Version
This is the Author’s Original version (preprint) of the following article. This version is defined in the NISO recommended practice RP-8-2008
http://www.niso.org/publications/rp/

Suggested Reference

Copyright
Items in ResearchSpace are protected by copyright, with all rights reserved, unless otherwise indicated. Previously published items are made available in accordance with the copyright policy of the publisher.

http://www.sherpa.ac.uk/romeo/issn/0305-0548/
https://researchspace.auckland.ac.nz/docs/uoa-docs/rights.htm
A Two-Phase Algorithm for the Biobjective Integer
Minimum Cost Flow Problem

Andrea Raith
Department of Engineering Science
The University of Auckland, New Zealand
email: a.raith@auckland.ac.nz

Matthias Ehrgott
Department of Engineering Science
The University of Auckland, New Zealand
email: m.ehrgott@auckland.ac.nz
and
Laboratoire d’Informatique de Nantes Atlantique
Université de Nantes, France
email: matthias.ehrgott@univ-nantes.fr

February 4, 2009
Abstract

We present an algorithm to compute a complete set of efficient solutions for the biobjective integer minimum cost flow problem. We use the two phase method with a parametric network simplex algorithm in phase 1 to compute all non-dominated extreme points. In phase 2, the remaining non-dominated points (non-extreme supported and non-supported) are computed using a $k$ best flow algorithm on single-objective weighted sum problems.

We implement the algorithm and report run-times on problem instances generated with a modified version of the NETGEN generator and also for networks with a grid structure.

Keywords: Biobjective integer minimum cost flow problem, two phase method, $k$ best flow algorithm.
1 Introduction

Single-objective integer minimum cost flow problems have received a lot of attention in the literature as they have various applications (see for example Ahuja et al. 1993). As with most real-world optimisation problems, there is usually more than one objective that has to be taken into account, thus leading to multiobjective integer minimum cost flow problems (MIMCF). We restrict our considerations to the biobjective case (BIMCF). The aim in BIMCF is to find efficient solutions. The problem of finding all efficient solutions of BIMCF is intractable, Ruhe (1988) presents an example problem with exponentially many efficient solutions. BIMCF is an \( \mathcal{NP} \)-hard problem, as the biobjective shortest path problem, a special case of BIMCF, was shown to be \( \mathcal{NP} \)-hard by Serafini (1986).

We propose to solve the BIMCF problem using an approach with two phases. In the first phase, extreme efficient solutions (efficient solutions which define extreme points of the convex hull of feasible solution vectors in objective space) are computed with a parametric network simplex algorithm (Sedeño-Noda and González-Martín 2000). Other efficient solutions are computed in the second phase using a ranking algorithm (Hamacher 1995) on restricted areas of the objective space.

We test our algorithm on different problem instances generated with the well known network generator \textsc{netgen} and also on networks with a grid structure.

The paper is organised as follows: In Section 2 basic concepts of BIMCF problems are introduced. Recent literature is discussed in Section 3. In Section 4 we present an algorithm to solve BIMCF. Finally, numerical results are presented in Section 5.

2 Biobjective Integer Minimum Cost Flow Problem

In this section, terminology and basic theory of biobjective integer minimum cost flow (BIMCF) problems are introduced.

Let \( N = (G,c,l,u) \) be a directed network where the graph \( G = (V,A) \) consists of a set of nodes or vertices \( V = \{1, \ldots, n\} \) and a set of arcs \( A \subseteq V \times V \) with \(|A| = m\). We denote by \( t(a) \) the tail node of arc \( a \in A \) and by \( h(a) \) the head node of \( a \). Two costs \( c_a = (c_a^1,c_a^2) \in \mathbb{Z} \times \mathbb{Z} \) are associated with each arc \( a \in A \). Furthermore, there are non-negative integer lower and upper bound capacities \( l_a \) and \( u_a \) with \( l_a \leq u_a \) on every arc \( a \). An integer numerical value \( b_i \), the balance, is associated with each node \( i \in V \). A value \( b_i > 0 \), \( b_i < 0 \), or \( b_i = 0 \) indicates that, at node \( i \), there exists a supply of flow, a demand of flow, or neither of the two (\( i \) is then called transshipment node). The BIMCF problem is defined by the following mathematical programme:
\[
\begin{align*}
\min \quad & z(x) = \begin{cases} 
z_1(x) = \sum_{a \in A} c_1^a x_a \\
z_2(x) = \sum_{a \in A} c_2^a x_a 
\end{cases} \\
\text{s.t.} \quad & \sum_{a \in A: t(a) = i} x_a - \sum_{a \in A: h(a) = i} x_a = b_i \quad \forall i \in V \\
& l_a \leq x_a \leq u_a \quad \text{for all } a \in A \\
& x_a \text{ integer} \quad \text{for all } a \in A.
\end{align*}
\]

