Instituto de Engenharia de Sistemas e Computadores de Coimbra

Institute of Systems Engineering and Computers

INESC - Coimbra

Teresa Gomes, José Craveirinha e Luísa Jorge

An effective algorithm for obtaining minimal cost pairs of disjoint paths with dual arc costs

No.5 2006

ISSN: 1645-2631

Instituto de Engenharia de Sistemas e Computadores de Coimbra

INESC - Coimbra

Rua Antero de Quental, 199; 3000-033 Coimbra; Portugal

www.inescc.pt

An effective algorithm for obtaining minimal cost pairs of disjoint paths with dual arc costs

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

[a] Dept. of Electrical and Computer Engineering, Pólo II of Coimbra University

3030-290 Coimbra, Portugal

[b] INESC-Coimbra, Rua Antero de Quental 199

3000-033 Coimbra, Portugal

[c] Polytechnic Institute of Bragança

Campus de St$^{\text{a}}$ Apolónia, 5301-857 Bragança, Portugal

Email: {teresa,jcrav}@deec.uc.pt   ljorge@inescc.pt

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.

In the first part of this report an exact algorithm for solving this NP-complete problem is proposed. A formal proof of the correctness of the algorithm is presented. Extensive experimentation is presented to show the effectiveness of this algorithm: most solutions are optimal and are calculated very quickly; only a very small fraction of the solutions is sub-optimal. In real time applications this algorithm can be modified to ensure that a solution (that is either optimal or sub-optimal) is obtained in a bounded CPU time.

In the second part of this report it is shown that the previous algorithm can be extended to collect all the minimal cost pair of disjoint paths. A proof of the correctness of this extension is presented. Experimental results are shown for directed networks.

Keywords OR in telecommunications, paths with minimal cost sum, dual arc costs, disjoint paths

[0]Work partially supported by programme POSI of the III EC programme cosponsored by FEDER and national funds.

Part I

The minimal cost pair of disjoint paths with dual arc costs

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 active path 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 [13] and a more efficient version for $k = 2$ was presented in [14]. In this type of problem the arc costs are uniform, that is they have the same value regardless of the considered path.

A similar problem, called the min-max problem [7] which involves the minimisation of the cost of the more expensive path of the disjoint path pairs for each node pair, was shown (Li *et al.* [7]) to be NP-complete even for $k = 2$, for the four possible variants of the problem: arc or node disjoint paths and directed or undirected networks.

Xu *et al.* [15] tackle a related problem, called min-min, which seeks to minimise the cost of the shorter path (of the disjoint path pair). These authors prove that the min-min problem is NP-complete, whether the network is directed or undirected, with uniform arc costs. Therefore this implies the problem to be NP-complete in dual arc cost networks.

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 [15] and [6], respectively.

In [15] an effective heuristic (designated as COLE – COnflicting Link Exclusion) for solving the min-min problem, is proposed. This approach is based on the use of a shortest path algorithm and on the identification of the so-called *conflicting link set*, with the links which should be avoided in the calculation of the AP, in order to guarantee that a disjoint BP can be found. This approach was also applied to a MSOD problem.

Several heuristics for obtaining an asymmetrically weighted pair of disjoint paths with minimal cost, a particular type of the MSOD problem, are described in [6]. The proposed heuristics are based on k-shortest path searching, Suurballe's algorithm [14], integer linear programming, linear relaxation and minimum cost network flow (MCNF) techniques. According to the computational results presented for networks up to 400 nodes, the MCNF heuristic yields the best performance in terms of cost and running time. Note that the problem addressed in the present paper considers arbitrary dual arc costs, unlike MSOD where an order relation has to be fulfilled by the dual costs.

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 [8]. 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.

In [5] 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 [5] it was considered that the worst case guarantee in [8] 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.* [4] 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 active path (this a problem similar to the one addressed in [5]). In [4] this problem is formulated as an Integer Linear Programming 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 [6] where it was designated as an Enhancement to the Two-Step-Approach (TSA). The TSA uses a shortest path algorithm for obtaining the active path, then removes the arcs of the active path from the network; finally the shortest path algorithm is used again for obtaining the backup path. ITSA iteratively inspects k-shortest paths as active paths, 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 [12], the ITSA can work extremely well in solving the diverse routing problem with shared protection.

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, provided the motivation for this report.

1.2 Contribution of the report

In the first part of this report we propose an exact algorithm for finding an arc disjoint path pair, with minimal cost in a network with dual arc costs. The algorithm is based on the resolution of two k-shortest path problems in an articulate manner, so that an optimal stopping condition is formulated.

Experimental results with random generated networks 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. If execution time is important, as in automatic routing calculation procedures with stringent resolution times, a CPU time limit can be implemented per node pair, leading to a *truncated* version of the algorithm.

An exact algorithm for obtaining the minimal cost pair of disjoint paths with dual arc costs and an extension of that algorithm for obtaining minimal cost pairs of disjoint paths with dual arc costs are presented, respectively in the first and second part of this report.

The first part is organised in the following manner. In the next section the problem is formalised, the notation is introduced and the algorithm is presented; the control conditions of the algorithm are also discussed. Section 2 also presents the proof of the correctness of the algorithm, as well as its application in the context of undirected networks and its variant for the problem with length constrained paths. Extensive experimental results with randomly generated directed networks having different cost ranges are presented and analysed in section 3. Finally, some conclusions are drawn in section 4 which concludes part 1 one the paper.

The second part consists in sections 5 and 6. In section 5 the algorithm for obtaining all the minimal cost pairs of disjoint paths with dual arc costs is introduced, followed by the proof of its correctness. Finally experimental results with randomly generated directed networks having different cost ranges are presented and analysed in section 6.

2 Proposed Approach

2.1 Problem formalisation

Let $G = (V, E)$ be a directed network with node set V and arc set E, where two different non-negative cost functions (or metrics) in the arcs are defined:

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

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

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

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

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 $V_s(p)$ and the set of arcs in p be $E_s(p)$. 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 $E_s(p) \cap E_s(q) = \emptyset$. Two paths p and q are disjoint if $V_s(p) \cap V_s(q) = \emptyset$, and are internally disjoint if $\{v_2, \ldots, v_{i-1}\} \cap V_s(q) = \emptyset$ [1]. 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 (node) disjoint paths which minimises the total (combined) cost of the pair defined by:

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

We recall that in [8] this problem was proved to be NP-complete.

An algorithm which solves this problem by using a k-shortest path enumeration algorithm (such as MPS [11] in its loopless version [10]), 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 [16, 9], we prefer to use MPS because, in [10] , experimental results show that this algorithm is more efficient than Yen's.

2.2 Notation

Let $p_h^{(1)}$ be the h-shortest (loopless) path from s to t with respect to metric $\eta^{(1)}$ (obtained by MPS, for example). Let $q(p_h^{(1)})^{(2)}$ 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 $p_h^{(1)}$.

Let $p_k^{(2)}$ be the k-shortest (loopless) path for s to t with respect to $\eta^{(2)}$. Let $q(p_k^{(2)})^{(1)}$ 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 $p_k^{(2)}$.

Let $A_h = (p_h^{(1)}, q(p_h^{(1)})^{(2)})$, $B_k = (q(p_k^{(2)})^{(1)}, p_k^{(2)})$, $C(A_h) = C^{(1)}(p_h^{(1)}) + C^{(2)}(q(p_h^{(1)})^{(2)})$, and $C(B_k) = C^{(1)}(q(p_k^{(2)})^{(1)}) + C^{(2)}(p_k^{(2)})$.

2.3 Main steps of the algorithm

A k-shortest path enumeration algorithm, using arc costs $c_{ij}^{(1)}$, obtains the paths $p_h^{(1)}$, $h = 1, 2, \ldots$. The arcs of each path $p_h^{(1)}$ are temporarily removed from the network graph and the shortest path with respect to metric $\eta^{(2)}$, $q(p_h^{(1)})^{(2)}$ (disjoint with $p_h^{(1)}$) is obtained in the resulting graph (using for example the Dijkstra algorithm) by using arc costs $c_{ij}^{(2)}$ (with the cost of the removed arcs equal to ∞). In this manner, (*procedure* $\mathcal{A}$) 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 $c_{ij}^{(2)}$ be used for obtaining the k shortest paths $p_k^{(2)}$, $k = 1, 2, \ldots$. For each $p_k^{(2)}$ all its arcs are temporarily removed from the network graph and the shortest path with respect to metric $\eta^{(1)}$, disjoint with $p_k^{(2)}$, is obtained. In this manner (*procedure* $\mathcal{B}$) 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 $\mathcal{B}$ can be considered as 'symmetrical' to procedure $\mathcal{A}$.

We will show that if these two procedures are executed in an articulate manner, when the current best pairs of paths have identical cost (in procedures $\mathcal{A}$ and $\mathcal{B}$), the optimal arc-disjoint path pair has been found (for networks with integral costs) if $C^{(1)}(p_h^{(1)}) = C^{(1)}(q(p_k^{(2)})^{(1)})$ and $C^{(2)}(p_k^{(2)}) = C^{(2)}(q(p_h^{(1)})^{(2)})$.

The main steps of the proposed algorithm, designated DP2LC, are:

```
1. Find the first path pair $A$ using procedure $\mathcal{A}$.
2. Find the first path pair $B$ using procedure $\mathcal{B}$.
3. If $A$ and $B$ satisfy the optimal path pair condition Then Stop Else
   (a) Repeat
         i. While $A$ has to be improved Do
            search for a new pair $A$ with cost lower than or equal to $B$, using procedure
            $\mathcal{A}$ EndWhileDo
        ii. While $B$ has to be improved Do
            search for a new pair $B$ with cost lower than or equal to $A$, using procedure
            $\mathcal{B}$ EndWhileDo
       Until $A$ and $B$ satisfy the optimal path pair condition.
   EndIf
```

The meaning of "$A(B)$ has to be improved" will be made clear in the next sub-section.

2.4 Detailed description of the algorithm

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 $\mathcal{A}$ and $\mathcal{B}$,respectively. The j-th element $(j = 1, 2)$ of A or B is the path which was obtained by using metric $\eta^{(j)}$. Whenever a path $p^{(1)}$ $(p^{(2)})$ does not have a disjoint path $q(p^{(1)})^{(2)})$ $(q(p^{(2)})^{(1)})$ the cost of A (B) is ∞.

The proof of the correctness of the proposed algorithm requires a more detailed description of the algorithm, which will now be presented:

```
1. 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. 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. Identify (and generate) an optimal pair of disjoint paths according to the rule:
If $\left[C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p_v^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p_v^{(2)}) = C^{(2)}(q(p_u^{(1)})^{(2)})\right]$
Then one pair ($p_u^{(1)} = q(p_v^{(2)})^{(1)}$, $p_v^{(2)} = q(p_u^{(1)})^{(2)}$) or two different pairs of identical cost are found and they are both optimal.
Else find an optimal pair:

```
(a) $A \leftarrow A_u$
(b) $B \leftarrow B_v$
(c) Repeat
    i. Procedure $\mathcal{A}$:
       While $[C(B) < C(A)] \vee \left[C^{(1)}(p^{(1)}) < C^{(1)}(q(p^{(2)})^{(1)}) \wedge C(A) = C(B)\right]$ Do
       A. $u \leftarrow u + 1$
       B. MPS (loopless) generates $p_u^{(1)}$, and Dijkstra generates $q(p_u^{(1)})^{(2)}$
       C. If $C(A_u) < C(A)$ Then $A \leftarrow A_u$
          Else If $C(A_u) = C(A) \wedge C(A) = C(B)$ Then $A \leftarrow A_u$ EndIf EndIf
       EndWhileDo
    ii. Procedure $\mathcal{B}$:
       While $[C(A) < C(B))] \vee \left[C^{(2)}(p^{(2)}) < C^{(2)}(q(p^{(1)})^{(2)}) \wedge C(A) = C(B)\right]$ Do
       A. $v \leftarrow v + 1$
       B. MPS (loopless) generates $p_v^{(2)}$, and Dijkstra generates $q(p_v^{(2)})^{(1)}$
```

