

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

Leizer Lima Pinto and Marta Pascoal

**Enhanced algorithms for tricriteria shortest path problems with two bottleneck  
objective functions**

No. 3

2009

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](http://www.inescc.pt)

# ENHANCED ALGORITHMS FOR TRICRITERIA SHORTEST PATH PROBLEMS WITH TWO BOTTLENECK OBJECTIVE FUNCTIONS

LEIZER LIMA PINTO<sup>(1,2)</sup> and MARTA M. B. PASCOAL<sup>\*(3,4)</sup>

<sup>(1)</sup>COPPE/UFRJ - Federal University of Rio de Janeiro, Brazil  
Department of Systems Engineering and Computer Science  
E-mail: leizer@cos.ufrj.br

<sup>(2)</sup>Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

<sup>(3)</sup>Departamento de Matemática da Universidade de Coimbra  
Apartado 3008, EC Universidade, 3001-454 Coimbra, Portugal  
E-mail: marta@mat.uc.pt

<sup>(4)</sup>Institute for Systems and Computers Engineering – Coimbra (INESCC)

March 2009

---

## Abstract

This paper deals with a tricriteria path problem involving two bottleneck objective functions and a cost. It presents two methods for computing shortest paths in subnetworks, obtained by restricting the set of arcs according to the bottleneck values in order to find the minimal complete set of Pareto-optimal solutions. These procedures are enhanced by using the objective values of the determined shortest paths to reduce the number of considered subnetworks, and thus the number of solved shortest path problems. Two algorithms are introduced, evaluated and compared with the previous literature. Results for random instances with 5 000 nodes, an average degree of 100 and 200 distinct bottleneck values show that one of them solves 15 times fewer shortest path problems than before, and the other 43 times fewer, while the CPU times are improved 11 and 29 times, respectively. On average the new algorithms computed the minimal complete set in 7 000 node networks with 200 bottleneck values in less than four minutes. Moreover two variants of these methods are introduced. Their aim is to choose the solutions with the best bottleneck values when the cost is the same. For random problems this leads to a maximum improvement of 30% and 9% for costs in [1,10] and [1,100], respectively.

**Keywords:** Tricriteria path problem, cost function, bottleneck function, Pareto-optimal solution.

---

## 1 Introduction

This paper deals with path problems involving two bottleneck functions, either of MaxMin or of MinMax type, and one additive cost function. Polynomial algorithms for path problems involving cost and bottleneck functions, have been presented by Hansen [3], Martins [4] and Berman *et al.* [1]. More recently the tricriteria path problem with a cost function and two bottleneck functions was analyzed by Pinto *et al.* [6, 7]. In these cases the goal is the generation of a set of paths, all

---

\*Corresponding author.

having Pareto-optimal objective values. The finite number of values that a bottleneck function can have yields algorithms with polynomial complexity order. The focus of this paper is to describe two algorithms for improving the method introduced in [6] and modified later [7]. We start by introducing some notation and by formulating the problem itself.

Consider a graph  $G = (\mathcal{N}, \mathcal{A})$  with  $|\mathcal{N}| = n$  and  $|\mathcal{A}| = m$ . For each arc  $(i, j) \in \mathcal{A}$ , let  $c_{ij}^k \in \mathbb{R}$  be its weights,  $k = 1, 2, 3$ . Given initial and terminal nodes in  $\mathcal{N}$ ,  $s$  and  $t$ , let  $p = \langle i_1 = s, i_2, \dots, i_\ell = t \rangle$ , with  $(i_k, i_{k+1}) \in \mathcal{A}$  for  $k = 1, \dots, \ell - 1$ , denote a path in  $G$ . For simplicity we write  $i \in p$  if  $i$  is a node in the sequence  $p$ , and  $(i, j) \in p$  if  $i$  immediately precedes  $j$  in  $p$ . Let  $\mathcal{P}$  stand for the set of paths from  $s$  to  $t$  in  $G$ .

As mentioned above we deal with two bottleneck functions and one cost function, therefore the *objective vector* associated with path  $p$  is given by  $c(p) = (c_1(p), c_2(p), c_3(p)) \in \mathbb{R}^3$ , where

$$c_1(p) = \max_{(i,j) \in p} \{c_{ij}^1\}, \quad c_2(p) = \max_{(i,j) \in p} \{c_{ij}^2\} \quad \text{and} \quad c_3(p) = \sum_{(i,j) \in p} c_{ij}^3.$$

For the sake of simplicity the bottleneck functions are considered as of MinMax type, yet there is no loss of generality in doing so. The tricriteria shortest path problem with two bottleneck functions (TSPPB) is then defined as

$$\min\{c(p) : p \in \mathcal{P}\}.$$

It is said that  $p \in \mathcal{P}$  *dominates*  $\bar{p} \in \mathcal{P}$  when  $c(p) \leq c(\bar{p})$  and  $c(p) \neq c(\bar{p})$ . A path  $p \in \mathcal{P}$  is *Pareto-optimal* if it is not dominated by any other path in  $\mathcal{P}$ . A set  $\mathcal{P}^* \subseteq \mathcal{P}$  of Pareto-optimal solutions is a *minimal complete set* if for each  $p, q \in \mathcal{P}^*$  we have  $c(p) \neq c(q)$  and for any Pareto-optimal solution  $p^* \in \mathcal{P}$  there exists  $\bar{p} \in \mathcal{P}^*$  so that  $c(p^*) = c(\bar{p})$ .

This paper contains three other sections. The next one introduces two algorithms to compute a minimal complete set as well as the use of a shortest path method that avoids ties in the costs in order to reduce the number of subroutine calls. Section 3 presents an algorithmic analysis. Finally, Section 4 reports and discusses computational results.

## 2 Algorithms for the TSPPB

The methods initially proposed to find the minimal complete set for the TSPPB, first MMS and later MMS-R, [6, 7], use the fact that fixing bounds on the two bottleneck functions,  $c_1, c_2$ , produces a subgraph of  $G$ . Therefore, Pareto-optimal solutions for this problem are amongst the shortest paths in each of these subgraphs. MMS determines the shortest path in subgraphs obtained by fixing all possible combinations of bounds on  $c_1$  and  $c_2$ . Following that procedure but fixing the bounds by decreasing order allows algorithm MMS-R to skip infeasible subproblems. In this section two new algorithms are proposed for the TSPPB. They are still based on the computation of shortest paths in subgraphs of  $G$ , but the number of subproblems to be solved is reduced by considering the objective values of the solutions that are obtained. A modification of these methods aimed at decreasing the number of shortest path problems is also proposed.

Let  $m_k$  be the number of different values of  $c_{ij}^k$ ,  $(i, j) \in \mathcal{A}$ , and suppose these are arranged in decreasing order, i.e.,  $c_1^k > c_2^k > \dots > c_{m_k}^k$ ,  $k = 1, 2$ . Given  $v = (v_1, v_2)$ , with  $v_k \in \{1, \dots, m_k\}$  and  $k = 1, 2$ , considering the subset of arcs

$$\mathcal{A}_v = \{(i, j) \in \mathcal{A} : c_{ij}^k \leq c_{v_k}^k, \quad k = 1, 2\},$$

the subgraph of  $G$ ,  $G_v = (\mathcal{N}, \mathcal{A}_v)$ , can be defined.