Here \( x \) is the vector of flow on the arcs, constraint (2) represents flow conservation at the different nodes, and we assume that \( \sum_{i \in V} b_i = 0 \) since otherwise the problem is infeasible. The feasible set \( X \) is described by constraints (2) – (4). Its image under the objective function is \( Z := z(X) \).

In case of positive lower bound capacities, the network can be transformed into a network with zero lower bound capacities as explained in Ahuja et al. (1993). Therefore, we assume \( l_a = 0 \) in the following.

In the remainder of this paper we use the following orders on \( \mathbb{R}^2 \):

\[
\begin{align*}
y^1 & \leq y^2 \quad \Leftrightarrow \quad y^1_k \leq y^2_k \quad k = 1, 2, \\
y^1 & \leq y^2 \quad \Leftrightarrow \quad y^1_k \leq y^2_k \quad k = 1, 2; \quad y^1 \neq y^2, \text{ and} \\
y^1 & < y^2 \quad \Leftrightarrow \quad y^1_k < y^2_k \quad k = 1, 2.
\end{align*}
\]

We are seeking those feasible solutions that do not allow to improve one component of the objective vector \( z(x) \) without deteriorating the other one.

**Definition 1** A feasible solution \( \hat{x} \in X \) is called efficient if there does not exist any \( x' \in X \) with \( (z_1(x'), z_2(x')) \leq (z_1(\hat{x}), z_2(\hat{x})) \). The image \( z(\hat{x}) = (z_1(\hat{x}), z_2(\hat{x})) \) of \( \hat{x} \) is called non-dominated. Let \( X_E \) denote the set of all efficient solutions and and let \( Z_N \) denote the set of all non-dominated points. We distinguish two different types of efficient solutions.

- **Supported efficient solutions** are those efficient solutions that can be obtained as optimal solutions to a (single objective) weighted sum problem

\[
\min_{x \in X} \lambda_1 z_1(x) + \lambda_2 z_2(x)
\]

for some \( \lambda_1 > 0, \lambda_2 > 0 \). The set of all supported efficient solutions is denoted by \( X_{SE} \), its non-dominated image \( Z_{SN} \). The supported non-dominated points lie on the boundary of the convex hull \( \text{conv}(Z) \) of the feasible set in objective space.

- **Supported efficient solutions which define an extreme point of \( \text{conv}(Z) \)** are called extreme efficient solutions.

- **The remaining efficient solutions in \( X_{NE} := X_E \setminus X_{SE} \)** are called non-supported efficient solutions. They cannot be obtained as solutions of a weighted sum problem as their image lies in the interior of \( \text{conv}(Z) \). The set of non-supported non-dominated points is denoted by \( Z_{NN} \). 

2
The two objective functions $z_1$ and $z_2$ do generally not attain their individual optima for the same values of $\hat{x}$. We will assume in the following that there exists no $\hat{x}$ such that $\hat{x} \in \text{argmin}\{z_1\}$ and $\hat{x} \in \text{argmin}\{z_2\}$ for a problem of the form (1) - (4).

**Definition 2** Two feasible solutions $x$ and $x'$ are called equivalent if $z(x) = z(x')$. A complete set $X_E$ is a set of efficient solutions such that all $x \in X \setminus X_E$ are either non-efficient or equivalent to at least one $x \in X_E$.

Another notion of optimality that is used in the context of biobjective optimisation is **lexicographic minimisation**. Here, we choose among all optimal feasible solutions for the preferred component $k$ of the objective vector one that is optimal for the other component $l$.

