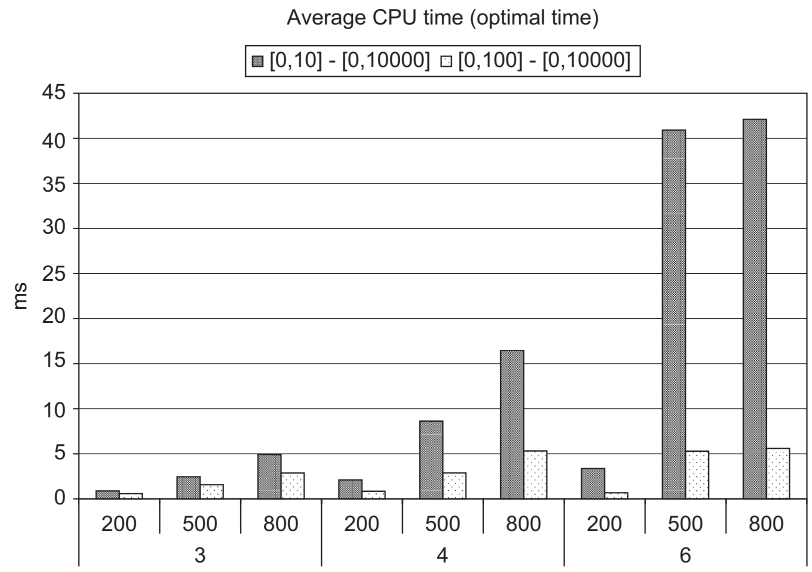

Fig. 8. Average CPU time per node pair with an optimal solution for $n = 200, 500, 800$, $m = 3n, 4n, 6n$, and ranges $\mathscr{L}\bar{\mathscr{E}}$.

frequently. Therefore, for ranges $\bar{\mathscr{E}}$, procedure $\mathscr{A}$ had very often the task of obtaining a path pair, A_h, which confirms the optimal cost is indeed equal to $C(B_1)$ $(C(B = B_1) = C^{(1)}(p_h^{(1)}) + C^{(2)}(p^{(2)}))$, where h may be a very large number, especially for ranges $\mathscr{L}\bar{\mathscr{E}}$.

For all networks with ranges $\mathscr{E}$ the frequency of occurrence of the optimal pair among the first pair identified by procedure $\mathscr{A}$ or by procedure $\mathscr{B}$ was in the interval $[0.931, 0.993]$, increasing with the ratio m/n, regardless of the network size. In Fig. 9 results are shown for all values of n and m but only for ranges $([1, 10], [1, 10\,000])$ and $([0, 100], [0, 100])$.

Also one should note that DP2LC does not enter cycle 5 if the first path identified in both procedures is optimal. An analysis of the frequency of this event was also made, and it was verified that it increased with the ratio m/n, and was practically insensitive to the range of the cost of the arcs and also to the network dimension. The average values were around 30%, 40% and close to 60% for $m = 3n$, $4n$ and $6n$, respectively.

3.2. Denser networks

In the previous set of experiments, networks were generated with a given average node degree by making $m = 3n, 4n, 6n$. Is this second set of experiments 10 different networks were randomly generated with density (d) equal to 10% and 20%, for $n = 50, 100, 200, 400, 800, 1600, 3200$, with the cost ranges given in Table 1. In this case (s, t) node pairs were randomly generated and the MPS algorithm did not re-use the tree of shortest paths from all nodes to t.

DP2LC and ILOG CPLEX 10.110 (see ILP formulation in Section 1.3) were used to solve the same problem for 100 randomly generated node pairs. CPLEX was not able to solve the problem ('Out of memory' error condition) for 20% density networks when $n = 3200$. Results for DP2LC and CPLEX will be presented in different figures because of the different time scales.

We verified that, using this new set of more dense networks, and testing only for 100 node pairs, the CPU time per node pair of