```
            C. If $C(B_v) < C(B)$ Then $B \leftarrow B_v$
               Else If $C(B_v) = C(B) \wedge C(A) = C(B)$ Then $B \leftarrow B_v$ EndIf EndIf
          EndWhileDo
      Until $\left[C^{(1)}(p^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p^{(2)}) = C^{(2)}(q(p^{(1)})^{(2)})\right]$
  EndIf
```

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 3c) seeks the improvement of A and B, until they become two pairs with identical cost (that is with equal total cost and with equal cost for each cost function).

The cycle in step 3(c)i is executed when $C(B) < C(A)$, so that the solution of procedure $\mathcal{A}$ catches up (or overcomes) the best current solution obtained by procedure $\mathcal{B}$. The cycle in step 3(c)i is also executed when $C(B) = C(A)$ and $C^{(1)}(p^{(1)}) < C^{(1)}(q(p^{(2)})^{(1)})$, so that procedure $\mathcal{A}$ may check whether or not there exists a path pair $A_u = (p_u^{(1)}, q(p_u(1)^{(2)}))$, with $C^{(1)}(p^{(1)}) < C^{(1)}(p_u^{(1)}) < C^{(1)}(q(p^{(2)})^{(1)})$, of cost lower than $C(A)$ or until $C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})$ and $C(A_u) = C(A)$. Step 3(c)iC guarantees that the best current pair A is always updated when a lower cost pair is found in procedure $\mathcal{A}$ or when $C(A) = C(B)$ and $C(A_u) = C(A)$. When cycle 3(c)i ends, A (the best current path pair of this procedure) is either such that $C(A) < C(B)$, and A has overcome B (the current best solution of procedure $\mathcal{B}$), or $C(A_u) = C(A) = C(B)$ and therefore A and B are both optimal path pairs because $C^{(1)}(p^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})$.

A similar argument can be made regarding step 3(c)ii. The cycle in step 3(c)ii is executed when $C(A) < C(B)$, so that procedure $\mathcal{B}$ improves its solution until it catches up (or overcomes) the best current solution of procedure $\mathcal{A}$. The cycle in step 3(c)ii is also executed when $C(B) = C(A)$ and $C^{(2)}(p^{(2)}) < C^{(2)}(q(p^{(1)})^{(2)})$, so that procedure $\mathcal{B}$ may check whether or not there exists a path pair $B_v = (q(p_v^{(2)})^{(1)}, p_v^{(2)})$, with $C^{(2)}(p^{(2)}) < C^{(2)}(p_v^{(2)}) < C^{(2)}(q(p^{(1)})^{(2)})$, with cost lower than $C(B)$, or until $C^{(2)}(p_v^{(2)}) = C^{(1)}(q(p^{(2)})^{(1)})$ and $C(B_v) = C(B)$. Step 3(c)iiC guarantees that the best current pair B is always updated when a lower cost pair is found in procedure $\mathcal{B}$ or when $C(A) = C(B)$ and $C(B_v) = C(B)$. When cycle 3(c)ii ends, B (the best current path pair of this procedure) is such that either $C(B) < C(A)$, and B has overcome A (the current best solution of procedure $\mathcal{A}$), or $C(B_v) = C(B) = C(A)$ and therefore A and B are both optimal path pairs because $C^{(2)}(p_v^{(2)}) = C^{(1)}(q(p^{(2)})^{(1)})$.

The output of the algorithm is composed of two pairs (A and B) which either are equal or have equal cost, and in both cases they are the (or an) optimal pair.

The previous approach can also be used for obtaining the minimum 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 [3]) for enumerating the k-shortest paths with length constraints was used instead of a k-shortest path enumeration algorithm (MPS loopless) and if the Bellman-Ford (see [2]) 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 Some comments on the control conditions of the algorithm

The optimal stopping condition of the algorithm (in step 3c) is

$$\left[C^{(1)}(p^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p^{(2)}) = C^{(2)}(q(p^{(1)})^{(2)})\right] \tag{5}$$

instead of $C(A) = C(B)$ and in steps 3(c)i, 3(c)ii is not simply $C(B) < C(A)$ and $C(A) < C(B)$, respectively, because when:

$$\begin{aligned} C^{(1)}(p_h^{(1)}) + C^{(2)}(q(p_h^{(1)})^{(2)}) &= C^{(2)}(p_k^{(2)}) + C^{(1)}(q(p_k^{(2)})^{(1)}) && (6)\\ C^{(1)}(p_h^{(1)}) &\neq C^{(1)}(q(p_k^{(2)})^{(1)}) && (7)\\ C^{(2)}(p_k^{(2)}) &\neq C^{(2)}(q(p_h^{(1)})^{(2)}) && (8) \end{aligned}$$

the algorithm could stop with a non optimal solution.

Consider the example: $C(A_u) = C(B_v) = 10$ but $C^{(1)}(p_u^{(1)}) = 2$, $C^{(2)}(q(p_u^{(1)})^{(2)}) = 8$, $C^{(2)}(p_v^{(2)}) = C^{(1)}(q(p_v^{(2)})^{(1)}) = 5$, when $A = A_u$ and $B = B_v$. A path pair A_w, $w > u$ such that $C^{(1)}(p_w^{(1)}) = 3$ and $C^{(2)}(q(p_w^{(1)})^{(2)}) = 6$, so that $C(A_w) = 9$ is less than $C(A_u)$ might exist and would not be found by the algorithm. If the stopping condition was simply $C(A) = C(B)$ the algorithm would terminate because $C(A_u) = C(B_v) = 10$ and the better solution with cost 9 would not be sought (a similar analysis could be made for the paths obtained in procedure $\mathcal{B}$).

It would not be sufficient to have the conjunction (5) as a stopping rule (the correct optimal stopping rule) and in steps 3(c)i, 3(c)ii to use the conditions $C(B) < C(A)$ and

$C(A) < C(B)$, respectively, because when $C(A) = C(B)$ but (7) and (8) are true, none of the procedures $\mathcal{A}$ or $\mathcal{B}$ would seek to improve its solution (because $C(A) = C(B)$), and the algorithm would be in an endless loop (because the optimal stopping condition in (5) would never be verified). For the same reason, when $C(A) = C(B)$ but the optimal stopping condition is not verified (but the minimal cost pair has been reached, although we have no proof of that at this step) the current best pair must be updated in step 3(c)iC (3(c)iiC)) when $C(A_u) = C(A)$ ($C(B_v) = C(B)$), changing A (B) so that a pair such that $C^{(1)}(p^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)}) \wedge C(A) = C(B)$ ($C^{(2)}(p^{(2)}) = C^{(2)}(q(p^{(1)})^{(2)}) \wedge C(A) = C(B)$), is obtained.

Finally, the reason why in steps 3(c)i and 3(c)ii the stopping condition to improve solutions in procedures $\mathcal{A}$ and $\mathcal{B}$ was written as:

$$[C(B) < C(A)] \quad \vee \quad \left[C^{(1)}(p^{(1)}) < C^{(1)}(q(p^{(2)})^{(1)}) \wedge C(A) = C(B)\right] \tag{9}$$

$$[C(A) < C(B)] \quad \vee \quad \left[C^{(2)}(p^{(2)}) < C^{(2)}(q(p^{(1)})^{(2)}) \wedge C(A) = C(B)\right] \tag{10}$$

will be justified. Consider that $C(A) = C(B)$ and that $p^{(1)} = p_h^{(1)}$ and $p^{(2)} = p_k^{(2)}$, but $C^{(1)}(p_h^{(1)}) \neq C^{(1)}(q(p_k^{(2)})^{(1)})$. If $C^{(1)}(p_h^{(1)}) > C^{(1)}(q(p_k^{(2)})^{(1)})$ (and $C^{(2)}(p_k^{(2)}) > C^{(2)}(q(p_h^{(1)})^{(2)})$) then $q(p_k^{(2)})^{(1)}$ must coincide with some $p_w^{(1)}$, $1 \leq w < h$, and therefore the pair A_w would have already been generated but it was not the optimal pair otherwise it would have been the stored path – a similar reasoning can be used for $q(p_h^{(1)})^{(2)}$. So in fact equations (6)-(8) can be written simply as:

$$C^{(1)}(p_h^{(1)}) + C^{(2)}(q(p_h^{(1)})^{(2)}) \;=\; C^{(2)}(p_k^{(2)}) + C^{(1)}(q(p_k^{(2)})^{(1)}) \tag{11}$$

$$C^{(1)}(p_h^{(1)}) \;<\; C^{(1)}(q(p_k^{(2)})^{(1)}) \tag{12}$$

$$C^{(2)}(p_k^{(2)}) \;<\; C^{(2)}(q(p_h^{(1)})^{(2)}) \tag{13}$$

This means that when the paths stored in A and B have equal costs but $C^{(1)}(p_h^{(1)}) < C^{(1)}(q(p_k^{(2)})^{(1)})$ (and $C^{(2)}(p_k^{(2)}) < C^{(2)}(q(p_h^{(1)})^{(2)})$) the stored path may not be the optimal one (as shown in the previous example), and path generation must be continued until either $C(A)$ (in step 3(c)i) or $C(B)$ (in step 3(c)ii) is improved or paths of equal cost are obtained in different phases of both procedures, so that the optimal stopping condition of the algorithm (see equation (5)) is satisfied.

2.6 Proof of the correctness of the algorithm

The proof takes into account the control and stopping conditions of the algorithm (explained in the previous section) and shows that if the pair(s) of paths found by the algorithm was (were) not optimal then an absurd situation would arise. The proof of the algorithm also assumes that $A = (p^{(1)}, q(p^{(1)})^{(2)})$ ($B = (q(p^{(2)})^{(1)}, p^{(2)})$), obtained in step 3(c)i (3(c)ii), corresponding to procedure $\mathcal{A}$ ($\mathcal{B}$), when u (v) paths have been generated by the k-shortest path enumeration algorithm, using metric $\eta^{(1)}$ ($\eta^{(2)}$), is such that $C(A) = \min_i C((p_i^{(1)}, q^{(2)}(p_i^{(1)}))$, $i = 1, \cdots, u$ ($C(B) = \min_i C((q(p^{(2)})^{(1)}, p^{(2)}))$, $i = 1, \cdots, v$).

Note that the first pair of disjoint paths was obtained, for procedures $\mathcal{A}$ and $\mathcal{B}$, in steps 1 and 2. In step 3, it is stated that if "$\left[C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p_v^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p_v^{(2)}) = C^{(2)}(q(p_u^{(1)})^{(2)})\right]$" then either $p_u^{(1)} = q(p_v^{(2)})^{(1)}$ and $p_v^{(2)} = q(p_u^{(1)})^{(2)}$ or two different pairs of identical cost have been found and they are both optimal.

Lemma 2.1 (Minimal cost of the disjoint path before step 3) *If, when entering step 3, the condition $\left[C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p_v^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p_v^{(2)}) = C^{(2)}(q(p_u^{(1)})^{(2)})\right]$ is true then pairs of disjoint paths in G have been found, A_u and B_v, and c_{opt}, the minimal cost of the pairs of disjoint paths in G, has been found: $c_{opt} = C(A_u) = C(B_v)$.*

Proof: By construction, paths $p_u^{(1)}$ ($p_v^{(2)}$) are obtained by non decreasing order of their cost, with respect to metric $\eta^{(1)}$ ($\eta^{(2)}$). Immediately after step 1, $p_u^{(1)}$ is the path of minimal cost with respect to metric $\eta^{(1)}$ which has a disjoint path in G. Using the Dijkstra algorithm the path $q(p^{(1)})^{(2)}$, of minimal cost with respect to metric $\eta^{(2)}$, was found in $G^{(1)}$. Perfectly analogous conclusions apply to paths $p_v^{(2)}$, $q(p_v^{(2)})^{(1)}$ obtained immediately after step 2. If $C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p_v^{(2)})^{(1)})$, then the first element of B_v is (by step 1) a path with cost equal to the cost of the first shortest path with respect to metric $\eta^{(1)}$ which had a disjoint path in G; the second element of B_v, $p_v^{(2)}$, is (by step 2) a path of minimal cost with respect to metric $\eta^{(2)}$, which had a disjoint path in G. Therefore B_v is formed by two disjoint paths $p_v^{(2)}$ and $q(p_v^{(2)})^{(1)}$, each with minimal possible cost with respect to metrics $\eta^{(j)}$, $j = 1, 2$, respectively. Therefore $C(B_v)$ is the minimal cost of any disjoint path pair with dual arc cost, c_{opt}. A similar argument can be made regarding path pair A_u. $\square$

We present a *reductio ad absurdum* proof of the correctness of the algorithm from step 3 onwards.

Proposition 2.1 (Correctness of the algorithm) *At the end of step 3 the minimum of the cost of all the disjoint path pairs in G, c_{opt}, was found, and an optimal path pair was identified.*

Proof: If the first two pairs, found in step 1 and 2, were such that when entering step 3, the condition $\left[C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p_v^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p_v^{(2)}) = C^{(2)}(q(p_u^{(1)})^{(2)})\right]$ is true, then by lemma 2.1 the minimal cost of any disjoint path pair with dual arc cost, $c_{opt} = C(A_u) = C(B_v)$ has been found; $A = A_u$ and $B = B_v$ are both optimal pairs of disjoint paths.

Otherwise the main cycle 3c is executed and when step 3 ends because the condition expressed by equation (5) is true, then the minimal cost of the disjoint path pairs, $c_{opt} = C(A) = C(B)$, in G, was found.

Let us assume that: $A = A_h = (p_h^{(1)}, q(p_h^{(1)})^{(2)})$, $B = B_k = (p_k^{(2)}, q(p_k^{(2)})^{(1)})$, which means that $p_h^{(1)}$ is the h-th shortest path in G with respect to metric $\eta^{(1)}$, $p_k^{(2)}$ is the k-th shortest path in G with respect to metric $\eta^{(2)}$ and that:

$$\left[C^{(1)}(p_h^{(1)}) = C^{(1)}(q(p_k^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p_k^{(2)}) = C^{(2)}(q(p_h^{(1)})^{(2)})\right] \tag{14}$$

This means that the minimum cost arc-disjoint path pair found by procedures $\mathcal{A}$ and $\mathcal{B}$ is A_h and B_k, respectively, and that the algorithm stopping rule in (14) is satisfied.

If the pairs of disjoint paths (of equal cost) found by the algorithm were not optimal, this would meant that a pair of disjoint paths of lower cost exists.

Firstly consider that the optimal disjoint path pairs, obtained by procedures $\mathcal{A}$ and $\mathcal{B}$, are identical:

$$p_h^{(1)} = q(p_k^{(2)})^{(1)} \tag{15}$$

$$p_k^{(2)} = q(p_h^{(1)})^{(2)} \tag{16}$$

For a better path to exist, there must exist a $p_i^{(1)}$, $i > h$ $(C^{(1)}(p_i^{(1)}) \geq C^{(1)}(p_h^{(1)}))$, such that:

$$C^{(1)}(p_i^{(1)}) + C^{(2)}(q(p_i^{(1)})^{(2)}) < C^{(1)}(p_h^{(1)}) + C^{(2)}(q(p_h^{(1)})^{(2)}) \tag{17}$$

Since $C^{(1)}(p_i^{(1)}) \geq C^{(1)}(p_h^{(1)})$ this implies that the only way the previous inequality can be true is if $C^{(2)}(q(p_i^{(1)})^{(2)}) < C^{(2)}(q(p_h^{(1)})^{(2)})$. But, by equation (16), $p_k^{(2)} = q(p_h^{(1)})^{(2)}$ and therefore $C^{(2)}(q(p_i^{(1)})^{(2)}) < C^{(2)}(p_k^{(2)})$ and $q(p_i^{(1)})^{(2)}$ would have to coincide with some $p_j^{(2)}$, $1 \leq j < k$.

But all paths $p_j^{(2)}$, $j = 1, 2, \ldots, k-1$, with cost lower than $p_k^{(2)}$, have already been generated and the corresponding minimal cost arc disjoint path has already been found (in procedure $\mathcal{B}$); the pair with cost $C(B_j)$, $j = 1, 2, \ldots, k-1$ was calculated and its cost was not the stored one, which means that its cost was higher than $C(B_k)$. Therefore no disjoint path pair A_i, with $C^{(1)}(p_i^{(1)}) + C^{(2)}(q(p_i)^{(2)}) < C(A_h)$, $i > h$, can exist.

Secondly, if

$$C^{(1)}(p_h^{(1)}) = C^{(1)}(q(p_k^{(2)})^{(1)}) \tag{18}$$

$$C^{(2)}(p_k^{(2)}) = C^{(2)}(q(p_h^{(1)})^{(2)}) \tag{19}$$

but the two disjoint pairs of paths are not equal,

$$A_h \neq B_k \tag{20}$$

then both disjoint pairs are still optimal (and more than one solution exists). To prove this let us assume that a better disjoint path pair can be found: $p_i^{(1)}$, $i > h$ ($C^{(1)}(p_i^{(1)}) \geq C^{(1)}(p_h^{(1)})$):

$$C^{(1)}(p_i^{(1)}) + C^{(2)}(q(p_i^{(1)})^{(2)}) < C^{(1)}(p_h^{(1)}) + C^{(2)}(q(p_h^{(1)})^{(2)}) \tag{21}$$

or

$$C(A_i) < C(A_h) \tag{22}$$

This inequality can only be true if $C^{(2)}(q(p_i^{(1)})^{(2)})) < C^{(2)}(q(p_h^{(1)})^{(2)})$, which implies, by equation (19), that $C^{(2)}(q(p_i^{(1)})^{(2)})) < C^{(2)}(p_k^{(2)})$. Therefore $q(p_i^{(1)})^{(2)}$ has to coincide with some $p_j^{(2)}$, with $j = 1, 2, \ldots, k-1$. So the disjoint path pair $B_j = (q(p_j^{(2)})^{(1)}), p_j^{(2)})$, must have been generated (in procedure $\mathcal{B}$) before B_k and its cost $C(B_j)$ could not be lower than $C(B_k)$ because otherwise it would have been the stored value B instead of B_k ($B = B_k$). Therefore no disjoint path pair A_i, with $C^{(1)}(p_i^{(1)}) + C^{(2)}(q(p_i)^{(2)}) < C(A_h)$, $i > h$, can exist.

Therefore at the end of step 3, $c_{opt} = C(A) = C(B)$ and A and B are (the) disjoint pair(s) of minimal cost. □

2.6.1 Improving the efficiency of the algorithm

The purpose of the cycle in step 3(c)i is to improve solution A. Consider that $C(B) < C(A)$. Every time a path pair A_u is found which improves $C(A)$, A is updated, even though

$C(A_u) > C(B)$, and the cycle in step 3(c)i must continue searching for a better A. If step 3(c)iC was rewritten as: "If $C(A_u) \leq C(B)$ Then $A \leftarrow A_u$ EndIf", A would be updated when A_u became better than B, and would also be updated when it became as good as B (or already was as good as B because $C(A) = C(B)$). Therefore by using this form for step 3(c)iC fewer updates of A will be required in some cases, and the final outcome of the cycle 3(c)iC is the same.

A similar argument could be made regarding step 3(c)iiC, which should be rewritten as: "If $C(B_v) \leq C(A)$ Then $B \leftarrow B_v$ EndIf",

The proof of the correctness of this version of the algorithm is similar, but it can no longer be said that A (B) contains at any time the best path in procedure $\mathcal{A}$ ($\mathcal{B}$), but only that when procedure $\mathcal{A}$ ($\mathcal{B}$) is executed, at the end of step 3(c)i (3(c)ii) A (B) is such that $C(A) = \min_i C((p_i^{(1)}, q^{(2)}(p_i^{(1)}))$, $i = 1, \cdots, u$ ($C(B) = \min_i C((q(p_i^{(2)})^{(1)}, p_i^{(2)}))$, $i = 1, \cdots, v$).

Experimental results will be presented for this version of the algorithm.

2.7 Pair of node-disjoint paths

If an optimal pair of node-disjoint path pair is desired, then, in procedure $\mathcal{A}$, 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 $\mathcal{B}$.

The previous algorithm (with this simple adaptation) would also solve the problem of obtaining the node-disjoint path pair of minimal cost.

2.8 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

	$\mathcal{E}$	$\bar{\mathcal{E}}$
$\mathcal{Z}$	$([0,100],[0,100])$	$([0,10],[0,10000])$
	$([0,10000],[0,10000])$	$([0,100],[0,10000])$
$\bar{\mathcal{Z}}$	$([1,100],[1,100])$	$([1,10],[1,10000])$
	$([1,10000],[1,10000])$	$([1,100],[1,10000])$

Table 1: Defining symbols to refer to cost range values

temporarily removed in the corresponding network directed graph, before executing the Dijkstra algorithm.

3 Experimental results

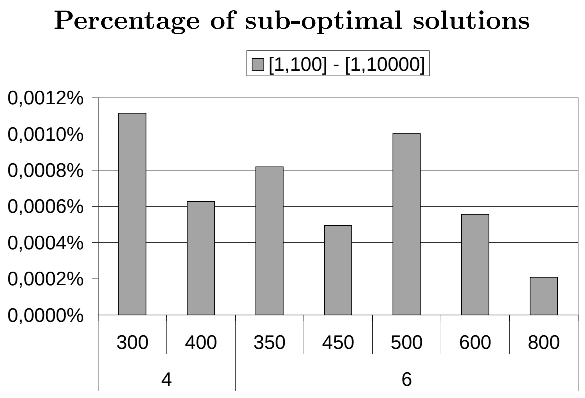
Extensive and systematic tests were carried out with the algorithm in a significant number of networks of different topologies and using various cost ranges.

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 ($n = 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 600, 800$) ten different networks were randomly generated[1] with the same number of arcs. All networks have arc-connectivity greater than one. For each of these networks the costs of the arcs were randomly generated in different ranges.

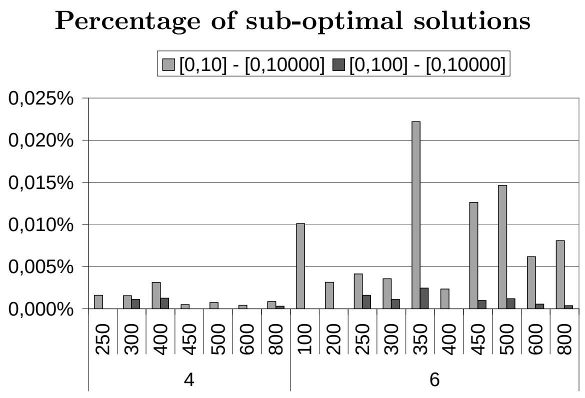
The symbols introduced in Table 1 will be used to refer to ranges of cost values. The first (second) range in each pair refers to the costs $c^{(1)}(c^{(2)})$. Identical (different) ranges for the two arc costs will be designated by $\mathcal{E}$ $(\bar{\mathcal{E}})$. Ranges with lower bound equal to 0 (1) will be designated by $\mathcal{Z}$ $(\bar{\mathcal{Z}})$. Finally $\mathcal{Z}\mathcal{E}$, $\mathcal{Z}\bar{\mathcal{E}}$, $\bar{\mathcal{Z}}\mathcal{E}$ and $\bar{\mathcal{Z}}\bar{\mathcal{E}}$, will identify the four groups (each group with two ranges) of ranges in the table.

It will be shown that algorithm DP2LC performs quite well and obtains the minimal cost pair of disjoint paths for almost every node pair. If a sub-set of node-pairs was randomly selected in each tested network (a procedure often adopted in this type of experimental study) the results could be misleading if the sub-set did not include any of the more “difficult” node pairs. Therefore it was decided that the DP2LC would be tested

[1]The used program for network generation was kindly borrowed from José Luis Santos.



(a)



(b)

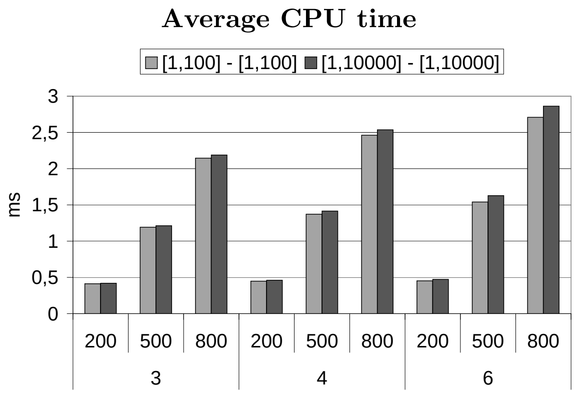
Figure 1: Percentage of sub-optimal solutions (DP2LC), for networks with sub-optimal solutions for $m = 4n, 6n$: (a) $\bar{\mathcal{Z}}\bar{\mathcal{E}}$ and (b) $\mathcal{Z}\bar{\mathcal{E}}$.

for all end-to-end node pairs ($n(n-1)$) in every network. Note that the total number of used networks was 2880.

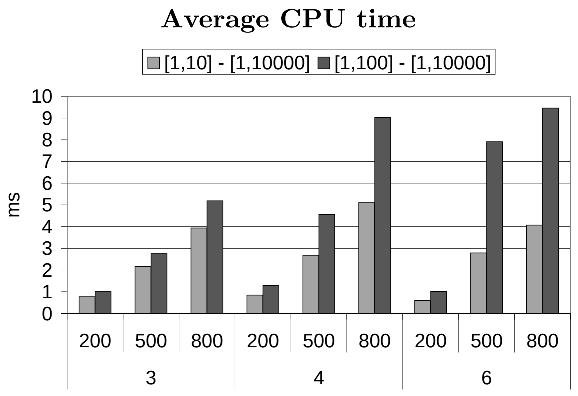
In all tested cases the algorithm always obtained a solution. Sub-optimal solutions are obtained when the optimal stopping condition cannot be verified, because memory is exhausted in one of the procedures (namely the allowed number of paths[2] that can be generated by MPS, is attained).

DP2LC only failed to find all optimal solutions for the ranges $\bar{\mathcal{Z}}$, in the case of range $([1, 100], [1, 10000])$ 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 checked) equal to 15, in approximately 235×10^6 node pairs considered in these tests (for all networks with ranges $\bar{\mathcal{Z}}$). The frequency of sub-optimal solutions for range $([1, 100], [1, 10000])$ is shown in figure 1(a). Note that in Figure 1 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 $m = 4$ each value in in Figure 1(a) corresponds to a single node pair (in a single network). In the case of ranges $\mathcal{Z}\bar{\mathcal{E}}$, for $m = 4n$ and $m = 6n$ the number of sub-optimal solutions is higher (a total of 1695 node pairs in approximately 235×10^6) which represents a small percentage of the tested node pairs (for all networks

[2]The allowed maximum number of paths ensures that only RAM memory is used and that no swapping takes place.



(a)



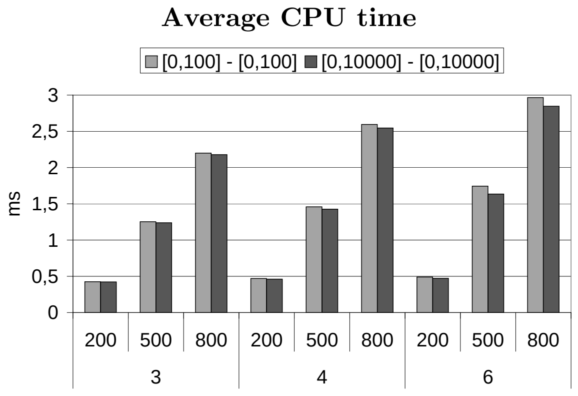
(b)

Figure 2: Average CPU time (DP2LC) per node pair (with optimal or sub-optimal solutions) for $n = 200, 500, 800$ and $m = 3n, 4n, 6n$: (a) ranges $\bar{\mathcal{Z}}\mathcal{E}$ (b) ranges $\bar{\mathcal{Z}}\bar{\mathcal{E}}$.

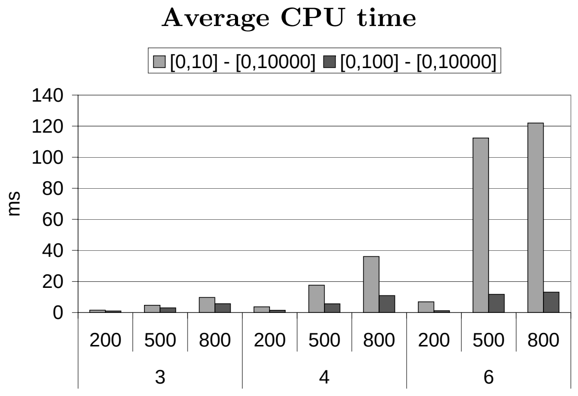
with ranges $\mathcal{Z}$). Figure 1(b) shows the percentage of sub-optimal solutions, for the networks where sub-optimal solutions were detected. The worst average value is 0.022% (the highest value in all tested cases was 0.1555% for $n = 350$ and $m = 6n$). Sub-optimal solutions occur more often for the costs in the range $([0, 10], [0, 10000])$.

CPU times were obtained in a Pentium IV at 3.2 GHz with 2 GB of RAM. The average CPU time was obtained per pair of disjoint paths for each node pair in the set of all s-t pairs with t fixed (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 Figures 2(a) and (b) average CPU times per node pair (with optimal or sub-optimal

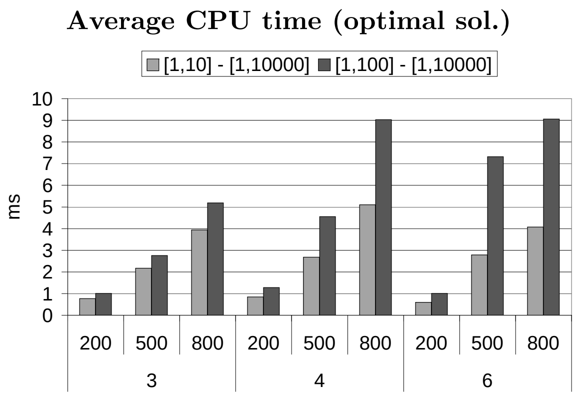


(a)

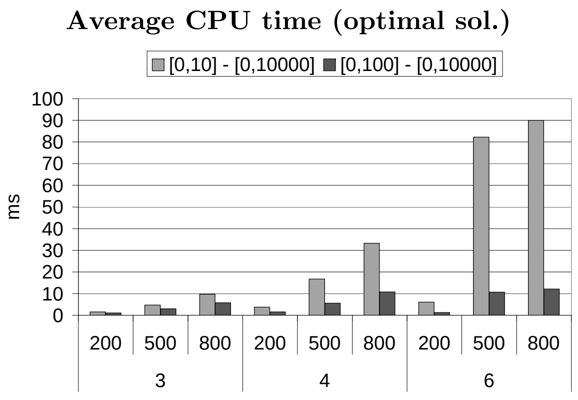


(b)

Figure 3: Average CPU time (DP2LC) per node pair (with optimal or sub-optimal solutions) for $n = 200, 500, 800$ and $m = 3n, 4n, 6n$: (a) ranges $\mathcal{Z}\mathcal{E}$ (b) ranges $\mathcal{Z}\bar{\mathcal{E}}$.



(a)



(b)

Figure 4: Average CPU time (DP2LC) per node pair with an optimal solution for $n = 200, 500, 800$ and $m = 3n, 4n, 6n$: (a) ranges $\bar{\mathcal{Z}}\bar{\mathcal{E}}$ and (b) ranges $\mathcal{Z}\bar{\mathcal{E}}$.

solutions) are presented for $n = 200, 500, 800$ and $m = 3n, 4n, 6n$ for costs ranges $\bar{\mathcal{Z}}\mathcal{E}$ and $\bar{\mathcal{Z}}\bar{\mathcal{E}}$. Two separate Figures are presented because in 2(b) the maximum CPU time is less than 3 ms and in 2(a) is less than 10 ms. Here it can be seen that the used CPU time is relatively higher in the case of different cost ranges.

In Figures 3(a) and (b) the average CPU times per node pair (with optimal or sub-optimal solutions) for $n = 200, 500, 800$ and $m = 3n, 4n, 6n$ for costs ranges $\mathcal{Z}\mathcal{E}$ and $\mathcal{Z}\bar{\mathcal{E}}$, are presented.

Recalling that DP2LC, for networks with arc costs in ranges $\mathcal{E}$, always obtained optimal solutions for all node pairs Figures 2(a) and 3(a) present the average CPU time for obtaining an optimal solution, per node pair.

From Figures 2(a) and 3(a) we conclude that the average CPU time for obtaining an optimal solution is similar in ranges $\mathcal{E}$ and approximately independent of the lower bound of the cost ranges. In Figure 2(b) a higher CPU time associated with different ranges for the arc costs $\bar{\mathcal{Z}}\bar{\mathcal{E}}$, is shown. In the case of $\mathcal{Z}\bar{\mathcal{E}}$ this CPU time increase is also visible in Figure 3(b). Results for ranges $([0, 100], [0, 10000])$ (in Figure 3(b)) and $([1, 100], [1, 10000])$ (in Figure 2(b)) are similar but a strong increase in used CPU time was detected for networks with arc costs in $([0, 10], [0, 10000])$ and $m = 4n, 6n$.

In Figures 4(a) and 4(b) the average used CPU time, per node pair, for obtaining an optimal solution for all considered networks with arc costs in ranges $\bar{\mathcal{E}}$, is presented (for networks with arc costs in ranges $\mathcal{E}$ all solutions were optimal). The results in Figures

2(b) and 4(a) are practically identical because of the negligible number of sub-optimal solutions in the ranges $\bar{\mathcal{Z}}$. However Figures 3(b) and 4(b) show some differences. In Figure 4(b) a visible decrease in CPU time can be seen when compared to 3(b), for $m = 6n$ and range $([0, 10], [0, 10000])$.

The number of sub-optimal solutions is rather small for $\mathcal{Z}\bar{\mathcal{E}}$ and negligible for ranges $\bar{\mathcal{Z}}\bar{\mathcal{E}}$ (as shown in Figure 1). 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 times can range from tens of seconds in a 100 node network with 600 arcs to hundreds of seconds in a network with 800 nodes and 4800 arcs, for cost range $([0, 10], [0, 10000])$. These high CPU times combined with a relative higher frequency (on average $\leq 0.0221\%$ as compared to $0, 0025\%$, the maximal average frequency of range $([0, 100], [0, 10000])$) of occurrence of sub-optimal solutions in the case of range $([0, 10], [0, 10000])$ have some impact in the average CPU time in Figure 3(b), for $n = 6m$.

A study was also made regarding the frequency of occurrence of the optimal pair among the first pair identified by procedure $\mathcal{A}$ or by procedure $\mathcal{B}$. The results showed that this frequency was greater than 99.469% for ranges $\bar{\mathcal{E}}$, regardless of network density or size. For ranges $\mathcal{E}$ this frequency was in the interval [0.93142,0.99266], increasing with the ratio m/n, regardless of the network size. In Figure 5 results are shown for all values of n and m but only for ranges $([1, 10], [1, 10000])$ and $([0, 100], [0, 100])$.

It was verified that, when optimal solutions were harder to find (usually for ranges $\bar{\mathcal{E}}$, and in particular for range $([0, 10], [0, 10000])$) one of the procedures had quickly found a very low cost path pair (procedure $\mathcal{B}$) which required the other procedure (procedure $\mathcal{A}$) to generate a large number of path pairs, in search for a better path pair.

Also one should note that DP2LC does not enter cycle 3c 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 cost range of the arcs and also to the network dimension. The average values were around 30%, 40% and close to 60% for $m = 3n$, $m = 4n$ and $m = 6n$, respectively.

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

- When arcs costs were in ranges $\mathcal{E}$ the algorithm managed to obtain optimal solutions

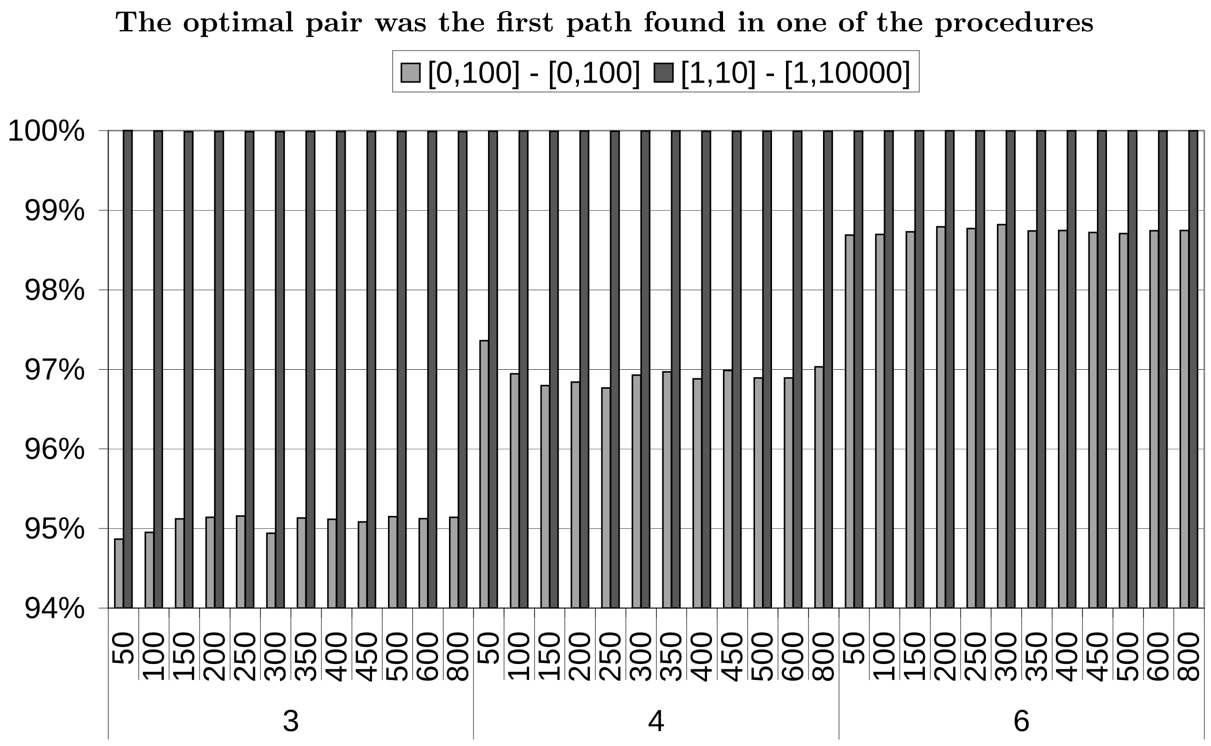


Figure 5: Frequency of occurrence of the optimal pair (DP2LC) among the first identified pair in procedures $\mathcal{A}$ or $\mathcal{B}$, for ranges $([1, 10], [1, 10000])$ and $([0, 100], [0, 100])$.

for all tested node pairs.

- When arc costs were in ranges $\bar{\mathcal{Z}}$, the algorithm obtained optimal solutions for almost every node pair, in all tested networks.

- In the case of networks with arc costs in ranges $\mathcal{Z}\bar{\mathcal{E}}$ the number of sub-optimal solutions was a little higher than in $\bar{\mathcal{Z}}\bar{\mathcal{E}}$ but nevertheless a very small percentage of the tested node pairs.

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

  Concerning average CPU time per node pair, for obtaining optimal solutions:

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

- For networks with ranges $\bar{\mathcal{Z}}\bar{\mathcal{E}}$, CPU times were a little higher than for ranges $\mathcal{Z}\mathcal{E}$, but still in the order of some milliseconds.
- For networks with ranges $\mathcal{Z}\bar{\mathcal{E}}$, CPU times were some milliseconds for range $([0,100],[0,10000])$ but several tens of milliseconds for $([0,10],[0,10000])$ (especially for $m = 6n$).

The complete set of results can be found in appendix C.

4 Conclusion

An exact algorithm, DP2LC, was proposed for finding an arc disjoint path pair, with minimal cost, in a network with dual arc costs. The algorithm is based on the resolution of two k-shortest path problems in an articulate manner, so that an optimal stopping condition is formulated. A proof of the correctness of the algorithm was also presented.

It was shown that 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. It was also shown that the proposed algorithmic procedure could be used for solving the same problem but with the additional constraint that a maximum number of arcs per path must not be exceeded.

Extensive experimentation with DP2LC was carried out in a significant number of randomly generated directed networks of different topologies and using eight cost ranges. A minimal cost disjoint path pair was sought for all node pairs of all the considered networks. Experimental results showed that DP2LC solved the problem exactly in practically all the cases in directed networks, the rare cases of failure being due to memory exhaustion alone. In the small percentage of cases in which an optimal solution was not attained, the algorithm always returned a sub-optimal solution.

The average CPU time per node pair, for obtaining an optimal solution, was shown to be a few milliseconds for all considered range costs, excepting for one cost range, where it sometimes required tens of milliseconds (for the larger and more dense networks). The algorithm is faster when the the dual arc costs have identical ranges. If execution time is critical, as in real time applications, a CPU time limit can be implemented per node pair, leading to a *truncated* version of the algorithm.

Finally, concerning applications of the algorithm, it must be noted that in the context of survivable routing, many approaches require a minimal cost pair of disjoint paths with dual arc costs. DP2LC, given its efficiency and exactitude, can be a very interesting solution in this and in similar contexts.

Part II

Set of all the minimal cost pair of disjoint paths with dual arc costs

5 Collecting all the alternative optimal solutions

If the addressed problem has more than one optimal solution, in a given network the previous approach can be used, with little modifications and some additional processing so that all optimal solutions can be obtained.

Let us define S_A and S_B as the stack of the current best candidate pairs found in approaches $\mathcal{A}$ and $\mathcal{B}$ respectively. The usual operations on a stack will be considered: push(S_A,A_i) (insert a new pair A_i in stack S_A), pop(S_A) (remove the top element from stack S_A), top(S_A) (return the top element from stack S_A, without removing it). The function "clear(S_A)" will pop all the elements in stack S_A until the stack becomes empty. The symbols $p^{(1)}$ ($q(p^{(1)})^{(2)}$) will represent the first (second) path of the pair top(S_A); similarly $q(p^{(2)})^{(1)}$ ($p^{(2)}$) will represent the first (second) path of top(S_B). Two functions $\gamma^{(j)}$, which return the maximumal cost, with respect to $\eta^{(j)}$, of all the elements in a stack S, will also be needed in the algorithm:

$$\gamma^{(j)} \quad : \quad S \rightarrow \mathbb{N}_0 \quad (j = 1, 2) \tag{23}$$

$$\gamma^{(1)}(S) = \max_{(f^{(1)}, g^{(2)}) \in S} C^{(1)}(f^{(1)}) \tag{24}$$

$$\gamma^{(2)}(S) = \max_{(f^{(1)}, g^{(2)}) \in S} C^{(2)}(g^{(2)}) \tag{25}$$

A sequence $U =< F_1, F_2, \ldots, F_{|U|} >$ of pairs of paths will also be required, where $|U|$ is the number of elements in the sequence. The function get(U) will return an element (pair of disjoint paths) from U by removing it. The function view(U, i), will produce a copy of element i in sequence U, without removing it.

5.1 Main steps of Set2LC

The proposed algorithm, designated Set2LC is more elaborate than DP2LC. Initially it uses two stacks, S_A and S_B for storing the best solutions (all with equal cost) obtained by procedures $\mathcal{A}$ and $\mathcal{B}$, respectively. Every time a new path pair A (B) is obtained with lower cost than top(S_A) (top(S_B)) the corresponding stack is cleared and the new found path pair becomes its only element. When top(S_A) and top(S_B) satisfy the optimal path pair condition, stacks S_A and S_B are possibly enlarged with additional path pairs of optimal cost. Then stacks S_A and S_B are merged and a sequence of all the different generated path pairs is obtained. Finally possibly missing path pairs are generated based on the pairs in that sequence.