Because our purpose is to find a minimal complete set, that is, one solution for each Pareto-optimal triple of objective values, there is at most one solution to be considered for each pair

$(v_1, v_2)$  of  $(c_1, c_2)$  values. If such a solution exists it is the shortest path in the subgraph defined by  $(v_1, v_2)$ . The presentation is simplified if the set of subgraphs  $G_v$  is represented as an  $m_1 \times m_2$  matrix, denoted by  $C$  and used to store the several obtained shortest paths.

The two methods below work in two phases. One solves shortest path problems and stores the solutions at a certain position of  $C$ . This phase is followed by another, common to both methods, for filtering possible dominated or equivalent solutions in  $C$ .

**Stair method** As mentioned above, in the previous algorithms all the possible combinations of  $c_{ij}^1$  and  $c_{ij}^2$ , for any  $(i, j) \in \mathcal{A}$ , were considered. The aim of this new version is to use the objective values of the computed shortest paths, in order to skip some subproblems, i.e., some positions in matrix  $C$ . A similar idea was exploited in [4] when dealing with only two criteria, one bottleneck and one additive. It is now extended for one more objective function.

For each value  $c_{u_1}^1$  fixed as a bound of  $c_{ij}^1$ , decreasing bounds are chosen for  $c_{ij}^2$ . Let  $p^*$  be the shortest path in  $G_u$  and  $c_k(p^*) = c_{v_k}^k$ ,  $k = 1, 2$ . Then  $v = (v_1, v_2)$  is called the *final position of  $p^*$* . Under these conditions it can be shown that  $c_k(p^*) = c_{v_k}^k \leq c_{u_k}^k$ ,  $k = 1, 2$ . Then, given some other path  $q$  in  $G_w$  with  $u \leq w \leq v$  two situations can occur:

1.  $c_2(q) < c_2(p^*)$ , which means that  $q$  is still a path in graph  $G_{u_1, v_2+1}$ , therefore it can be found in a subsequent iteration.
2.  $c_2(q) \geq c_2(p^*)$ , therefore two new cases can happen, now in terms of  $c_1$ :
  - (a)  $c_1(q) < c_1(p^*)$ , then  $q$  is still a path in  $G_{u_1+1, v_2}$ , and it can be determined later, or else
  - (b)  $c_1(q) \geq c_1(p^*)$ , then  $q$  is dominated or equivalent to  $p^*$ , as  $p^*$  is the shortest path in  $G_u$  and  $q$  is in  $G_u$ .

As a consequence  $p^*$  can be stored at position  $(u_1, v_2)$  and the next subgraph to consider is  $G_{u_1, v_2+1}$ . Under certain conditions the jump can be applied to more than one row of  $C$  at a time, as in fact these conclusions can be extended to the subgraphs of  $G_u$  until  $G_v$ . This is implemented by using an auxiliary array  $b$ , with  $m_1$  elements, that marks the beginning of the  $c_2$  indexes to be scanned for every row in  $C$ . A second array  $a$ , of the same size, is used to aid updating  $b$ , namely to ensure that all the paths are computed.

Algorithm 1 summarizes the method based on scanning matrix  $C$  as a stair that has just been described. In this pseudo-language code,  $Q$  is an  $m_1 \times m_2$  matrix that stores the  $c_3$  values of the solutions obtained at the corresponding positions. Variable  $ctrl$  is used to skip subproblems that have no solution.

**Algorithm 1.** *Finding Pareto-optimal candidates with stairs*

```

For  $i = 1, \dots, m_1$  Do
   $b_i \leftarrow 1$ ;  $a_i \leftarrow m_1$ 
  For  $j = 1, \dots, m_2$  Do  $C_{ij} \leftarrow \emptyset$ ;  $Q_{ij} \leftarrow \infty$ 
 $ctrl \leftarrow m_2$ ;  $i \leftarrow 1$ 
While  $i \leq m_1$  and  $ctrl \neq 0$  Do
  While  $b_i \leq ctrl$  Do
     $p \leftarrow$  shortest path in  $G_{ib_i}$  in terms of  $c_3$ 
    If  $p$  is not defined Then  $ctrl \leftarrow b_i - 1$ 
  Else
     $v_k \leftarrow$  indexes such that  $c_k(p) = c_{v_k}^k$ ,  $k = 1, 2$ 
     $a_i \leftarrow \min\{v_1, a_i\}$ ;  $C_{a_i v_2} \leftarrow p$ ;  $Q_{a_i v_2} \leftarrow c_3(p)$ 
    For  $k = i, \dots, a_i$  Do  $b_k \leftarrow v_2 + 1$ ;  $a_k \leftarrow a_i$ 
   $i \leftarrow i + 1$ 

```

Assume, for instance, that  $p^*$  is a shortest path in  $G_{11}$  and  $(1, 1)$  is  $p^*$ 's final position. Following the algorithm,  $a$  and  $b$  are updated as  $a_1 = 1$  and  $b_1 = 2$ , and the next step is to consider  $G_{12}$ . Let now  $\bar{p}$  be the shortest path in this graph, with  $c_k(\bar{p}) = c_{v_k}^k$ ,  $k = 1, 2$ , and  $v_1 = 2$ ,  $v_2 = 3$ . If  $a_1$  had not been updated during the first iteration, then after the second iteration  $b_2 = 4$  and the shortest path in  $G_{21}$  would not have been considered. However, according to situations 1 and 2 above, only the iterations between  $(1, 2)$  and  $(2, 3)$ , that is,  $(1, 2)$ ,  $(1, 3)$ ,  $(2, 2)$  and  $(2, 3)$ , can be skipped. This is accomplished by taking into account that in the second iteration  $a_1 = \min\{v_1, a_1\} = \min\{2, 1\} = 1$ , and therefore  $b_1$  is assigned the value 4 while  $b_2$  remains 1.

**Blocks method** The idea of this second method is the same as that used for the stair method, but it considers both the values  $c_1$  and  $c_2$  of the computed paths at the same time. A binary matrix  $R$ , with dimension  $m_1 \times m_2$ , marks the subproblems (or iterations) that have to be solved. The shortest path problem in  $G_v$  is solved if and only if  $R_v = 1$ ; otherwise the next position in  $C$  is considered. The solutions that are obtained are inserted in the final position in matrix  $C$ . Moreover, given  $p$  the shortest path in  $G_{\bar{v}}$  and  $v$  its final position, then  $R_{\hat{v}}$  is set to 0 for every  $\hat{v}$  such that  $\bar{v} \leq \hat{v} \leq v$ . The justification for skipping the shortest path problem resolutions associated with  $\hat{v}$  is similar to that of the stair method.

Algorithm 2 outlines the blocks method in pseudo-code language.

