Finding extreme supported solutions of biobjective network flow problems: an enhanced parametric programming approach.

ANDREA RAITH (a.raith@auckland.ac.nz)
Corresponding Author. Department of Engineering Science, The University of Auckland, Private Bag 92019, Auckland, New Zealand. Phone +64 (9) 923 1977.

ANTONIO SEDEÑO-NODA (asedeno@ull.edu.es)
Departamento de Matemáticas, Estadística e Investigación Operativa. Universidad de La Laguna, C.P. 38271, San Cristóbal de La Laguna, Santa cruz de Tenerife, España.

Abstract: We address the problem of determining a complete set of extreme supported efficient solutions of biobjective minimum cost flow (BMCF) problems. A novel method improving the classical parametric method for this biobjective problem is proposed. The algorithm runs in O($Nn(m+n\log n)$) time determining all extreme supported non-dominated points in the outcome space and one extreme supported efficient solution associated with each one of them. Here n is the number of nodes, m is the number of arcs and N is the number of extreme supported non-dominated points in outcome space for the BMCF problem. The memory space required by the algorithm is O($n+m$) when the extreme supported efficient solutions are not required to be stored in RAM. Otherwise, the algorithm requires O($N+m$) space. Extensive computational experiments comparing the performance of the proposed method and a standard parametric network simplex method are presented.

Keywords Biobjective minimum cost flow problem; Extreme supported efficient solutions; Network flow algorithm; Parametric simplex method.

1. Introduction

In this paper we study biobjective minimum cost network flow problems. Minimum cost network flow (MCF) problems are widely applied network optimization models where a single commodity is moved through a capacitated network at minimum overall cost. Many algorithms to solve MCF have been proposed and tested, see [1,2]. In applications often more than one objective function needs to be considered leading to biobjective minimum cost network flow (BMCF) problems, or even to multiobjective MCF if more than two objectives are to be included. Two BMCF problems should be distinguished here. The first one has continuous variables and its set of efficient solutions consists of supported efficient solutions only, which can all be obtained by solving an MCF with a weighted sum objective function.

Secondly, flow in BMCF could be restricted to be integer-valued (problem BIMCF) in which case both supported and non-supported efficient solutions exist.

Here, we propose a variation of the parametric network simplex method for BMCF that obtains a complete set of extreme supported efficient solutions of continuous BMCF problems, and of BIMCF (as long as all capacities in the problem are also integer-valued). Extreme supported efficient solutions are faster and easier to identify than a complete set of efficient solutions of BIMCF and would allow decision makers to obtain an impression of available solutions quickly, which is particularly important for larger, and thus computationally challenging, problem instances. Extreme supported efficient solutions also need to be identified in the first phase of a two-phase approach, as described in [3] for BIMCF, where they help reduce the search space when other efficient solutions are sought in the second phase.

A comprehensive review of algorithms for BMCF is published in [4], where the authors comment on a lack of algorithms for multiobjective MCF at the time. The most promising methods for BMCF are applying a parametric network simplex method [3,5] or solving a sequence of scalarizations [6,7,8]. In [6] problems are solved by a network simplex method, and in [7] by an interior point method, however, the authors remark on performance problems in the latter case. In [8] scalarizations are applied to biobjective problems, and the resulting single-objective problems are solved in parallel, but only tested on shortest path problem instances. Out-of-kilter methods have also been proposed, see [4], but have not been shown to be computationally superior to network simplex methods. More recently, a primal-dual simplex method was developed for BMCF but the authors conclude that it generally performs worse than a (primal) parametric simplex method for most of their test instances [9]. In [10] an approach to find all supported efficient solutions of multiobjective MCF (assuming extreme supported solutions and corresponding weight vectors are given) is introduced, which is based on zero-cost cycles in the incremental graph associated with the corresponding weighted sum problems. Other work focuses on integer biobjective or multiobjective MCF where non-supported efficient solutions generally are the most challenging to identify [3,11,12,13].

Our proposed improvement of the classical parametric network simplex method extends ideas we first proposed in [14] for a labelling algorithm to solve biobjective shortest path problems. Where a parametric network simplex method has to scan all non-basic arcs as candidates to enter the basis associated with a current solution, we are able to show that only a

subset of arcs needs to be considered as their ratio changes, which we exploit in the formulation of a new ratio-labelling algorithm.

The paper is organized as follows: Section 2 describes the BMCF problem and introduces some known results from the literature. In Section 3, BMCF is formulated as a parametric programming problem and details of improvements to aspects of the parametric network simplex method are presented when the aim is to compute all extreme supported non-dominated points in the outcome space of the BMCF problem, and one extreme supported efficient solution associated with each of them. Section 4 proposes a ratio-labeling algorithm where the nodes have an associated ratio, instead of associating this ratio with arcs. This is done in order to improve the computational effort in the operations that select the entering arc with minimum ratio of reduced cost in each iteration. Section 4 also provides the worst-case time and space complexities of the proposed new algorithm. In Section 5, computational experiments comparing performance of the proposed algorithms and other known algorithms are discussed. Finally, Section 6 concludes with final comments and possible future avenues of investigation.

2. The biobjective minimum cost flow problem.

Given a directed network $G = (V, A)$, let $V = \{1,...,n\}$ be the set of nodes and A be the set of m arcs. For each node $i \in V$, let the integer b_i be the supply/demand of the node i and for each arc $(i, j) \in A$ let u_{ij} and l_{ij} be the upper bound and the lower bound on flow through arc (i, j), respectively. Let c_{ij}^v be the cost per unit of flow on arc (i, j) in the v-th objective function, $v \in \{1, 2\}$.

If x_{ij} denotes the amount of flow on an arc (i, j), $\Gamma_i^+ = \{ j \in V \mid (i, j) \in A \}$ and $\Gamma_i^- = \{ j \in V \mid (j, i) \in A \}$, the BMCF problem can be formalized in the following way:

$$\text{Minimize} \quad c(x) = (c^1(x), c^2(x)) = \left(\sum_{i \in V} \sum_{j \in \Gamma_i^+} c_{ij}^1 x_{ij}, \sum_{i \in V} \sum_{j \in \Gamma_i^+} c_{ij}^2 x_{ij} \right) \tag{1}$$

$s.t$:

$$\sum_{j \in \Gamma_i^+} x_{ij} - \sum_{j \in \Gamma_i^-} x_{ji} = b_i, \ \forall i \in V \tag{2}$$

$$l_{ij} \le x_{ij} \le u_{ij}, \ \forall (i, j) \in A \tag{3}$$

Let X be the polyhedron defined by constraints (2)-(3) (feasible set in decision space) and let its image under the objective function be $C = c(X)$ (feasible set in outcome space). The above problem is a biobjective version of the minimum cost flow (MCF) problem. The network simplex method solves the MCF problem by taking advantage of the fact that every basis in the MCF problem is also a spanning tree T of G (see [1,15]). A spanning tree T of G is a sub-graph of G with n-1 arcs that contains no cycle. We denote by $V(T)$ the set of nodes included in a tree T and by $A(T)$ the set of arcs in T.

Definition 1. A feasible solution $x \in X$ is called *efficient* if there does not exist any $x' \in X$ with $c^1(x') \leq c^1(x)$ and $c^2(x') \leq c^2(x)$ with at least one inequality being strict. The image $c(x)$ of an efficient solution x is called *non-dominated point*.

Definition 2. *Supported efficient solutions* are those efficient solutions that can be obtained as optimal solutions of a weighted sum problem $\min_{x \in X} \left(\lambda_1 c^1(x) + \lambda_2 c^2(x) \right)$ for some $\lambda_1 > 0$ and $\lambda_2 > 0$. All other efficient solutions are called *non-supported*.

The *supported non-dominated points* lie on the lower-left boundary of the convex hull (*conv*(C)) of the feasible set C in outcome space of a biobjective optimization problem such as BMCF. The (continuous) BMCF problem, as stated in (1)-(3), only has supported efficient solutions, and $C = conv(C)$. If problem (1)-(3) is stated with the additional requirement that all flow variables are integer valued, then non-supported solutions may exist which lie in the interior of *conv(C)*. Both cases are illustrated in Figure 1.

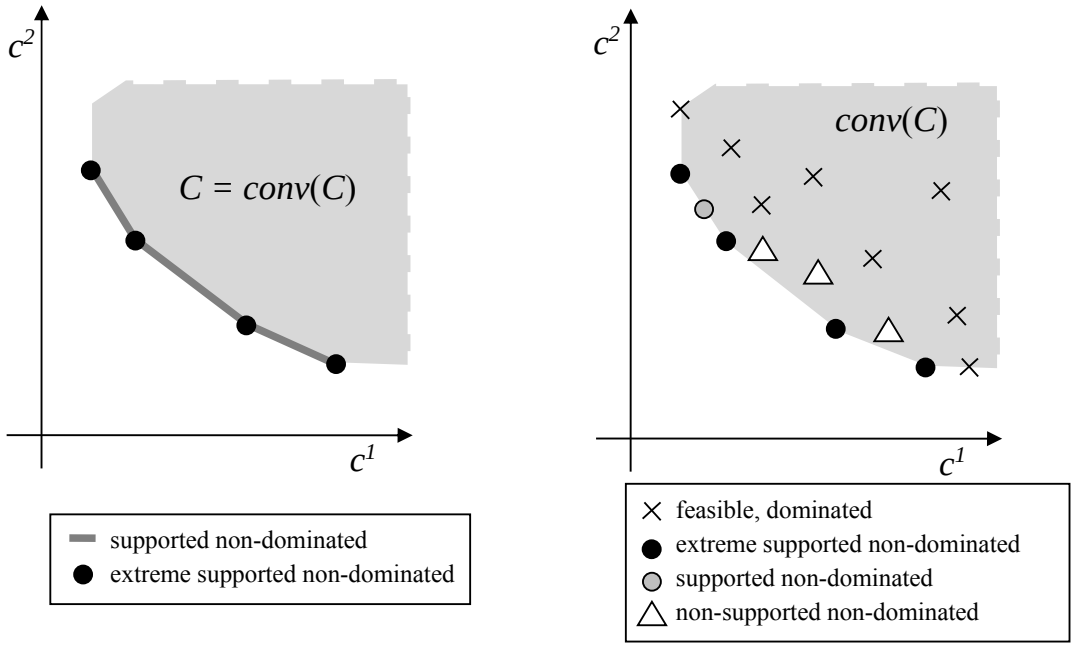

Figure 1. Illustration of *conv*(*C*) in outcome space in continuous (left) and integer case (right).

The focus of this paper is to design a fast algorithm to determine feasible solutions associated with *extreme supported non-dominated points* in the BMCF problem. Those feasible solutions are denoted *extreme supported efficient solutions*. That is, we determine supported efficient solutions whose images are extreme points of the convex hull of the supported non-dominated points. We ensure the proposed algorithm computes one extreme supported efficient solution for each extreme supported non-dominated point. Once those extreme supported efficient solutions are known, other (non-extreme) supported efficient solutions of BMCF can be obtained as convex combinations of extreme supported efficient solutions. This means we can derive a *complete* set of efficient solutions of BMCF from the obtained extreme supported efficient solutions, that is one supported efficient solution per supported non-dominated point in *C*.

3. Solving the BMCF problem with an enhanced parametric programming approach.

Instead of solving the weighted sum problem $\min_{x\in X}\left(\lambda_1 c^1(x)+\lambda_2 c^2(x)\right)$ with $\lambda_1 \geq 0, \lambda_2 \geq 0$, the problem $\min_{x\in X}\left(c^1(x)+\theta c^2(x)\right)$ with $\theta \in [0,+\infty)$ can be solved alternatively. For the linear programming formulation of BMCF, this leads to a *parametric linear program*.

Solving a biobjective linear program by solving its associated parametric linear program works by initially obtaining a lexicographically optimal solution to the problem, that is an

efficient solution is obtained which is optimal for $\theta = 0$ (see [16,17]). Such a lexicographically optimal solution is minimal with respect to c^1, with smallest possible c^2–value among minimizers of c^1. The parametric simplex method then solves the problem by iteratively pivoting variables while ensuring that all optimal solutions (again, one per optimal point in objective space) for increasing values of θ are obtained. These solutions correspond to supported efficient solutions of the biobjective linear program. This process continues until ultimately the other lexicographically optimal point, which minimizes c^2, is reached. An important aspect of the parametric simplex method is that all non-basic variables need to be inspected in every iteration to ensure that variables that are pivoted into the basis lead to increasing values of θ, and, most importantly, that a complete set of extreme supported solution is obtained. In the case of a network flow problem, a variant of the parametric method for network flow problems is used, the parametric network simplex method [3,5].