```
1. Find the first path pair $A$ using procedure $\mathcal{A}$.

2. Find the first path pair $B$ using procedure $\mathcal{B}$.

3. Create Empty stacks $S_A$ and $S_B$. Execute: push($S_A$, $A$) and push($S_B$, $B$).

4. If top($S_A$) and top($S_B$) satisfy the optimal path pair condition Then Stop Else

    (a) Repeat

        i. While top($S_A$) has to be improved Do
           search for a new pair $A$ with cost lower than or equal to top($S_B$), using
           procedure $\mathcal{A}$, and update $S_A$.

        ii. While top($B$) has to be improved Do
           search for a new pair $B$ with cost lower than or equal to top($S_A$), using
           procedure $\mathcal{B}$, and update $S_B$.

       Until top($S_A$) and top($S_B$) satisfy the optimal path pair condition.

5. Seek to increase the size of stacks $S_A$ and $S_B$, obtaining additional path pairs:
```

   (a) `While` using procedure $\mathcal{A}$, paths pairs A can be obtained such that $C^{(1)}(p^{(1)}) \leq \gamma^{(1)}(S_B)$ `Do`
      generate such path pairs and `If` $C(A) = C(\text{top}(S_A))$ `Then` push(S_A, A).

   (b) `While` using procedure $\mathcal{B}$, paths pairs B can be obtained such that $C^{(2)}(p^{(2)}) \leq \gamma^{(2)}(S_A)$ `Do`
      generate such path pairs and `If` $C(B) = C(\text{top}(S_B))$ `Then` push(S_B, B).

6. Create the union of pairs in stacks S_A and S_B and store the result (in any order) in sequence U (no two path pairs are identical in U). Create an empty stack S_U.

7. `While` U is not empty `Do`

   (a) Remove a path pair from U and push it into S_U. Let $(p_g^{(1)}, q_g^{(2)}) \leftarrow \text{top}(S_U)$.

   (b) Generate possibly missing optimal path pairs, by interlacing $p_g^{(1)}$ and $q_g^{(2)}$ with paths in the pairs existing in U, and push them into stack S_U.

5.2 Detailed steps of Set2LC

The steps of the algorithm (Set2LC) are as follows:

1. Find the first pair of disjoint paths with MPS using costs $c_{ij}^{(1)}$ and Dijkstra using costs $c_{ij}^{(2)}$, in the pruned network:

   (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. Find the first pair of disjoint paths with MPS using costs $c_{ij}^{(2)}$ and Dijkstra using costs $c_{ij}^{(1)}$, in the pruned network:

   (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. Store each first candidate pair in the corresponding stack:

   (a) Creates empty stacks S_A and S_B.

   (b) push(S_A, A_u)

   (c) push(S_B, B_v)

4. Identify (and generate) optimal pair(s) of disjoint paths according to the rule:

   If $\left[C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p_v^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p_v^{(2)}) = C^{(2)}(q(p_u^{(1)})^{(2)})\right]$
   Then one pair ($p_u^{(1)} = q(p_v^{(2)})^{(1)}$ and $p_v^{(2)} = q(p_u^{(1)})^{(2)}$) or two different pairs of identical cost were found and they are both optimal.
   Else find an optimal pair (and its cost):

```
(a) Repeat
    i. Procedure $\mathcal{A}$:
       While $[C(\text{top}(S_B)) < C(\text{top}(S_A))] \vee$
             $\left[C^{(1)}(p^{(1)}) < C^{(1)}(q(p^{(2)})^{(1)}) \wedge C(\text{top}(S_A)) = C(\text{top}(S_B))\right]$ Do
       A. $u \leftarrow u + 1$
       B. MPS (loopless) generates $p_u^{(1)}$, and Dijkstra generates $q(p_u^{(1)})^{(2)}$
       C. If $C(A_u) \leq C(\text{top}(S_B))$ Then
               If $C(A_u) < C(\text{top}(S_A))$ Then clear$(S_A)$ EndIf
              push$(S_A, A_u)$
          EndIf
       EndWhileDo
    ii. Procedure $\mathcal{B}$:
       While $[C(\text{top}(S_A)) < C(\text{top}(S_B))] \vee$
             $\left[C^{(2)}(p^{(2)}) < C^{(2)}(q(p^{(1)})^{(2)}) \wedge C(\text{top}(S_A)) = C(\text{top}(S_B))\right]$ Do
       A. $v \leftarrow v + 1$
       B. MPS (loopless) generates $p_v^{(2)}$, and Dijkstra generates $q(p_v^{(2)})^{(1)}$
```

```
        C. If $C(B_v) \leq C(\text{top}(S_A))$ Then
               If $C(B_v) < C(\text{top}(S_B))$ Then clear$(S_B)$ EndIf
              push$(S_B, B_v)$
           EndIf
        EndWhileDo
    Until $\left[C^{(1)}(p^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p^{(2)}) = C^{(2)}(q(p^{(1)})^{(2)})\right]$
  EndIf
```

5. Seek the increase of the size of stack S_A (and S_B) with all paths $p_i^{(1)}$ ($p_j^{(2)}$) such that a disjoint path $q(p_i^{(1)})^{(2)}$ ($q(p_j^{(2)})^{(1)}$) might exist, with cost equal to the optimal cost:

```
  (a) Procedure $\mathcal{A}$:
      i. $c_A \leftarrow \gamma^{(1)}(S_B)$
      ii. Do
          A. $u \leftarrow u + 1$
          B. MPS (loopless) generates $p_u^{(1)}$, and Dijkstra generates $q(p_u^{(1)})^{(2)}$
          C. If $C(A_u) = C(\text{top}(S_A))$ Then push$(S_A, A_u)$ EndIf
          While $C^{(1)}(p_u^{(1)}) \leq c_A$

  (a) Procedure $\mathcal{B}$:
      i. $c_B = \gamma^{(2)}(S_A)$
      ii. Do
          A. $v \leftarrow v + 1$
          B. MPS (loopless) generates $p_v^{(2)}$, and Dijkstra generates $q(p_v^{(2)})^{(1)}$
          C. If $C(B_v) = C(\text{top}(S_B))$ Then push$(S_B, B_v)$ EndIf
          While $C^{(2)}(p_v^{(2)}) \leq c_B$
```

6. Create the union of pairs in stack S_A and stack S_B and store the result in sequence U (in any order).

7. Generate missing optimal path pairs based on the elements in U, and store them in stack S_U.