**Definition 3** Let $k \in \{1, 2\}$ and $l \in \{1, 2\}\setminus\{k\}$. Then $z(\hat{x}) \leq_{\text{lex}(k,l)} z(x')$ if either $z_k(\hat{x}) < z_k(x')$ or both $z_k(\hat{x}) = z_k(x')$ and $z_l(\hat{x}) \leq z_l(x')$. We call $\hat{x}$ a $\text{lex}(k,l)$-best solution if $z(\hat{x}) \leq_{\text{lex}(k,l)} z(x)$ for all $x \in X$ and denote it by $x_{\text{lex}(k,l)}$.

3 Literature

An excellent and very recent review on multiobjective minimum cost flow problems is given by Hamacher et al. (2007). We will therefore only briefly mention relevant literature. To our knowledge, there is no published work on MIMCF, so the following is restricted to BIMCF. All exact solution approaches to find a (complete) set of efficient solutions for BIMCF, i.e. supported and non-supported efficient solutions, consist of two phases, also known as the two phase method. In the first phase the set of supported efficient solutions, or at least a complete set of extreme efficient solutions, is computed. In the second phase other efficient solutions are computed.

In case all capacities, supplies, and demands are integer, which we assume in this paper, any approach to solve the biobjective continuous network flow problem can be used in phase 1 of BIMCF to find a complete set of (extreme) supported efficient solutions, e.g. Lee and Pulat (1991); Pulat et al. (1992); Sedeño-Noda and González-Martín (2000, 2003). This is because (5) is a single-objective minimum cost flow problem and because of total unimodularity of the matrix defining constraints (2). To solve the continuous problem it is sufficient to generate a complete set of extreme efficient solutions because all non-extreme supported non-dominated points are convex combinations of extreme ones. The algorithms presented by Lee and Pulat (1991); Pulat et al. (1992) may generate some non-extreme supported efficient solutions, whereas the algorithms by Sedeño-Noda and González-Martín (2000, 2003) generate extreme efficient solutions only. Lee and Pulat (1991) claim that their procedure can be extended to generate all integer efficient solutions with image on the edges of $\text{conv}(Z)$, i.e. all supported efficient solutions. Every efficient solution found by their algorithm corresponds to a basic feasible solution of (5), represented by a spanning tree in $N$. Two solutions $x^1$ and $x^2$ are called adjacent if the two corresponding
trees have \( n - 2 \) arcs in common. For adjacent solutions \( x^1 \) and \( x^2 \), the arc that is in the basis of \( x^2 \), but not in the basis of \( x^1 \), can be introduced into the basic tree of \( x^1 \) resulting in a cycle. As much flow as possible is sent along that cycle until the flow on one of the arcs in the cycle reaches its upper or lower bound, this determines the flow change \( \delta \) along the cycle. The arc with flow at upper or lower bound leaves the tree, resulting in the tree of \( x^1 \)'s adjacent solution, \( x^2 \). Whenever the flow changes by \( \delta \) along the cycle, when moving from one efficient solution to an adjacent one, the authors propose to increase the flow stepwise by \( 1, 2, \ldots, \delta - 1 \) to obtain intermediate solutions and claim to obtain all supported efficient solutions this way. This claim is incorrect, as not all non-extreme supported efficient solutions can be obtained as intermediate solutions of two adjacent basic efficient solutions, an example is given by Eusébio and Figueira (2006).

Several papers are dedicated to the computation of non-supported efficient solutions of BIMCF, assuming all non-dominated extreme points are known. Lee and Pulat (1993) perform an explicit search of the solution space, by using intermediate solutions between adjacent basic solutions (which is not sufficient, see remark above) and modifying upper and lower bounds of arcs. They assume non-degeneracy of the problem. Huarng et al. (1992) extend this algorithm to allow degeneracy in the problems.