In the following it is explained how a lexicographically optimal solution is first found, and a discussion of enhancements of the pivoting steps follows.

3.1 *Solving* $\operatorname{lex}\min_{x\in X}\left(c^1(x), c^2(x)\right)$.

To compute an initial extreme supported efficient solution x^* we solve $\operatorname{lex}\min_{x\in X}\left(c^1(x), c^2(x)\right)$. In this way, we obtain the strongly feasible spanning tree structure (*T, L, U*) and the node potentials π^1 and π^2 (dual variables) with respect to the two objective functions (see, for example [1] and [15]). (*T, L, U*) refers to the following three sets: *T* is the basis (tree), *L* is the set of arcs $(i,j) \notin A(T)$ such that $x^*_{ij} = l_{ij}$ and *U* is the set of arcs $(i,j) \notin A(T)$ such that $x^*_{ij} = u_{ij}$. Any arc (i,j) in $A(T)$ satisfies $l_{ij} \le x^*_{ij} \le u_{ij}$. Clearly, the flow x^* must satisfy constraint (2). Based on node potentials π^1 and π^2, the reduced costs are defined as $\bar{c}^v_{ij} = c^v_{ij} - \pi^v_i + \pi^v_j$ for all $(i,j) \in A(T)$ and for $v \in \{1,2\}$. Then, a strongly feasible spanning tree structure (*T, L, U*) will be optimal for the $\operatorname{lex}\min_{x\in X}\left(c^1(x), c^2(x)\right)$ problem when the reduced costs and flows satisfy the following optimality conditions:

$$\bar{c}^1_{ij} > 0 \text{ or } (\bar{c}^1_{ij} = 0 \text{ and } \bar{c}^2_{ij} \ge 0) \quad \forall (i,j) \in L \qquad \text{(a)}$$
$$\bar{c}^1_{ij} < 0 \text{ or } (\bar{c}^1_{ij} = 0 \text{ and } \bar{c}^2_{ij} \le 0) \quad \forall (i,j) \in U \qquad \text{(b)}$$
$$\bar{c}^1_{ij} = 0 \text{ and } \bar{c}^2_{ij} = 0 \qquad \forall (i,j) \in A(T) \qquad \text{(c)}$$

It is easy to adapt the network simplex method to obtain an optimal strongly feasible spanning tree structure (*T, L, U*) and node potentials π^1 and π^2 satisfying (a), (b) and (c), for the $\operatorname{lex}\min_{x\in X}\left(c^1(x), c^2(x)\right)$ problem, see for example [1].

3.2 Obtaining a complete set of extreme supported efficient solutions.

Let x^* be an extreme supported efficient solution contained in the feasible set X in decision space ($c(x^*)$ is an extreme supported non-dominated point of the feasible set C in objective space) and (*T, L, U*), π^1 and π^2 be the corresponding optimal spanning tree structure and node potentials obtained after solving problem $\operatorname{lex}\min_{x\in X}\left(c^1(x), c^2(x)\right)$. The next step in the parametric network simplex method consists of finding an extreme non-dominated point of the objective space that is adjacent to $c(x^*)$. Note that $\bar{c}_{ij}^1 = 0$ and $\bar{c}_{ij}^2 = 0$ for any arc $(i, j) \in A(T)$ and that the arcs in L and U satisfy (a) and (b), respectively. The current optimal solution of the parametric linear program $\min_{x\in X}\left(c^1(x) + \theta c^2(x)\right)$ remains optimal as long as the reduced costs remain non-negative for the set L and non-positive for the set U. This is the case as long as the following remains true:

$$\begin{aligned} \bar{c}_{ij}^1 + \theta \bar{c}_{ij}^2 &\geq 0 \quad \forall (i, j) \in L \\ \bar{c}_{ij}^1 + \theta \bar{c}_{ij}^2 &\leq 0 \quad \forall (i, j) \in U \end{aligned},$$

where the value of θ must be non-negative, $\theta \in [0, +\infty)$. Moreover, we have $\bar{c}_{ij}^1 + \theta \bar{c}_{ij}^2 = 0$ for all $(i, j) \in A(T)$. To shorten our exposition and be able to make similar statements for both the arcs in L and U, we define modified reduced costs for any arc (i, j) in these sets:

$$\hat{c}_{ij}^v = \begin{cases} \bar{c}_{ij}^v & \text{if } (i, j) \in L \\ -\bar{c}_{ij}^v & \text{if } (i, j) \in U \end{cases} \text{ with } v \in \{1, 2\}$$

In order to move from the current efficient solution to another efficient one the first objective has to worsen, or $\hat{c}_{ij}^1 > 0$, and the second one has to improve, or $\hat{c}_{ij}^2 < 0$. Note that the flow of the arcs in L increases and the flow of the arcs in U decreases when they are

introduced into *T*, unless in the case where there is no flow change. Therefore, structure (*T, L, U*) remains optimal for all θ in the range $0 \le \theta \le \theta^k$ where $\theta^k = \min_{(i,j)\in A}\left\{\theta_{ij}^k\right\}$ with

$$\theta_{ij}^k = \begin{cases} -\hat{c}_{ij}^1 / \hat{c}_{ij}^2 & \text{if } \hat{c}_{ij}^2 < 0 \\ +\infty & \text{otherwise} \end{cases}. \qquad (4)$$

That is, if $\hat{c}_{ij}^2 < 0$ the ratio associated with arc (i, j) in iteration *k* (where *k* = 1 initially) is $-\hat{c}_{ij}^1 / \hat{c}_{ij}^2$; otherwise this ratio is infinite. We refer to the interval of values of θ for which an efficient solution remains optimal as *optimality interval.* To determine the (strongly feasible) spanning tree corresponding to the next supported non-dominated point, the next arc to be introduced into the basis is identified as:

$$(x, y) = \arg\operatorname{lex}\min_{(i,j)\in L\cup U}\left\{(\theta_{ij}^k, \hat{c}_{ij}^2) : \theta_{ij}^k < +\infty\right\}.$$

That is, we must determine an arc with minimum finite ratio (4), which then enters the basis (tree *T*), and an arc from the current basis (tree *T*) leaves it. We call this operation *pivot operation* (see [1]). It is also possible that this pivot operation leads to the same arc entering and leaving the basis. In this case, the arc with minimum ratio moves from *L* to *U* or *vice versa*. If more than one arc with minimum ratio exists, we select one with smallest (or most negative) value of $\hat{c}_{ij}^2$ (this is why lexmin appears in the above expression). Then, a pivot operation is performed with the entering arc (x, y) giving the next efficient solution with index *k* + 1. We have the following result:

Theorem 1. *The set of extreme supported non-dominated points of the BMCF problem can be determined starting from an optimal tree structure minimizing* $\operatorname{lex}\min(c^1(x), c^2(x))$ *and making a sequence of pivot operations where the entering arc is* (x, y)*, with* $(x, y) = \arg\operatorname{lex}\min_{(i,j)\in A}\left\{(\theta_{ij}, \bar{c}_{ij}^2) : \theta_{ij} < +\infty\right\}$*, until the ratio of any arc in A is* $+\infty$*.*

Proof. Immediate from parametric linear programming theory (see also [3,5,18]).

Now, once a pivot operation is performed with the arc (x, y) and the node potentials are updated, we know that $\theta^k \le \theta^{k+1}$ from Dantzig and Thapa [18]. The process of identifying an

arc with minimum ratio (4) continues in the standard parametric network simplex method by inspecting, in each iteration, the ratio of all non-basic arcs. In our enhancement shown in the following it is analysed how ratios change to identify situations in which ratios do not have to be re-computed. For instance, in [3] it is shown that any arc (u,v) with ratio $\theta_{uv}^k = \theta^k$ before the pivot operation keeps the minimum ratio value (unless it becomes ineligible to enter the basis), that is, $\theta^k = -\dfrac{\hat{c}_{uv}^1}{\hat{c}_{uv}^2}$, where $\hat{c}_{uv}^1$ and $\hat{c}_{uv}^2$ are the new reduced costs for the arc (u,v) once the pivot operation is made. If (u,v) becomes ineligible to enter the basis, due to $\hat{c}_{uv}^1 \le 0$ and $\hat{c}_{uv}^2 \ge 0$ then $\theta_{uv}^{k+1} = +\infty$ in our definition of the ratios (4). Instead of following the standard steps of a parametric network simplex method from here, we further investigate how the new ratios for all arcs in *L* and *U* change, once a pivot operation with the entering arc (x,y) is completed.

In order to compute the new ratios when a pivot operation is performed, we need to recall how the node potentials change (see page 419, [1]) once this operation is made. Let *T* be the current basis tree, (x,y) the entering arc and (p,q) the leaving arc. Assume that $(x,y) \neq (p,q)$ then we update the node potentials as follows. The deletion of the arc (p,q) from *T* partitions the tree into two subtrees, one, T_1, containing the root node and the other, T_2, the complementary set containing nodes $V(T) - V(T_1)$. Then the node potentials are updated according to one of two cases. We use the update process A or B when appropriate (this is shown later).

(A) If $x \in V(T_1)$ then $\pi_i^v = \pi_i^v - \bar{c}_{xy}^v$ for all $i \in V(T_2)$ with $v \in \{1,2\}$. If $x \notin V(T_1)$ then $\pi_i^v = \pi_i^v + \bar{c}_{xy}^v$ for all $i \in V(T_2)$ with $v \in \{1,2\}$. (This update procedure of T_2 is called *process* A).

Alternatively, the following update of node potentials is applied.

(B) If $x \in V(T_1)$ then $\pi_i^v = \pi_i^v + \bar{c}_{xy}^v$ for all $i \in V(T_1)$ with $v \in \{1,2\}$. If $x \notin V(T_1)$ then $\pi_i^v = \pi_i^v - \bar{c}_{xy}^v$ for all $i \in V(T_1)$ with $v \in \{1,2\}$. (This update procedure of T_1 is called *process* B).

In summary, process A updates potentials of nodes $i \in V(T_2)$ by subtracting the reduced costs of arc (x, y) if $x \in V(T_1)$, and adding them if $x \notin V(T_1)$, whereas process B updates potentials of $i \in V(T_1)$ by adding the reduced costs of arc (x, y) if $x \in V(T_1)$, and subtracting them otherwise.

Now, we analyze the different cases of the computation of new ratios for all arcs $(i, j) \notin A(T)$ when $A(T) = A(T) - \{(p, q)\} + \{(x, y)\}$ with $(x, y) \neq (p, q)$. From the calculations shown in *Appendix A* it can be observed under which circumstances we have $\theta_{ij}^{k+1} \geq \theta_{xy}^{k}$ for any arc (i, j) in set *L* or *U*. Moreover, Table 1 summarizes the relationship between the ratio of arc (i, j) before and after a pivot operation with the entering arc (x, y) is made. The different cases arise taking into account the membership of arcs (i, j) and (x, y) of the sets *L* and *U*. In addition, it is necessary to distinguish which set $V(T_1)$ or $V(T_2)$ contains the nodes *x*, *i* and *j*. In Table 1 the node potential update process that is carried out in each case is also shown.

Table 1. Relationship between the ratios of arc (i, j) before and after node potential update operation.

Set for node x		$x \in V(T_1)$		$x \in V(T_2)$	
Entering arc		$(x, y) \in L$	$(x, y) \in U$	$(x, y) \in L$	$(x, y) \in U$
Update potentials process		A	B	B	A
$(i, j) \in L$	$i \in V(T_2)$, $j \in V(T_1)$	$\theta_{ij}^{k+1} \leq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \geq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \geq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \leq \theta_{ij}^{k}$
	$i \in V(T_1)$, $j \in V(T_2)$	$\theta_{ij}^{k+1} \geq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \leq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \leq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \geq \theta_{ij}^{k}$
$(i, j) \in U$	$i \in V(T_2)$, $j \in V(T_1)$	$\theta_{ij}^{k+1} \geq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \leq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \leq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \geq \theta_{ij}^{k}$
	$i \in V(T_1)$, $j \in V(T_2)$	$\theta_{ij}^{k+1} \leq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \geq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \geq \theta_{ij}^{k}$	$\theta_{ij}^{k+1} \leq \theta_{ij}^{k}$

The selection of the update process A or B can be extracted from Table 1. The table also summarizes changes of ratios (4) after node update process A or B are applied. For example, for any arc $(i, j) \in L$ such that $\theta_{ij}^{k+1} \geq \theta_{ij}^{k}$ the node potential of node *j* was modified. Additionally, for any arc $(i, j) \in U$ such that $\theta_{ij}^{k+1} \geq \theta_{ij}^{k}$ the node potential of node *i* was modified. The new ratios for the remaining arcs keep their previous value or decrease. The observations listed in Table 1 will allow us to identify for each node *i* of *V* a subset of non-tree arcs that must be explored whenever the node potential of node *i* changes, and others that do not need to be considered (as their ratio only increases). This relation will be used in

Algorithm 1 introduced in Section 4 to limit the number of ratios associated with arcs that need to be computed in each iteration.