**Algorithm 2.** *Finding Pareto-optimal candidates with blocks*

```

For  $i = 1, \dots, m_1$  Do
  For  $j = 1, \dots, m_2$  Do  $C_{ij} \leftarrow \emptyset$ ;  $Q_{ij} \leftarrow \infty$ ;  $R_{ij} \leftarrow 1$ 
   $ctrl \leftarrow m_2$ ;  $i \leftarrow 1$ 
  While  $i \leq m_1$  and  $ctrl \neq 0$  Do
     $j \leftarrow 1$ 
    While  $j \leq ctrl$  Do
      If  $R_{ij} = 1$  Then
         $p \leftarrow$  shortest path in  $G_{ij}$  in terms of  $c_3$ 
        If  $p$  is not defined Then  $ctrl \leftarrow j - 1$ 
      Else
         $v_k \leftarrow$  indexes such that  $c_k(p) = c_{v_k}^k$ ,  $k = 1, 2$ 
        If  $c_3(p) < Q_{v_1 v_2}$  Then
           $C_{v_1 v_2} \leftarrow p$ ;  $Q_{v_1 v_2} \leftarrow c_3(p)$ 
          For  $g = i, \dots, v_1$  Do
            For  $h = j, \dots, v_2$  Do  $R_{gh} \leftarrow 0$ 
         $j \leftarrow j + 1$ 
     $i \leftarrow i + 1$ 

```

The matrix  $C$  resulting from Algorithms 1 and 2 may contain some dominated or equivalent paths. The process that eliminates such solutions is common to both the stair and blocks methods, and is outlined in Algorithm 3. After this procedure has been applied, given the positions  $v$  and  $\bar{v}$  in  $C$ , such that  $v \leq \bar{v}$  and  $v \neq \bar{v}$ , the correspondent paths  $p$  and  $\bar{p}$  satisfy  $c_3(p) < c_3(\bar{p})$ , therefore  $p$  is not dominated by  $\bar{p}$ . This is done by checking all the objective values of the paths stored in  $C$ , that is, the values in matrix  $Q$ . Assume Algorithm 1 (or 2) has been applied and let  $C_v = p$  and  $C_{\bar{v}} = \bar{p}$ , for  $1 \leq \bar{v}_k \leq v_k = m_k$ , therefore  $c_k(p) \leq c_k(\bar{p})$ ,  $k = 1, 2$ . Thus  $Q_v \leq Q_{\bar{v}}$  means  $p$  dominates or is equivalent to  $\bar{p}$ , so  $\bar{p}$  can be deleted in  $C$ .

**Algorithm 3.** *Filtering Pareto-optimal solutions in  $C$*

```

For  $i = m_1, \dots, 1$  Do

```

For  $j = m_2, \dots, 1$  Do  
 If  $Q_{ij} \neq \infty$  Then  
   If  $j > 1$  and  $Q_{ij} \leq Q_{ij-1}$  Then  $C_{ij-1} \leftarrow \emptyset$ ;  $Q_{ij-1} \leftarrow Q_{ij}$   
   If  $i > 1$  and  $Q_{ij} \leq Q_{i-1j}$  Then  $C_{i-1j} \leftarrow \emptyset$ ;  $Q_{i-1j} \leftarrow Q_{ij}$

**Alternative method** Aimed at reducing the number of shortest path problems that have to be solved, the routine used by the stair and the blocks methods can be replaced by a variant, adapted from the algorithm described in [5], that chooses the best option whenever there is a tie in the cost. This procedure is more demanding than a regular shortest path algorithm, as it implies storing and updating the bottleneck function values, besides the additive cost. Still it can avoid the computation of certain paths and, for some cases, outperforms the previous version, as the empirical tests reported in Section 4 show. Denoting by  $\pi_i^k$  the value of  $c_k$  associated with a path from  $s$  to a node  $i$  at a certain step of this labeling algorithm,  $k = 1, 2, 3$ , the shortest path algorithm variant consists in rewriting the labeling test as:

If  $(\pi_j^3 > \pi_i^3 + c_{ij}^3)$  or  $(\pi_j^3 = \pi_i^3 + c_{ij}^3$  and  $(\pi_j^1 > \max\{\pi_i^1, c_{ij}^1\}$  or  $(\pi_j^1 = \max\{\pi_i^1, c_{ij}^1\}$  and  $\pi_j^2 > \max\{\pi_i^2, c_{ij}^2\}))$ )) Then  
 Node  $j$  is labeled using arc  $(i, j)$   
 $\pi_j^1 \leftarrow \max\{\pi_i^1, c_{ij}^1\}$ ;  $\pi_j^2 \leftarrow \max\{\pi_i^2, c_{ij}^2\}$ ;  $\pi_j^3 \leftarrow \pi_i^3 + c_{ij}^3$

### 3 Algorithms analyses

In this section the stair and blocks method are illustrated. Also the correction of those algorithms and their complexity are analyzed.

Let us consider the graph  $G$  depicted in Figure 1. For this instance of the TSPPB  $m_1 = 5$  and  $m_2 = 4$ , with  $c_{ij}^1 \in \{9, 7, 6, 3, 2\}$  and  $c_{ij}^2 \in \{8, 5, 4, 3\}$ , for any  $(i, j) \in \mathcal{A}$ .

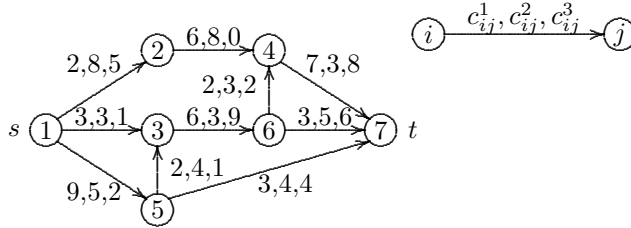


Figure 1: Graph  $G$

Tables 1 and 2 show a list of the shortest paths and the matrix  $C$  obtained when applying to  $G$  Algorithms 1 and 2 followed by Algorithm 3, respectively. Columns  $p$ ,  $v$ ,  $IP$  and  $FP$  present the computed paths, the iteration for obtaining them and their insertion and final positions, respectively. The symbol ‘ $\emptyset$ ’ means there is no path in graph  $G_v$ , therefore no new solution is found at iteration  $v$ .

In Algorithm 1 the first path to be found is  $p_1 = \langle 1, 5, 7 \rangle$ , the shortest path from 1 to 7 in  $G_{11}$ . As  $c_1(p_1) = c_1^1$  and  $c_2(p_1) = c_2^2$ ,  $p_1$  is stored at position  $C_{12}$ . This means that the shortest path problem in  $G_{12}$  can be skipped and the next graph to be considered is  $G_{13}$ .

Note that for  $p_2$  and  $p_4$  the insertion positions are not final. In fact these paths are obtained again at iterations (2, 3) and (3, 1), respectively. The repetitions are eliminated during Algorithm 3.