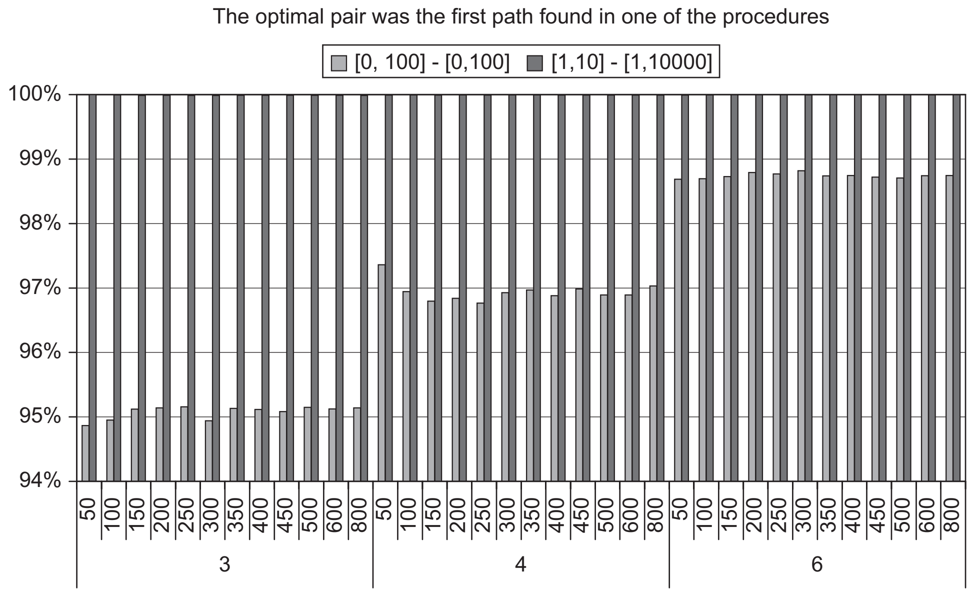

Fig. 9. Frequency of occurrence of the optimal pair among the first identified pair in procedures $\mathscr{A}$ or $\mathscr{B}$, for ranges ([1, 10], [1, 10 000]) and ([0, 100], [0, 100]).

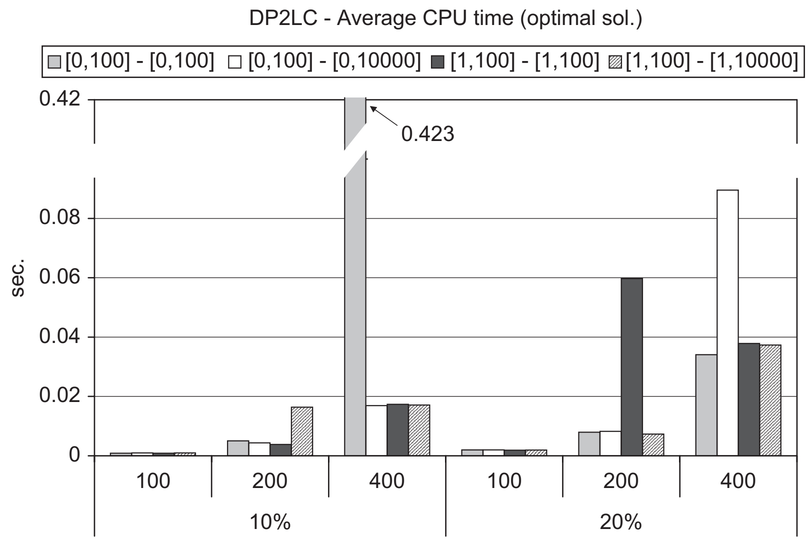

Fig. 10. Average CPU time per node pair (with optimal solutions) for $n = 100, 200, 400$, $d = 10\%, 20\%$ and four cost ranges.

DP2LC no longer presented a clear correlation with the range of the costs. Therefore four ranges will be shown in Figs. 10 and 12 and the remaining four in Figs. 11 and 13.

From Figs. 10 and 11 it can be observed that DP2LC is apparently more sensitive to the number of nodes than to the network density: for the same n the CPU time approximately doubles with network density and for the same density it multiplies by four when n doubles.

Observing the relative performance of CPLEX and of DL2LC for smaller networks it can be seen that DP2LC is usually much faster than CPLEX: up to 25 times, for 100 nodes networks with 20% density. There is an exception for n=400, d=10% and range ([0, 100], [0, 100]), where it is only 1.5 faster, but for the remaining ranges (of $n = 400$, $d = 10\%$) it is at least 6.4 times faster than CPLEX. CPU times for n=50 are not shown because they would not be visible in Figs. 10 or 12—for DP2LC the CPU time was always below 0.5 ms and for CPLEX it was around 10 ms.

In Figs. 12 and 13 the average CPU time used by CPLEX to solve the same problems is presented. The average CPU time per node pair, for obtaining optimal solutions in DP2LC, was less than 1.65 s for n=3200 and 10% density networks. CPLEX required in average more than 35 s for the same networks. When the density increased to 20% and the number of nodes was 3200, the CPU time of DP2LC was just below 3.5 s for all cost ranges, and CPLEX did not manage to solve this problem. Looking at the largest problem CPLEX was capable of solving, in the case of 20% density networks, we can see the average CPU time per node pair was close to 16 s while DP2LC took less than 0.79 s in average for obtaining optimal solutions, for all cost ranges.