4. A ratio-labeling algorithm for BMCF problem.

Assume an optimal solution (*T*, *L*, *U*) of the lexicographic optimization problem and the associated node potentials π^1 and π^2 with respect to the two objective functions are given. The previous results allow us to develop Algorithm 1, which is introduced in detail below.

Let $A_i = \{\{(j,i) \in L\} \cup \{(i,j) \in U\}\}$ be the set of incoming arcs at node *i* in the set *L* plus the set of outgoing arcs from node *i* in the set *U*. That is, A_i contains the set of non-tree arcs that can carry flow to node *i* when a pivot operation is made with this arc as entering arc. This set allows to associate the following label with any node *i* in *V*:

$$\left(\theta_i, \hat{c}_i^2\right) = \operatorname*{lex\,min}_{(u,v)\in A_i} \left\{\left(\theta_{uv}, \hat{c}_{uv}^2\right)\right\}$$

This label θ_i indicates the current minimum ratio of any arc in set A_i associated with node *i*. Remember that $\theta_{uv} = -\hat{c}_{uv}^1 / \hat{c}_{uv}^2$ is the ratio of the arc (*u*,*v*) and takes a finite value if and only if $\hat{c}_{uv}^2 < 0$. The implementation of Algorithm 1 uses a heap *H* to store the labels of each node *i* in *V*. That is, the algorithm stores in *H* the minimum ratio θ_i attained for the arcs in A_i, for any node *i*. Here $\hat{c}_i^2 = \hat{c}_{uv}^2$ for the arc (u,v) in A_i that has the minimum ratio, that is, with $\theta_i = \theta_{uv}$. In case that two or more arcs in A_i have the same value of θ_i, the algorithm stores the ratio of an arc with smallest value of the second reduced cost, that is, $\hat{c}_i^2 = \min_{(u,v)\in A_i} \left\{\hat{c}_{uv}^2 : \theta_{uv} = \theta_i\right\}$. The key of each element in *H* is the pair of values $(\theta_i, \hat{c}_i^2)$. Additionally, the algorithm tracks for each candidate node *i* the following information: the ratio θ_i, the reduced costs $\hat{c}_i = (\hat{c}_i^1, \hat{c}_i^2)$, the candidate predecessor J_i (to identify the entering arc) and the label Low_i indicating if the minimum is attained in an arc in set *L* (Low_i = True) or in an arc in set *U* (Low_i = False) to be able to update labels easily. The following heap operations are needed in our algorithm (see [19]): *CreateHeap*(*H*), *Insert*({*label*}, *H*), *Find-min*(*H*), *Decrease-key*({*label*}, *H*), and *Delete-min*(*H*).

The Ratio-Labeling BMCF (RLBMCF) algorithm (Algorithm 1) maintains a heap with minimum ratio associated with each node. When a new entering variable (arc) is selected in each iteration, it is not necessary to compute the arc's ratio for every arc as in the standard

parametric network simplex method. Instead a minimum label is extracted from the heap representing the node (and associated arc) with minimum ratio. Since we identified which arcs need updating as their ratio could increase in the previous section this can be exploited when node labels are updated in each iteration. This leads to fewer ratio calculations (or the same number of them). The algorithm is outlined in the following, and pseudo-code is shown below.

The RLBMCF algorithm (Algorithm 1) starts by solving the lexmin problem (line (1)). The image of this first extreme supported efficient solution is stored in the variables c^1 and c^2. This solution is printed on the screen (or stored), although its optimality interval (or at least its upper limit) remains unknown. We use the variable *lastratio* to store the lower limit of the optimality interval of the current solution $\overline{x}$. Initially, *lastratio* is zero. Next, the heap *H* is created and the labels for all nodes *i* in *V* are calculated by calling Procedure 1 *ComputeLabels* (Lines (2), (4) and (5)). Now, the algorithm starts a loop that ends when the heap *H* is empty. In each iteration, the node *i* with minimum ratio is extracted from the heap. This operation allows to identify the entering arc (x, y) (Line (8)). The leaving arc (p, q) and the maximum amount of flow δ that can be sent along the cycle $A(T) \cup (x, y)$ are identified in Line (9), a standard operation in a network simplex method. Next, the current flow $\overline{x}$ and the structure (*T*, *L*, *U*) are modified by a standard pivot operation (lines (10) and (11)). At this point, we know the image of the next supported efficient solution but it does not have to be an extreme supported efficient solution. If the ratio θ_i is greater than *lastratio* and the value of δ is greater than 0, the algorithm has identified the next extreme supported efficient solution $\overline{x}$ and the upper limit of the optimality interval of the previous extreme solution is θ_i. Lines (13) and (14) print (or store) this information. If the entering arc equals the leaving arc, only a new label for the candidate node *i* extracted from the heap needs to be computed (line (15)). Otherwise, we must identify the set *S* that contains the nodes whose node potentials need to be modified. Note that *S* is $V(T_2)$ or $V(T_1)$ when process A or process B, respectively, must be applied. The identification of this set (in the pseudo-code) follows the characterization in Table 1 (see Lines (17)-(19)). We also show an alternative approach to identifying *S* in Procedure 2 *IdentifyS*. Once the set *S* is identified, the node potential of any node in this set decreases by $\hat{c}_i$ (line (20)).

Algorithm 1 *Ratio-Labeling BMCF* (RLBMCF)

(1) Let $\overline{x}$, (T, L, U), $\pi = (\pi^1, \pi^2)$ be the information of the optimal solution of $\operatorname{lex}\min_{x \in X}\left(c^1(x), c^2(x)\right)$;

(2) *CreateHeap*(H); *lastratio* $= 0$; $c^1 = c^1(\overline{x})$; $c^2 = c^2(\overline{x})$;

(3) **Print** the current flow $\overline{x}$ with objective $\left(c^1, c^2\right)$ as an extreme supported efficient flow;

(4) Set $\theta_i = +\infty; J_i = 0; Low_i = True$; for all $i \in V$;

(5) **For** all $i \in V$ **do** *ComputeLabels*($i, \theta_i, \hat{c}_i^1, \hat{c}_i^2, J_i, Low_i, H$);

(6) **While** ($H \neq \varnothing$) **do**

(7) $\left\{\theta_i, \hat{c}_i^2, i\right\}$ = *Find-min*(H); *Delete-min*(H);

(8) **If** ($Low_i == True$) **then** $(x, y) = (i, J_i)$ **Else** $(x, y) = (J_i, i)$;

(9) Let (p,q) be the leaving arc and δ the maximum amount of flow that can be sent along the cycle $A(T) \cup (x, y)$;

(10) $c^1 = c^1 + \hat{c}_i^1 \delta$; $c^2 = c^2 + \hat{c}_i^2 \delta$;

(11) $A(T) = A(T) - (p, q) + (x, y)$; Update L, U; // Make a pivot operation

(12) **If** ($\theta_i >$ *lastratio*) and ($\delta > 0$) **then**

(13) **Print** the optimal interval of the current flow is [lastratio, θ_i]; *lastratio* $= \theta_i$;

(14) **Print** the current flow $\overline{x}$ with objective $\left(c^1, c^2\right)$ as an extreme supported efficient flow;

(15) **If** ($(x, y) == (p, q)$) **then** *ComputeLabels*($i, \theta_i, \hat{c}_i^1, \hat{c}_i^2, J_i, Low_i, H$);

(16) **Else**

(17) Identify T_1 and T_2 in $T - (p, q)$;

(18) **If** ((($x \in V(T_1)$) **and** ($Low_i == False$)) **or** (($x \in V(T_2)$) **and**($Low_i == True$))) **then** $S = V(T_1)$;

(19) **Else** $S = V(T_2)$;

(20) **For** all $k \in S$ **do** $\pi_k = \pi_k - \hat{c}_i$;

(21) **For** all $k \in S$ **do** *ComputeLabels*($k, \theta_k, \hat{c}_k^1, \hat{c}_k^2, J_k, Low_k, H$);

(22) **For** all $k \in S$ **do**

(23) **For** all $j \in \Gamma_k^+ \cap (V - S)$ **do**

(24) **If** ($(k, j) \in L$) and ($\overline{c}_{kj}^2 < 0$) and ($\overline{c}_{kj}^1 > 0$)) **then**

(25) **If** (($-\overline{c}_{kj}^1 / \overline{c}_{kj}^2 < \theta_j$) **or** (($-\overline{c}_{kj}^1 / \overline{c}_{kj}^2 == \theta_j$) **and** ($\overline{c}_{kj}^2 < \hat{c}_j^2$))) **then**

(26) **If** ($J_j == 0$) **Then** *Insert*($\{-\overline{c}_{kj}^1 / \overline{c}_{kj}^2, \overline{c}_{kj}^2, j\}$, H);

(27) **Else** *Decrease-key*($\{-\overline{c}_{kj}^1 / \overline{c}_{kj}^2, \overline{c}_{kj}^2, j\}$, H);

(28) $\theta_j = -\overline{c}_{kj}^1 / \overline{c}_{kj}^2; \hat{c}_j = \overline{c}_{kj}; J_j = k; Low_j = True$;

(29) **For** all $j \in \Gamma_k^- \cap (V - S)$ **do**

(30) **If** ($(j, k) \in U$) and ($\overline{c}_{jk}^2 > 0$) and ($\overline{c}_{jk}^1 < 0$)) **then**

(31) **If** (($-\overline{c}_{jk}^1 / \overline{c}_{jk}^2 < \theta_j$) **or** (($-\overline{c}_{jk}^1 / \overline{c}_{jk}^2 == \theta_j$) **and** ($-\overline{c}_{jk}^2 < \hat{c}_j^2$))) **then**

(32) **If** ($J_j == 0$) **Then** *Insert*($\{-\overline{c}_{jk}^1 / \overline{c}_{jk}^2, -\overline{c}_{jk}^2, j\}$, H);

(33) **Else** *Decrease-key*($\{-\overline{c}_{jk}^1 / \overline{c}_{jk}^2, -\overline{c}_{jk}^2, j\}$, H);

(34) $\theta_j = -\overline{c}_{jk}^1 / \overline{c}_{jk}^2; \hat{c}_j = -\overline{c}_{jk}; J_j = k; Low_j = False$;

(35) **Print** the optimal interval of the current flow is [lastratio, $+\infty$];

Next, Procedure 1 *ComputeLabels* computes the minimum ratio and the corresponding label for a node *k* whenever the value of its node potential changed (line (21)). Note that the new ratio for each node *k* in *S* keeps its previous value or increases following the arguments in Section 3.2. If the label changes, we need to make a delete operation in the heap to delete node *k* (and the label of node *k*) whenever node *k* is in the heap. This operation is implemented as the two operations in Line (2) of Procedure 1 *ComputeLabels*. The lines (22) to (34) of the RLBMCF algorithm check if the ratios of the nodes $j \in V - S$ decrease when the arcs (*k*, *j*) and (*j*, *k*) are investigated for all *k* in *S*, again taking into account the characterization summarized in Table 1. In this case, the corresponding *Insert*({*label, j*}, *H*) or *Decrease-key*({*label, j*}, *H*) operation must be performed. Finally, once the heap is empty, the algorithm prints (or stores) the last optimality interval (line (35)).