Consider now the application of Algorithm 2 to the same graph. The resulting paths are presented in Table 2, where ‘-’ means the shortest path problem was not solved in that iteration, that is  $R_v = 0$ .

| $p$                                   | $c_1$ | $c_2$ | $c_3$ | $v$   | $IP$  | $FP$  |
|---------------------------------------|-------|-------|-------|-------|-------|-------|
| $p_1 = \langle 1, 5, 7 \rangle$       | 9     | 5     | 6     | (1,1) | (1,2) | (1,2) |
| $p_2 = \langle 1, 3, 6, 4, 7 \rangle$ | 7     | 3     | 20    | (1,3) | (1,4) | (2,4) |
| $p_3 = \langle 1, 2, 4, 7 \rangle$    | 7     | 8     | 13    | (2,1) | (2,1) | (2,1) |
| $p_4 = \langle 1, 3, 6, 7 \rangle$    | 6     | 5     | 16    | (2,2) | (2,2) | (3,2) |
| $p_5 = \langle 1, 3, 6, 4, 7 \rangle$ | 7     | 3     | 20    | (2,3) | (2,4) | (2,4) |
| $p_6 = \langle 1, 3, 6, 7 \rangle$    | 6     | 5     | 16    | (3,1) | (3,2) | (3,2) |
| $\emptyset$                           | —     | —     | —     | (3,3) | —     | —     |
| $\emptyset$                           | —     | —     | —     | (4,1) | —     | —     |

|   | 1     | 2                           | 3 | 4                           |
|---|-------|-----------------------------|---|-----------------------------|
| 1 |       | $p_1$                       |   | <del><math>p_1</math></del> |
| 2 | $p_3$ | <del><math>p_3</math></del> |   | $p_5$                       |
| 3 |       | $p_6$                       |   |                             |
| 4 |       |                             |   |                             |
| 5 |       |                             |   |                             |

Table 1: Result of Algorithms 1 and 3

| $p$                                   | $c_1$ | $c_2$ | $c_3$ | $v$   | $IP$  | $FP$  | $R_v$ |
|---------------------------------------|-------|-------|-------|-------|-------|-------|-------|
| $p_1 = \langle 1, 5, 7 \rangle$       | 9     | 5     | 6     | (1,1) | (1,2) | (1,2) | 1     |
| —                                     | —     | —     | —     | (1,2) | —     | —     | 0     |
| $p_2 = \langle 1, 3, 6, 4, 7 \rangle$ | 7     | 3     | 20    | (1,3) | (2,4) | (2,4) | 1     |
| —                                     | —     | —     | —     | (1,4) | —     | —     | 0     |
| $p_3 = \langle 1, 2, 4, 7 \rangle$    | 7     | 8     | 13    | (2,1) | (2,1) | (2,1) | 1     |
| $p_4 = \langle 1, 3, 6, 7 \rangle$    | 6     | 5     | 16    | (2,2) | (3,2) | (3,2) | 1     |
| —                                     | —     | —     | —     | (2,3) | —     | —     | 0     |
| —                                     | —     | —     | —     | (2,4) | —     | —     | 0     |
| $p_5 = \langle 1, 3, 6, 7 \rangle$    | 6     | 5     | 16    | (3,1) | —     | (3,2) | 1     |
| —                                     | —     | —     | —     | (3,2) | —     | —     | 0     |
| $\emptyset$                           | —     | —     | —     | (3,3) | —     | —     | 1     |
| $\emptyset$                           | —     | —     | —     | (4,1) | —     | —     | 1     |

|   | 1     | 2     | 3 | 4     |
|---|-------|-------|---|-------|
| 1 |       | $p_1$ |   |       |
| 2 | $p_3$ |       |   | $p_2$ |
| 3 |       | $p_4$ |   |       |
| 4 |       |       |   |       |
| 5 |       |       |   |       |

Table 2: Result of Algorithms 2 and 3

Like before, at the first iteration  $p_1$  is obtained and is inserted in position  $C_{12}$ . Instead of jumping to step (1,3) the next iteration is (1,2), but before  $R_{12}$  is set to 0, which means the shortest path problem does not need to be solved at this iteration. Now  $p_2 = \langle 1, 3, 6, 4, 7 \rangle$  is stored in  $C_{24}$ , because  $c_1(p_2) = 7$  and  $c_2(p_2) = 3$ . This means with this method paths  $p_2$ , and  $p_4$ , are inserted in the final position, (2,4) and (3,2) respectively, from the moment they are obtained. Matrix  $C$  is unchanged after Algorithm 3. Note also that when  $p_2$  is stored,  $R_{ij}$  is set to 0, for  $i = 1, 2$  and  $j = 3, 4$ .

It is worth noting that in this example Algorithm 2 performs twelve iterations, against eight iterations for Algorithm 1. However, some of the iterations of Algorithm 2 (five in this case) consist simply of checking the value of  $R_v$ , which is less demanding than solving a shortest path problem. In fact, the number of shortest path problems solved by Algorithm 2 (which is seven) is smaller than that solved by Algorithm 1 (with eight). The former procedures, MMS and MMS-R, would solve 20 and 12 shortest path problems, respectively.

The following results show that the algorithms presented are capable of finding the minimal complete set of paths. In order to prove some auxiliary results we first note that the final position of a path obtained by Algorithm 1 (or 2) always has components greater than or equal to the iteration where it is computed. Let  $p$  be any path, obtained at iteration  $\hat{v}$ , and let  $v$  be its final

position. Since  $p$  is a path in  $G_{\hat{v}}$  then  $c_{v_k}^k = c_k(p) \leq c_{\hat{v}_k}^k$ ,  $k = 1, 2$ , thus  $v \geq \hat{v}$ . Lemma 1 ensures that the insertion row is always greater than or equal to the iteration.

**Lemma 1.** *In Algorithm 1  $a_k \geq k$  always holds, for  $k = 1, 2, \dots, m_1$ .*

*Proof.* The  $m_1$  inequalities are satisfied at the beginning of the algorithm because in the initialization  $a_k = m_1$ ,  $k = 1, 2, \dots, m_1$ . Let  $p$  be a path obtained at iteration  $(i, b_i)$  and assume that at previous iterations the same property was valid, that is,  $a_k \geq k$ ,  $k = 1, 2, \dots, m_1$ . According to the remark above  $v_1 \geq i$ , where  $v$  is  $p$ 's final position. Combining the two sets of inequalities we can say that at iteration  $(i, b_i)$ , after setting  $a_i = \min\{v_1, a_i\}$ , we have  $a_i \geq i$  and afterwards  $a_k$  is updated as  $a_k = a_i$ , for  $k = i, \dots, a_i$ . As  $k \leq a_i$  then  $a_k \geq k$  for  $k = i, \dots, a_i$  and  $a_k$  is not changed for  $k \in \{1, \dots, i-1, a_i+1, \dots, m_1\}$ , therefore at the iteration after  $(i, b_i)$  we still have  $a_k \geq k$ ,  $k = 1, 2, \dots, m_1$ .  $\square$

Lemma 2 shows that in Algorithms 1 and 2 the insertion position is always between the iteration number and the final position.

**Lemma 2.** *Let  $p$  be a path obtained in iteration  $\hat{v}$  of Algorithm 1 (or 2). Then  $\hat{v} \leq \bar{v} \leq v$ , where  $v$  and  $\bar{v}$  are  $p$ 's final and insertion positions, respectively.*

*Proof.* We first note that  $\hat{v} \leq v$ . Besides, for Algorithm 2  $\bar{v} = v$ , while for Algorithm 1  $\bar{v}_2 = v_2$  and  $\bar{v}_1 = a_{\hat{v}_1} = \min\{v_1, a_{\hat{v}_1}\}$ , which implies that  $\bar{v}_1 \leq v_1$ . Since by Lemma 1 we have  $a_{\hat{v}_1} \geq \hat{v}_1$ , then  $\bar{v}_1 \geq \hat{v}_1$ , and therefore  $\hat{v} \leq \bar{v} \leq v$  is valid for both algorithms.  $\square$

Theorem 1 shows that after Algorithm 3 is applied all paths in  $C$  are at their final positions and are Pareto-optimal.