From Figs. 11 and 12 it can be verified that DP2LC obtains optimal solutions much more quickly than CPLEX. In the largest problem solved by DP2LC and CPLEX ($n = 3200$, $d = 10\%$), DP2LC was at least 10 times faster than CPLEX, in all cost ranges. The best relative performance of DP2LC, when $n = 800, 1600, 3200$, occurred for $n = 800$, $d = 10\%, 20\%$, where DP2LC was at least 34 faster for all cost ranges.

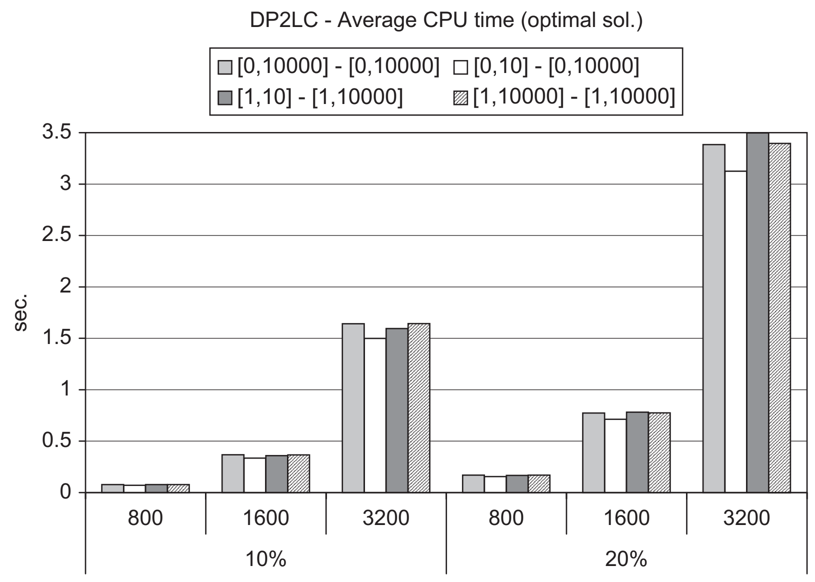

Fig. 11. Average CPU time per node pair (with optimal solutions) for $n = 800, 1600, 3200$, $d = 10\%, 20\%$ and four cost ranges.

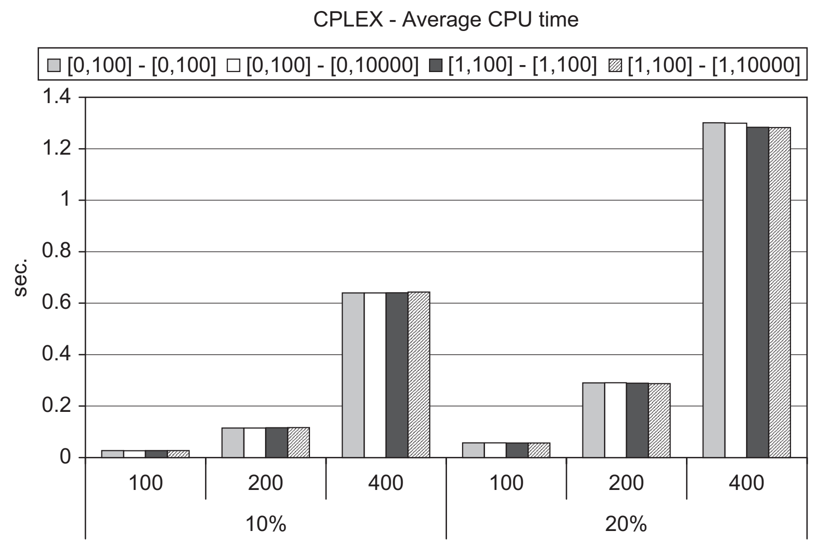

Fig. 12. Average CPU time per node pair for $n = 100, 200, 400$, $d = 10\%, 20\%$ and four cost ranges.

In the tested networks, and for the randomly selected node pairs, some solutions were sub-optimal. The grand total of sup-optimal solutions was 115 in a total of 112 000 node pairs (close to 0.1%). In these cases CPU time can be much larger than the values presented in Figs. 10 and 11. Nevertheless, because DP2LC is very efficient for almost every node pair, and obtains optimal solutions in a short time, even for large problems, it is clearly a preferable approach to CPLEX. In practical situations, and to avoid possible long running times, in the rare occasions in which the optimal stopping condition takes too long to be obtained, a very effective approach would be to enforce a CPU time limit per node pair on DP2LC. Also recall that, in these cases, the quality of every sub-optimal solution can be known, based on the provided upper and lower bounds of the optimal cost.