Procedure 1 $ComputeLabels(i, \theta_i, \hat{c}_i^1, \hat{c}_i^2, J_i, Low_i, H)$;

(1) **If** $(J_i \neq 0)$ **then** // since the ratio of node i must be re-computed, it must be deleted from H;
(2) $Decrease\text{-}key(\{-1, 0, i\}, H)$; $Delete\text{-}min(H)$;
(3) $\theta_i = +\infty; J_i = 0;$
(4) **For** all $j \in \Gamma_i^-$ **do**
(5) **If** $((j,i) \in L)$ and $(\overline{c}_{ji}^2 < 0)$ and $(\overline{c}_{ji}^1 > 0))$ **then**
(6) **If** $((-\overline{c}_{ji}^1 / \overline{c}_{ji}^2 < \theta_i)$ **or** $((-\overline{c}_{ji}^1 / \overline{c}_{ji}^2 == \theta_i)$ **and** $(\overline{c}_{ij}^2 < \hat{c}_i^2)))$ **then**
(7) $\theta_i = -\overline{c}_{ji}^1 / \overline{c}_{ji}^2; \hat{c}_i = \overline{c}_{ji}; J_i = j; Low_j = True;$
(8) **For** all $j \in \Gamma_i^+$ **do**
(9) **If** $((i,j) \in U)$ and $(\overline{c}_{ij}^2 > 0)$ and $(\overline{c}_{ij}^1 < 0))$ **then**
(10) **If** $((-\overline{c}_{ij}^1 / \overline{c}_{ij}^2 < \theta_i)$ **or** $((-\overline{c}_{ij}^1 / \overline{c}_{ij}^2 == \theta_i)$ **and** $(-\overline{c}_{ji}^2 < \hat{c}_i^2)))$ **then**
(11) $\theta_i = -\overline{c}_{ij}^1 / \overline{c}_{ij}^2; \hat{c}_i = -\overline{c}_{ij}; J_i = j; Low_j = False;$
(12) **If** $(J_i \neq 0)$ **then** $Insert(\{\theta_i, \hat{c}_i^2, i\}, H)$;

In the implementation of the RLBMCF algorithm, the operations in line (21) and lines (22)-(34) are performed concurrently so that the sets of successor and predecessor nodes of any node *k* in *S* are examined only once. Also, the operation modifying potentials is implemented together with the operation identifying the set $V(T_1)$ or $V(T_2)$ (set *S* in general). In other words, lines (17)-(20) in the algorithm are implemented simultaneously. The scheme uses Procedure 2 *IdentifyS* based on a Depth-First Search (DFS) approach (see [1]) with a vector *InS* which takes the value 1 in position *k* if and only if node *k* belongs to *S*.

Procedure 2 $IdentifyS(k,\ end\ ,\ change)$;

(1) $InS_k = 1\ ;\ \pi_k = \pi_k - change\ ;$

(2) **For** all $(k, j) \in A(T)$ **do If** $((\ InS_j == 0\ )$ **and** $(\ j \neq end\ ))$ **Then** $IdentifyS(j,\ end,\ change)$;

(3) **For** all $(j,k) \in A(T)$ **do If** $((\ InS_j == 0\ )$ **and** $(\ j \neq end\ ))$ **Then** $IdentifyS(j,\ end,\ change)$;

The pivot process swapping the entering arc (x, y) and the leaving arc (p, q) appears in line (11) of Algorithm 1. The algorithm keeps *depth* labels for the current tree *T*. These labels inform us of the depth of each node from the root node in the new tree. For example, if *depth*[*x*] < *depth*[*y*], we know the node *y* will be situated in the subtree of node *x* in the new tree. The *depth* and *Low* labels (Low_i of the candidate node *i* where *i* could be *x* or *y*) are used to identify whether the process A or B must be applied, following the cases in Table 1. Table 2 summarizes the cases. The first two rows of Table 2 show the relationship between the cases in Table 1 and the corresponding values of *Low* and *depth* labels. The third row in Table 2 shows how Procedure 2 *IdentifyS* is called in each case.

Table 2. Summary of how the procedure *IdentifyS* must be called to determine the set *S*.

$x \in V(T_1)$ and $(x, y) \in L$ update process is A	$x \in V(T_1)$ and $(x, y) \in U$ update process is B	$x \in V(T_2)$ and $(x, y) \in L$ update process is B	$x \in V(T_2)$ and $(x, y) \in U$ update process is A
Low_y = True and *depth*[*x*] < *depth*[*y*] in *T*	Low_x = False and *depth*[*x*] < *depth*[*y*] in *T*	Low_y = True and *depth*[*x*] > *depth*[*y*] in *T*	Low_x = False and *depth*[*x*] > *depth*[*y*] in *T*
IdentifyS(*root*, *y*, *change*) Finds $S = V(T_1)$	*IdentifyS*(*y*, *x*, *change*) Finds $S = V(T_2)$	*IdentifyS*(*x*, *y*, *change*) Finds $S = V(T_2)$	*IdentifyS*(*root*, *x*, *change*) Finds $S = V(T_1)$

Procedure 2 *IdentifyS* is called following the scheme of Table 2 where the appropriate choice of arguments *k* and *end* is given in the function call of *IdentifyS*. The node *end* and the nodes in the subtree of node *end* are never reached in the search process. The parameter *change* refers to $\hat{c}_i = (\hat{c}_i^1, \hat{c}_i^2)$. Any other variable used in this procedure is a global variable (such as *T*, *InS*, and the node potentials).

4.1 Worst-case complexity of the RLBMCF algorithm.

We consider that any heap operation takes constant time with the exception of the *Delete-min* operation which requires O(log*n*) time when a Fibonacci heap is used (see [19]). The worst-case complexity of Procedure 1 *ComputeLabels* is $O(\log n + |\Gamma_i^-| + |\Gamma_i^+|)$ time since it performs one *Delete-min* operation and examines the set of the predecessor and the successor nodes of node *i*. Also, the complexity of Procedure 2 *IdentifyS* is O(*n*) time, because in the

worst-case all nodes are considered and the number of arcs in $A(T)$ is n-1. Now, if we denote by N the number of extreme supported non-dominated points in the outcome space of the BMCF problem we obtain Theorem 2.

Theorem 2. *The RLBMCF algorithm runs in* O(*MCF*+ *Nn*(*m*+*n*log*n*)) *time and uses* O(*n*+*m*) *space*.

Proof. The solution of the $\operatorname{lex}\min_{x\in X}\left(c^1(x), c^2(x)\right)$ problem requires the same time as solving the MCF problem using a state-of-the-art algorithm (see [1,2]). We denote this effort by O(*MCF*). Note that the size of the heap H in the RLBMCF algorithm is at most n as the label associated with a node is contained in the heap at most once at any point during the algorithm's execution. The number of iterations performed by the algorithm is O(*Nn*) since the ratio of a node can take, in the worst-case, N different *minimum* values, that is, as many values as the number of different extreme supported non-dominated points in the outcome space. All pivot operations require O(n) time. Therefore, the greatest time employed in an iteration is due to the execution of Lines (21)-(34). This corresponds to $O(n+\sum_{k\in S}(\log n+\left|\Gamma_k^-\right|+\left|\Gamma_k^+\right|))$, where the $\log n$ term corresponds to the *Delete-min* operations in Procedure 1 *ComputeLabels*. We have mentioned that in the implementation the execution of line (21) and Lines (22)-(34) is done simultaneously. The term O(n) is the time spent in lines (9), (11), (17) and (20). Remember that Procedure 2 *IdentifyS* performs (17) and (20), simultaneously. The time $O(\sum_{k\in S}(\left|\Gamma_k^-\right|+\left|\Gamma_k^+\right|))$ corresponds to the effort to scan the arcs leaving from and arriving at any node k in S (including the effort in Procedure 1 *ComputeLabels*). In each iteration, the algorithm requires an additional O(m) time to print the flows. Thus, in the worst-case, the complexity of the algorithm is $O(\sum_{iter=1}^{nN}(m+n+\sum_{k\in S}(\log n+\left|\Gamma_k^-\right|+\left|\Gamma_k^+\right|)))$. Therefore, the worst-case time complexity of the algorithm is O(*MCF*+ *Nn*(*m*+*n*log*n*)) since $\left|S\right| < n$ in any iteration. Finally, it is easy to observe that the space used by the algorithm is O(n+m) since only the current extreme supported efficient flow x is stored. □

Often, in practice, the size of the set S is much smaller than n. Note that our algorithm scans all the arcs in the network twice when $\left|S\right|$ is close to n, that is, it takes $2m$ effort in the operations in Lines (21)-(34). However, the classical parametric method always performs m

operations when it examines all arcs to find the minimum reduced cost ratio. In a particular iteration, whenever $|S|$ is small compared to n, the proposed algorithm requires O(n) time (here we are not considering the time to print the flow). In this case, the proposed method has an advantage compared to the parametric method. For this reason, Lines (21)-(34) are performed when $|S| < n/8$ in the implementation of the RLBMCF algorithm. Otherwise, the whole set of entering arcs is considered once following the procedure applied by the parametric method.

5. Computational Results

In this section, we examine the performance of the presented RLMCF algorithm.

5.1 Problem Instances

Test instances were generated based on the single-objective minimum cost flow (MCF) problems used in [2]. This extensive computational comparison of MCF algorithms considers many different algorithms and implementations (a total of 15 solvers), and tests these on a large set of problem instances of varying characteristics and size, enabling the analysis of asymptotic behavior. A main conclusion is that cost-scaling algorithms and the primal network simplex method perform best in general (with exceptions for special network types). Test problems are generated using standard generators, namely NETGEN, GRIDGEN, GOTO, GRIDGRAPH, and other networks are also tested in [2]. Test problems are available online[*] and form the basis of our computational analysis. All instances involve solely integer data.

Problem instance characteristics from [2] are summarized in Table 3. In the instances either the number of nodes n or the number of arcs m varies. A certain number of nodes acts as supply and demand nodes, shown as #supp in the table. For GRIDGRAPH instances the number of rows (W) and the number of columns (L) of the grid are listed. The overall supply in the network is *Supp*. The network generators select costs and capacities randomly and uniformly between 1 and the maximum cost and capacity listed in Table 3. There are five randomly generated instances per set of problem parameters.

The original problem instances from [2] have a single objective function. We generate biobjective problems as follows. From the five instances for each set of parameters, denoted *a, b, c, d* and *e* we generate biobjective instances by combining the costs of pairs of the

[*] http://lemon.cs.elte.hu/trac/lemon/wiki/MinCostFlowData

original instances: We combine an original instance, say instance *a*, with the costs from another instance, say instance *b*, obtaining one new biobjective instance, instance *ab* in this case. We obtain five new instances with combined cost coefficients for each set of problem parameters: *ab, bc, cd, de*, and *ea*. We use a much broader set of test instances than is usually applied to test BMCF algorithms where most researchers use NETGEN instances only.

Table 3. Summary of problem instance characteristics from [2].

	n	*m*	#supp	*Supp*	Max cost	Max cap	type
NETGEN-8	$n = 2^i, i = 8,...,15$	$8n$	$\sqrt{n}$	10^3	10^4	10^3	sparse
NETGEN-SR	$n = 2^i, i = 8,...,12$	$\sim n\sqrt{n}$	$\sqrt{n}$	10^3	10^4	10^3	dense
NETGEN-LO-8	$n = 2^i, i = 8,...,15$	$8n$	$\sqrt{n}$	10	10^4	10^3	low supply
NETGEN-LO-SR	$n = 2^i, i = 8,...,12$	$\sim n\sqrt{n}$	$\sqrt{n}$	10	10^4	10^3	low supply
NETGEN-DEG	4,096	$4n$ to n^2	$\sqrt{n}$	10^3	10^4	10^3	increasing density
GRIDGEN-8	$n = 2^i, i = 8,...,15$,	$m = 8n$	$\sqrt{n}$	10^3	10^4	10^3	sparse
GRIDGEN-SR	$n = 2^i, i = 8,...,13$	$\sim n\sqrt{n}$	$\sqrt{n}$	10^3	10^4	10^3	dense
GRIDGEN-DEG	4,096	$4n$ to n^2	$\sqrt{n}$	10^3	10^4	10^3	increasing density
GOTO-8	$n = 2^i, i = 8,...,15$	$8n$	1	Increase with *n*: ~60,000-210,000	10^4	10^3	sparse
GOTO-SR†	$n = 2^i, i = 8,...,12$	$\sim n\sqrt{n}$	1	Increase with *n*: ~10,000-1,750,000	10^4	10^3	dense
GRIDGRAPH-WIDE	$n = WL + 2$; $L = 16$; $W = 16 \cdot 2^i, i = 0,1,...,$	~2*n*	1	Increase with *W*: ~3,000 to ~4,230,000	10^4	10^3	wide; very sparse
GRIDGRAPH-LONG	$n = WL + 2$; $W = 16$ $L = 16 \cdot 2^i, i = 0,1,...,13$	~2*n*	1	*Decrease* with *L*: ~3,000 (smallest) to ~16 (largest)	10^4	10^3	long; very sparse
GRIDGRAPH-SQUARE	$n = WL + 2$; $L = W \sim \sqrt{WL}$ for *W*, *L* as other GRIDGRAPH instances	~2*n*	1	Increases with $L = W$; ~3,000 to ~70,000	10^4	10^3	square; very sparse
ROAD 01-04	9,559 / 49,109 / 116,920 / 261,155	29,766 / 120,576 / 265,402 / 620,924	6 - 50	~10,000 - ~45,000	10^{5}‡	10^3	Real; very sparse

To include more problem instances based on real-world network data to the computational experiments, we adapt some of the social network datasets from the Stanford Large Network

† Note that the overall supply for all GOTO-SR instances was scaled by a factor of 1/10.