```
(a) Creates empty stack $S_U$.
(b) While $U \neq \emptyset$ Do
    i. $(p_g^{(1)}, q_g^{(2)}) \leftarrow \text{get}(U)$
    ii. $\text{push}(S_U, (p_g^{(1)}, q_g^{(2)}))$
    iii. $i = |U|$
    iv. While $i > 0$ Do
        A. $(p_i^{(1)}, q_i^{(2)}) \leftarrow \text{view}(U, i)$
        B. If $C^{(1)}(p_g^{(1)}) = C^{(1)}(p_i^{(1)}) \wedge p_g^{(1)} \neq p_i^{(1)} \wedge q_g^{(2)} \neq q_i^{(2)}$ Then
            • If ( $p_g^{(1)}$ is disjoint with $q_i^{(2)}$ ) Then $\text{push}(S_U, (p_g^{(1)}, q_i^{(2)}))$ EndIf
            • If ( $q_g^{(2)}$ is disjoint with $p_i^{(1)}$ ) Then $\text{push}(S_U, (p_i^{(1)}, q_g^{(2)}))$ EndIf
           EndIf
        C. $i \leftarrow i - 1$
       EndWhileDo
    EndWhileDo
```

At the end of step 4 the stacks S_A and S_B contain a path pair or a set of paths pairs with optimal cost (the minimal cost). One of the stacks has a single element and the other stack may have one or more elements, but they all have the same (optimal cost). In step 5 procedure $\mathcal{A}$ ($\mathcal{B}$) ensures that all paths $p_u^{(1)}$ ($p_v^{(2)}$) such that a disjoint path $q(p_u^{(1)})^{(2)}$ ($q(p_v^{(2)})^{(1)}$) might exist with $C(A_u)$ ($C(B_v)$) equal to the optimal cost, are generated and the corresponding optimal pair (if it exists) is stored in S_A (S_B).

The algorithm does not end at step 6 because some paths $p^{(1)}$ ($p^{(2)}$) obtained in procedure $\mathcal{A}$ ($\mathcal{B}$) may have more than one disjoint path, and any missing pair will be obtained in step 7.

5.3 Proof of the algorithm correctness

When only an optimal path pair was sought, the condition expressed by equation (5) and present in step 3 (of algorithm DP2LC) was used to detect that a disjoint path pair of optimal cost was found, so it was also the optimal stopping condition of the algorithm.

In the present case we wish to obtain the set of all the minimal cost pairs of disjoint paths in the network. The condition expressed by equation (5) and used in step 4 of algorithm Set2LC is now the detection condition of the minimal cost of disjoint paths in the network.

Immediately after step 3, stacks S_A and S_B each has a pair of disjoint paths (we are assuming a least one such a pair always exists). If the first path pairs obtained in step 1 and 2, verified the condition expressed by (5), then the optimal path cost has been found, as proved in lemma A.1.

If the first path pairs verified the condition expressed by (5), then step 4 only confirms that and proceeds to step 5. Otherwise the main cycle (step 4a) of step 4 is executed; when the algorithms exits cycle 4a, stacks S_A and S_B will contain a single path pair or a set of paths pairs (see appendix B) with optimal cost as stated in lemma A.3.

Lets assume that $p^{(1)} = p_u^{(1)}$ was the u-shortest (loopless) path from s to t with respect to $\eta^{(1)}$ obtained in procedure $\mathcal{A}$ and that $p^{(2)} = p_v^{(2)}$ was the v-shortest (loopless) path from s to t with respect to $\eta^{(2)}$ obtained in procedure $\mathcal{B}$, just after step 4. Let the optimal cost be $c_{opt} = C(\text{top}(S_A)) = C(\text{top}(S_B))$ and $c_{opt}^{(1)} = C^{(1)}(p^{(1)})$, $c_{opt}^{(1)} = C^{(2)}(p^{(2)})$ ($c_{opt} = c_{opt}^{(1)} + c_{opt}^{(2)}$), the cost of the first and second elements of the top pairs of stacks S_A and S_B, respectively, just after step 4. Lemma A.3 proves that, at the end of step 4 of algorithm Set2LC, c_{opt} is indeed the minimal cost of any pair of disjoint paths with dual arc costs in G. Lemma A.4 (A.5) proves that if a path $p_i^{(1)}$, $i = 1, 2, \ldots, u-1$ ($p_j^{(2)}$, $j = 1, 2, \ldots, v-1$), where u is the order of the first element of $\text{top}(S_A)$ (where v is the order of the first element of $\text{top}(S_B)$) is such that at least a minimal cost disjoint path exists in G, the first (second) element of which is $p_i^{(1)}$ ($p_j^{(2)}$) then a path pair $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$ ($B_j = (q(p_j^{(2)})^{(1)}), p_j^{(2)})$), with $C(A_i) = c_{opt}$ ($C(B_j) = c_{opt}$) will belong to S_A (S_B), at the end of step 4 (just before step 5).

Although the minimal cost of a disjoint path in network G has certainly been found at the end of step 4, there may still exist path pairs $(p_i^{(1)}, q(p_i^{(1)})^{(2)})$, $i > u$ and path pairs $(q(p_j^{(2)})^{(1)}, p_j^{(2)})$, $j > v$ with cost equal to the optimal cost c_{opt}.

An example of the situation pointed in the previous paragraph follows. Let $c_A = \gamma^{(1)}(S_B)$ and $c_B = \gamma^{(1)}(S_A)$. If S_A (S_B) has a single element then $c_B = C^{(2)}(q(p^{(1)})^{(2)})$ ($c_A = C^{(1)}(q(p^{(2)})^{(1)})$). Lets assume that stack S_A has more than one element; if at some time a path pair $A_a = (p_a^{(1)}, q(p_a^{(1)})^{(2)})$ ($a < u$, $C(A_a) = c_{opt}$) was generated and placed in stack S_A, such that $C^{(1)}(p_a^{(1)}) < C^{(1)}(p^{(1)})$ and $C^{(2)}(q(p_a^{(1)})^{(2)}) > C^{(2)}(p^{(2)})$, then c_B will be larger than $C^{(2)}(p^{(2)})$. A similar reasoning can be used when S_B has more than one element (and S_A has a single pair).

In short, procedures $\mathcal{A}$ and $\mathcal{B}$ must now be used to generate all the next k-shortest paths with cost less or equal to c_A and c_B, respectively, obtain the corresponding disjoint path, and if the pair exist and has cost equal to c_{opt}, store it in stacks S_A and S_B, respectively. This is done in step 5.

So just after step 5 we have in stack S_A path pairs $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$, such that the first element $p_i^{(1)}$ has cost $C^{(1)}(p_i^{(1)}) \in \left[c_{opt}^{(1)}, c_A\right]$ (for all $p_i^{(1)}$ such that a disjoint path A_i exists $C(A_i) = c_{opt}$) and in stack S_B the path pairs $B_j = (q(p_j^{(2)})^{(1)}, p_j^{(2)})$, $C(B_j) = c_{opt}$, such that the second element $p^{(2)}$ has cost $C^{(2)}(p_j^{(2)}) \in \left[c_{opt}^{(2)}, c_B\right]$ (for all $p_j^{(2)}$ such that a least a disjoint path B_j exist with $C(B_j) = c_{opt}$). Procedure $\mathcal{A}$ ($\mathcal{B}$) in step 5 does not need to generate any more path pairs, because by lemma A.6 it is not possible to obtain a path pair with cost equal to c_{opt}, if its first (second) element, $p_i^{(1)}$ ($p_j^{(2)}$), is such that $C^{(1)}(p_i^{(1)}) > c_A$ ($C^{(2)}(p_j^{(2)}) > c_B$).

Immediately after step 5 we know the algorithm has generated all paths $p_i^{(1)}$ ($p_j^{(2)}$), that may be the first (second) element of a disjoint path pair of optimal cost is stored in S_A (S_B). However we still may not have the complete set of optimal disjoint paths. If a path $p_i^{(1)}$ ($p_j^{(2)}$) has more than one disjoint path in G (such that the path pair cost is optimal), procedure $\mathcal{A}$ ($\mathcal{B}$) only gets one of them: $q(p_i^{(1)})^{(2)}$ ($q(p_j^{(2)})^{(1)}$). Also, some of the elements in stack S_A may also be in stack S_B, therefore the union of the elements of both stacks is performed, and is stored sequence U (which will not have any repeated pair), in step 6.

We can now proceed to obtain the missing pairs of disjoint paths. Given two disjoint path pairs of optimal cost, $D = (d^{(1)}, e^{(2)})$ and $F = (f^{(1)}, g^{(2)})$, a new disjoint path pair may exist if $C^{(1)}(d^{(1)}) = C^{(1)}(f^{(1)})$ (or $C^{(2)}(e^{(2)}) = C^{(2)}(g^{(2)})$) and if $d^{(1)} \neq f^{(1)}$ and $e^{(2)} \neq g^{(2)}$; if $d^{(1)}$ is disjoint with $g^{(2)}$, then a new pair can be obtained: $(d^{(1)}, g^{(2)})$; if $f^{(1)}$ is disjoint with $e^{(2)}$, then a new pair can be obtained:$(f^{(1)}, e^{(2)})$.

Step 7 ensures all such missing pairs are obtained and added to the set of optimal solutions by removing each element from the the sequence U, storing it in S_U and then comparing this pair with all the remaining pairs in U. Each comparison may generate new path pairs if the conditions presented in the previous paragraph are true.

Finally the algorithm ends with an optimal and complete set of all minimal cost pairs of disjoint paths with dual arc costs in stack S_U.

A formal proof of the correctness of algorithm Set2LC will now be given.

Proposition 5.1 (Correctness of algorithm Set2LC) *At the end of step 7 all the disjoint path pairs in G, c_{opt}, with minimal cost have been found.*

Proof: Immediately after step 3, stacks S_A and S_B each has a pair of disjoint paths (we are assuming a least one such a pair always exists). If the first path pairs obtained in step 1 and 2, verified the condition expressed by (5), then the optimal path cost has been found, as proved in lemma A.1.