Allowing DP2LC to run to the limit of available resources, including the use of swapping to increase memory space up to a total amount of 5 GB, we were able to obtain 20 new optimal solutions. These new optimal solutions were found in 11 networks, with $n = 100, 200, 400$ (in the larger networks the number of sub-optimal solutions remained unchanged). The CPU times for these additional optimal solutions were quite high as compared to the values in Fig. 10, ranging from tens to hundreds of seconds, which again reinforces the need for a CPU time limit, per node pair, in practical situations.

To summarise, the following conclusions can be drawn from this extensive experimental study:

- In the case of low density networks a correlation was observed between the range of the costs and DP2LC performance:
    - When arcs costs were in ranges $\mathscr{E}$ the algorithm managed to obtain optimal solutions for all tested node pairs.
    - When arc costs were in ranges $\bar{\mathscr{Z}}$, the algorithm obtained optimal solutions for almost every node pair, in all tested networks.
    - In the case of networks with arc costs in ranges $\mathscr{Z}\bar{\mathscr{E}}$ the number of sub-optimal solutions was a little higher than in $\bar{\mathscr{Z}}\mathscr{E}$ but nevertheless a very small percentage of the tested node pairs.

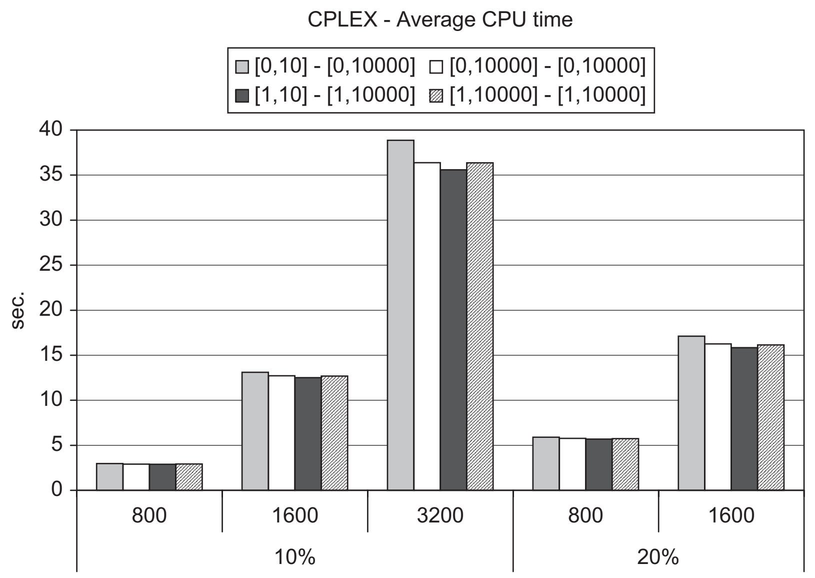

Fig. 13. Average CPU time per node pair for $n = 800, 1600, 3200$, $d = 10\%$, $n = 800, 1600$, $d = 20\%$, and four cost ranges.

- ∘ It was also verified that DP2LC used relatively higher CPU time per node pair for ranges $\bar{\mathscr{E}}$ than for ranges $\mathscr{E}$. This effect was more pronounced when the range of the costs started with zero.

Concerning average CPU time per node pair, for obtaining optimal solutions it can be concluded that:

For networks with ranges $\mathscr{E}$, the average CPU time per node pair was very small (a few milliseconds).

For networks with ranges $\bar{\mathscr{Z}}\bar{\mathscr{E}}$, CPU times were a little higher than for ranges $\mathscr{Z}\mathscr{E}$, but still in the order of some milliseconds.

For networks with ranges $\mathscr{Z}\bar{\mathscr{E}}$, CPU times were some milliseconds for range $([0, 100], [0, 10\,000])$ but several tens of milliseconds for $([0, 10], [0, 10\,000])$ (especially for $m = 6n$).

- In the case of networks with 10% and 20% density the correlation between the range of the costs and DP2LC performance was no longer clear.

The time required by DP2LC to obtain an optimal solution grows more steeply with the number of nodes than with the network density.

DP2LC presents significantly lower average CPU times per node pair (for obtaining optimal solutions), and solves larger problems than CPLEX.

4. Conclusions

An exact and effective algorithm, DP2LC, for finding an arc-disjoint path pair, with minimal cost, in a network with dual arc costs, was proposed. The algorithm is based on the resolution of two k-shortest path problems in an articulate manner, so that an optimal stopping condition can be attained. This optimality condition is based on the calculation of upper and lower bounds on the optimal cost. A proof of the correctness of the algorithm was also presented.