**Theorem 1.** *For any  $C_v = p \neq \emptyset$ , after Algorithm 1 (or Algorithm 2) followed by Algorithm 3:*

1.  $v$  is the final position of  $p$  in  $C$ , and
2.  $p$  is Pareto-optimal.

*Proof.* 1. The proof of this point is immediate for Algorithm 2, because all insertions are made in the final position.

Let us consider the same case for Algorithm 1. Given a path  $p$  in  $C_v$ , assume  $\bar{v}$  is its final position, that is,  $c_k(p) = c_{\bar{v}_k}^k$ ,  $k = 1, 2$ . As paths are inserted at their final column, then  $\bar{v}_2 = v_2$ . Furthermore, by Lemma 2,  $\bar{v}_1 \geq v_1$ , which implies  $c_1(p) = c_{\bar{v}_1}^1 \leq c_{v_1}^1$ . We now prove, by contradiction, that  $c_{\bar{v}_1}^1 = c_{v_1}^1$ . If  $c_{\bar{v}_1}^1 < c_{v_1}^1$ , then

$$\bar{v}_1 > v_1 \Rightarrow \bar{v}_1 \geq v_1 + 1 \Rightarrow c_{\bar{v}_1}^1 \leq c_{v_1+1}^1,$$

thus  $p$  is a path in  $G_{v^+}$ , for  $v^+ = (v_1 + 1, v_2)$ . Two cases can be considered for iteration  $v^+$ :

- (a) If the shortest path problem is solved in  $G_{v^+}$ , the solution is  $p^+$  such that  $c_3(p^+) \leq c_3(p)$ , therefore by Lemma 2  $p^+$  is inserted in  $C_{\bar{v}}$ , for some  $\tilde{v} \geq v^+$ . Furthermore  $\tilde{v} \geq v$  with  $\tilde{v}_1 \geq v_1 + 1$ , thus in Algorithm 3, when  $i = v_1 + 1$  and  $j = v_2$  we have  $Q_{ij} \leq c_3(p^+) \leq c_3(p) = Q_v = Q_{i-1j}$ , then  $C_v$  is set to  $\emptyset$ .
- (b) Else in some iteration  $\hat{v}$  such that  $\hat{v} \leq v^+$ , a path  $p^+$  is obtained and inserted in  $C_{\bar{v}}$  for some  $\tilde{v} \geq v^+$ . We have  $p$  in  $G_{\hat{v}}$  because  $\hat{v} \leq v^+$ , then  $c_3(p^+) \leq c_3(p)$ . Using the same proof we again conclude that  $C_v = \emptyset$ .

Thus  $c_{\bar{v}_1}^1 < c_{v_1}^1$  implies  $C_v = \emptyset$ , which contradicts  $C_v = p \neq \emptyset$ , therefore  $c_1(p) = c_{\bar{v}_1}^1 = c_{v_1}^1$ .

2. Path  $p$  is the shortest in some graph  $G_{\bar{v}}$  such that  $\bar{v} \leq v$  and, by item 1,  $c(p) = (c_{v_1}^1, c_{v_2}^2, c_3(p))$ . Assume, by contradiction, that there is a path  $\hat{p}$  that dominates  $p$ , that is,

$$c(\hat{p}) = (c_{\hat{v}_1}^1, c_{\hat{v}_2}^2, c_3(\hat{p})) \leq c(p) = (c_{v_1}^1, c_{v_2}^2, c_3(p)) \quad \text{and} \quad c(\hat{p}) \neq c(p).$$

Then  $\bar{v} \leq v \leq \hat{v}$ , therefore  $\hat{p}$  is also a path in  $G_{\bar{v}}$ . On the one hand  $p$  is the shortest path in that graph, and on the other hand  $c(p) \geq c(\hat{p})$ , thus  $c_3(\hat{p}) = c_3(p)$ . Together with  $c(\hat{p}) \leq c(p)$  and  $c(\hat{p}) \neq c(p)$  this leads to  $c_{\hat{v}_1}^1 < c_{v_1}^1$  or  $c_{\hat{v}_2}^2 < c_{v_2}^2$ .

Let us consider  $c_{\hat{v}_1}^1 < c_{v_1}^1$  (the other case can be treated similarly). Then  $\hat{v} \geq v$  with  $\hat{v}_1 > v_1$ , therefore  $\hat{p}$  is a path in  $G_{v^+}$ , where  $v^+ = (v_1 + 1, v_2)$ . Now following the same steps as before a contradiction is found, therefore  $p$  is Pareto-optimal.  $\square$

The next result ensures that after the two phases of the algorithms are applied  $c(p_1) \neq c(p_2)$ , for any two distinct paths  $p_1$  and  $p_2$  stored at  $C$ .

**Proposition 1.** *After Algorithms 1 (or 2) and 3 the paths in  $C$  have distinct objective values  $c$ .*

*Proof.* At the end of the algorithms, by Theorem 1, all paths are in the final positions and there is at most one path in each  $C$  position, therefore there are no two paths with exactly the same values in  $c_1, c_2$ .  $\square$

Lemma 3 is an auxiliary result for proving Theorem 2.

**Lemma 3.** *Let  $p^*$  be a Pareto-optimal path inserted at its final position in  $C$  by Algorithm 1 (or 2). After the application of Algorithm 3  $C_v = p^*$ , where  $v$  is  $p^*$ 's final position.*

*Proof.* In Algorithm 1 all solutions are inserted in distinct positions, therefore it is not necessary to delete any solution. In Algorithm 2 a solution  $p$  is only replaced by another solution  $\bar{p}$ , when  $\bar{p}$  dominates  $p$ , and thus a Pareto-optimal path is never deleted. A path  $p$  is only deleted in Algorithm 3, where  $p$  was inserted at its final position, if  $p$  is dominated by another path in  $C$ . Therefore a Pareto-optimal path inserted at its final position is never removed from  $C$ .  $\square$

Finally we prove that Algorithms 1 and 2, together with Algorithm 3, find a minimal complete set of solutions.

**Theorem 2.** *Let  $p^*$  be a Pareto-optimal path and  $v$  be such that  $c_k(p^*) = c_{v_k}^k$ ,  $k = 1, 2$ . After the application of Algorithms 1 (or 2) and 3  $C_v = p$ , with  $p$  some path in  $\mathcal{P}$  such that  $c(p) = c(p^*)$ .*

*Proof.* Two cases have to be analyzed:

1. The shortest path problem in  $G_v$  is not solved.

In this case a path  $p$ , the shortest in  $G_{\bar{v}}$  where  $\bar{v} \leq v$ , is inserted in position  $\hat{v}$  such that  $\hat{v} \geq v$ . Let  $\tilde{v}$  be  $p$ 's final position. We have  $\hat{v} \geq v$  and by Lemma 2  $\tilde{v} \geq \hat{v}$ , then  $c_k(p) = c_{\tilde{v}_k}^k \leq c_{\hat{v}_k}^k \leq c_{v_k}^k = c_k(p^*)$ ,  $k = 1, 2$ . Since  $\bar{v} \leq v$  the path  $p^*$  is in graph  $G_{\bar{v}}$ , and therefore  $c_3(p) \leq c_3(p^*)$ , hence  $c(p) \leq c(p^*)$ . As  $p^*$  is Pareto-optimal, then  $c(p) = c(p^*)$  and  $\tilde{v} = \hat{v} = v$ . This means  $p$  is also Pareto-optimal,  $v$  is its final position and  $p$  is stored in  $C_v$  at iteration  $\bar{v}$ .