If the first path pairs verified the condition expressed by (5), then step 4 only confirms that and proceeds to step 5. Otherwise the main cycle (step 4a) of step 4 is executed; when the algorithms exits cycle 4a, stacks S_A and S_B will contain a single path pair or a set of paths pairs (see appendix B) with optimal cost, c_{opt}, as stated in lemma A.3.

Lemma A.4 (A.5) proves that if a path $p_i^{(1)}$, $i = 1, 2, \dots, u-1$ ($p_j^{(2)}$, $j = 1, 2, \dots, v-1$), where u is the order of the first element of top(S_A) (where v is the order of the first element of top(S_B)) is such that at least a minimal cost disjoint path exists in G, the first (second) element of which is $p_i^{(1)}$ ($p_j^{(2)}$) then a path pair $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$ ($B_j = (q(p_j^{(2)})^{(1)}), p_j^{(2)})$), with $C(A_i) = c_{opt}$ ($C(B_j) = c_{opt}$) will belong to S_A (S_B), at the end of step 4.

By lemma A.6 it is not possible to obtain a path pair with cost equal to c_{opt}, if its first (second) element, $p_i^{(1)}$ ($p_j^{(2)}$), is such that $C^{(1)}(p_i^{(1)}) > c_A$ ($C^{(2)}(p_j^{(2)}) > c_B$). Immediatly after step 5 we have in stack S_A path pairs $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$, such that the first element $p_i^{(1)}$ has cost $C^{(1)}(p_i^{(1)}) \in \left[c_{opt}^{(1)}, c_A\right]$ (for all $p_i^{(1)}$ such that a disjoint path A_i exists $C(A_i) = c_{opt}$) and in stack S_B path pairs $B_j = (q(p_j^{(2)})^{(1)}, p_j^{(2)})$, $C(B_j) = c_{opt}$, such that the second element $p^{(2)}$ has cost $C^{(2)}(p_j^{(2)}) \in \left[c_{opt}^{(2)}, c_B\right]$ (for all $p_j^{(2)}$ such that a least a disjoint path B_j exist with $C(B_j) = c_{opt}$). Therefore by lemma A.6, procedure $\mathcal{A}$ ($\mathcal{B}$) in step 5 have obtained the remaining existing paths that can be the first (second) element of pairs of disjoint paths with cost c_{opt}, and has stored them in S_A (S_B), whenever such a disjoint path pair was obtained.

Because some of the elements in stack S_A may also be in stack S_B, the union of the elements of both stacks is performed, and is stored sequence U (which will have no repeated path pair), in step 6.

Immediately after step 5 we know the algorithm has generated all paths $p_i^{(1)}$ ($p_j^{(2)}$), that may be the first (second) element of a disjoint path pair of optimal cost is stored in S_A (S_B), therefore the same statement is still valid for the elements in U, obtained in step 6.

Finally in 7 the (possibly) still missing pairs of disjoint paths, resulting form the interlacing of paths in U are obtained. Given two disjoint path pairs of optimal cost, $D = (d^{(1)}, e^{(2)})$ and $F = (f^{(1)}, g^{(2)})$, a new disjoint path pair may exist if $C^{(1)}(d^{(1)}) = C^{(1)}(f^{(1)})$ (or $C^{(2)}(e^{(2)}) = C^{(2)}(g^{(2)})$) and if $d^{(1)} \neq f^{(1)}$ and $e^{(2)} \neq g^{(2)}$; if $d^{(1)}$ is disjoint with $g^{(2)}$, then a new pair can be obtained: $(d^{(1)}, g^{(2)})$; if $f^{(1)}$ is disjoint with $e^{(2)}$, then a new pair can be obtained:$(f^{(1)}, e^{(2)})$. Step 7 ensures all such missing pairs are obtained and added to the set of optimal solutions by removing each element from the the sequence U, storing it in S_U and then comparing this pair with all the remaining pairs in U. Each comparison may generate new path pairs if the conditions presented in the previous paragraph are true.

Finally the algorithm ends with an optimal and complete set of all minimal cost pairs of disjoint paths with dual arc costs in stack S_U. $\square$

6 Experimental results

In section 3 experimental results showed that sometimes algorithm DP2LC had difficulty finding the optimal condition, and that in those cases a long CPU was often required. Therefore experimental results for Set2LC will presented, with a limit of CPU time per node pair of 50ms (as soon as 50 ms are exceeded the algorithm terminates), for directed networks. So three types of situations may occur:

1. The complete set of optimal solutions.

2. An incomplete set of optimal solutions.

3. A set of sub-optimal solutions: the returned set is a set of pairs, the optimality of which could not be verified.

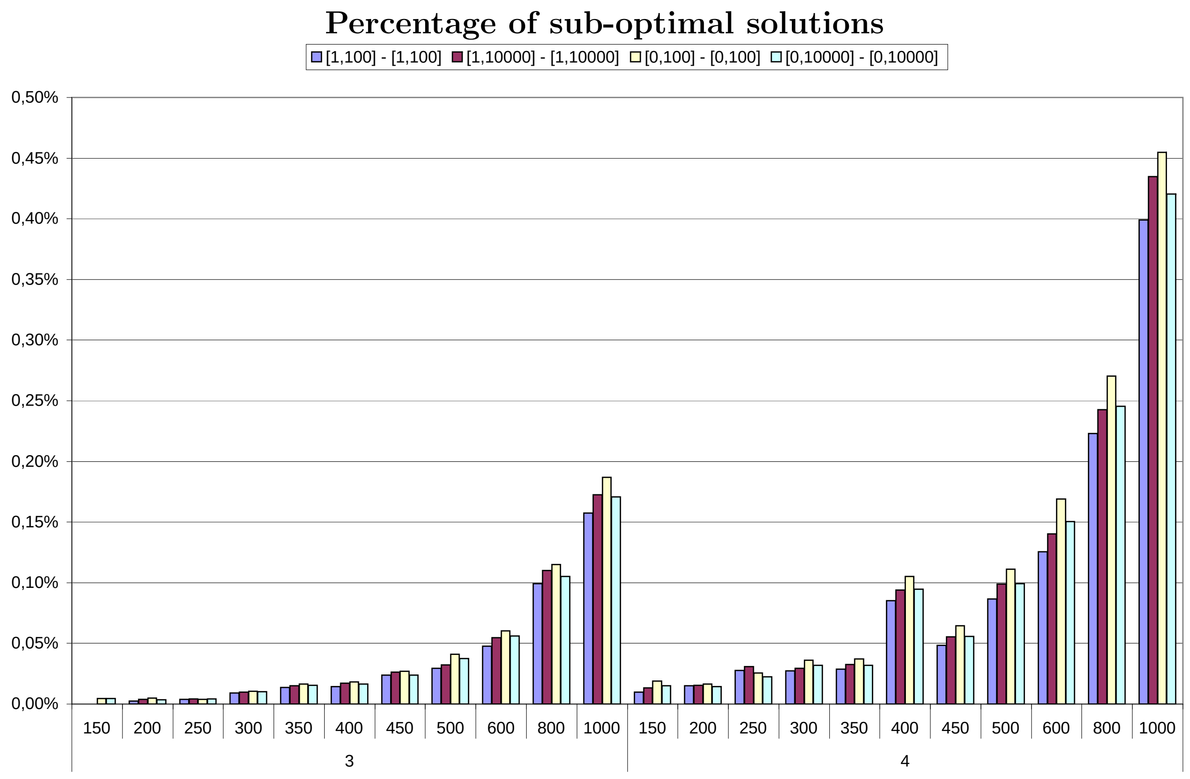


Figure 6: Percentage of sub-optimal solutions (Set2LC), for networks with sub-optimal solutions for $m = 3n, 4n$ and ranges $\mathcal{E}$.

Networks with number or arcs $m = 3n, 4n$ were used for evaluating the algorithm's performance (recall that n is the number of nodes in the network).

Experiments showed that, in the case of Set2LC, the algorithm behaviour depends more on the fact that ranges for the costs of the arcs are similar (ranges $\mathcal{E}$) versus different (ranges $\bar{\mathcal{E}}$), than on the fact that the lower bound of the ranges is zero or not.

In Figures 6 and 7 the percentage of suboptimal solutions for networks with sub-optimal solutions is presented. For less dense networks the number of sub-optimal solutions was lower. In Figure 6 the results were similar regardless of $\bar{\mathcal{Z}}$ or $\mathcal{Z}$ and always less than 0.5%; in Figures 7 the number of sub-optimal solutions is approximately ten times higher, and results are worse for the costs in range $([0, 10], [0, 10000])$.

CPU time per node pair is presented in Figures 8 and 9, and it was consistently lower for costs in ranges $\mathcal{E}$. Note that although the algorithm's runs were stopped only when 50 ms were exceed, the values in figures 8 and 9 are always less than 3.5 ms and 8 ms, respectively.

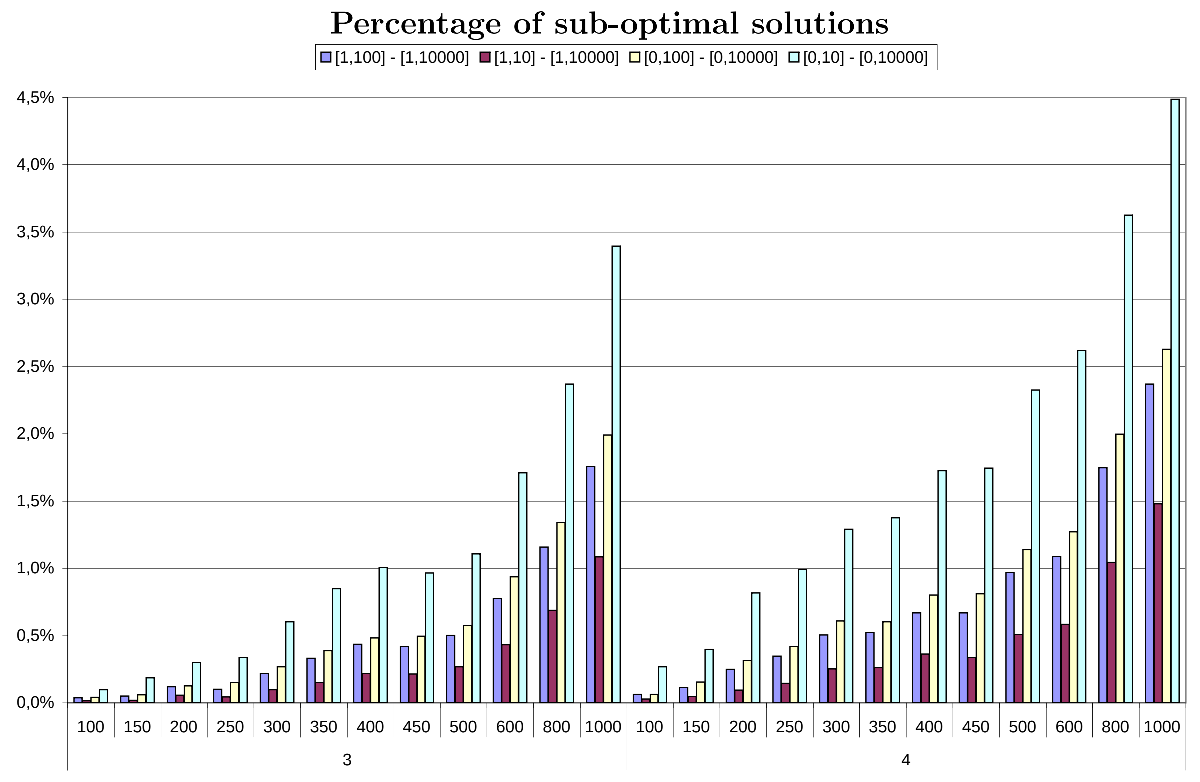


Figure 7: Percentage of sub-optimal solutions (Set2LC), for networks with sub-optimal solutions for $m = 3n, 4n$ and ranges $\bar{\mathcal{E}}$.

The percentage of incomplete sets with optimal disjoint pairs was very small: less than 0.03% for for all ranges except for $([0, 10], [0, 10000])$, where it was always less than 0.7%.

The average number of optimal solutions per node pair was also analysed. The results are quite similar for $m = 3n$ and m= $4n$ for all ranges. There is however a slight increase in this average for the two more dissimilar cost ranges in $\bar{\mathcal{E}}$, as can be seen in figure 12. Values are shown only for ranges $\bar{\mathcal{E}}$, because the results for ranges $\mathcal{E}$ were very similar to $([0, 100][0, 10000])$ and $([1, 100], [1, 10000])$, and the corresponding average values were always in the interval $[1, 1.05[$.

The average for the maximum number solutions is again presented in Figure 13 for $m = 4n$ and ranges $([0, 10][0, 10000])$ and $([1, 10], [1, 10000])$, where an error bar was added, centred in the average μ of the collected samples (one sample per network) which goes from $\max(1, \mu - \sigma)$ to $\mu + \sigma$, where σ is the standard deviation of the sample. The purpose of this bar was to show the low variability of the results.

Although the average number of optimal solutions per node pair was close to one, the average number of maximum solutions (for some nome pair(s)) obtained for the ten

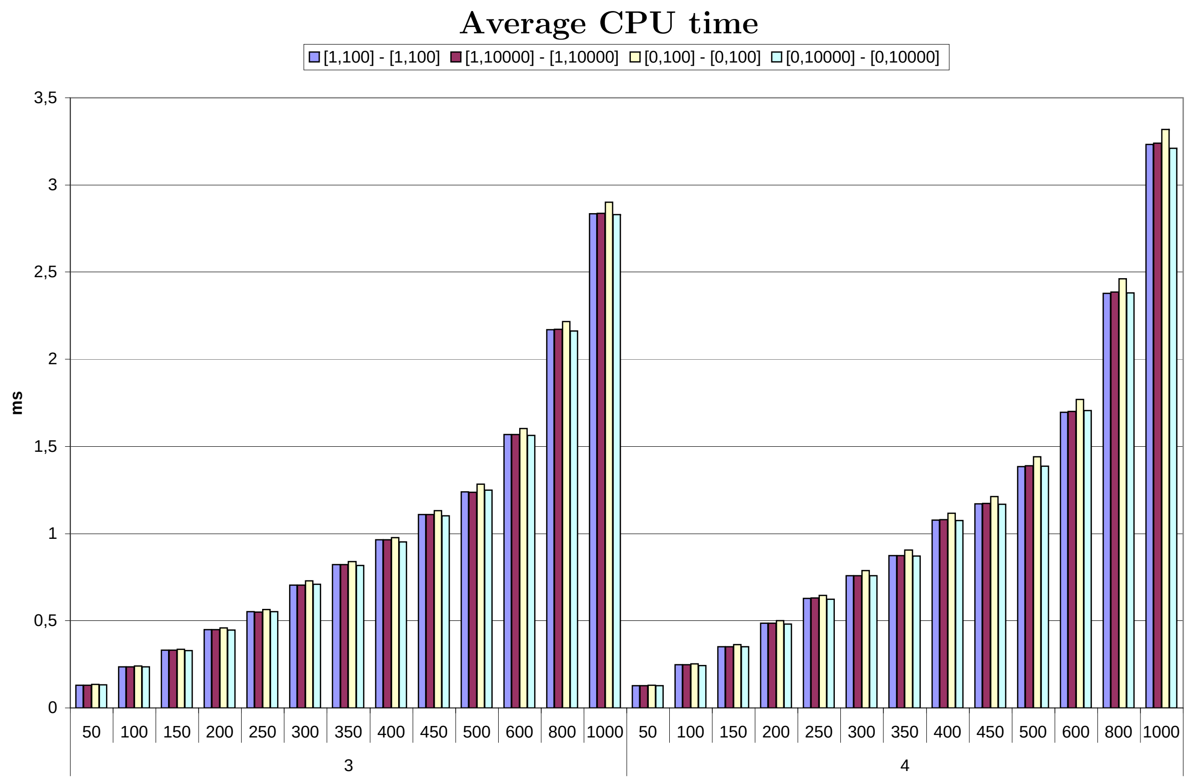


Figure 8: Average CPU time (Set2LC) per node pair (with optimal or sub-optimal solutions) for $m = 3n, 4n$ and ranges $\mathcal{E}$.

networks in each experimentwas higher as shown in figures 14 and 15. Ranges $\bar{\mathcal{E}}$ presented higher values than range $\mathcal{E}$. Also note that ranges $([0, 10][0, 10000])$ and $([1, 10], [1, 10000])$ presented the higher average for the maximum number of sets, which is consistent with the results in figure 12.

A Auxiliary lemmas to prove Set2LC correction

Lemma A.1 *Immediately after step 3, S_A has a single path pair, A_u, and S_B has also a single path pair, B_v. If $C(A_u) = C(B_v)$ and $C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p_v^{(2)})^{(1)})$ (and therefore $C^{(2)}(p_v^{(2)}) = C^{(2)}(q(p_u^{(1)})^{(2)})$) then $c_{opt} = C(A_u) = C(B_v)$ and the algorithm has found two optimal disjoint path pairs and the value of the minimal path cost of any pair of disjoint paths in G is c_{opt}.*

Proof: Immediately after step 1, $p_u^{(1)}$ is the first path of minimal cost with respect to metric $\eta^{(1)}$ which has a disjoint path in G. Using Dijkstra algoritm the path, $q(p_u^{(1)})^{(2)}$,

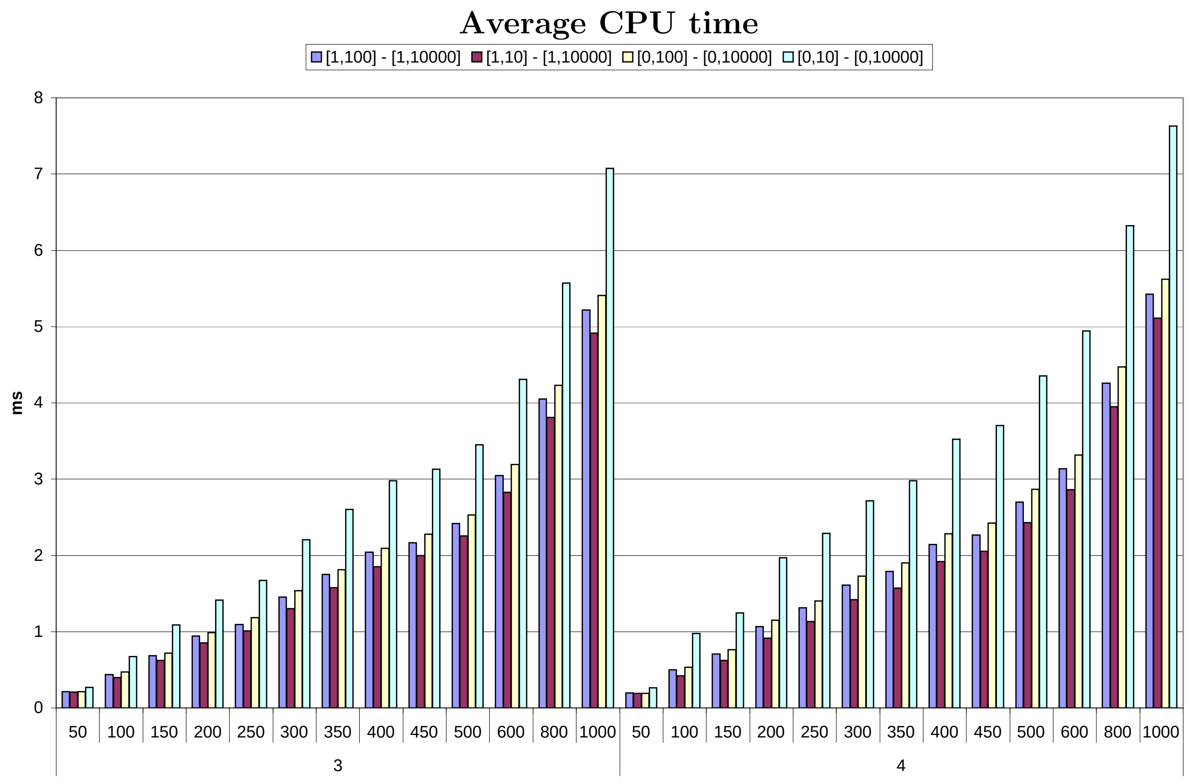


Figure 9: Average CPU time (Set2LC) per node pair (with optimal or sub-optimal solutions) for $m = 3n, 4n$ and ranges $\bar{\mathcal{E}}$.