‡ Both physical distance and transit time are available for ROAD networks from [20], these two arc costs are used in our biobjective instances, and there is only one network instance for each of the four road networks. For each road network five sets of instances are considered each with different supply and demand.

Dataset Collection [23]. The social network datasets for facebook and wikipedia votes are used in the following. As these instances only describe the network structure, the arc capacities, arc costs and node supply / demand must be created. Capacities are uniformly distributed between 0 and 100. Supply and demand is randomly assigned to nodes, and we vary the total supply in the network to be 100, 500, 1,000, 2,500, 5,000 and 10,000. The first arc cost component is always uniformly distributed between 0 and 100. We create three types of cost instances: random (both costs uniformly distributed), correlated and anti-correlated. For the correlated instances the second cost is normally distributed with mean c^1 and standard deviation 10, whereas the mean is 100 - c^1 in the anti-correlated instances. We create 5 problem instances per set of problem parameters with the same network but capacities, costs and flow balances are randomly generated for each instance. To ensure feasibility of instances we also add a cycle connecting all the nodes in the network with capacity 10,000 and costs 10,000 to penalize their use. The WIKI-VOTE network is directed, but the FACEBOOK network is undirected. We hence convert FACEBOOK to a directed network by creating two directed arcs for each undirected one. Table 4 lists instance characteristics, where #supp is the average number of supply nodes.

Table 4. Summary of problem instance characteristics from [23].

	n	*m*	#supp	*Supp*	Max cost	Max cap	type
FACEBOOK	4,039	180,507	~2, 10, 20, 50, 100, 200	100, 500, 1,000; 2,500; 5,000; 10,000	100	100	Real; dense
WIKI-VOTE	8,298	111,987	~2, 10, 20, 50, 100, 200	100, 500, 1,000; 2,500; 5,000; 10,000	100	100	Real; dense

5.2 Implementation issues and computational setup

RLBMCF builds on the idea of the parametric network simplex method for biobjective network flow problems, but reduces the number of arcs that need to be scanned to identify the entering variable (arc) in each iteration of the parametric network simplex method. To establish performance of RLBMCF we compare it to an implementation of the parametric network simplex method, denoted Para in the following, implemented as described for the Phase 1 algorithm in [3], which builds on an implementation of the single-objective network simplex method called MCF [21]. Since a complete set of extreme supported solutions of BMCF can also be found as solutions of a sequence of single-objective weighted sum problems, we also explore this so-called dichotomic approach as described, for instance, in [22]. This approach is denoted as Dicho in the following. The single-objective problems in

Dicho are again solved by the MCF implementation [21] with the advantage that speed-up techniques such as partial pricing can be used. Both Para and RLBMCF do not need to store flow solutions as the algorithm runs, instead solutions can be output or stored as soon as they are obtained. Dicho on the other hand is required to manage solutions for later exploration in subsequent single-objective solves which may pose additional effort.

We test two implementations of RLBMCF. RLBMCF^f is the direct implementation of the proposed algorithm using a binary heap with real-valued keys, requiring floating point arithmetic for comparison of ratios (4). Since the costs in all our instances are integer valued, we have implemented a version of the proposed algorithm that only applies integer arithmetic. We denote this version as RLBMCF^i. Moreover, RLBMCF^i uses a vector *VC* of size *n* where the values $\hat{c}_i = (\hat{c}_i^1, \hat{c}_i^2)$ associated with the minimum ratio (4) for each node *i* are stored. In RLBMCF^i, the comparisons of ratios are done in the following way: When we want to check if $a/b < c/d$, the comparison $ad < bc$ is performed instead. In order to select the entering arc in each iteration, a linear search of vector *VC* is made. This way, all operations performed by RLBMCF^i are fixed point operations.

A computer running Ubuntu 14.04 with Intel(R) Core(TM) i7-4610 CPU @ 3.00 GHz, and 16GB RAM was used for computational experiments. The algorithms are implemented in C, and compiled with the gcc compiler (version 4.8.4) and –O4 compile option.

5.3 Experimental Results

In the following we discuss results of NETGEN, GRIDGEN and GOTO instances, and briefly comment on GRIDGRAPH and ROAD type instances. We report average runtimes and numbers of solutions found by the algorithms for each type of instance and problem size, that is for each set of five instances with same value of *i* in Table 3. We list the number of nodes (*n*) and arcs (*m*), the average cardinality of the set of extreme supported efficient solutions obtained (*#solutions*), and runtimes for all four tested algorithms. In the interest of brevity, we will refer to the number of solutions (*#solutions*) found by the algorithm throughout Section 5.3 rather than (more precisely) referring to the average cardinality of the set of extreme supported efficient solutions obtained by the algorithms. We also track the number of times the ratio for an arc is updated or computed (*#arc ops*) for RLBMCF^f and the parametric network simplex method, and list the ratio of the two with the results. This indicates how many arc operations can be saved by the proposed RLBMCF^f algorithm. Note that the number of arc operations for RLBMCF^i and RLBMCF^f is identical. Some instances

took an excessive time to solve, in which case only a single instance was solved to obtain a sense of runtime without running the experiment for every instance. Those cases are marked with an asterisk in Tables 5-8.

The results of the five different types of NETGEN instances are summarized in Table 5. In general, RLBMCF^{i} performs better than RLBMCF^{f}, as our instances have only integer costs and capacities, as noted above. Also, Para performs better than Dicho (unless there are few extreme supported efficient solutions). Low density instances NETGEN_8 are solved almost equally well by RLBMCF^{i} and Para for small instances, but RLBMCF^{i} works better for larger ones. Both Dicho and RLBMCF^{f} take longer by a factor of at least 2. The low supply version, NETGEN_LO_8 on the other hand is best solved by a dichotomic version of the network simplex method, also due to a fairly low number of solutions (*#solutions*). High density NETGEN_SR instances are best solved with RLBMCF^{i} which shows much better performance than its closest competitor, Para. Due to a large number of arcs runtimes increase dramatically from one instance to the next for these instances. For the low supply version of the dense instances, NETGEN_LO_SR, RLBMCF^{i} again performs best but the gap to other algorithms is smaller. We also note that Dicho performs a lot better here than for NETGEN_SR, due to a small number of solutions (*#solutions*). Finally, the NETGEN_DEG instances, with increasing number of arcs for fixed number of nodes, show that RLBMCF^{i} outperforms the other approaches, and that it scales better with increasing network density than the other algorithms.

Table 5. NETGEN results.

				Runtime (CPU second)				*#arc ops*
	n	*m*	*#solutions*	*RLBMCF^{f}*	*RLBMCF^{i}*	*Dicho*	*Para*	*RLBMCF^{i}/Para*
NETGEN_8	256	2,048	354.6	**0.0**	**0.0**	**0.0**	**0.0**	0.54
	512	4,096	671.2	0.1	**0.0**	0.1	**0.0**	0.50
	1,024	8,192	1,145.0	0.7	**0.2**	0.3	**0.2**	0.44
	2,048	16,384	2,011.8	2.9	1.2	1.7	**0.9**	0.39
	4,096	32,768	3,568.4	12.3	5.3	11.1	**5.1**	0.35
	8,192	65,536	5,818.6	78.2	**25.5**	53.3	30.1	0.30
	16,384	131,072	9,750.8	433.0	**148.1**	343.1	156.8	0.27
	32,768	262,144	16,529.8	3,215.9	**1,023.2**	1,491.2	1,053.3	0.24
NETGEN_SR	256	4,096	549.6	0.1	**0.0**	0.1	**0.0**	0.49
	512	11,585	1,143.0	0.5	**0.2**	0.5	**0.2**	0.50
	1,024	32,768	2,295.6	2.8	**1.0**	5.0	1.4	0.42
	2,048	92,682	4,333.2	20.7	**6.0**	51.2	14.9	0.38

	4,096	262,144	8,037.4	210.4	**38.6**	443.9	123.0	0.34
	8,192	741,455	14,738.2	1,375.3	**272.7**	2,497.8	968.7	0.35
	16,384	2,095,152	26,418.2	9,086.7*	**1,826.8**	3,601.0*	7,232.5*	0.34
NETGEN_LO_8	256	2,048	82.2	**0.0**	**0.0**	**0.0**	**0.0**	0.35
	512	4,096	137.2	0.1	**0.0**	**0.0**	**0.0**	0.30
	1,024	8,192	202.0	0.3	**0.1**	**0.1**	**0.1**	0.23
	2,048	16,384	293.0	1.3	0.7	**0.4**	0.7	0.20
	4,096	32,768	456.2	6.3	3.3	**2.3**	4.2	0.18
	8,192	65,536	631.2	36.4	15.5	**8.6**	24.5	0.16
	16,384	131,072	929.6	199.9	90.8	**51.3**	135.7	0.13
	32,768	262,144	1,340.4	1,477.6	608.6	**168.2**	726.4	0.11
NETGEN_LO_SR	256	4,096	118.4	**0.0**	**0.0**	**0.0**	**0.0**	0.31
	512	11,585	204.2	0.2	**0.1**	**0.1**	**0.1**	0.30
	1,024	32,768	350.4	1.2	**0.4**	0.9	1.0	0.22
	2,048	92,682	553.8	9.2	**2.8**	9.5	10.6	0.22
	4,096	262,144	895.2	97.1	**16.7**	76.6	91.2	0.18
	8,192	741,455	1,452.2	673.3	**137.3**	348.7	740.5	0.17
	16,384	2,095,152	2,262.6	3,554.2	**829.1**	1,799.6	5,618.0*	NA*
NETGEN_DEG	4,096	8,192	616.8	0.8	0.5	0.5	**0.2**	0.67
	4,096	16,384	1,964.2	4.5	2.3	2.2	**1.2**	0.37
	4,096	32,768	3,568.4	12.1	5.3	10.5	**5.0**	0.35
	4,096	65,536	5,213.0	31.3	**10.4**	42.3	18.2	0.34
	4,096	131,072	6,275.4	83.8	**19.0**	175.9	49.6	0.33
	4,096	262,144	8,037.4	204.4	**39.0**	449.3	125.5	0.34
	4,096	524,288	9,589.2	539.1	**84.5**	1,056.0	299.4	0.33
	4,096	1,048,576	10,602.2	1,152.2	**165.7**	2,596.0	700.4	0.34

RLBMCF^{i} reduces the number of arc operations that need to be performed, or the number of times the ratio for an arc is calculated, for all NETGEN instances. However, this does not lead to a significant decrease in runtime in all instances when compared to Para. For NETGEN_8 instances there is only a slight runtime advantage for RLBMCF^{i} throughout the different instances. Comparing the two for low supply instances NETGEN_LO_8 we observe a slightly higher advantage of RLBMCF^{i} compared to Para, although neither is the best performing algorithm for these instances. The effect of reducing the number of arc operations is best seen for high density instances NETGEN_SR, NETGEN_LO_SR and NETGEN_DEG where the reduction in arc operations in RLBMCF^{i} leads to much improved runtimes. NETGEN_DEG illustrates this point well, where all instances have the same number of nodes but the number of arcs doubles from one instance to the next. Here, the number of arc operations in RLBMCF^{i} drops to about a third of those needed in Para, and the fraction of

runtime reduction increases from each instance to the next. To summarize we show the number of arc operations needed by Para and RLBMCF for each type of NETGEN instances in Table 6, where we focus on the instance groups with 4,096 nodes, and we choose the instance with most arcs for NETGEN_DEG. The table shows that a reduction of arc operations to about a third has a different impact for NETGEN_8 (only ~491 million arc operations) compared to NETGEN_SR (almost 11 billion arc operations) and NETGEN_DEG (almost 63 billion arc operations) and this is reflected in runtime reduction.

Table 6. Average number of arc operations for NETGEN instances with n = 4,096.