2. The shortest path problem in  $G_v$  is solved.

$p^*$  is a path in graph  $G_v$ . If  $p$  is the shortest path obtained at iteration  $v$ , then  $c_3(p) \leq c_3(p^*)$ . As  $v$  is  $p^*$ 's final position and  $p \in G_v$ , then  $c_k(p) \leq c_k(p^*)$ ,  $k = 1, 2$ , therefore  $c(p) \leq c(p^*)$ . On the other hand  $p^*$  is Pareto-optimal, therefore  $c(p) = c(p^*)$  must hold, which implies  $p$  is a Pareto-optimal too and  $v$  is its final position. Finally  $C_v = p$  at the end of iteration  $v$  (for Algorithm 2  $R_v = 1$  implies  $Q_v = \infty$ ).

In both cases, by Lemma 3  $C_v = p$  holds. □

The number of operations performed by Algorithms 1 and 2 depends on the number of distinct values of the bottleneck objective functions,  $m_1, m_2 \leq m$ , which are highly correlated with the number of Pareto-optimal solutions, and thus the number of shortest path problems that have to be solved. As a result the new methods share the same worst-case complexity order of previous algorithms in the literature,  $\mathcal{O}(m_1 m_2 c(n))$ , where  $c(n)$  is the number of operations needed to find the shortest path in a network with  $n$  nodes. The same bound is valid for the variant that avoids ties in the paths' bottleneck values, as in a worst-case there is a non-dominated solution for every pair of bottleneck values  $c_1$  and  $c_2$ .

## 4 Computational experiments

Computational experiments were carried out to evaluate the performance of the new methods, as well as to compare them with previous approaches. Six codes were implemented in C: the methods described in the literature, MMS and MMS-R [6, 7], the stair method, MMS-S, and the blocks method, MMS-B, as well as the variants of the latter two which deal with cost ties, MMS-ST and MMS-BT. In order to determine the shortest path the first four programs used Dijkstra's algorithm [2], while the others used its modified version described at the end of Section 2. The codes ran on an Intel Core 2 Duo 2.0GHz with 3GB of RAM.

The first set of tests aimed to compare MMS-S and MMS-B against the previous methods, MMS and MMS-B. Random networks with 500, 2000 and 5000 nodes,  $dn$  arcs, for densities  $d = 20, 100$ , and integer  $c_{ij}^3$ , uniformly generated in  $[1, 10000]$  were considered. The bottleneck values  $c_{ij}^k$ , are integers randomly generated in  $[1, m_k]$ ,  $k = 1, 2$ , where  $m_1 = m_2$  may be 10, 50 and 200. For any node  $i$ ,  $d$  successors  $j$  are randomly chosen and the arcs  $(i, j)$  are created in  $\mathcal{A}$ . The results below are average values for 10 different instances of each data set dimension.

| $n$  | $m$    | $m_1$ | # PO | MMS     |       | MMS-R   |       | MMS-S  |      | MMS-B  |     |
|------|--------|-------|------|---------|-------|---------|-------|--------|------|--------|-----|
|      |        |       |      | Time    | SPP   | Time    | SPP   | Time   | SPP  | Time   | SPP |
| 500  | 9796   | 10    | 36   | 0.14    | 100   | 0.12    | 83    | 0.08   | 49   | 0.07   | 42  |
| 500  | 9791   | 50    | 106  | 2.96    | 2500  | 2.67    | 1684  | 0.49   | 248  | 0.30   | 145 |
| 500  | 9792   | 200   | 140  | 43.55   | 40000 | 39.73   | 26408 | 0.91   | 450  | 0.42   | 206 |
| 500  | 45264  | 10    | 50   | 0.20    | 100   | 0.19    | 99    | 0.14   | 66   | 0.12   | 54  |
| 500  | 45245  | 50    | 153  | 3.82    | 2500  | 3.78    | 2313  | 0.81   | 386  | 0.46   | 205 |
| 500  | 45243  | 200   | 321  | 60.17   | 40000 | 58.76   | 35891 | 2.53   | 1203 | 1.02   | 457 |
| 2000 | 39791  | 10    | 41   | 1.71    | 100   | 1.61    | 77    | 1.36   | 55   | 1.22   | 48  |
| 2000 | 39788  | 50    | 148  | 42.09   | 2500  | 38.12   | 1741  | 9.11   | 330  | 5.58   | 194 |
| 2000 | 39788  | 200   | 245  | 609.91  | 40000 | 559.77  | 27710 | 23.52  | 850  | 10.44  | 354 |
| 2000 | 195037 | 10    | 56   | 2.16    | 100   | 2.14    | 99    | 1.80   | 73   | 1.62   | 61  |
| 2000 | 195021 | 50    | 256  | 60.53   | 2500  | 59.08   | 2306  | 18.30  | 573  | 11.01  | 322 |
| 2000 | 195037 | 200   | 474  | 792.38  | 40000 | 781.29  | 36226 | 56.70  | 1966 | 20.60  | 670 |
| 5000 | 99790  | 10    | 46   | 11.97   | 100   | 11.72   | 81    | 9.64   | 59   | 8.87   | 53  |
| 5000 | 99780  | 50    | 206  | 242.11  | 2500  | 221.66  | 1785  | 72.81  | 451  | 45.03  | 265 |
| 5000 | 99795  | 200   | 289  | 3789.81 | 40000 | 3518.27 | 26817 | 178.83 | 1028 | 77.41  | 426 |
| 5000 | 494924 | 10    | 59   | 14.25   | 100   | 14.19   | 99    | 12.22  | 73   | 11.31  | 64  |
| 5000 | 494972 | 50    | 296  | 361.43  | 2500  | 352.87  | 2315  | 125.61 | 664  | 73.48  | 372 |
| 5000 | 494940 | 200   | 602  | 4709.67 | 40000 | 4609.85 | 36346 | 430.59 | 2379 | 158.50 | 836 |

Table 3: Average results for codes MMS, MMS-R, MMS-S and MMS-B

Table 3 presents for this set of tests the final number of Pareto-optimal solutions, the CPU times (in seconds) and the number of shortest path problems solved. We first note that the number

of Pareto-optimal solutions for this set of problems is far from the theoretical upper bound  $m_1m_2$  [6]. For  $m_1 = m_2 = 10$  that number is between 36% and 59% of  $m_1m_2$ , for  $m_1 = m_2 = 50$  between 4% and 12% and for  $m_1 = m_2 = 200$  between 0.4% and 1.5% of  $m_1m_2$ . Still concerning the number of solved shortest path problems, it is always  $m_1m_2$  with MMS while for MMS-R it is greater than 65% $m_1m_2$ . With MMS-S and MMS-B these values are between 10% and 27%, and between 6% and 15% for  $m_1 = m_2 = 50$ , respectively. The numbers decrease with  $m_1, m_2$  and are between 1.1% and 6% for MMS-S and between 0.5% and 2.1% for MMS-B, when  $m_1 = m_2 = 200$ . Finally it should be noted that the number of subproblems and the number of Pareto-optimal solutions are very close, which indicates that not many unnecessary shortest path problems are being computed.