The proposed algorithm (with minor changes) can be used in directed and undirected networks, for obtaining either an arc-disjoint or a node-disjoint minimal cost path pair.

Extensive experimentation with DP2LC was carried out in a significant number of randomly generated directed networks of different topologies, representative of possible telecommunication networks, and using eight ranges for the costs. Experimental results showed that DP2LC solved the problem exactly in practically all the cases in directed networks, the rare exceptions being due to memory exhaustion alone. In the small percentage of cases in which an optimal solution was not attained, the algorithm always returns a sub-optimal solution. Another important advantage of the algorithm is the possibility of calculating upper and lower bounds for the optimal cost as well as the relative error of the sub-optimal solutions, in the very rare cases in which only these solutions are computed.

In low density networks, the average CPU time for obtaining an optimal solution was shown to be a few milliseconds, per node pair, for all considered ranges of the costs, excepting for one cost range, where it required (in some cases) tens of milliseconds (for the larger and more dense networks). The algorithm is faster when the dual arc costs have identical ranges.

In the case of 10% and 20% density networks, DP2LC was capable of obtaining optimal solutions in much shorter CPU times than CPLEX, as expected. Moreover it was capable of solving larger problems than CPLEX.

Finally, concerning applications of the algorithm, it must be noted that in the context of survivable routing in communication networks, many approaches require a minimal cost pair of disjoint paths with dual arc costs. DP2LC, given its efficiency and exactitude, as well as its capability to compute bounds on the optimal cost can provide a very interesting solution in this and similar contexts.

References

[1] Suurballe JW. Disjoint paths in networks. Networks 1974;4:125–45.
[2] Suurballe JW, Tarjan RE. A quick method for finding shortest pairs of disjoint paths. Networks 1984;14(2):325–36.
[3] Xu D, Chen Y, Xiong Y, Qiao C, He X. On finding disjoint paths in single and dual link cost networks. In: IEEE INFOCOM 2004. New York: IEEE; 2004.
[4] Laborczi P, Tapolcai J, Ho P-H, Cinkler T, Recski A, Mouftah HT. Algorithms for asymmetrically weighted pair of disjoint paths in survivable networks. In: Cinkler T, editor. Proceedings of design of reliable communication networks (DRCN 2001). October 7–10 2001. p. 220–7.
[5] Li C-L, McCormick ST, Simchi-Levi D. Finding disjoint paths with different path costs: complexity and algorithms. Networks 1992;22:653–67.
[6] Kodialam M, Lakshman TV. Dynamic routing of restorable bandwidth-guaranteed tunnels using aggregated network resource usage information. IEEE/ACM Transactions on Networking 2003;11(3): 399–410.
[7] Ho P-H, Tapolcai J, Mouftah HT. On achieving optimal survivable routing for shared protection in survivable next-generation internet. IEEE Transactions on Reliability 2004;53(2):216–25.

[8] Mouftah HT, Ho P-H. Optical networks—architecture and survivability. Dordrecht: Kluwer Academic Publishers; 2003.
[9] Bang-Jensen J, Gutin G. Digraphs: theory, algorithms and applications, Springer monographs in mathematics. Berlin: Springer; 2002.
[10] Laborczi P, Tapolcai J, Cinkler T. Efficient algorithms for physically-disjoint routing in survivable networks. In: Networks 2004. June 13–16, 2004. p. 185–91.
[11] Martins E, Pascoal M, Santos J. An algorithm for ranking loopless paths. Technical Report 99/007, CISUC, 1999 ⟨http://www.mat.uc.pt/~marta/Publicacoes/mps2.ps⟩.
[12] Martins E, Pascoal M, Santos J. Deviation algorithms for ranking shortest paths. International Journal of Foundations of Computer Science 1999;10(3):247–63.
[13] Yen JY. Finding the *k* shortest loopless paths in a network. Management Science 1971;17(11):712–6.
[14] Martins E, Pascoal M. A new implementation of Yen's ranking loopless paths algorithm. 4OR—Quarterly Journal of the Belgian, French and Italian Operations Research Societies 2003;1(2):121–34.
[15] Gomes T, Martins L, Craveirinha J. An algorithm for calculating the *k* shortest paths with a maximum number of arcs. Investigação Operacional 2001;21(2):235–44.
[16] Ford Jr LR, Fulkerson DR. Network flows. Princeton, NJ: Princeton University Press; 1962.
[17] Gomes T, Craveirinha J, Jorge L. An effective algorithm for obtaining minimal cost pairs of disjoint paths with dual arc costs. Technical Report 5, INESC—Coimbra, Coimbra, Portugal; 2006. ISSN: 1645-2631.