			Runtime		*# arc ops*		
NETGEN Type	*m*	*#solutions*	*RLBMC^i*	*Para*	*RLBMC^i*	*Para*	*ratio*
8	32,768	3,568.4	5.3	**5.1**	170,146,148.8	490,933,373.4	0.35
SR	262,144	8,037.4	**38.6**	123.0	3,761,309,899.0	10,967,657,331.4	0.34
LO_8	32,768	456.2	**3.3**	4.2	75,057,960.6	406,305,521.4	0.18
LO_SR	262,144	895.2	**16.7**	91.2	1,471,151,770.4	8,253,646,447.0	0.18
DEG	1,048,576	10,602.2	**165.7**	700.4	21,544,262,682.8	62,912,017,449.6	0.34

In Figure 2 we give one example of runtimes for each of the instances solved for the NETGEN_DEG problem class. In the figure runtimes for the smallest instances (1-5) are not shown as they are < 0.1*s*. The figure shows that runtimes for sets of five instances with the same number of nodes and arcs are similar, and that they increase with problem size. This is generally the case for all problem instances we test and we hence only report on averages in the remainder of this section.

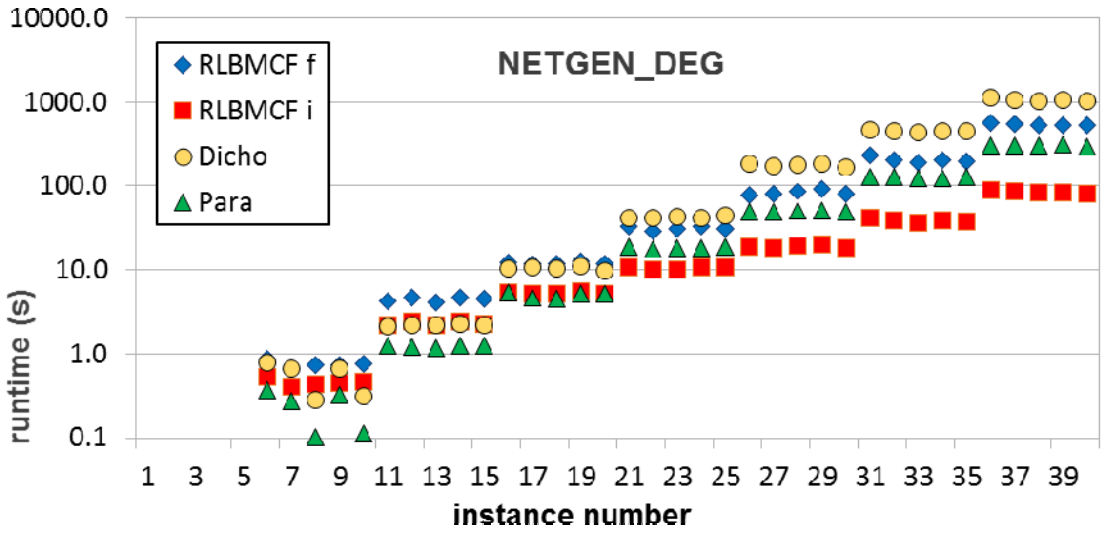

Figure 2. Runtimes (in seconds on logarithmic scale) of different algorithms per NETGEN_DEG instance.

Next, we consider the three sets of GRIDGEN instances. The observations shown in Table 7 are similar to those made for NETGEN instances above. For the low density instances, GRIDGEN_8, RLBMCF^{i} shows best performance, especially as instances grow in size. For high density instances GRIDGEN_SR and GRIDGEN_DEG, RLBMCF^{i} shows superior performance throughout the instances with biggest gains for larger instances.

Table 7. GRIDGEN results.

				Runtime (CPU second)				*#arc ops*
	n	*m*	*#solutions*	*RLBMCF^{f}*	*RLBMCF^{i}*	*Dicho*	*Para*	*RLBMCF^{i}/Para*
GRIDGEN_8	257	2,056	375.4	**0.0**	**0.0**	**0.0**	**0.0**	0.55
	507	4,056	651.4	0.1	**0.0**	0.1	**0.0**	0.46
	1,025	8,200	1,234.2	0.6	**0.2**	0.4	**0.2**	0.44
	2,071	16,568	2,195.6	2.8	1.2	2.2	**1.0**	0.38
	4,097	32,776	3,556.6	12.8	**5.4**	11.9	**5.4**	0.34
	8,191	65,528	6,212.0	77.6	**26.0**	61.0	32.4	0.31
	16,385	131,080	9,566.6	407.8	**144.5**	347.8	179.0	0.25
	32,762	262,096	16,745.0	2,268.2	**942.8**	1,597.0	1,036.3	0.23
GRIDGEN_SR	257	4,112	604.2	0.1	**0.0**	0.1	**0.0**	0.54
	507	11,661	1,163.4	0.5	**0.2**	0.5	**0.2**	0.48
	1,025	32,800	2,255.8	3.3	**1.1**	5.4	1.8	0.40
	2,071	93,195	4,207.2	26.5	**6.9**	58.1	19.4	0.36
	4,097	262,208	7,825.4	279.1	**44.3**	435.0	171.6	0.32
	8,191	745,381	14,848.2	1,989.8	**324.5**	2,740.8	1,399.7	0.29
GRIDGEN_DEG	4,097	16,388	707.0	1.0	0.7	0.7	**0.4**	0.20
	4,097	32,776	3,556.6	12.5	**5.4**	11.5	5.7	0.34
	4,097	65,552	5,035.0	33.5	**10.9**	45.4	23.2	0.32
	4,097	131,104	6,499.6	103.6	**21.8**	191.4	68.5	0.32
	4,097	262,208	7,825.4	289.8	**45.1**	451.2	182.8	0.32
	4,097	524,416	9,516.4	725.1	**97.7**	1,180.1	417.1	0.33
	4,097	1,046,632	10,968.8	1,691.2	**208.4**	2,675.6	975.1	0.34

Results for GOTO instances are shown in Table 8. For the lower density instances, GOTO_8 with node degree 8, Para performs best, and we note that RLBMCF is only able to reduce the number of arc operations to 65-94%, the smallest reduction seen in computational experiments so far. This is likely due to a combination of network structure and the total supply of the instances, which is significantly higher than the supply in NETGEN and GRIDGEN instances. Consistent with results for NETGEN and GRIDGEN instances,

RLBMCF^{i} performs best for the higher density instances GOTO_SR where it is able to solve the largest problems considered within an hour, which the other approaches are unable to do. GOTO instances also have solutions (*#solutions*) which means that Dicho struggles with these instances as they grow larger.

Table 8. GOTO results.

				Runtime (CPU second)				*#arc ops*
	n	*m*	*#solutions*	*RLBMCF^{f}*	*RLBMCF^{i}*	*Dicho*	*Para*	*RLBMCF^{i}/Para*
GOTO_8	256	2,048	328.4	**0.0**	**0.0**	**0.0**	**0.0**	0.65
	512	4,096	818.2	0.1	**0.0**	0.1	**0.0**	0.67
	1,024	8,192	1,511.0	0.2	**0.1**	0.3	**0.1**	0.69
	2,048	16,384	2,909.8	1.0	0.4	1.9	**0.2**	0.70
	4,096	32,768	10,211.0	10.0	4.2	21.6	**2.7**	0.74
	8,192	65,536	10,757.6	30.0	8.1	52.2	**6.1**	0.74
	16,384	131,072	19,070.4	143.3	52.1	282.4	**28.5**	0.78
	32,768	262,144	59,417.4	2,447.9	906.1	3,254.4	**302.5**	0.94
GOTO_SR	256	4,096	2,460.0	0.1	**0.1**	0.2	0.1	0.62
	512	11,585	7,719.4	1.2	**0.5**	3.3	0.6	0.65
	1,024	32,768	23,668.6	10.3	**4.6**	46.6	7.5	0.70
	2,048	92,682	66,687.4	112.9	**36.2**	558.6	73.4	0.75
	4,096	262,144	223,657.8	1,423.3	**456.7**	7,173.0*	784.5	0.59
	8,192	741,455	518,105.8	> 3,600*	**3,529.2**	> 3,600*	> 3,600*	NA*

For GRIDGRAPH networks we briefly report on the results without listing details in a table. GRIDGRAPH instances have very low density with an average node degree of two (see Table 3). Many solutions (*#solutions*) are found for most GRIDGRAPH instances. Consistent with our previous results (Tables 5-8), low density instances with many solutions (*#solutions*) are best solved using Para, as the Dicho simplex method does not work well when *#solutions* is large, and versions of RLBMCF perform best for high density networks making the latter two approaches less suitable for GRIDGRAPH instances. It can be observed that LONG instances are generally easiest with fewest solutions (*#solutions*), and SQUARE and WIDE instances are increasingly difficult to solve with more solutions (*#solutions*). For GRIDGRAPH instances, the reduction of arc operations achieved by RLBMCF is at best a reduction to 80-90% for the more difficult square and wide instances.

For ROAD instances we again have low density with an average node degree between 2 and 3, hence RLBMCF does not perform well here. As ROAD instances have relatively few solutions (*#solutions*), Dicho is the fastest solution algorithm, followed by Para, see Table 9.

Table 9. ROAD results.

				Runtime (CPU second)				*#arc ops*
	n	*m*	*#solutions*	*RLBMCF^f*	*RLBMCF^i*	*Dicho*	*Para*	*RLBMCF^i/Para*
ROAD	9,559	29,766	64.4	2.0	1.6	**0.3**	0.4	0.29
	49,109	120,576	133.4	64.5	55.1	**6.4**	10.0	0.30
	116,920	265,402	300.8	483.5	430.7	**34.8**	54.2	0.43
	261,155	620,924	325.2	2,246.1	2,214.3	**92.4**	190.1	0.29

The social media networks all have the same number of nodes and arcs listed in Table 4. Since supply was varied here, we list *Supp* and *#solutions* for each instance group in Tables 10 and 11. Firstly, it can be observed that networks with anti-correlated costs lead to the highest average numbers of *#solutions*, followed by networks with independent randomly generated costs. Networks with correlated arc costs have the lowest number of *#solutions*. RLBMCF^i performs best for random and anti-correlated instances, particularly with lower supplies, for both FACEBOOK and WIKI-VOTE networks. In these networks RLBMCF^i reduces the number of arc operations significantly, particularly for low supplies. While for networks with random costs RLBMCF^i and Para perform similarly (with Para only slightly worse) for large total supply, RLBMCF^i is consistently the superior approach when costs are anti-correlated. It is interesting to note that Dicho is not the best approach to use even when there are relatively few solutions (*#solutions*), due to network density which makes solving single objective network flow problems computationally expensive. FACEBOOK-CORR instances with supply of at least 2,500 have a ratio of arc operations exceeding 1. This is because the parametric network simplex implementation as described in [3] does not always have to scan all non-basic arcs. It maintains a list of all non-basic arcs with minimum ratio found in an iteration. In the next iteration ratios for the arcs on this list are updated, and those arcs whose ratio remains unchanged can enter the basis. It is not necessary to scan the set of non-basic arcs in this case. With most of the instances above this makes very little difference. For the correlated instances shown below, it has a major impact: For example, in the first instance with supply 10,000 we have only 806 extreme supported efficient solutions, and these are found in 8,275 iterations, hence there are many iterations that do not lead to an

extreme supported nondominated point, and arcs that enter the basis with the same ratio. In 6,579 of these iterations the arc to enter the basis is selected from the candidate arc list from the previous iterations rather than by scanning all non-basic arcs. This reduces the number of scanned arcs significantly compared to the worst-case for parametric simplex. This explains *#arc ops* ratios exceeding 1 for correlated FACEBOOK instances.

Table 10. FACEBOOK results.