of minimal cost with respect to metric $\eta^{(2)}$ was found in $G^{(1)}$. Immediately after step 2, $p_v^{(2)}$ is the first path of minimal cost with respect to metric $\eta^{(2)}$ which has a disjoint path in G. Using Dijkstra algoritm the path, $q(p_v^{(2)})^{(1)}$, of minimal cost wit respect to metric $\eta^{(1)}$ was found in $G^{(2)}$. If $C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p_v^{(2)})^{(1)})$, then the first element of B_v is a path of minimal cost with respect to metric $\eta^{(1)}$, that has a disjoint path in G, and the second element of B_v, $p_v^{(2)}$, is (by step 2) the path of minimal cost with respect to metric $\eta^{(2)}$, that has a disjoint path in G. Therefore B_v is formed by two disjoint paths, $p_v^{(2)}$ and $q(p_v^{(2)})^{(1)}$, each with minimal possible cost with respect to metric $\eta^{(j)}$, $j = 2, 1$, respectively, and therefore $C(B_v)$ is the minimal cost of any disjoint path pair with dual arc cost, c_{opt}.

A similar argument can be made regarding path pair A_u. $\square$.

So, the first part of step 4 is proved:

“ `If` $\left[C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p_v^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p_v^{(2)}) = C^{(2)}(q(p_u^{(1)})^{(2)})\right]$

`Then` one pair ($p_u^{(1)} = q(p_v^{(2)})^{(1)}$ and $p_v^{(2)} = q(p_u^{(1)})^{(2)}$) or two different pairs of

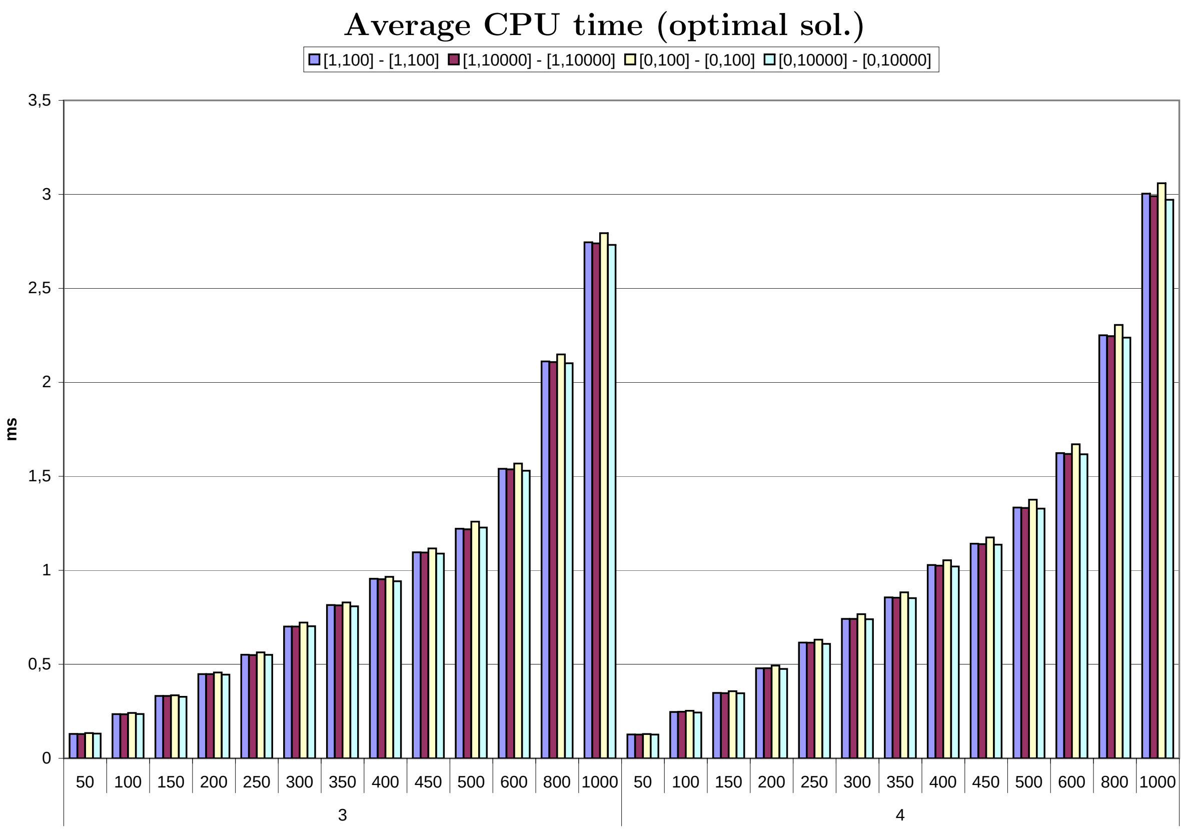


Figure 10: Average CPU time (Set2LC) per node pair with an optimal solution for $m = 3n, 4n$ and ranges $\mathcal{E}$.

identical cost were found and they are both optimal."

The main cycle in step 4a is entered only when the optimal cost detection condition was not true for the first pairs.

Lemma A.2 *Whenever step 4 tests the condition:*

$$\left[C^{(1)}(p^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p^{(2)}) = C^{(2)}(q(p^{(1)})^{(2)})\right]$$

S_A and S_B contain the, and only the, current best pairs of disjoint paths, from steps 1 and 2, or found using procedures $\mathcal{A}$ (step 4(a)i) and $\mathcal{B}$ (step 4(a)ii), respectively.

Proof: When the main cycle of step 4a is executed for the first time, S_A and S_B contain the first and therefore the best current pair of disjoint paths found using procedure $\mathcal{A}$ (step 1) and $\mathcal{B}$ (step 2), respectively.

When the main cycle of step 4a is executed, procedure $\mathcal{A}$ (step 4(a)i), will only be entered when the solution in S_A needs to be improved because $C(\text{top}(S_B)) < C(\text{top}(S_A))$

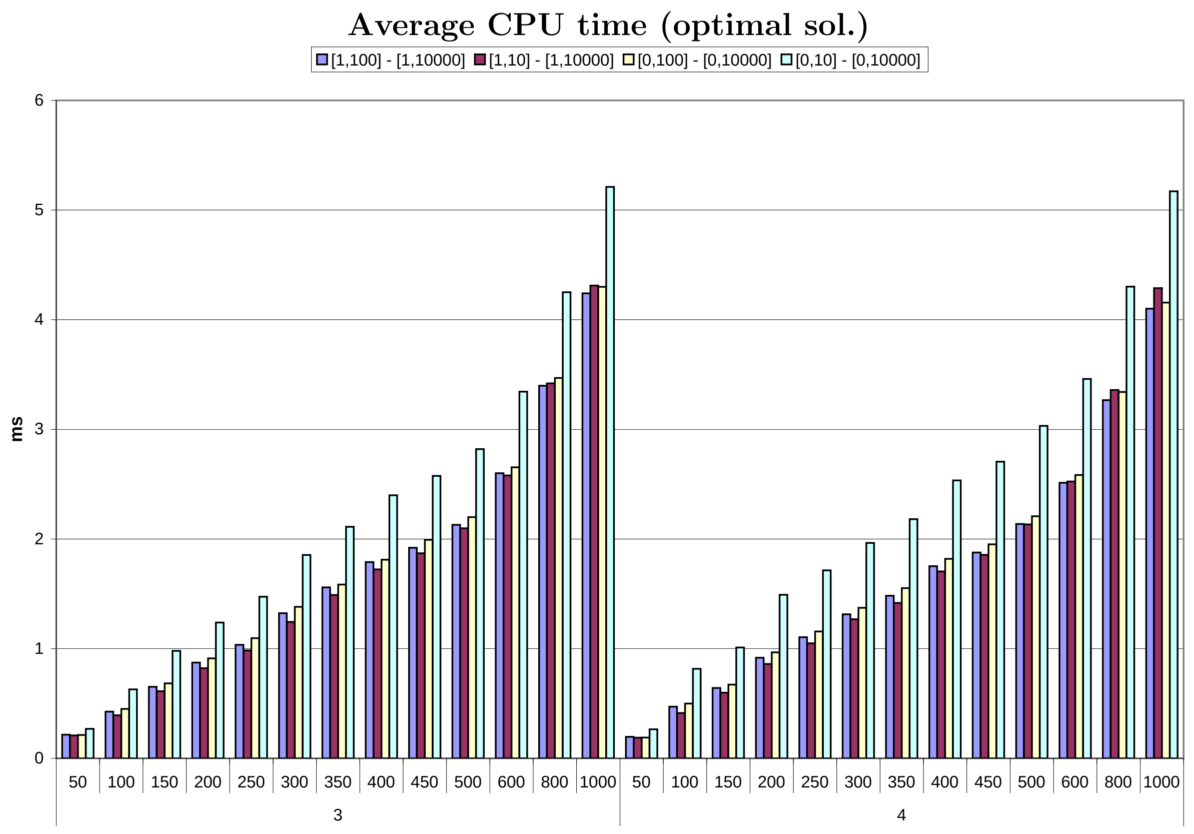


Figure 11: Average CPU time (Set2LC) per node pair with an optimal solution for $m = 3n, 4n$ and ranges $\bar{\mathcal{E}}$.

or because $C(\text{top}(S_B)) = C(\text{top}(S_A))$ and $C^{(1)}(p^{(1)}) < C^{(1)}(q(p^{(2)})^{(1)})$. In step 4(a)i stack S_A will be updated whenever $C(A_u) \leq C(\text{top}(S_B))$, but before pushing A_u into S_A, the algorithm ensures that at end of cycle (step 4(a)i) S_A always has path pairs of identical and current minimal cost, so if $C(Au) < C(\text{top}(S_A))$ stack S_A is cleared before pusing A_u into S_A. The cycle in step 4(a)i terminates when pair A_u, such that $C(A_u) < C(\text{top}(S_B))$ or when A_u, such that $C(A_u) = C(\text{top}(S_B))$ with $C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})$, is pushed into S_A. Therefore if the cycle in step 4(a)i is entered, then when it terminates S_A has the current best pair (or pairs) found so far, using prodecure $\mathcal{A}$, and all pairs in S_A have the same cost.

Similarly for procedure $\mathcal{B}$ (step 4(a)ii). $\square$

Lemma A.3 *If condition (in equation (5)):*

$$\left[C^{(1)}(p^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})\right] \wedge \left[C^{(2)}(p^{(2)}) = C^{(2)}(q(p^{(1)})^{(2)})\right]$$

is true at the end of step 4, then the minimal cost of the disjoint path pairs, c_{opt} =

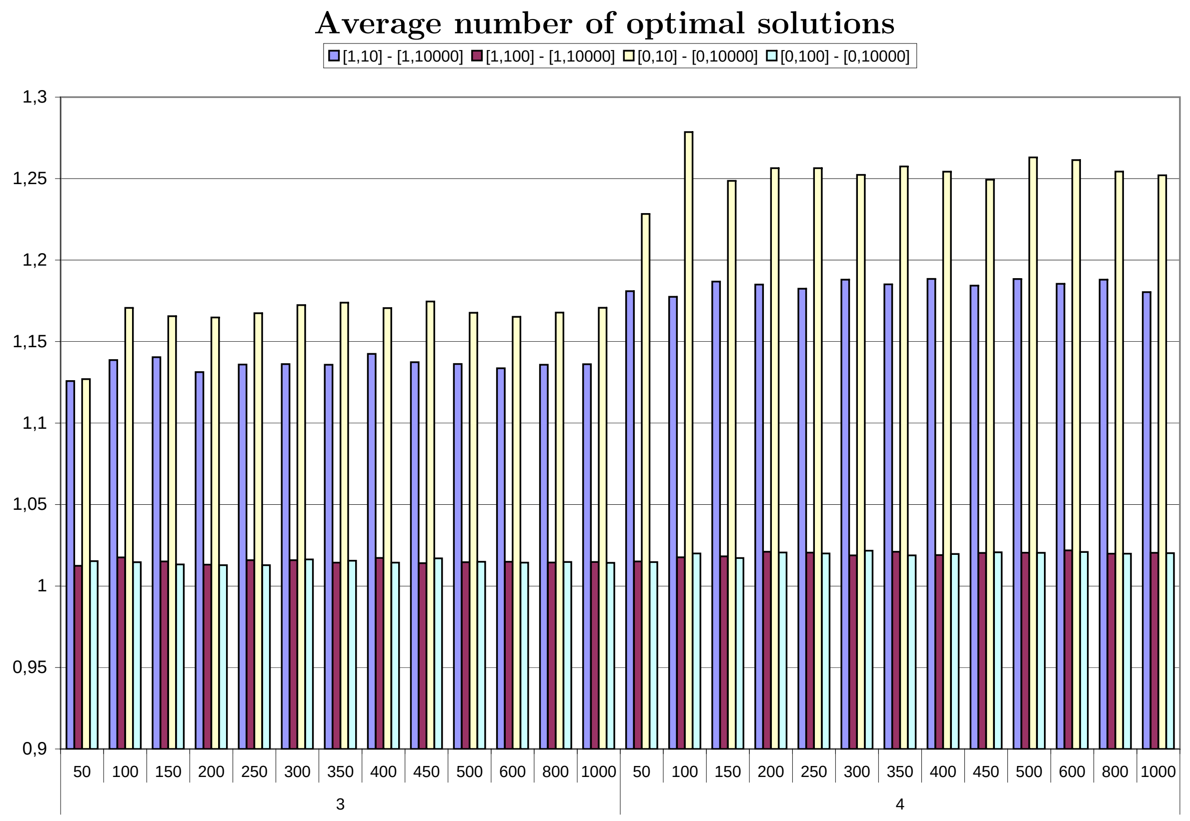


Figure 12: Average number of optimal solutions (Set2LC) for $m = 3n, 4n$ and ranges $\bar{\mathcal{E}}$.

$C(\text{top}(A)) = C(\text{top}(B))$, *in* G *was found, and all path pairs in* S_A *and* S_B *have cost* c_{opt}.

Proof: If the first pairs found in 1 an 2 are optimal (see lemma A.1) then c_{opt} has been found.

If the first pairs found in 1 an 2 do not satisfy equation (5) then the main cycle 4a of step 4 will be executed until equation (5) becomes true. By lemma A.2 stacks S_A and S_B have at the time of evaluation of equation (5) in the step 4a the best current path pairs, therefore if equation (5) is true they must contain optimal path pairs, unless a better path could yet be found.

Consider that a better path pair $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$, with $i > u$, exists, where u is the order of the last path generated in procedure $\mathcal{A}$ and $A_u = \text{top}(S_A)$. Then $C(A_i) < C(\text{top}(S_A))$ if and only if $C^{(1)}(p_i^{(1)}) \geq C^{(1)}(p_u^{(1)})$ (by lemma A.2 and because $i > u$)and $C^{(2)}(q(p_i^{(1)})^{(2)}) < C^{(2)}(q(p_u^{(1)})^{(2)})$. But if $C^{(2)}(q(p_i^{(1)})^{(2)}) < C^{(2)}(q(p_u^{(1)})^{(2)})$, then $q(p_u^{(1)})^{(2)}$ must coincide with some $p_k^{(2)}$, $k < v$, and if its cost was lower than $C(\text{top}(S_B))$ it would have been stored in S_B and $C(\text{top}(S_B)) \neq C(\text{top}(S_A))$. So no such path A_i can exist.

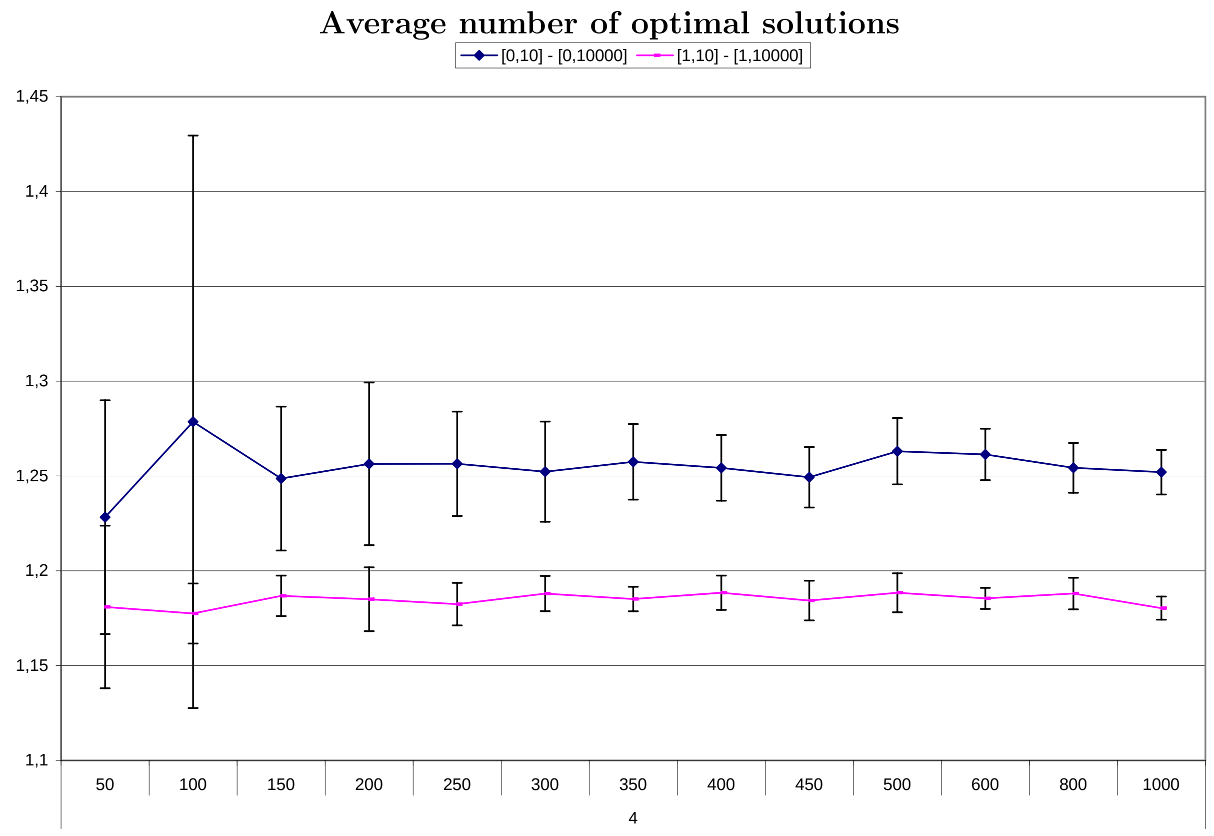


Figure 13: Average number of optimal solutions (Set2LC) for $m = 4n$ and ranges ($[0,10][0,10000]$) and ($[1,10],[1,10000]$).

Similarly, consider that a better path pair $B_j = (q(p_j^{(2)})^{(1)}, p_j^{(2)})$, with $j > v$ where v is the order of the last path generated in procedure $\mathcal{B}$ and $B_v = \text{top}(S_B)$. Then $C(B_j) < C(\text{top}(S_B))$ if and only if $C^{(2)}(p_j^{(2)}) \geq C^{(2)}(p_v^{(2)})$ (because $j > v$ and of lemma A.2) and $C^{(1)}(q(p_j^{(2)})^{(1)}) < C^{(2)}(q(p_v^{(2)})^{(1)})$. But if $C^{(1)}(q(p_j^{(2)})^{(1)}) < C^{(2)}(q(p_v^{(2)})^{(1)})$, then $q(p_v^{(2)})^{(1)}$ must coincide with some $p_k^{(1)}$, $k < u$, and if its cost was lower than $C(\text{top}(S_A))$ it would have been stored in S_A and $C(\text{top}(S_B)) \neq C(\text{top}(S_A))$. So no such path B_j can exist. And this concludes the proof. $\Box$

Lemma A.3 (wich uses lemmas A.1 and A.2) will now be used to prove that for all paths $p_i^{(1)}$, $i = 1, 2, \ldots, u$ (where u is the order of the first element of $\text{top}(S_A)$), a disjoint path pair $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$ of minimal cost was stored in S_A. And similarly for paths $p^{(2)}$.

Lemma A.4 *If a path* $p_i^{(1)}$, $i = 1, 2, \ldots, u-1$ *(where* u *is the order of the first element of* $\text{top}(S_A)$*) is such that at least a minimal cost disjoint path pair exists in* G*, the first*

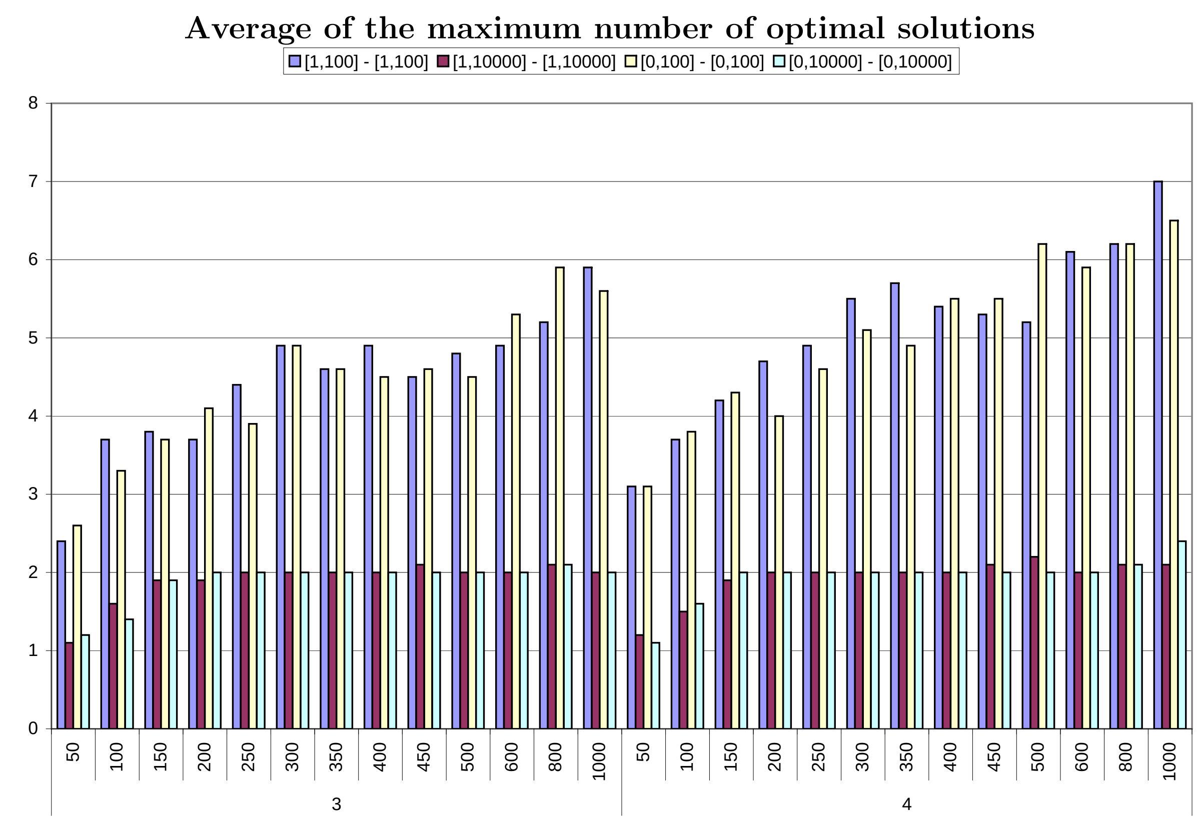


Figure 14: Average of the maximum number of optimal solutions (Set2LC) for $m = 3n, 4n$ and ranges $\mathcal{E}$.

element of wchich is $p_i^{(1)}$, then a path pair $A_i = ((p_i^{(1)}, q(p_i^{(1)})^{(2)}))$, with $C(A_i) = c_{opt}$ will belong to S_A, at the end of step 4 (just before step 5) in algorithm Set2LC.

Proof: If at the begining of step 4 the condition expressed by equation (5) is true then, by lemma A.1, stacks S_A and S_B each contains a single (optimal) pair. In this case this lemma is true, because the first element of S_A is the first shortest path, with repect to metric $\eta^{(1)}$, for which a disjoint path exists.

If at the begining of step 4 the condition expressed by equation (5) is not true, then the main cycle 4a is entered. This lemma would be false if and only if a path pair existed, $A_k = (p_k^{(1)}, q_k^{(2)})$, such that $C(A_k) = c_{opt}$, with $k \in [1, u-1]$ and the path $p_k^{(1)}$ did not coincide with the first element of any path pair in S_A.

By lemma A.3, c_{opt} is the minimal cost of any pair of disjoint paths with dual arc cost in G, therefore, at most $C(A_k) = c_{opt}$.

Recall that all paths $p_i^{(1)}$, $i = 1, 2, \cdots, u$, were sequentially generated by a k-shortest path enumeration algorithm (in our case MPS), and that a disjoint path (the shortest

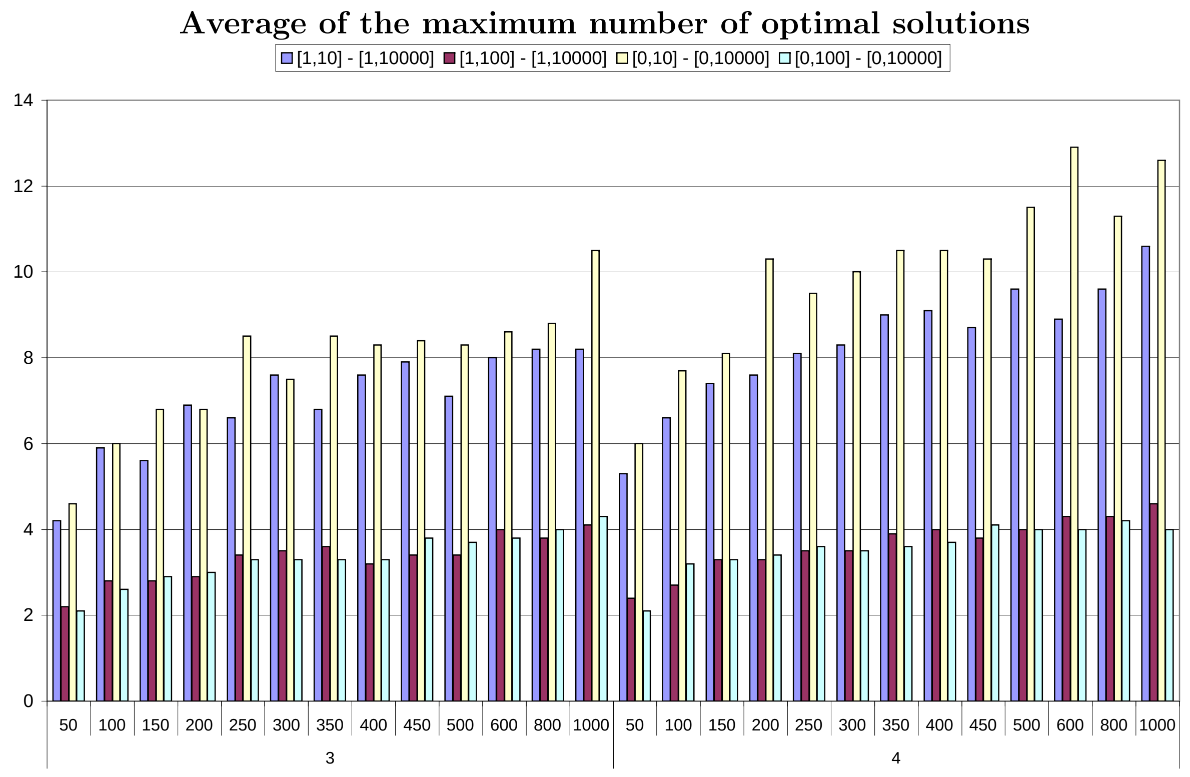


Figure 15: Average of the maximum number of optimal solutions (Set2LC) for $m = 3n, 4n$ and ranges $\bar{\mathcal{E}}$.