Sedeño-Noda and González-Martín (2001) argue that these two papers are incorrect and present an approach that is based on the basic tree structure of solutions. Having found a complete set of extreme efficient solutions in phase 1, the algorithm by Sedeño-Noda and González-Martín (2001) moves from one efficient solution to adjacent solutions, in order to identify efficient ones among them. Przybylski et al. (2006) give an example of a network where one efficient solution is not adjacent to any of the other efficient solutions, hence showing that the approach by Sedeño-Noda and González-Martín (2001) cannot generate a complete efficient set. The same example can also be used to show that the approach by Lee and Pulat (1993) is incorrect.

Figueira (2002) present an approach where \( \varepsilon \)-constraint problems, \( \min \{ z_1(x) : x \in X, z_2(x) \leq \varepsilon \} \), are repeatedly solved via branch-and-bound to obtain non-supported efficient solutions.

The following reports were not mentioned in Hamacher et al. (2007).

Eusébio and Figueira (2006) give examples of networks, where for a supported extreme and supported non-extreme non-dominated point, both basic and non-basic supported efficient solutions exist. It is known from linear programming that there is always a basic feasible solution for every extreme non-dominated point, but the authors show that there may be other non-basic efficient solutions that lead to the same point. The network simplex method can be used to identify all basic efficient solutions at that point. The non-basic solutions can be obtained as convex combinations of the basic ones, but not by the network simplex algorithm itself. Eusébio and Figueira (2006) also give a network in which supported efficient solutions exist that cannot be obtained as intermediate solutions between two extreme efficient solutions.

In a more recent report, Eusébio and Figueira (2007) illustrate and prove that supported efficient solutions are indeed connected via chains of zero-cost
cycles in the incremental graph constructed from basic feasible solutions corresponding to extreme efficient solutions. They use this relationship to characterise all supported efficient solutions to a BIMCF problem. The same result can be obtained by considering a weighted sum objective (5) for which two extreme efficient solutions corresponding to two consecutive non-dominated extreme points are optimal. The non-dominated points on the edge of \( \text{conv}(Z) \) connecting the two extreme non-dominated points can be obtained by applying the \( k \) best flow algorithm by Hamacher (1995) to the problem with weighted sum objective. The \( k \) best flow algorithm is also based on cycles in the incremental graph. We explain how to apply the \( k \) best flow algorithm in Section 4.2.

4 A Two Phase Algorithm to Solve BIMCF

We solve the BIMCF problem with the two phase method. A description of the two phase method for general multiobjective combinatorial optimisation problems can be found in Ulungu and Teghem (1995).

The two phase method is based on computing supported and non-supported non-dominated points separately. In phase 1 extreme efficient solutions (see Figure 1) are computed. There are two main approaches: One is taking advantage of the fact that supported solutions are obtainable as solutions to the weighted sum problem (5), refer to Sedeño-Noda and González-Martín (2003). The other main approach is based on the network simplex method where extreme efficient solutions are generated in a right-to-left (or left-to-right) fashion, e.g. Sedeño-Noda and González-Martín (2000). In phase 2 the remaining supported and non-supported non-dominated points are computed with an enumerative approach, as there is no theoretical characterisation for their efficient calculation. The search space in phase 2 can be restricted to triangles given by two consecutive supported non-dominated points as indicated in Figure 2. It is expected that the search space in phase 2 is highly restricted due to information obtained in phase 1 so that the associated problems can be solved quickly.

Figure 1: Supported non-dominated points.

Figure 2: All non-dominated points.
4.1 Phase 1 – Parametric Simplex

In phase 1 of the two phase method, we compute a complete set of extreme efficient solutions of the problem. As mentioned above, any solution method to solve the biobjective continuous minimum cost flow problem can be used here. We use a parametric simplex method by Sedeño-Noda and González-Martín (2000). Initially, one of the two lexicographically optimal solutions, e.g. the \( lex(1,2) \)-best solution, is obtained with a single-objective network simplex algorithm with \( lex(1,2) \) objective. The procedure generates a complete set of extreme efficient solutions moving in a left-to-right fashion.

In the single-objective network simplex (Helgason and Kennington 1995, for example), each basic feasible solution (BFS) is represented by a tree given by a set of basic arcs with flow \( 0 \leq x_a \leq u_a \); all other (non-basic) arcs have a flow of either \( x_a = 0 \) or \( x_a = u_a \). In each step in the network simplex algorithm, a non-basic arc is introduced into the basis (chosen depending on its reduced cost), resulting in a cycle. As much flow as possible is sent along the cycle: The amount is determined by the cycle arc that first reaches its lower or upper limit. One of the arcs that first reach their limit will leave the basis.