			Runtime (CPU second)				*#arc ops*
	Supp	*#solutions*	*RLBMCF^{f}*	*RLBMCF^{i}*	*Dicho*	*Para*	*RLBMCF^{i}/Para*
FACEBOOK-RANDOM	100	143.8	4.6	**2.1**	9.9	7.0	0.10
	500	465.6	12.5	**3.7**	27.0	7.6	0.27
	1,000	763.6	15.4	**4.2**	40.5	8.2	0.32
	2,500	1575.4	26.9	**6.6**	71.4	9.9	0.47
	5,000	2454.8	37.4	**8.9**	101.9	11.5	0.57
	10,000	4011.8	52.4	**12.1**	145.8	13.1	0.72
FACEBOOK-CORR	100	71.2	1.7	**0.7**	4.4	0.9	0.31
	500	166.8	3.3	**1.0**	8.9	**1.0**	0.56
	1,000	262.8	5.6	1.3	12.9	**1.1**	0.86
	2,500	424.6	8.6	1.9	19.0	**1.4**	1.13
	5,000	556.8	11.2	2.3	23.6	**1.5**	1.33
	10,000	777.8	16.9	3.2	30.8	**1.9**	1.62
FACEBOOK-ANTI	100	136.2	4.5	**2.4**	9.6	12.1	0.05
	500	524.8	17.3	**5.2**	31.7	14.1	0.21
	1,000	848.2	23.4	**6.5**	46.1	15.0	0.26
	2,500	1788.6	32.7	**8.7**	86.3	16.8	0.34
	5,000	3074.4	49.4	**12.5**	134.0	19.6	0.45
	10,000	4926.6	65.2	**16.3**	189.3	22.0	0.54

Table 11. WIKI-VOTE results.

			Runtime (CPU second)				*#arc ops*
	Supp	*#solutions*	*RLBMCF^{f}*	*RLBMCF^{i}*	*Dicho*	*Para*	*RLBMCF^{i}/Para*
WIKI-VOTE-RANDOM	100	92.4	3.1	**2.2**	3.6	4.9	0.05
	500	369.4	6.4	**3.1**	12.0	6.1	0.13
	1,000	598.0	9.3	**3.7**	18.3	6.8	0.19
	2,500	1,216.2	14.9	**4.7**	32.2	8.0	0.28
	5,000	2,032.6	20.0	**6.0**	47.3	9.0	0.38
	10,000	3,226.2	28.8	**7.9**	69.3	11.1	0.47

WIKI-VOTE-CORR	100	43.0	1.2	0.7	1.5	**0.6**	0.17
	500	137.6	2.0	**0.9**	4.2	**0.9**	0.30
	1,000	220.8	2.7	**1.1**	6.4	**1.1**	0.37
	2,500	388.2	4.9	**1.3**	9.7	**1.3**	0.57
	5,000	550.8	6.4	1.7	13.6	**1.6**	0.66
	10,000	765.2	8.1	2.1	16.9	**1.9**	0.79
WIKI-VOTE-ANTI	100	83.4	4.1	**3.2**	3.5	8.6	0.03
	500	451.0	10.4	**4.8**	15.8	11.0	0.11
	1,000	745.6	15.1	**5.7**	24.0	12.2	0.15
	2,500	1,763.2	24.4	**7.7**	46.9	14.2	0.25
	5,000	2,696.4	31.6	**9.1**	65.9	15.6	0.31
	10,000	4,390.6	44.4	**11.9**	98.1	18.3	0.41

In summary, we observe that RLBMCF does reduce the number of arc operations for all problem instances tested. This is reflected in faster computation times when problem instances have a high density. For some instance types this reduction leads to a significant decrease of runtime by a factor of up to 4.2 / 3.6 / 5.5 / 4.7 / 3.9 / 2.0 / 5.0 / 3.6 for NETGEN_DEG / NETGEN_SR / NETGEN_LO_SR / GRIDGEN_DEG / GRIDGEN_SR / GOTO_SR / FACEBOOK / WIKI-VOTE instances. When RLBMCF is not the best solution algorithm, e.g. when problems have low density, then the parametric network simplex method generally is the best approach to choose. When dense instances have very high supply, as demonstrated for FACEBOOK and WIKI-VOTE instances, the parametric network simplex can be preferable, especially when costs are correlated. We also note that the parametric simplex method is preferable to a dichotomic network simplex method when there are many solutions (*#solutions*) to be found. While a dichotomic simplex method can take advantage of speed-up techniques such as partial pricing, it has to manage problems to be solved leading to higher memory consumption and additional runtime which an implementation of the parametric simplex method and RLBMCF can both avoid. Finally, our computational experiments confirm that RLBMCF is able to reduce the number of arc operations needed in all test instances, where reductions in the number of arc operations needed by RLBMCF are often significant when compared to the parametric network simplex method. We observe that these reductions in arc operations lead to an advantage in terms of overall runtime especially when the network density is high. Compared to an efficient implementation of the parametric network simplex method, RLBMCF does need to track additional sets and data, and the associated effort is best offset in higher density networks. For instance, for the last set of

NETGEN_DEG instances RLBMCF^{i} only needs to perform a third of the arc operations leading to an average runtime reduction by a factor of 4.2.

6. Conclusions.

We propose a novel ratio-labeling algorithm to find a complete set of extreme supported efficient solutions of BMCF problems. The algorithm is based on the ideas of a parametric network simplex method for biobjective linear programs. The proposed method enhances this classical parametric method by associating, with each node, so-called ratio labels (the slope or trade-off of the two objective functions between two consecutive extreme supported non-dominated points). Instead of examining the complete set of arcs to find the entering arc with minimum ratio, the proposed method maintains a heap from which the node that allows to determine this arc is extracted. To do this we investigate how the ratios of the arcs change when a pivot with this entering arc is performed in order to adequately update the ratio labels of the nodes, and details of the derivation of necessary label updates are contained in this paper (and Appendix A). The result is an efficient algorithm improving the parametric method to find a complete set of extreme supported efficient solutions to the BMCF problem. Moreover, the space needed by our algorithm is minimal since it does not need to store the flows associated with each extreme supported solution. We present extensive computational experiments for five different types of test networks to demonstrate the superior performance of the proposed ratio-labeling approach, when network density is high.

In the future algorithms for BMCF could be further improved by exploiting potential for parallelization. While this is relatively straightforward for scalarization based approaches such as the dichotomic approach tested here, good strategies for parallelization in other methods are worth investigating and may lead to significant improvements in runtime.

Acknowledgments

The authors thank University of Auckland summer student Samuel Ridler for testing and calibration of the dichotomic algorithm implementation. The research of the second author was partially supported by MTM2016-74877-P.

References

[1] Ahuja R, Magnanti T, Orlin JB. Network Flows: theory, algorithms and applications. Englewood Cliffs, New Jersey: Prentice-Hall; 1993.

[2] Kovács, P. Minimum-cost flow algorithms: an experimental evaluation. Optimization Methods and Software; 30:1, 94-127; 2015.

[3] Raith A, Ehrgott M. A two-phase algorithm for the biobjective integer minimum cost flow problem, Computers & Operations Research; 36:1945-1954; 2009.
[4] Hamacher HW, Roed Pedersen C, Ruzika, S. Multiple objective minimum cost flow problems: a reviw. European Journal of Operational Research; 176, 1404-1422; 2007.
[5] Sedeño-Noda A, González-Martín C. The Biobjective minimum cost flow problem, European Journal of Operational Research; 124:591-600; 2000.
[6] Sedeño-Noda A, González-Martín C. An alternative method to solve the biobjective minimum cost flow problem, Asia-Pacific Journal of Operational Research; 20:241-260; 2003.
[7] Fonseca M, Figueira JR, Resende MGC. Solving scalarized multi-objective network flow problems using an interior point method, International Transactions of Operational Research; 17:607-636; 2010.
[8] Medrano FA and Church RL. A parallel computing framework for finding the supported solutions to a biobjective network optimization problem, Journal of Multi-Criteria Decision Analysis; 22:244-259; 2015.
[9] Eusébio A, Figueira JR, Ehrgott M. A primal-dual simplex algorithm for bi-objective network flow problems, 4OR; 7:255-273; 2009.
[10] Eusébio A, Figueira JR. On the computation of all supported efficient solutions in multi-objective integer network flow problems, European Journal of Operational Research; 199:68-76; 2009.
[11] Eusébio A, Figueira JR. Finding non-dominated solutions in bi-objective integer network flow problems, Computers & Operations Research; 36:2254-2564; 2009.
[12] Sun M. Finding integer efficient solutions for multiobjective network programming problems, Networks; 57:362-375; 2011.
[13] Eusébio A, Figueira JR, Ehrgott M. On finding representative non-dominated points for bi-objective integer network flow problems, Computers & Operations Research; 48:1-10; 2014.
[14] Sedeño-Noda A, Raith A. A Dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem, Computers & Operations Research; 57:83-94; 2015.
[15] Cunningham, WH. A Network Simplex Method. Mathematical Programming; 11:105-106; 1976.
[16] Steuer R. Multiple Criteria Optimization. Theory, Computation, and Application. New York: Wiley;1985.
[17] Isermann H. Proper efficiency and the linear vector maximum problem. Operations Research; 22:189-191; 1974.
[18] Dantzig GB, Thapa MN. Linear Programming 2: Theory and Extensions. Springer Series in Operations Research. Springer; 1997.
[19] Fredman ML, Tarjan RE. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM; 34(3):596-615; 1987.
[20] DIMACS, 9th DIMACS Implementation Challenge – Shortest Paths, Available at http://www.dis.uniroma1.it/challenge9 ; 2005-2006.
[21] Löbel, A. MCF version 1.3 – a network simplex implementation. Available for academic use free of charge via WWW at www.zib.de, 2004.
[22] Cohon JL, Church RL. Generating multiobjective trade-offs: an algorithm for bicriterion problems, Water Resources Research; 15:1001-1010; 1979.
[23] Leskovec, J., Krevl A. SNAP Datasets: Stanford Large Network Dataset Collection http://snap.stanford.edu/data ; 2014.

Appendix A

We analyse values of the new ratios of the arc $(i,j) \notin A(T)$ when $A(T) = A(T) - \{(p,q)\} + \{(x,y)\}$ with $(x,y) \neq (p,q)$, that is, (x,y) is the entering arc and (p,q) is the leaving arc in the pivot operation. Remember that

$$\text{(A1)}\ \theta_{xy}^k = -\frac{\hat{c}_{xy}^1}{\hat{c}_{xy}^2} \leq \theta_{ij}^k = -\frac{\hat{c}_{ij}^1}{\hat{c}_{ij}^2}$$

is always satisfied for the following expressions. The following cases have to be considered.

Case 1) $x \in V(T_1)$ and $(x,y) \in L$. In this case, node potentials in $V(T_2)$ are updated following process A. Now, we distinguish the next subcases:

1.1) $(i,j) \in L$, $i \in V(T_2)$ and $j \in V(T_1)$. Then the reduced cost of this arc becomes $\bar{c}_{ij}^v + \bar{c}_{xy}^v = \hat{c}_{ij}^v + \hat{c}_{xy}^v$ with $v \in \{1,2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{\hat{c}_{ij}^1 + \hat{c}_{xy}^1}{\hat{c}_{ij}^2 + \hat{c}_{xy}^2}$ whenever $\hat{c}_{ij}^2 + \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 + \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain that:

$$\text{(A2)}\ \theta_{ij}^{k+1} = -\frac{\hat{c}_{ij}^1 + \hat{c}_{xy}^1}{\hat{c}_{ij}^2 + \hat{c}_{xy}^2} = \frac{\hat{c}_{xy}^1}{\hat{c}_{xy}^2}\frac{\left(-\frac{\hat{c}_{ij}^1}{\hat{c}_{xy}^1}\hat{c}_{xy}^2 - \hat{c}_{xy}^2\right)}{\hat{c}_{ij}^2 + \hat{c}_{xy}^2} \geq \frac{\hat{c}_{xy}^1}{\hat{c}_{xy}^2}\frac{\left(-\hat{c}_{ij}^2 - \hat{c}_{xy}^2\right)}{\hat{c}_{ij}^2 + \hat{c}_{xy}^2} = -\frac{\hat{c}_{xy}^1}{\hat{c}_{xy}^2} = \theta_{xy}^k$$

$$\text{(A3)}\ \theta_{ij}^{k+1} = -\frac{\hat{c}_{ij}^1 + \hat{c}_{xy}^1}{\hat{c}_{ij}^2 + \hat{c}_{xy}^2} = \frac{\hat{c}_{ij}^1}{\hat{c}_{ij}^2}\frac{\left(-\hat{c}_{ij}^2 - \hat{c}_{ij}^2\frac{\hat{c}_{xy}^1}{\hat{c}_{ij}^1}\right)}{\hat{c}_{ij}^2 + \hat{c}_{xy}^2} \leq \frac{\hat{c}_{ij}^1}{\hat{c}_{ij}^2}\frac{\left(-\hat{c}_{ij}^2 - \hat{c}_{xy}^2\right)}{\hat{c}_{ij}^2 + \hat{c}_{xy}^2} = -\frac{\hat{c}_{ij}^1}{\hat{c}_{ij}^2} = \theta_{ij}^k$$

(A4) If $\theta_{xy}^k = \theta_{ij}^k$, then $\theta_{ij}^{k+1} = \theta_{xy}^k = \theta_{ij}^k$ following the same arguments as in (A3).