one in $G^{(1)}$) for each $p_i^{(1)}$ was obtained using Dijkstra algorithm, so the same procedure must have been used for $p_k^{(1)}$, generating a candidate path pair A_k. If $C(A_k) = c_{opt}$ then $C(A_k) \leq C(\text{top}(S_A))$, for S_A at the time of A_k generation. If $C(A_k) = C(\text{top}(S_A))$, for S_A at the time of A_k generation, than A_k is simply pushed into S_A. If $C(A_k) < C(\text{top}(S_A))$, for S_A at the time of A_k generation, than S_A is cleared and A_k is pushed into S_A, becoming its only element. Other path pairs, A_i, $i = k+1, \cdots, u$, of cost c_{opt} may aftwards be pushed into S_A; A_k will only be removed from S_A if S_A is cleared, and for that to happen a path a $p_i^{(1)}$, $i = k+1, \cdots, u$ would have to be obtained such that a path pair $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$ existed with $C(A_i) < c_{opt}$. But then this would mean that c_{opt} was not the minimal cost of any pair of disjoint paths with dual arc cost in G, and by lemma A.3 this can not be so. Therefore path A_k does not exist, and the lemma must be true. □

Lemma A.5 *If a path $p_j^{(2)}$, $j = 1, 2, \ldots, v$ (where v is the order of the first element of* top(S_B)*) is such that at least a minimal cost disjoint path exists in G, the first element of wchich is $p_j^{(1)}$, then a path pair $B_j = (p(p_j^{(2)})^{(2)}, p^{(2)})$, with $C(B_i) = c_{opt}$ will belong to*

S_B, at the end of step 4.

Proof: The proof is similar to A.4 and therefore will not be repeated here. □

Lemma A.6 *Lets assume that $p^{(1)} = p_u^{(1)}$ was the u-shortest (loopless) path from s to t with respect to $\eta^{(1)}$ obtained in procedure $\mathcal{A}$ and that $p^{(2)} = p_v^{(2)}$ was the v-shortest (loopless) path from s to t with respect to $\eta^{(2)}$ obtained in procedure $\mathcal{B}$, immediately after step 4.*

There may still exist path pairs $(p_i^{(1)}, q(p_i^{(1)})^{(2)})$, $i > u$ and path pairs $(q(p_j^{(2)})^{(1)}, p_j^{(2)})$, $j > v$ with cost equal to the optimal cost c_{opt}.

Let $c_A = \gamma^{(1)}(S_B)$ and $c_B = \gamma^{(1)}(S_A)$. There is no path $p_i^{(1)}$, $C^{(1)}(p_i^{(1)}) > c_A$ such that a disjoint path pair $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$ with $C(A_i) \leq c_{opt}$, might exist. There is no path $p_j^{(2)}$, $C^{(2)}(p_j^{(2)}) > c_B$ such that a disjoint path pair $B_j = (q(p_j^{(2)})^{(1)}, p_j^{(2)})$ with $C(B_j) \leq c_{opt}$, might exist.

Proof: Let the optimal cost be $c_{opt} = C(\text{top}(S_A)) = C(\text{top}(S_B))$ and $c_{opt}^{(1)} = C^{(1)}(p^{(1)})$, $c_{opt}^{(1)} = C^{(2)}(p^{(2)})$ $(c_{opt} = c_{opt}^{(1)} + c_{opt}^{(2)})$, the cost of the first and second elements of the top pairs of stacks S_A and S_B, respectively, immediatly after step 4.

By lemma A.3 the minimal cost is c_{opt} therefore any pair of disjoint paths, that may be generated after step 4 will have cost greater or equal to c_{opt}.

To prove that immediatly after step 4 there may still exist path pairs $(p_i^{(1)}, q(p_i^{(1)})^{(2)})$, $i > u$ and path pairs $(q(p_j^{(2)})^{(1)}, p_j^{(2)})$, $j > v$ with cost equal to the optimal cost c_{opt}, if suffices to consider the following situations:

- there are paths $p_i^{(1)}$, $i > u$, $C^{(1)}(p_i^{(1)}) = c_{opt}^{(1)}$, then it is possible that a path pair $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$, with $C^{(2)}(q(p_i^{(1)})^{(2)}) = c_{opt}^{(2)}$ exists, and therefore $C(A_i) = c_{opt}$.

- there are paths $p_j^{(2)}$, $j > v$, $C^{(2)}(p_j^{(2)}) = c_{opt}^{(2)}$, then it is possible that a path pair $B_j = (q(p_j^{(2)})^{(1)}, p_j^{(2)})$, with $C^{(1)}(q(p_j^{(2)})^{(1)}) = c_{opt}^{(1)}$ exists, and therefore $C(B_j) = c_{opt}$.

To prove that there is no path $p_i^{(1)}$, $C^{(1)}(p_i^{(1)}) > c_A$ such that a disjoint path pair $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$ with $C(A_i) \leq c_{opt}$, might exist, we must first recall that by lemma A.3 $C(A_i) \geq c_{copt}$. If $C(A_i) > c_{copt}$ then A_i does not belong to the optimal set. If $C(A_i) = c_{copt}$ and $C^{(1)}(p_i^{(1)}) > c_A \geq c_{opt}^{(1)}$, then $C^{(2)}(q(p_i^{(1)})^{(2)}) < c_{opt} - c_A$ and $C^{(2)}(q(p_i^{(1)})^{(2)}) < c_{opt}^{(2)}$. If path pair A_i existed, then in S_B there should exist a path pair $B_j = (q(p_j^{(2)})^{(1)}, p_j^{(2)})$, with

$j < v$, such that $p_j^{(2)} = q(p_i^{(1)})^{(2)}$, and $C^{(1)}(q(p_j^{(2)})^{(1)}) > c_A$, which by definition of c_A is impossible. So the path A_i can not exist.

Similarly, to prove that there is no path $p_j^{(2)}$, $C^{(2)}(p_j^{(2)}) > c_B$ such that a disjoint path pair $B_j = (q(p_j^{(2)})^{(1)}, p_j^{(2)})$ with $C(B_j) \leq c_{opt}$, might exist, we must first recall that by lemma A.3 $C(B_j) \geq c_{copt}$. If $C(B_j) > c_{copt}$ then B_j does not belong to the optimal set. If $C(B_j) = c_{copt}$ and $C^{(2)}(p_j^{(2)}) > c_B \geq c_{opt}^{(2)}$, then $C^{(1)}(q(p_j^{(2)})^{(1)}) < c_{opt} - c_B$ and $C^{(1)}(q(p_j^{(2)})^{(1)} < c_{opt}^{(1)}$. If path pair B_j existed, then in S_A there should exist a path pair $A_i = (p_i^{(1)}, q(p_i^{(1)})^{(2)})$, with $i < u$, such that $p_i^{(1)} = q(p_j^{(2)})^{(1)}$, and $C^{(2)}(q(p_j^{(1)})^{(2)}) > c_B$, which by definition of c_B is impossible. So the path B_j can not exist. $\square$

B Size of the stacks immediatly after step 4 of algorithm Set2LC

At the beginning of execution of step 4, stack S_A and S_B each have a single element, and two possibilities may occur:

1. If $C(\text{top}(S_A)) = C(\text{top}(S_B))$ then three possibilities (mutually exclusive) may arise:
   (a) If $C(\text{top}(S_A)) = C(\text{top}(S_B))$ and $C^{(1)}(p^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})$, then condition expressed by (5 is true (and step 4 ends).

   If S_A and S_B contain minimal cost pairs of disjoint paths (from steps 1 and 2) it has already been shown that $c_{opt} = C(\text{top}(S_A)) = C(\text{top}(S_B))$.

   If S_A and S_B have been found using procedure $\mathcal{A}$ (step 4(a)i) and $\mathcal{B}$ (step 4(a)ii), respectively, then as shown in lemma A.3 minimal cost pairs of disjoint paths have been found.

   In this case at the end of step 4 both stacks S_A and stack S_B will have a single element each.

   (b) If $C^{(1)}(p^{(1)}) < C^{(1)}(q(p^{(2)})^{(1)})$ two possibilities may still occur:
      i. If $C^{(1)}(p^{(1)}) < C^{(1)}(q(p^{(2)})^{(1)})$ but $C(\text{top}(S_A))$ (equal to $C(\text{top}(S_B)))$ is the minimal path pair cost (although not yet confirmed), then procedure $\mathcal{A}$ will not be able to improve $C(S_A)$.

Procedure $\mathcal{A}$ will generate path pairs, and if their cost is equal to $C(\text{top}(S_A))$ they will be pushed into S_A until a path pair A_u such that $C(\text{top}(S_A)) = C(A_u)$ and $C^{(1)}(p_u^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})$ is obtained and pushed into stack S_A, resulting in $C^{(1)}(p^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})$.

In this case at the end of step 4 stack S_A will have one or more elements (all with minimal cost), and stack S_B will have a single element.

ii. If $C^{(1)}(p^{(1)}) < C^{(1)}(q(p^{(2)})^{(1)})$ but $C(\text{top}(S_A))$ (equal to $C(\text{top}(S_B))$) is not the minimal cost of the pairs of the sought set, then procedure $\mathcal{A}$ will be able to improve $C(\text{top}(S_A))$.

Paths will be generated until a path pair A_u is found, with $C(A_u) < C(\text{top}(S_B))$ When such a path is found stack S_A is cleared and the recently found pair becomes its only element.

At this point stack S_A has one element, stack S_B has one element and $C(\text{top}(S_A)) < C(\text{top}(S_B))$.

From the point of view of the stack size evolution, the situation is similar to step 4 starting point when stack S_A and stack S_B have each a single element (the first pairs) and $C(\text{top}(S_A)) < C(\text{top}(S_B))$.

(c) If $C^{(2)}(p^{(2)}) < C^{(2)}(q(p^{(1)})^{(2)})$, a similar argument as for case 1b can be made. Again two cases may occur:

i. If $C^{(2)}(p^{(2)}) < C^{(2)}(q(p^{(1)})^{(2)})$ but $C(\text{top}(S_B))$ (equal to $C(\text{top}(S_A))$) is the minimal path pair cost (although not yet confirmed), then procedure $\mathcal{B}$ will not be able to improve $C(S_B)$.

Procedure $\mathcal{B}$ will generate path pairs, and if their cost is equal to $C(\text{top}(S_B))$ they will be pushed into S_B until a path pair B_v such that $C(\text{top}(S_B)) = C(B_v)$ and $C^{(1)}(p_v^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})$ is obtained and pushed into stack S_B, resulting in $C^{(1)}(p^{(1)}) = C^{(1)}(q(p^{(2)})^{(1)})$.

In this case at the end of step 4 stack S_B will have one or more elements (all with minimal cost), and stack S_A will have a single element.

ii. If $C^{(2)}(p^{(2)}) < C^{(2)}(q(p^{(1)})^{(2)})$ but $C(\text{top}(S_B))$ (equal to $C(\text{top}(S_A))$) is not the minimal cost of the pairs of the sought set, then procedure $\mathcal{B}$ will be able to improve $C(\text{top}(S_B))$. Paths will be generated until a path pair B_v