To solve phase 1 of the BIMCF problem, we use a variation of the network simplex algorithm, starting from the initial solution \( x^0 = x^{lex(1,2)} \). As there are two cost components \( (c_1^a, c_2^a) \) associated with each arc \( a \) in the network, in the network simplex algorithm the reduced cost of each arc also consists of two components \( (\bar{c}_1^a, \bar{c}_2^a) \). In iteration \( j \) of the network simplex algorithm, candidate entering arcs (contained in \( S_j \)) are selected with minimal ratio of their reduced cost derived from the current supported efficient solution \( x^j \) as indicated in Procedure 1 (Sedeño-Noda and González-Martín 2000). Note that we denote by \( L_j \) and \( U_j \) sets of non-basic arcs with flow at their lower and upper bound, respectively:

\[
L_j = \{ a \in A : a \text{ is non-basic in BFS } x^j \text{ with } x^j_a = l_a \}
\]
\[
U_j = \{ a \in A : a \text{ is non-basic in BFS } x^j \text{ with } x^j_a = u_a \}
\]

**Procedure 1 compute_entering_arcs(\( \bar{c}, x^j \))**

1. **input**: Reduced costs \( \bar{c} = (\bar{c}_1^a, \bar{c}_2^a) \) derived from current efficient solution \( x^j \)
2. \( \mu_j = \min \{ \frac{\bar{c}_2^a}{\bar{c}_1^a} : a \in L_j \text{ with } \bar{c}_2^a < 0 \text{ and } \bar{c}_1^a > 0, \frac{\bar{c}_2^a}{\bar{c}_1^a} : a \in U_j \text{ with } \bar{c}_2^a > 0 \text{ and } \bar{c}_1^a < 0 \} \)
3. Let \( S_j \subseteq L_j \cup U_j \) be the set of non-basic arcs for which \( \mu_j \) is attained.
4. **output**: Minimal ratio \( \mu_j \) and set of non-basic candidate basic-entering arcs \( S_j \)

One of the candidate arcs \( a \in S_j \) is removed from \( S_j \) and enters the basis. By performing a simplex-pivot with entering arc \( a \), i.e. introducing \( a \) into the basis and removing the leaving arc from the basis, the reduced costs may change. The reduced costs of all arcs remaining in \( S_j \) are updated according to the BFS obtained by pivoting \( a \) into \( x^j \). As long as there are arcs remaining in \( S_j \) with

---

6
\(c_a^2 < 0, c_a^1 > 0\) and \(a \in L_j\) or \(c_a^2 > 0, c_a^1 < 0\) and \(a \in U_j\), the process repeats as in Procedure 2 (Sedoño-Noda and González-Martín 2000).

**Procedure 2 compute_next_BFS\([S_j, \tilde{c}, x^j]\)**

1. **input**: set of candidate basic-entering arcs \(S_j\) and reduced costs \(\tilde{c} = (\tilde{c}_a^1, \tilde{c}_a^2)\) derived from current efficient solution \(x^j\)
2. **while** \(S_j \neq \emptyset\) **do**
3. Let \(a\) be the first arc in \(S_j\), set \(S_j = S_j \setminus \{a\}\)
4. **if** \(\tilde{c}_a^2 < 0, \tilde{c}_a^1 > 0\) and \(a \in L_j\) \(\text{OR} \) \(\tilde{c}_a^2 > 0, \tilde{c}_a^1 < 0\) and \(a \in U_j\) **then**
5. Perform simplex-pivot with entering arc \(a\)
6. **end if**
7. **end while**
8. **output**: Next BFS \(x^{j+1}\)

The next BFS \(x^{j+1}\) might define an extreme point \(z(x^{j+1}) \in Z\). We denote the last extreme efficient solution that was found by \(x_k\). If for the new minimal ratio \(\mu_{j+1}\) we have \(\mu_{j+1} \neq \mu_k\), then \(x^{j+1}\) corresponds to an extreme efficient solution. If, on the other hand, \(\mu_{j+1} = \mu_k\), then \(x^{j+1}\) is not extreme, i.e. \(z(x^{j+1})\) corresponds to a supported non-extreme non-dominated point.