1.2) $(i,j) \in L$, $i \in V(T_1)$ and $j \in V(T_2)$. Then the reduced cost of this arc becomes $\bar{c}_{ij}^v - \bar{c}_{xy}^v = \hat{c}_{ij}^v - \hat{c}_{xy}^v$ with $v \in \{1,2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{\hat{c}_{ij}^1 - \hat{c}_{xy}^1}{\hat{c}_{ij}^2 - \hat{c}_{xy}^2}$ whenever $\hat{c}_{ij}^2 - \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 - \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain that:

(A5) $\theta_{ij}^{k+1} = -\dfrac{\hat{c}_{ij}^1 - \hat{c}_{xy}^1}{\hat{c}_{ij}^2 - \hat{c}_{xy}^2} = \dfrac{\hat{c}_{xy}^1}{\hat{c}_{xy}^2} \dfrac{\left(-\dfrac{\hat{c}_{ij}^1}{\hat{c}_{xy}^1} \hat{c}_{xy}^2 + \hat{c}_{xy}^2 \right)}{\hat{c}_{ij}^2 - \hat{c}_{xy}^2} \geq \dfrac{\hat{c}_{xy}^1}{\hat{c}_{xy}^2} \dfrac{\left(-\hat{c}_{ij}^2 + \hat{c}_{xy}^2 \right)}{\hat{c}_{ij}^2 - \hat{c}_{xy}^2} = -\dfrac{\hat{c}_{xy}^1}{\hat{c}_{xy}^2} = \theta_{xy}^k$

(A6) $\theta_{ij}^{k+1} = -\dfrac{\hat{c}_{ij}^1 - \hat{c}_{xy}^1}{\hat{c}_{ij}^2 - \hat{c}_{xy}^2} = \dfrac{\hat{c}_{ij}^1}{\hat{c}_{ij}^2} \dfrac{\left(-\hat{c}_{ij}^2 + \hat{c}_{ij}^2 \dfrac{\hat{c}_{xy}^1}{\hat{c}_{ij}^1} \right)}{\hat{c}_{ij}^2 - \hat{c}_{xy}^2} \geq \dfrac{\hat{c}_{ij}^1}{\hat{c}_{ij}^2} \dfrac{\left(-\hat{c}_{ij}^2 + \hat{c}_{xy}^2 \right)}{\hat{c}_{ij}^2 - \hat{c}_{xy}^2} = -\dfrac{\hat{c}_{ij}^1}{\hat{c}_{ij}^2} = \theta_{ij}^k$

(A7) If $\theta_{xy}^k = \theta_{ij}^k$, then $\theta_{ij}^{k+1} = +\infty$, because $\hat{c}_{ij}^2 \geq \hat{c}_{xy}^2$ and $\hat{c}_{ij}^2 - \hat{c}_{xy}^2 \geq 0$

(A8) If $\theta_{ij}^k = +\infty$, then $\theta_{ij}^{k+1} = +\infty$, because if $\hat{c}_{ij}^2 \geq 0$ then $\hat{c}_{ij}^2 - \hat{c}_{xy}^2 > 0$ or if $\hat{c}_{ij}^1 \leq 0$ then $\hat{c}_{ij}^1 - \hat{c}_{xy}^1 < 0$.

1.3) $(i, j) \in U$, $i \in V(T_2)$ and $j \in V(T_1)$. Then the reduced cost of this arc becomes $\overline{c}_{ij}^v + \overline{c}_{xy}^v = -\hat{c}_{ij}^v + \hat{c}_{xy}^v$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{-(-\hat{c}_{ij}^1 + \hat{c}_{xy}^1)}{-(-\hat{c}_{ij}^2 + \hat{c}_{xy}^2)}$ whenever $\hat{c}_{ij}^2 - \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 - \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain the same result as in (A5), (A6), (A7) and (A8).

1.4) $(i, j) \in U$, $i \in V(T_1)$ and $j \in V(T_2)$. Then the reduced cost of this arc becomes $\overline{c}_{ij}^v - \overline{c}_{xy}^v = -\hat{c}_{ij}^v - \hat{c}_{xy}^v$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{-(-\hat{c}_{ij}^1 - \hat{c}_{xy}^1)}{-(-\hat{c}_{ij}^2 - \hat{c}_{xy}^2)}$ whenever $\hat{c}_{ij}^2 + \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 + \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A2), (A3) and (A4).

Case 2) $x \in V(T_1)$ and $(x, y) \in U$. In this case, the node potentials in $V(T_1)$ are updated following process B. Now, we distinguish the following subcases:

2.1) $(i, j) \in L$, $i \in V(T_2)$ and $j \in V(T_1)$. Then the reduced cost of this arc becomes $\overline{c}_{ij}^v + \overline{c}_{xy}^v = \hat{c}_{ij}^v - \hat{c}_{xy}^v$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{\hat{c}_{ij}^1 - \hat{c}_{xy}^1}{\hat{c}_{ij}^2 - \hat{c}_{xy}^2}$ whenever $\hat{c}_{ij}^2 - \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 - \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A5), (A6), (A7) and (A8).

2.2) $(i, j) \in L$, $i \in V(T_1)$ and $j \in V(T_2)$. Then the reduced cost of this arc becomes $\overline{c}_{ij}^v - \overline{c}_{xy}^v = \hat{c}_{ij}^v + \hat{c}_{xy}^v$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{\hat{c}_{ij}^1 + \hat{c}_{xy}^1}{\hat{c}_{ij}^2 + \hat{c}_{xy}^2}$

whenever $\hat{c}_{ij}^2 + \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 + \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A2), (A3) and (A4).

2.3) $(i, j) \in U$, $i \in V(T_2)$ and $j \in V(T_1)$. Then the reduced cost of this arc becomes $\bar{c}_{ij}^v + \bar{c}_{xy}^v = -\hat{c}_{ij}^v - \hat{c}_{xy}^v$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{-(-\hat{c}_{ij}^1 - \hat{c}_{xy}^1)}{-(-\hat{c}_{ij}^2 - \hat{c}_{xy}^2)}$ whenever $\hat{c}_{ij}^2 + \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 + \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A2), (A3) and (A4).

2.4) $(i, j) \in U$, $i \in V(T_1)$ and $j \in V(T_2)$. Then the reduced cost of this arc becomes $\bar{c}_{ij}^v - \bar{c}_{xy}^v = -\hat{c}_{ij}^v + \hat{c}_{xy}^v$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{-(-\hat{c}_{ij}^1 + \hat{c}_{xy}^1)}{-(-\hat{c}_{ij}^2 + \hat{c}_{xy}^2)}$ whenever $\hat{c}_{ij}^2 - \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 - \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A5), (A6), (A7) and (A8).

Case 3) $x \in V(T_2)$ and $(x, y) \in L$. Node potentials in $V(T_1)$ are updated following process B. Now, we distinguish the following subcases:

3.1) $(i, j) \in L$, $i \in V(T_2)$ and $j \in V(T_1)$. Then the reduced cost of this arc becomes $\bar{c}_{ij}^v - \bar{c}_{xy}^v = \hat{c}_{ij}^v - \hat{c}_{xy}^v$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{\hat{c}_{ij}^1 - \hat{c}_{xy}^1}{\hat{c}_{ij}^2 - \hat{c}_{xy}^2}$ whenever $\hat{c}_{ij}^2 - \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 - \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A5), (A6), (A7) and (A8).

3.2) $(i, j) \in L$, $i \in V(T_1)$ and $j \in V(T_2)$. Then the reduced cost of this arc becomes $\bar{c}_{ij}^v + \bar{c}_{xy}^v = \hat{c}_{ij}^v + \hat{c}_{xy}^v$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{\hat{c}_{ij}^1 + \hat{c}_{xy}^1}{\hat{c}_{ij}^2 + \hat{c}_{xy}^2}$ whenever $\hat{c}_{ij}^2 + \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 + \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A2), (A3) and (A4).

3.3) $(i, j) \in U$, $i \in V(T_2)$ and $j \in V(T_1)$. Then the reduced cost of this arc becomes $\bar{c}_{ij}^v - \bar{c}_{xy}^v = -\hat{c}_{ij}^v - \hat{c}_{xy}^v$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{-(-\hat{c}_{ij}^1 - \hat{c}_{xy}^1)}{-(-\hat{c}_{ij}^2 - \hat{c}_{xy}^2)}$ whenever $\hat{c}_{ij}^2 + \hat{c}_{xy}^2 < 0$ and $\hat{c}_{ij}^1 + \hat{c}_{xy}^1 > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A2), (A3) and (A4).

3.4) $(i, j) \in U$, $i \in V(T_1)$ and $j \in V(T_2)$. Then the reduced cost of this arc becomes $\overline{c}_{ij}^{v} + \overline{c}_{xy}^{v} = -\hat{c}_{ij}^{v} + \hat{c}_{xy}^{v}$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{-(-\hat{c}_{ij}^{1} + \hat{c}_{xy}^{1})}{-(-\hat{c}_{ij}^{2} + \hat{c}_{xy}^{2})}$ whenever $\hat{c}_{ij}^{2} - \hat{c}_{xy}^{2} < 0$ and $\hat{c}_{ij}^{1} - \hat{c}_{xy}^{1} > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A5), (A6), (A7) and (A8).

Case 4) $x \in V(T_2)$ and $(x, y) \in U$. We suppose that the node potentials in $V(T_2)$ are updated following process A. Now, we distinguish the following subcases:

4.1) $(i, j) \in L$, $i \in V(T_2)$ and $j \in V(T_1)$. Then the reduced cost of this arc becomes $\overline{c}_{ij}^{v} - \overline{c}_{xy}^{v} = \hat{c}_{ij}^{v} + \hat{c}_{xy}^{v}$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{\hat{c}_{ij}^{1} + \hat{c}_{xy}^{1}}{\hat{c}_{ij}^{2} + \hat{c}_{xy}^{2}}$ whenever $\hat{c}_{ij}^{2} + \hat{c}_{xy}^{2} < 0$ and $\hat{c}_{ij}^{1} + \hat{c}_{xy}^{1} > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A2), (A3) and (A4).

4.2) $(i, j) \in L$, $i \in V(T_1)$ and $j \in V(T_2)$. Then the reduced cost of this arc becomes $\overline{c}_{ij}^{v} + \overline{c}_{xy}^{v} = \hat{c}_{ij}^{v} - \hat{c}_{xy}^{v}$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{\hat{c}_{ij}^{1} - \hat{c}_{xy}^{1}}{\hat{c}_{ij}^{2} - \hat{c}_{xy}^{2}}$ whenever $\hat{c}_{ij}^{2} - \hat{c}_{xy}^{2} < 0$ and $\hat{c}_{ij}^{1} - \hat{c}_{xy}^{1} > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A5), (A6), (A7) and (A8).

4.3) $(i, j) \in U$, $i \in V(T_2)$ and $j \in V(T_1)$. Then the reduced cost of this arc becomes $\overline{c}_{ij}^{v} - \overline{c}_{xy}^{v} = -\hat{c}_{ij}^{v} + \hat{c}_{xy}^{v}$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{-(-\hat{c}_{ij}^{1} + \hat{c}_{xy}^{1})}{-(-\hat{c}_{ij}^{2} + \hat{c}_{xy}^{2})}$ whenever $\hat{c}_{ij}^{2} - \hat{c}_{xy}^{2} < 0$ and $\hat{c}_{ij}^{1} - \hat{c}_{xy}^{1} > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A5), (A6), (A7) and (A8).

4.4) $(i, j) \in U$, $i \in V(T_1)$ and $j \in V(T_2)$. Then the reduced cost of this arc becomes $\overline{c}_{ij}^{v} + \overline{c}_{xy}^{v} = -\hat{c}_{ij}^{v} - \hat{c}_{xy}^{v}$ with $v \in \{1, 2\}$. We obtain that the new ratio is $\theta_{ij}^{k+1} = -\dfrac{-(-\hat{c}_{ij}^{1} - \hat{c}_{xy}^{1})}{-(-\hat{c}_{ij}^{2} - \hat{c}_{xy}^{2})}$ whenever $\hat{c}_{ij}^{2} + \hat{c}_{xy}^{2} < 0$ and $\hat{c}_{ij}^{1} + \hat{c}_{xy}^{1} > 0$; otherwise it becomes +∞. From (A1), we obtain the same results as in (A2), (A3) and (A4).