is found, with $C(B_v) < C(\text{top}(S_A))$

When such a path is found stack S_B is cleared and the recently found pair becomes its only element.

At this point stack S_B has one element, stack S_A has one element and $C(\text{top}(S_B)) < C(\text{top}(S_A))$.

From the point of view of the stack size evolution, the situation is similar to step 4 starting point when stack S_A and stack S_B have each a single element (the first pairs) and $C(\text{top}(S_B)) < C(\text{top}(S_A))$

2. If $C(\text{top}(S_B)) \neq C(\text{top}(S_A))$, only two (symmetrical) possibilities exist:

   (a) If $C(\text{top}(S_B)) < C(\text{top}(S_A))$ then 4(a)i will iterate until a path pair A_u is found such that $C(A_u) \leq C(\text{top}(S_B))$; when that happens, S_A is cleared and A_u becomes is only element.

   At this point stacks S_A and S_B have each one element:

      i. If $C(\text{top}(S_B)) > C(\text{top}(S_A))$, step 4(a)i ends and procedure $\mathcal{B}$ (in step 4(a)ii) will improve S_B.

      From the point of view of the stack size evolution, the situation is similar to step 4 starting point when stack S_A and stack S_B have each a single element (the first pairs) and $C(\text{top}(S_B)) > C(\text{top}(S_A))$.

      ii. If $C(\text{top}(S_B)) = C(\text{top}(S_A))$, then the algorithm will remain in procedure $\mathcal{A}$ if $C^{(1)}(p^{(1)}) < C^{(1)}(q(p^{(2)})^{(1)})$, and everything will happen as explained in 1b.

      If $C(\text{top}(S_B)) = C(\text{top}(S_A))$, but $C^{(2)}(p^{(2)}) < C^{(2)}(q(p^{(1)})^{(2)})$, step 4(a)i ends and the algorithm will proceed to step 4(a)i, where procedure $\mathcal{B}$ will try to improve $C(\text{top}(S_B))$, and everything will happen as explained in 1c.

      From the point of view of the stack size evolution, the situation is similar to step 4 starting point when stack S_A and stack S_B have each a single element (the first pairs) and $C(\text{top}(S_B)) = C(\text{top}(S_A))$.

   (b) If $C(\text{top}(S_B)) > C(\text{top}(S_A))$ then 4(a)ii will iterate until a path pair B_v is found such that $C(B_v) \leq C(\text{top}(S_A))$; when that happens, S_B is cleared and B_u becomes is only element.

The rest of the explanation is similar to 2a

We have shown that at the beginning of step 5 (after step 4) two possibilities exist:

1. The first path pairs found in steps 1 to 3 were optimal (see lemma A.1), and stacks A and B have each a single element.

2. The first path pairs found in steps 1 to 3 might not have been optimal, and step 4 was executed until the condition expressed by (5) became true.

   In this case one of the stacks will have a single element and the other will have one or more elements and all pairs in both stacks will have cost c_{opt}, by lemma A.3.

C Additional set of results for directed networks and DP2LC

In this appendix, the full set of results obtained using algorithm DP2LC, is presented. Only a sub-set of these resutls was used in the first part of this report.

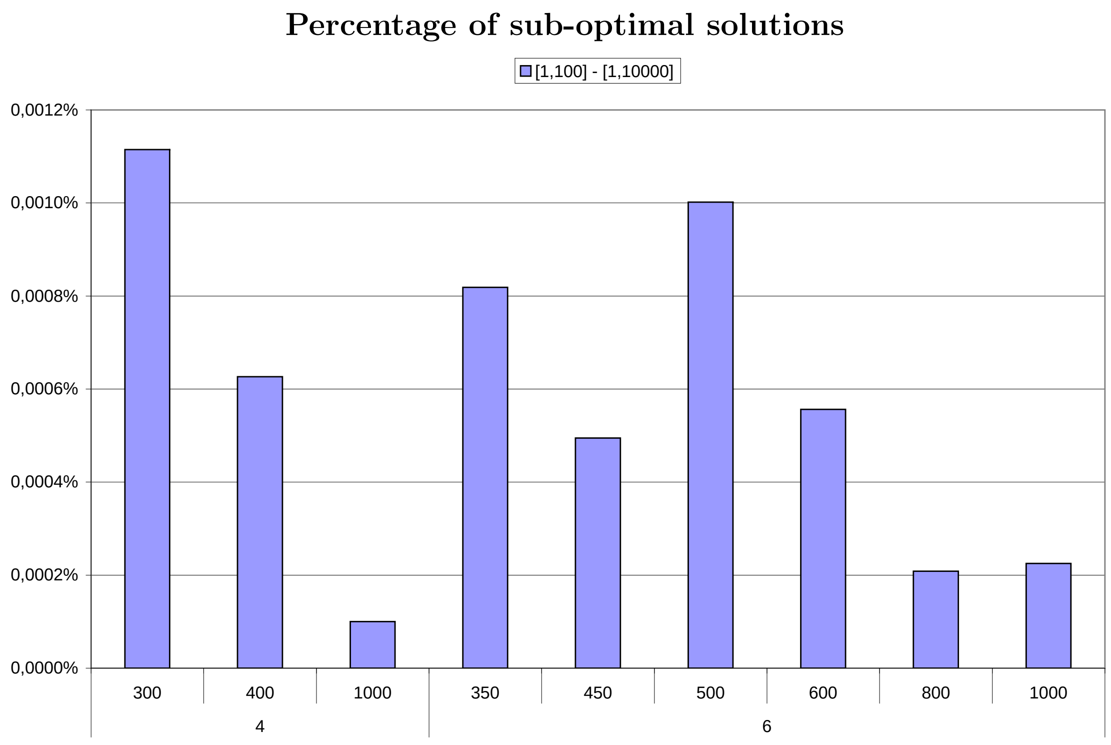


Figure 16: Percentage of sub-optimal solutions (DP2LC) for networks with sub-optimal solutions for $m = 4n, 6n$, and ranges $\bar{\mathcal{Z}}\bar{\mathcal{E}}$.

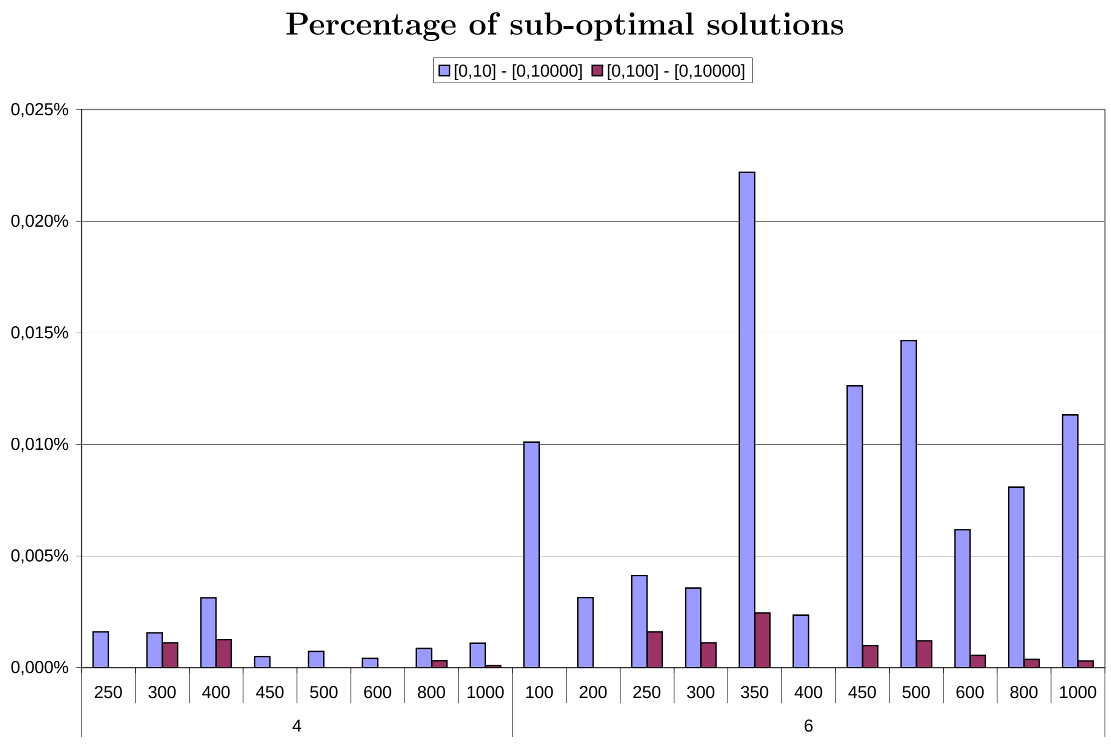


Figure 17: Percentage of sub-optimal solutions (DP2LC), for networks with sub-optimal solutions for $m = 4n, 6n$ and ranges $\mathcal{Z}\bar{\mathcal{E}}$.

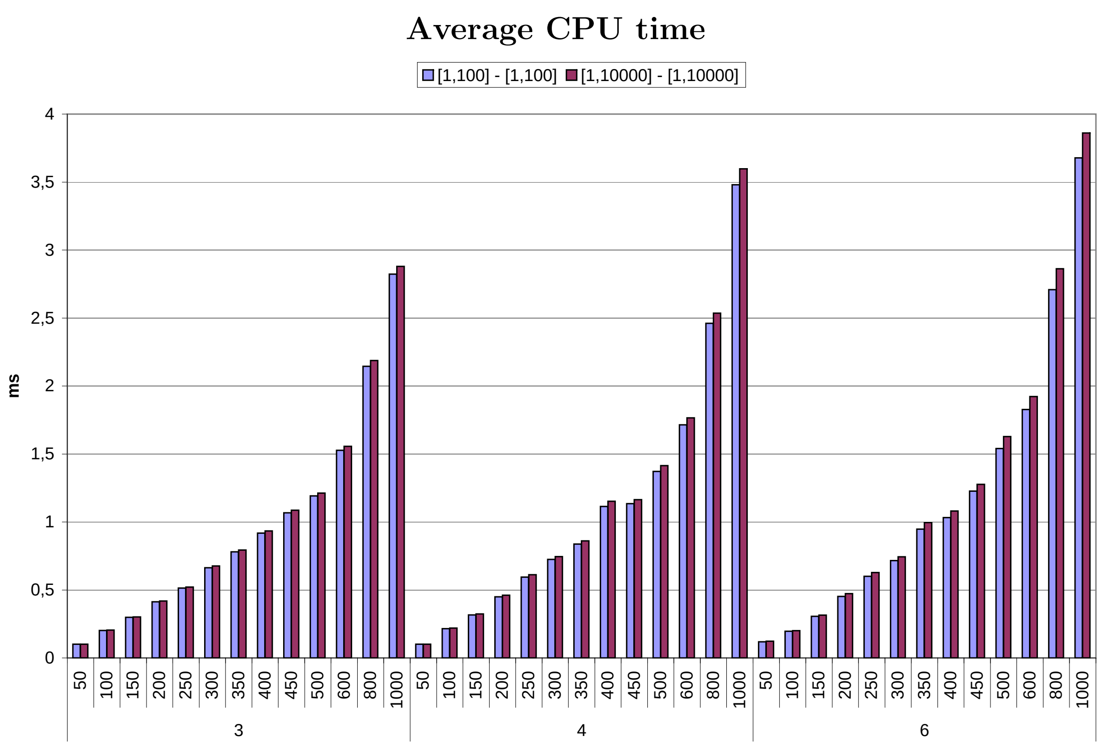


Figure 18: Average CPU time (DP2LC) per node pair (all solutions were optimal solutions) for $m = 3n, 4n, 6n$ and ranges $\bar{\mathcal{Z}}\mathcal{E}$.

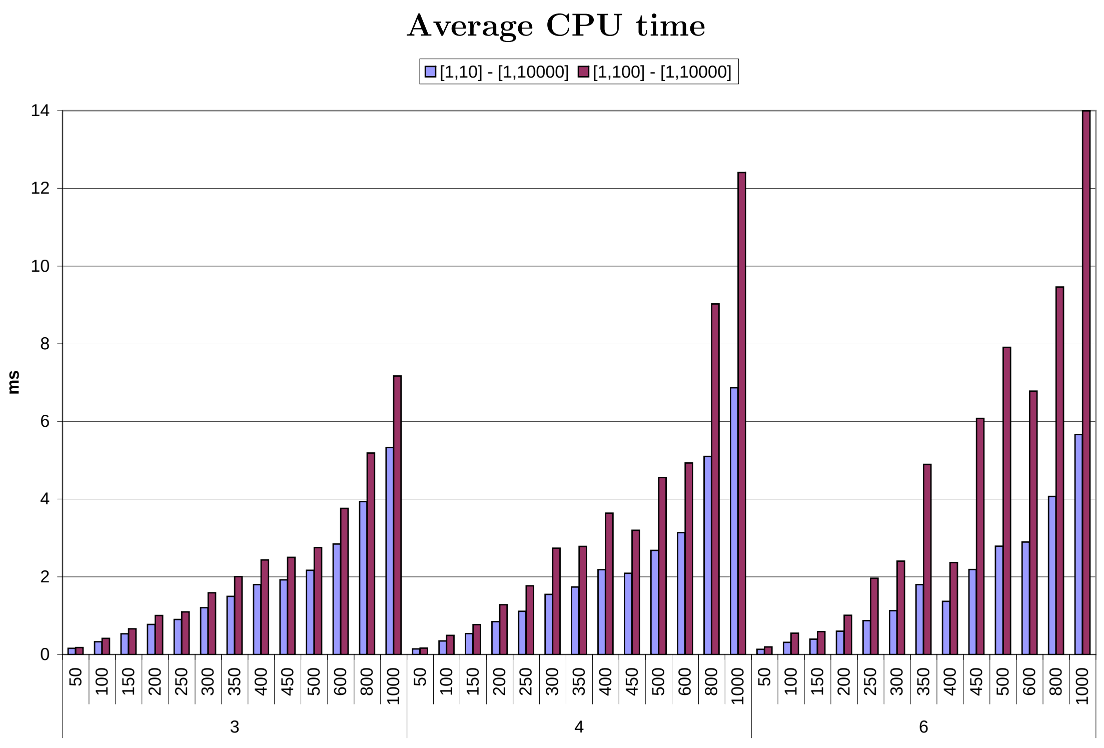


Figure 19: Average CPU time (DP2LC) per node pair (with optimal or sub-optimal solutions) for $m = 3n, 4n, 6n$ and ranges $\bar{\mathcal{Z}}\bar{\mathcal{E}}$.

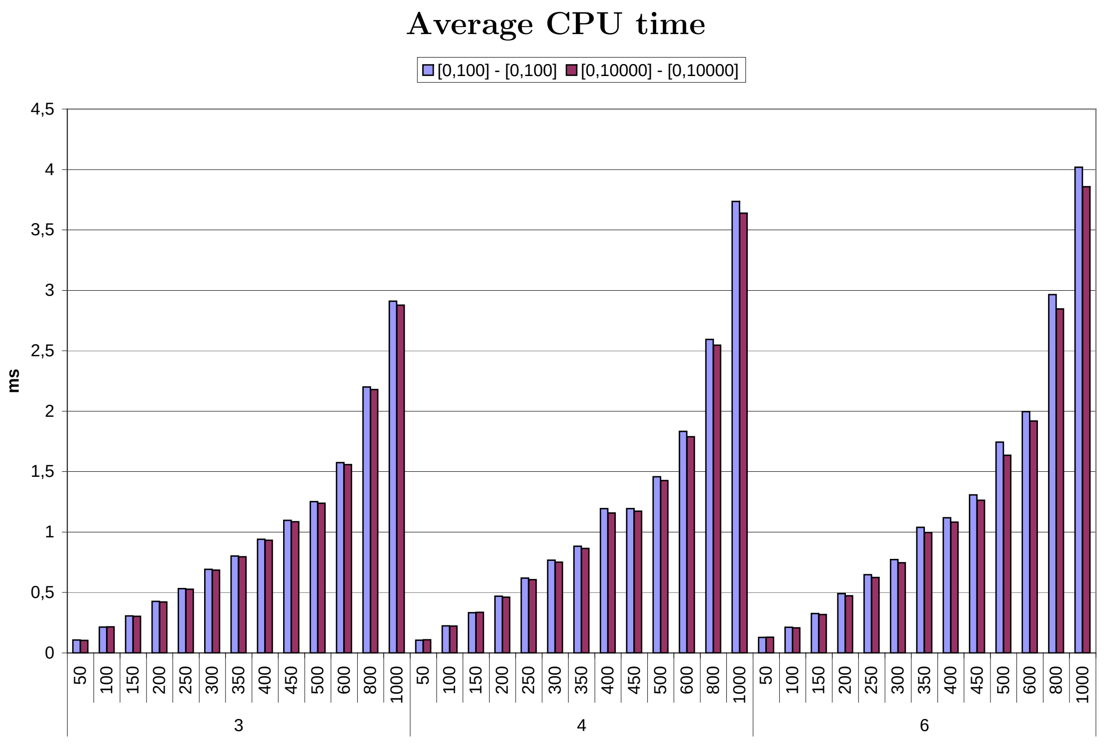


Figure 20: Average CPU time (DP2LC) per node pair (with optimal or sub-optimal solutions) for $m = 3n, 4n, 6n$ and ranges $\mathcal{Z}\mathcal{E}$.

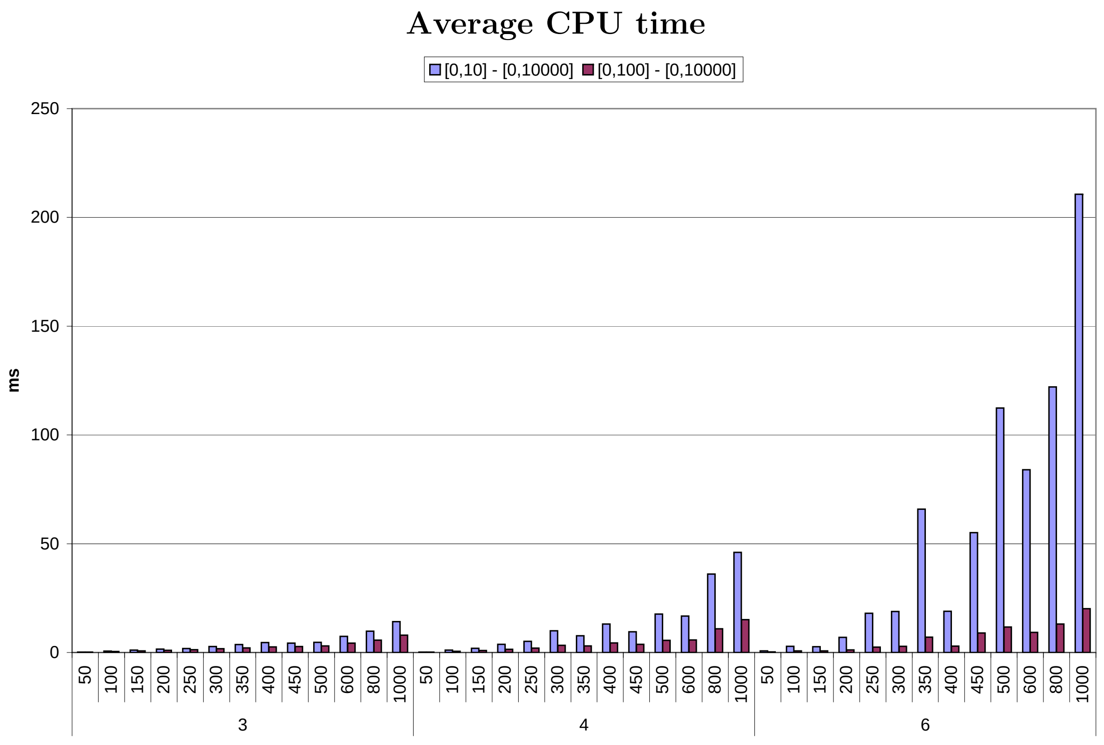


Figure 21: Average CPU time (DP2LC) per node pair (all solutions were optimal solutions) for $m = 3n, 4n, 6n$ and ranges $\mathcal{Z}\bar{\mathcal{E}}$.

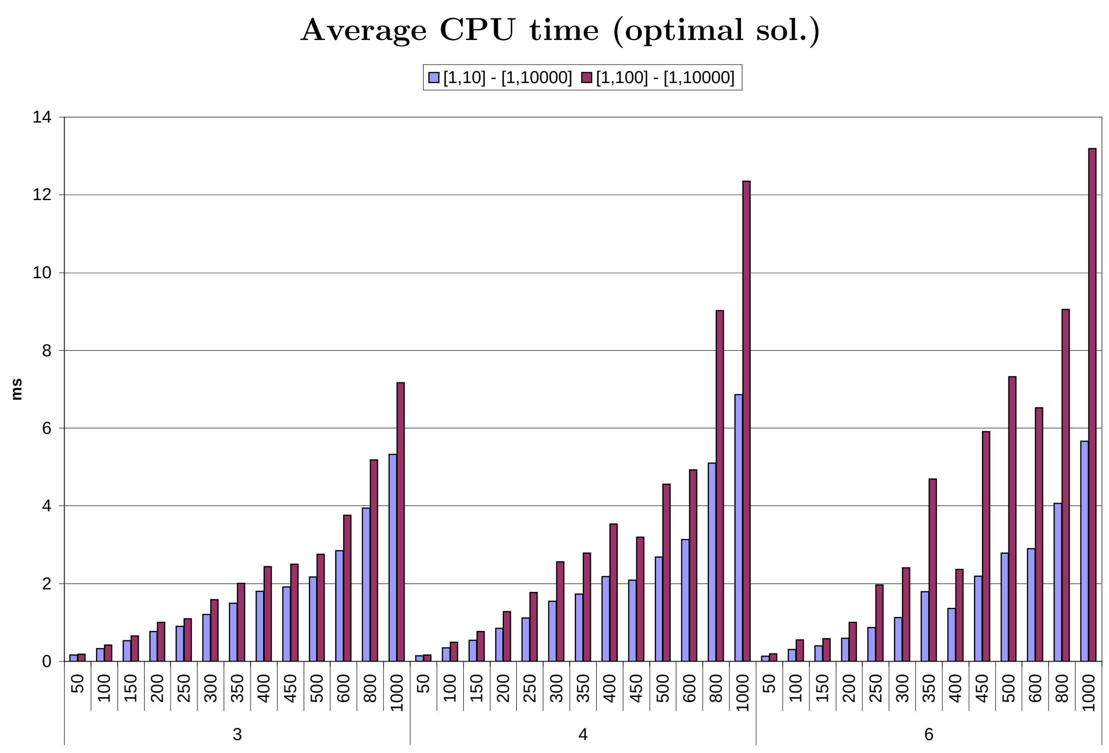


Figure 22: Average CPU time (DP2LC) per node pair with an optimal solution for $m = 3n, 4n, 6n$ and ranges $\bar{\mathcal{Z}}\bar{\mathcal{E}}$.

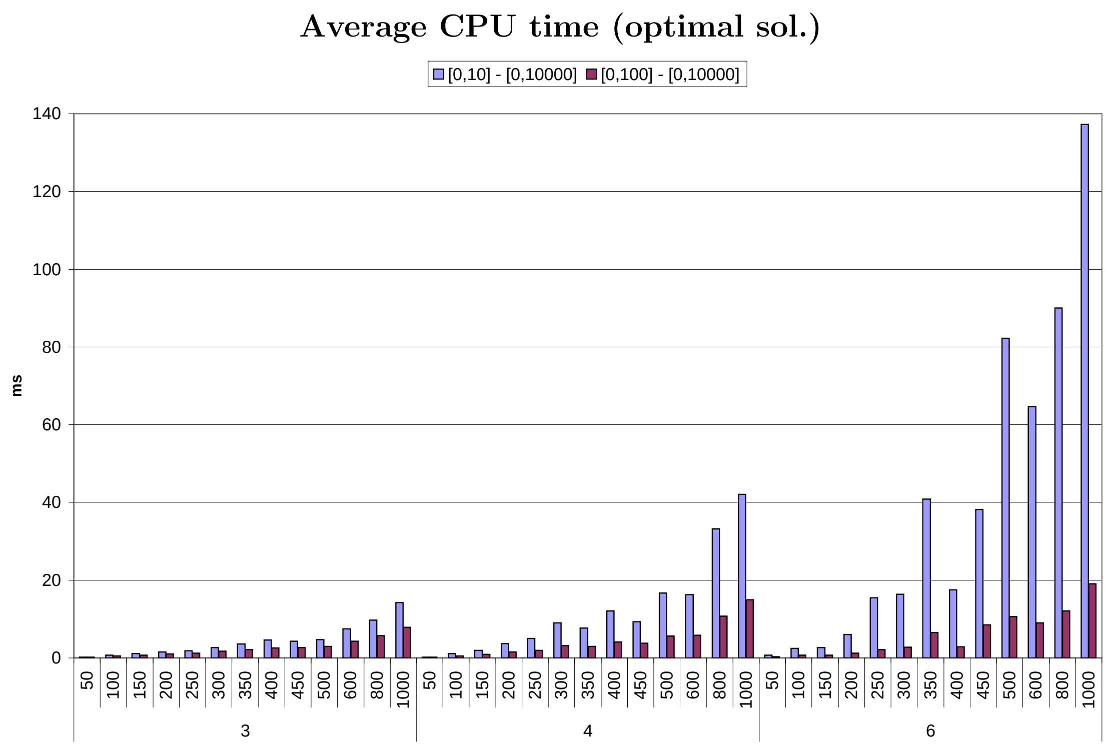


Figure 23: Average CPU time (DP2LC) per node pair with an optimal solution for $m = 3n, 4n, 6n$ and ranges $\mathcal{Z}\bar{\mathcal{E}}$.

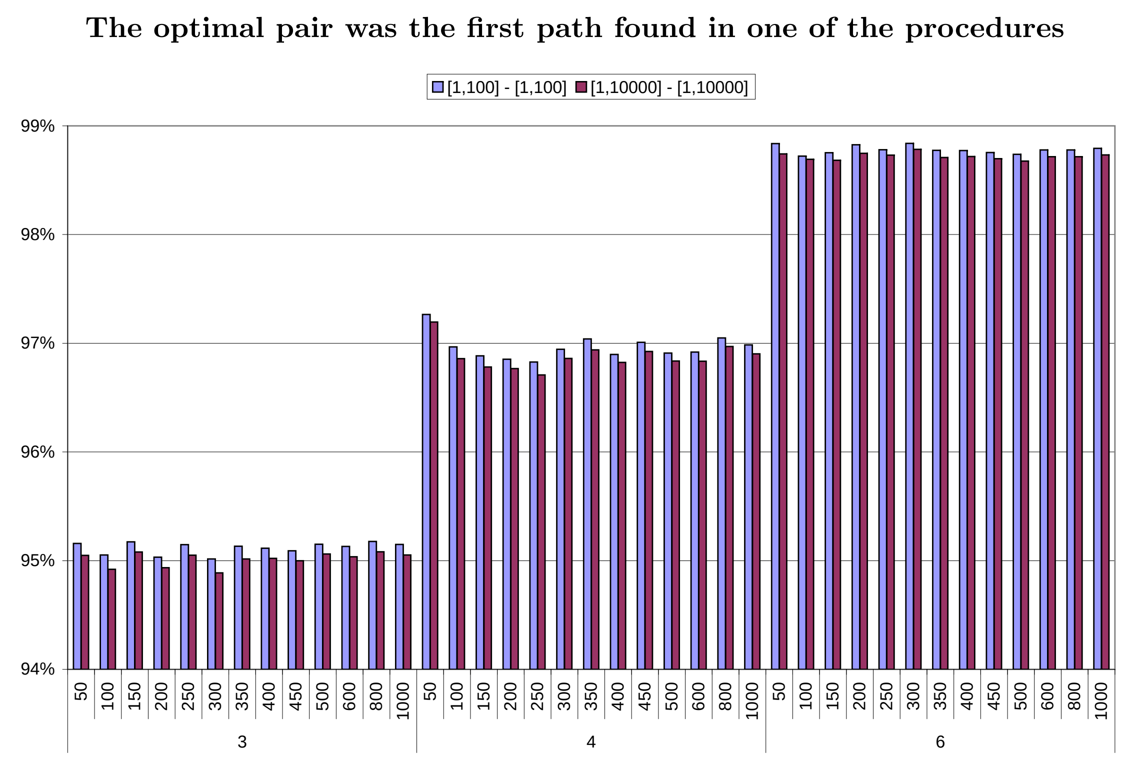


Figure 24: Frequency of occurrence of the optimal pair (DP2LC) among the first identified pair in procedures $\mathcal{A}$ or $\mathcal{B}$, for $m = 3n, 4n, 6n$ and ranges $\bar{\mathcal{Z}}\mathcal{E}$.

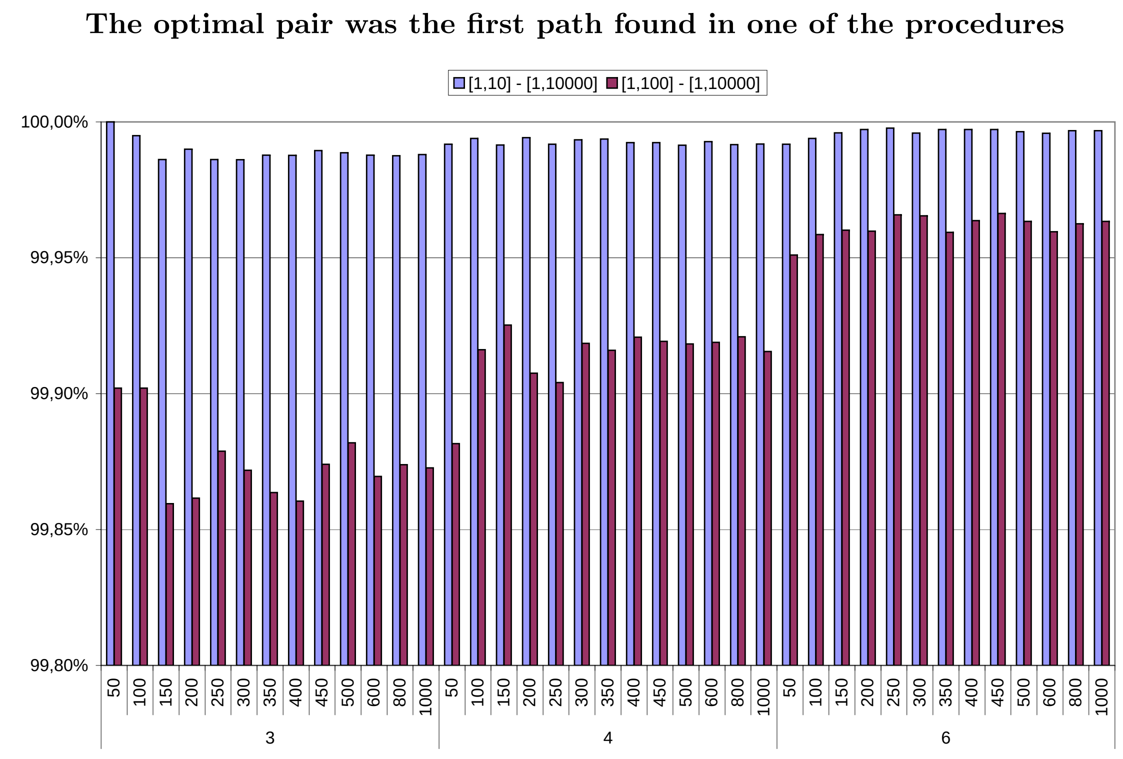


Figure 25: Frequency of occurrence of the optimal pair (DP2LC) among the first identified pair in procedures $\mathcal{A}$ or $\mathcal{B}$, for for $m = 3n, 4n, 6n$ and ranges $\bar{\mathcal{Z}}\bar{\mathcal{E}}$.

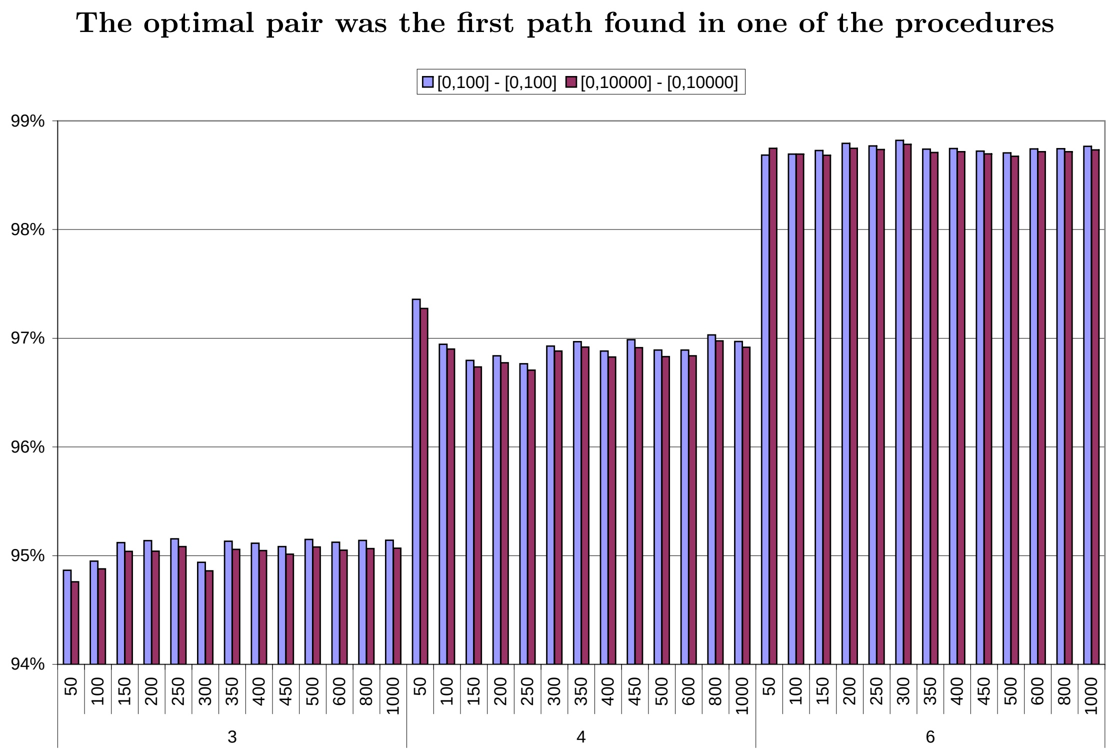


Figure 26: Frequency of occurrence of the optimal pair (DP2LC) among the first identified pair in procedures $\mathcal{A}$ or $\mathcal{B}$, for $m = 3n, 4n, 6n$ and ranges $\mathcal{ZE}$.

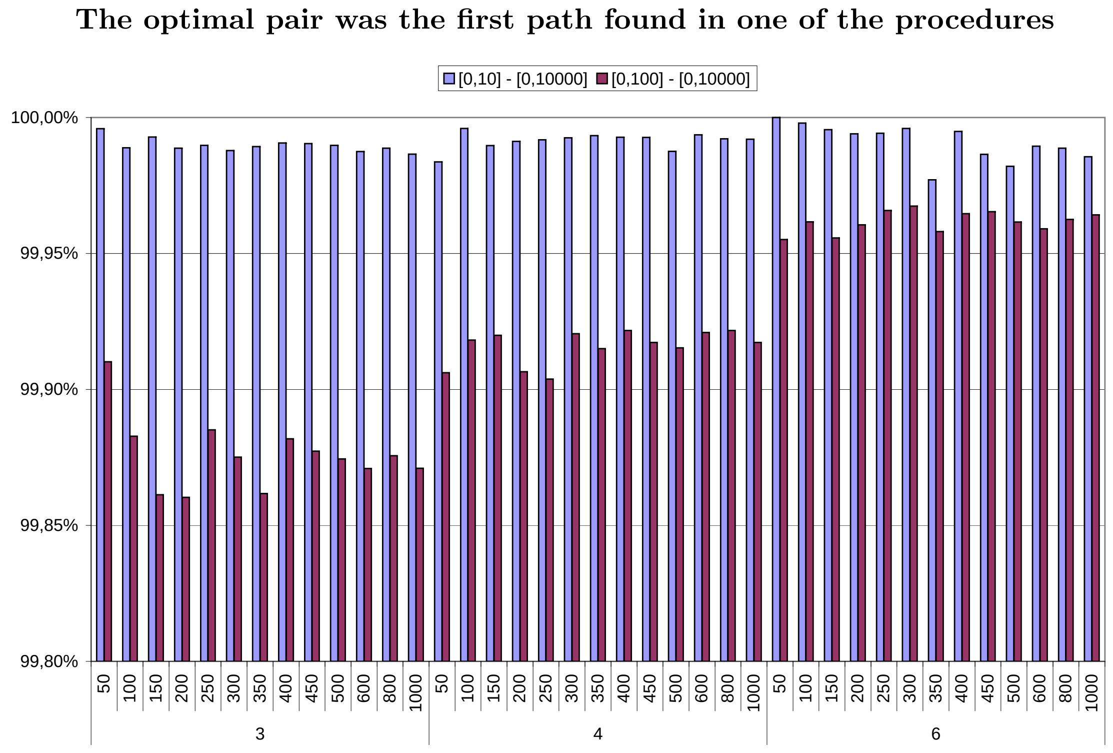


Figure 27: Frequency of occurrence of the optimal pair (DP2LC) among the first identified pair in procedures $\mathcal{A}$ or $\mathcal{B}$, for $m = 3n, 4n, 6n$ and ranges $\mathcal{Z}\bar{\mathcal{E}}$.

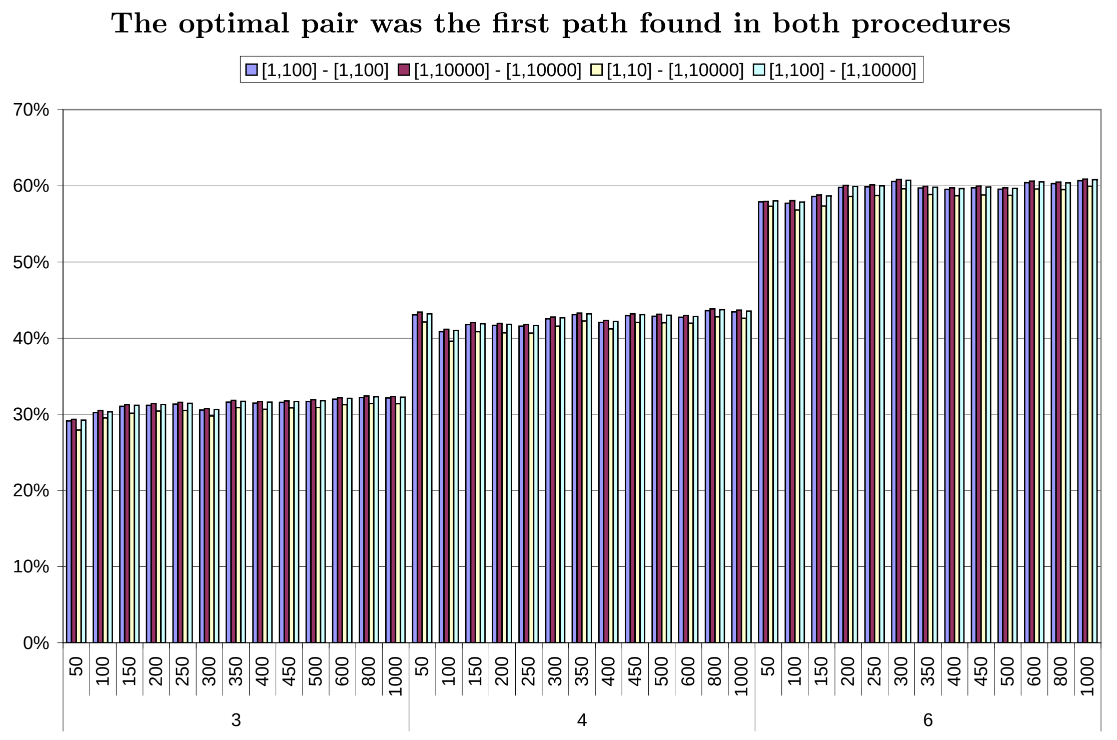


Figure 28: Frequency of occurrence of the optimal pair (DP2LC) among the first identified pair in both procedures $\mathcal{A}$ and $\mathcal{B}$, for $m = 3n, 4n, 6n$ and ranges $\bar{\mathcal{Z}}$.

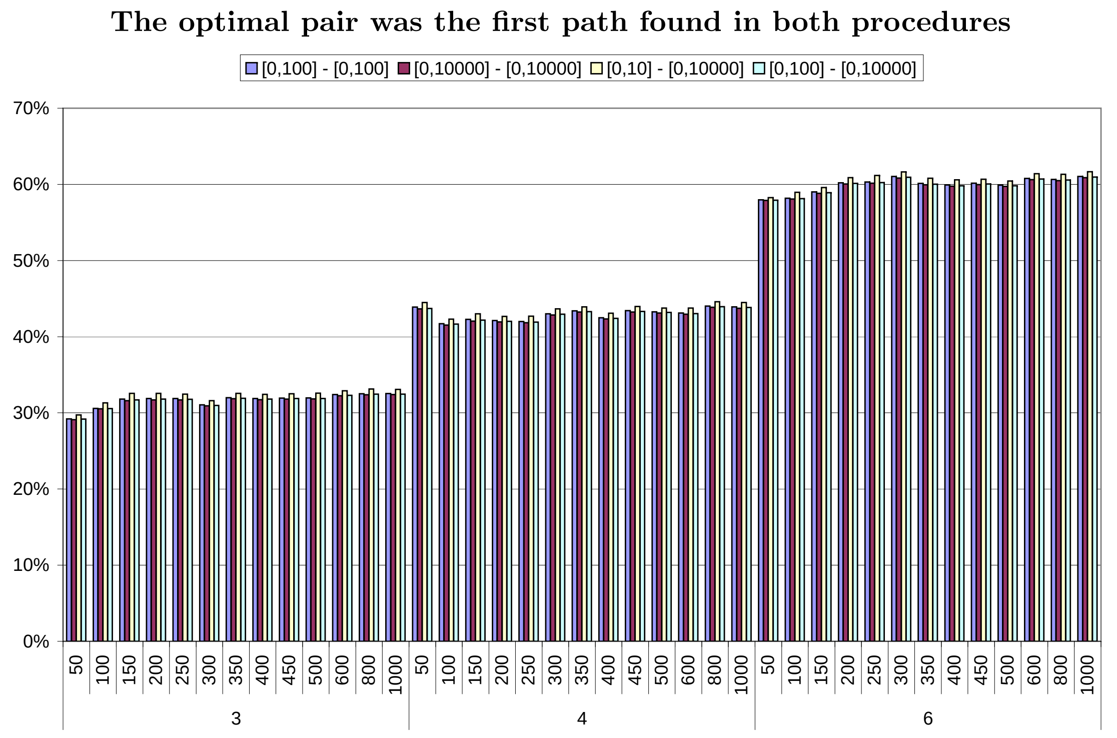


Figure 29: Frequency of occurrence of the optimal pair (DP2LC) among the first identified pair in both procedures $\mathcal{A}$ and $\mathcal{B}$, for $m = 3n, 4n, 6n$ and ranges $\mathcal{Z}$.

References

[1] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications.* Springer Monographs in Mathematics. Springer-Verlag, May 2002.

[2] D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall, 1992.

[3] T. Gomes, L. Martins, and J. Craveirinha. An algorithm for calculating the k shortest paths with a maximum number of arcs. *Investigação Operacional*, 21(2):235–244, 2001.

[4] P.-H. Ho, J. Tapolcai, and H. T. Mouftah. On achieving optimal survivable routing for shared protection in survivable next-generation internet. *IEEE Transactions on Reliability*, 53(2):216–225, June 2004.

[5] M. Kodialam and T. V. Lakshman. Dynamic routing of restorable bandwidth-guaranteed tunnels using aggregated network resource usage information. *IEEE/ACM Transactions on Networking*, 11(3):399–410, June 2003.

[6] P. Laborczi, J. Tapolcai, P.-H. Ho, T. Cinkler, A. Recski, and H. T. Mouftah. Algorithms for asymmetrically weighted pair of disjoint paths in survivable networks. In T. Cinkler, editor, *Proceedings of Design of Reliable Communication Networks (DCRN 2001)*, pages 220–227, October 7-10 2001.

[7] C. L. Li, S. T. McCormick, and D. Simchi-Levi. The complexity of finding two disjoint paths with min-max objective function. *Discrete Applied Mathematics*, 26(1):105–115, 1990.

[8] C.-L. Li, S. T. McCormick, and D. Simchi-Levi. Finding disjoint paths with different path costs: complexity and algorithms. *Networks*, 22:653–667, 1992.

[9] E. Martins and M. Pascoal. A new implementation of Yen's ranking loopless paths algorithm. *4OR – Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2):121–134, 2003.

[10] E. Martins, M. Pascoal, and J. Santos. An algorithm for ranking loopless paths. Technical Report 99/007, CISUC, 1999. http://www.mat.uc.pt/˜marta/Publicacoes/mps2.ps.

[11] E. Martins, M. Pascoal, and J. Santos. Deviation algorithms for ranking shortest paths. *International Journal of Foundations of Computer Science*, 10(3):247–263, 1999.

[12] H. T. Mouftah and P.-H. Ho. *Optical networks – arquitecture and survivability.* Kluwer Academic Publishers, 2003.

[13] J. W. Suurballe. Disjoint paths in networks. *Networks*, 4:125–145, 1974.

[14] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.

[15] D. Xu, Y. Chen, Y. Xiong, C. Qiao, and X. He. On finding disjoint paths in single and dual link cost networks. In *IEEE INFOCOM 2004*. IEEE, 2004.

[16] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, July 1971.