The parametric network simplex approach is given in Algorithm 1.

**Algorithm 1 Phase 1 BIMCF**

1. **input**: Network \((G, c, l, u)\) with \(c = (c^1, c^2)\)
2. Compute \(x^1 = x^{\text{lex}_1}(1, 2)\)
3. \(E_{ex} = \{x^1\}\)
4. Compute reduced costs \(\tilde{c}\) for \(x^1\)
5. \((\mu_1, S_1) = \text{compute_entering_arcs}[\tilde{c}, x^1]\)
6. \(j = 1, k = 1\)
7. **while** \(S_j \neq \emptyset\) **do**
8. \(x^{j+1} = \text{compute_next_BFS}[S_j, \tilde{c}, x^j]\)
9. Update \(\tilde{c}\) for \(x^{j+1}\)
10. \((\mu_{j+1}, S_{j+1}) = \text{compute_entering_arcs}[\tilde{c}, x^{j+1}]\)
11. **if** \(\mu_{j+1} \neq \mu_k\) **then**
12. \(\mu_{k+1} = \mu_{j+1}\) and \(x^{k+1} = x^{j+1}\)
13. \(E_{ex} = E_{ex} \cup \{x^{k+1}\}\)
14. \(k = k + 1\)
15. **end if**
16. \(j = j + 1\)
17. **end while**
18. **output**: Complete set of extreme efficient solutions \(E_{ex} = \{x^1, x^2, \ldots, x^{s-1}, x^s\}\)

The parametric simplex algorithm finishes when no candidate arcs to enter the basis can be found, i.e. when \(S_j = \emptyset\), and the \(\text{lex}(2, 1)\)-best solution is obtained.
The algorithm was originally proposed by Sedeño-Noda and González-Martín (2000). The authors do not include a check whether an efficient solution is extreme in their algorithm, but claim that every solution with \( x^{j+1} \neq x^j \) obtained by one execution of Procedure 2 is an extreme efficient solution. It can be easily seen that this is not true. A BFS with \( x^{j+1} \neq x^j \) is not necessarily an extreme efficient solution, i.e. \( z(x^{j+1}) \) is not necessarily an extreme point in \( Z \). The BFS \( x^{j+1} \) might represent a supported non-extreme efficient solution with \( z(x^{j+1}) \) on the facet between two extreme points in \( Z \). It is therefore incorrect that the computation of one set \( S_j \) per extreme efficient solution \( x^j \) is sufficient. In Sedeño-Noda et al. (2005) the algorithm is corrected as follows: The computation of a new point (Procedure 2) is modified by updating \( S_j \) when updating the reduced costs \( \bar{c} \) as long as the ratio \( \mu_j \) does not change. To make our paper self-contained, we include a proof of the correctness of Algorithm 1.

**Theorem 1** The set \( E_{ex} \) generated by Algorithm 1 is a complete set of extreme efficient solutions of BIMCF.

**Proof** Correctness of the parametric (network) simplex: By definition, \( X_{SE} \) is the set of optimal solutions to (5) for \( \lambda_1, \lambda_2 > 0 \). Dividing (5) by \( \lambda_1 \), we obtain that (5) is equivalent to

\[
\min \{(c^1 + \theta c^2)x : x \in X \} \quad \text{and} \quad \theta > 0.
\]

Since the constraint matrix of (2) is totally unimodular it is sufficient to find one optimal solution to

\[
\min \{(c^1 + \theta c^2)x : (2), (3) \}
\]

for each \( \theta > 0 \). This is a parametric linear programme which can be solved by the parametric simplex algorithm (Dantzig and Thapa 1997).

As an initial solution, we use \( x^{lex(1,2)} \), an extreme efficient solution. Clearly, \( x^{lex(1,2)} \) is efficient and an optimal solution to (5) with \( \lambda_1 = 1 \) and \( \lambda_2 = \epsilon \) for sufficiently small \( \epsilon > 0 \) (Isermann 1974), i.e. it is a supported efficient solution. Thus, there exists \( \theta > 0 \) such that \( x^{lex(1,2)} \) is an optimal solution to (6).