|      |        |       | CPU                                 |                                     |                                     | SPP                                 |                                     | 1000 × CPU/PO |        |
|------|--------|-------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|---------------|--------|
| $n$  | $m$    | $m_1$ | $\frac{\text{MMS-R}}{\text{MMS-S}}$ | $\frac{\text{MMS-R}}{\text{MMS-B}}$ | $\frac{\text{MMS-S}}{\text{MMS-B}}$ | $\frac{\text{MMS-R}}{\text{MMS-S}}$ | $\frac{\text{MMS-R}}{\text{MMS-B}}$ | MMS-S         | MMS-B  |
| 500  | 9796   | 10    | 1.50                                | 1.71                                | 1.14                                | 1.69                                | 1.98                                | 2.22          | 1.94   |
| 500  | 9791   | 50    | 5.45                                | 8.90                                | 1.63                                | 6.79                                | 11.61                               | 4.62          | 2.83   |
| 500  | 9792   | 200   | 43.66                               | 94.60                               | 2.17                                | 58.68                               | 128.19                              | 6.50          | 3.00   |
| 500  | 45264  | 10    | 1.36                                | 1.58                                | 1.17                                | 1.50                                | 1.83                                | 2.80          | 2.40   |
| 500  | 45245  | 50    | 4.67                                | 8.22                                | 1.76                                | 5.99                                | 11.28                               | 5.29          | 3.01   |
| 500  | 45243  | 200   | 23.23                               | 57.61                               | 2.48                                | 29.83                               | 78.54                               | 7.88          | 3.18   |
| 2000 | 39791  | 10    | 1.18                                | 1.32                                | 1.11                                | 1.40                                | 1.60                                | 33.17         | 29.76  |
| 2000 | 39788  | 50    | 4.18                                | 6.83                                | 1.63                                | 5.28                                | 8.97                                | 61.55         | 37.70  |
| 2000 | 39788  | 200   | 23.80                               | 53.62                               | 2.25                                | 32.60                               | 78.28                               | 96.00         | 42.61  |
| 2000 | 195037 | 10    | 1.19                                | 1.32                                | 1.11                                | 1.36                                | 1.62                                | 32.14         | 28.93  |
| 2000 | 195021 | 50    | 3.23                                | 5.37                                | 1.66                                | 4.02                                | 7.16                                | 71.48         | 43.01  |
| 2000 | 195037 | 200   | 13.78                               | 37.93                               | 2.75                                | 18.43                               | 54.07                               | 119.62        | 43.46  |
| 5000 | 99790  | 10    | 1.22                                | 1.32                                | 1.09                                | 1.37                                | 1.53                                | 209.57        | 192.83 |
| 5000 | 99780  | 50    | 3.04                                | 4.92                                | 1.62                                | 3.96                                | 6.74                                | 353.45        | 218.59 |
| 5000 | 99795  | 200   | 19.67                               | 45.45                               | 2.31                                | 26.09                               | 62.95                               | 618.79        | 267.85 |
| 5000 | 494924 | 10    | 1.16                                | 1.25                                | 1.08                                | 1.36                                | 1.55                                | 207.12        | 191.69 |
| 5000 | 494972 | 50    | 2.81                                | 4.80                                | 1.71                                | 3.49                                | 6.22                                | 424.36        | 248.24 |
| 5000 | 494940 | 200   | 10.71                               | 29.08                               | 2.72                                | 15.28                               | 43.48                               | 715.27        | 263.29 |

Table 4: Comparison of the results for codes MMS-R, MMS-S and MMS-B

Following the trends indicated in [7] Table 3 reports version MMS-R solves between 1%, when  $m_1 = m_2 = 10$ , and 51%, when  $m_1$  and  $m_2$  are greater, less shortest path problems than the original version, which corresponds to time improvements between 1% and 17%, greater for small size instances. Table 4 completes the comparison of MMS-R, MMS-S and MMS-B by showing the ratios of running times and shortest path subroutine calls. Like before the new algorithms' performance is worse for larger problems as shortest path problems are harder to compute in these cases and, in general, there are more Pareto-optimal paths to be found. Whether it is measured in terms of CPU times or in terms of the number of subroutine calls, the improvement of the new approaches is obvious. In the case of MMS-S it is from 1.15 to 43.66 times faster than MMS-R. For MMS-B the speed-up is even greater, these values are between 1.25 and 94.60. The results concerning the number of shortest paths found are very similar. The decrease on both time and number of subproblems is specially dependent on  $m_1, m_2$ , while it remains nearly constant when  $n$  increases.

The high dependence on  $m_1, m_2$  reflects the complexity order of the algorithms. Also the outperformance of MMS-B over MMS-S was expected, as the fact that with the first paths are stored in the final position when they are computed allows more subproblems to be skipped. This is confirmed by Tables 3 and 4. Again the difference between the two methods is more evident for higher  $m_1, m_2$  values and the weight of this factor is larger for instances with fewer nodes. The

ratio of the stair and the blocks methods running times is around 1.10 for  $m_1 = m_2 = 10$ , while for  $m_1 = m_2 = 200$  it ranges between 2.17 to 2.75.

| $n$   | $m$     | $m_1$ | # PO | MMS-S  |       | MMS-ST |       | MMS-B  |     | MMS-BT |     |
|-------|---------|-------|------|--------|-------|--------|-------|--------|-----|--------|-----|
|       |         |       |      | Time   | SPP   | Time   | SPP   | Time   | SPP | Time   | SPP |
| 1 000 | 19 793  | 50    | 99   | 1.75   | 237   | 1.55   | 209   | 1.15   | 148 | 1.03   | 134 |
| 1 000 | 95 112  | 200   | 234  | 8.09   | 953   | 6.26   | 741   | 3.55   | 401 | 2.99   | 342 |
| 5 000 | 99 783  | 50    | 116  | 51.02  | 284   | 45.61  | 250   | 32.56  | 176 | 29.31  | 157 |
| 5 000 | 494 953 | 200   | 322  | 279.54 | 1 423 | 213.38 | 1 097 | 113.97 | 558 | 94.95  | 466 |
| 7 000 | 139 778 | 50    | 154  | 130.31 | 376   | 117.64 | 334   | 79.57  | 221 | 73.88  | 203 |
| 7 000 | 694 876 | 200   | 354  | 556.85 | 1 433 | 444.77 | 1 139 | 238.23 | 594 | 202.18 | 502 |

Table 5: Average results for codes MMS-S, MMS-ST, MMS-B and MMS-BT, in instances with  $c_{ij}^3 \in [1, 10]$

The last two columns in Table 4 depict the average time in milliseconds needed to obtain each Pareto-optimal path. If on the one hand these values show a small  $m_1, m_2$  dependence and, for the performed tests, they were steady to the variation of  $d$ ; on the other hand they are highly sensitive to  $n$ , once it affects the time for solving shortest path problems.

On this set of instances the codes MMS-ST and MMS-BT were always less efficient than their original versions. In fact, the number of solved shortest path problems coincided and, as expected, the running times were higher for the codes that deal with ties on the paths cost. On a second test set tighter ranges of the  $c_{ij}^3$  values,  $(i, j) \in \mathcal{A}$ , were considered. This set was formed by random networks with 1 000, 5 000 and 7 000 nodes, densities of  $d = 20, 100$ , and  $c_{ij}^3$  uniformly generated in  $[1, M]$ , for  $M = 10, 100$ . The results, summarized in Tables 5 and 6, are again averages for 10 problems.

| $n$   | $m$     | $m_1$ | # PO | MMS-S  |       | MMS-ST |       | MMS-B  |     | MMS-BT |     |
|-------|---------|-------|------|--------|-------|--------|-------|--------|-----|--------|-----|
|       |         |       |      | Time   | SPP   | Time   | SPP   | Time   | SPP | Time   | SPP |
| 1 000 | 19 796  | 50    | 114  | 1.84   | 258   | 1.78   | 253   | 1.14   | 157 | 1.10   | 155 |
| 1 000 | 95 140  | 200   | 351  | 12.04  | 1 345 | 11.01  | 1 262 | 4.90   | 515 | 4.62   | 499 |
| 5 000 | 99 780  | 50    | 174  | 76.39  | 412   | 75.89  | 411   | 45.23  | 237 | 44.53  | 234 |
| 5 000 | 494 987 | 200   | 543  | 413.26 | 2 154 | 377.08 | 2 032 | 160.02 | 793 | 149.07 | 766 |
| 7 000 | 139 798 | 50    | 181  | 148.91 | 423   | 145.78 | 415   | 85.77  | 239 | 84.48  | 237 |
| 7 000 | 694 944 | 200   | 551  | 828.49 | 2 378 | 759.24 | 2 261 | 305.40 | 811 | 283.46 | 783 |

Table 6: Average results for codes MMS-S, MMS-ST, MMS-B and MMS-BT, in instances with  $c_{ij}^3 \in [1, 100]$

Tables 5 and 6 show that for costs in  $[1, 10]$  and  $[1, 100]$  avoiding ties has a positive impact over both the number of shortest path algorithm calls and the running times. The general tendency of increasing CPU time and number of subproblems with  $n$  and, in particular, with  $m_1$  and  $m_2$  is still observed for MMS-ST and MMS-BT. This modification has more impact on the stair method than on the blocks method.

When  $M = 10$  MMS-ST solved between 1.13 and 1.30 times less shortest path problems than MMS-S. These values are smaller for the blocks methods, between 1.09 and 1.20. In what concerns the running time, this results in a speedup between 1.11 and 1.31 for the stair methods and between 1.08 and 1.20 for the blocks methods. The improvement is not so sharp for the wider range of costs,  $M = 100$ . In fact with MMS-ST the number of subproblems is identical in one of the cases, for the others it decreases up to 1.07 times. The reduction obtained by MMS-BT is also small, between 1.01 and 1.04. In terms of time MMS-ST is from 1.01 to 1.10 times faster than its original version. The range is from 1.02 to 1.08 for MMS-BT.

## 5 Final remarks

We have introduced two methods for finding the minimal complete set of Pareto-optimal paths for a tricriteria path problem involving two bottleneck objective functions and a cost, the stair method and the blocks method. These methods enhance the procedures presented recently in [6] and [7] by using the objective values of the obtained paths to reduce the number of shortest path subproblems that have to be solved. The number of operations performed by the algorithms depends on the number of distinct values of the bottleneck objective functions,  $m_1, m_2$ , which are correlated with the number of Pareto-optimal solutions, and thus the number of shortest path problems that need to be solved. As a result the new methods have the same worst-case complexity order as previous algorithms,  $\mathcal{O}(m_1 m_2 c(n))$ , where  $c(n)$  is the number of operations needed to find the shortest path in a network with  $n$  nodes. Empirical tests showed that in practice this bound is far from being attained and in random instances with  $n = 5\,000$ , average degree 100 and  $m_1 = m_2 = 200$  the stair method improved the previous solving about 15 times less shortest path problems than before, while the blocks method solved about 43 times less. As a result the CPU times were improved in about 11 and 29 times, respectively. The blocks method was able to compute the complete minimal set in instances of  $n = 7\,000$  and  $m_1 = m_2 = 200$  in less than 4 minutes.

Variants of the two methods with the purpose of reducing the number of subproblems that have to be solved were also proposed. To this end the labeling test used in the shortest path algorithm is modified so that every time two paths ending at a given node have the same cost, the one with the best bottleneck value is chosen. For the experiments performed on random instances the percentage of improvement with these variants was between 10% and 30% for arc costs in  $[1, 10]$  and between 1% and 9% for arc costs in  $[1, 100]$ . The improvement is more significant for the stair than for the blocks method.

Finally, it should be noted that the idea used to develop the stair and blocks methods defines a way to search for the best solutions in matrix  $C$ . Therefore it can be applied to other combinatorial problems with one additive and two bottleneck functions. Moreover these procedures can be extended to the multi-objective bottleneck network problem, similarly to what was done in [8].

**Acknowledgments** This work was developed during a visit of L. Pinto and M. Pascoal to CIRRELT, Montréal, Canada. The work of M. Pascoal was partially funded by the FCT Portuguese Foundation of Science and Technology (Fundação para a Ciência e a Tecnologia) under grant SFRH/BSAB/830/2008. The authors also thank Gilbert Laporte for comments and suggestions on a preliminary version of this manuscript.

## References

- [1] Berman, O., Einav, D., Handler, G. “The constrained bottleneck problem in network”. *Operations Research* 38, pp. 178-181, 1990.
- [2] Dijkstra, E. “A note on two problems in connection with graphs”. *Numerical Mathematics* 1, pp. 395-412, 1959.
- [3] Hansen, P. “Bicriterion path problems”. In: *Multicriteria decision making: theory and applications*, Lecture Notes in Economics and Mathematical Systems, 177, G. Fandel and T. Gal (eds.), Springer, Heidelberg, pp. 109-127, 1980.
- [4] Martins, E. “On a special class of bicriterion path problems”. *European Journal of Operational Research* 17, pp. 85-94, 1984.

- [5] Martins, E., Pascoal, M., Rasteiro, D., Santos, J. “The optimal path problem”. *Investigação Operacional* 19, pp. 43-60, 1999. ([www.mat.uc.pt/~marta/Publicacoes/opath.ps.gz](http://www.mat.uc.pt/~marta/Publicacoes/opath.ps.gz)).
- [6] Pinto, L., Bornstein, C., Maculan, N. “The tricriterion shortest path problem with at least two bottleneck objective functions”. to appear in *European Journal of Operational Research*, 2008.
- [7] Pinto, L., Bornstein, C., Maculan, N. “Um problema de caminho tri-objetivo”. *Anais do XL SBPO*, João Pessoa-PB, pp. 1032-1042, 2008.
- [8] Pinto, L., Bornstein, C., Maculan, N. “A polynomial algorithm for the multi-objective bottleneck network problem with one MinSum objective function”. Submitted to *Networks*, 2008.