The entering variables in the parametric network simplex algorithm (in the case of upper bounds) are chosen from \( S_j \). At termination, it yields, for each \( \theta > 0 \) one optimal solution (a BFS) to (6), the parametric linear programme. Unlike in the parametric network simplex algorithm, we obtain a set of candidate arcs \( S_j \) and introduce the arcs contained in it into the basis one after the other (if they are still eligible to enter the basis). Therefore, the correctness of the algorithm critically depends on the fact that after pivoting \( s \in S_j \) into the BFS \( x^j \), the remaining arcs \( a \in S_j \setminus \{s\} \) that are still eligible, i.e. satisfy \( \bar{c}^a_a < 0, \bar{c}^a_a > 0 \) for \( a \in L_j \cap S_j \) or \( \bar{c}^a_a > 0, \bar{c}^a_a < 0 \) for \( a \in U_j \cap S_j \) (according to updated reduced costs) and still have the minimal ratio \( \mu_j \). Note that when pivoting an arc \( s \) with minimal ratio into the BFS of \( x^j \), the new minimal ratio of all eligible arcs is equal to or greater than the previous one, i.e. \( \mu_j \leq \mu_{j+1} \), which again follows from Dantzig and Thapa (1997) as the optimal value of the parametric linear programme (6) is a continuous piecewise linear concave
function of the parameter \( \theta \) (in our case with slopes given by the different values of \( \mu \)).

We need to show that \( a \in S_j \setminus \{ s \} \) either remains eligible with minimal ratio after the pivot, or is not eligible to enter the basis any more. If none or both dual variables \( \pi_{t(a)} \) and \( \pi_{h(a)} \) change, the reduced costs computed by \( c_a^k = c_a^k - \pi_{t(a)}^k + \pi_{h(a)}^k \), \( k = 1, 2 \) do not change (as both \( \pi_{t(a)} \) and \( \pi_{h(a)} \) change by the same amount), and therefore the ratio remains the same.

If only one of the dual variables \( \pi_{t(a)} \) changes to \( (\pi_{t(a)}^1 + \tilde{e}_s^1, \pi_{t(a)}^2 + \tilde{e}_s^2) \), updating reduced costs yields

\[
(\tilde{e}_a^k)_{\text{new}} = c_a^k - (\pi_{t(a)}^k + (\tilde{e}_s^k)_{\text{old}}) + \pi_{h(a)}^k = (\tilde{e}_a^k)_{\text{old}} - (\tilde{e}_s^k)_{\text{old}} \quad \text{for} \quad k = 1, 2.
\]

For both \( s, a \in S_j \), with respect to the “old” reduced costs, we have:

\[
\mu_j = \frac{(\tilde{e}_a^2)_{\text{old}}}{(\tilde{e}_a^1)_{\text{old}}} = \frac{(\tilde{e}_s^2)_{\text{old}}}{(\tilde{e}_s^1)_{\text{old}}} \Rightarrow (\tilde{e}_a^2)_{\text{old}} = \mu_j(\tilde{e}_a^1)_{\text{old}} \quad \text{and} \quad (\tilde{e}_s^2)_{\text{old}} = \mu_j(\tilde{e}_s^1)_{\text{old}}.
\]

We assume that the updated reduced costs of \( a \) still satisfy \( \tilde{e}_a^2 < 0, \tilde{e}_a^1 > 0 \) for \( a \in L_j \cap S_j \) or \( \tilde{e}_a^2 > 0, \tilde{e}_a^1 < 0 \) for \( a \in U_j \cap S_j \). It follows for the new ratio of \( \tilde{e}_a^2 \) and \( \tilde{e}_a^1 \):

\[
(\tilde{e}_a^2)_{\text{new}} = \frac{(\tilde{e}_a^2)_{\text{old}} - (\tilde{e}_s^2)_{\text{old}}}{(\tilde{e}_a^1)_{\text{old}} - (\tilde{e}_s^1)_{\text{old}}} = \frac{(\tilde{e}_a^1)_{\text{old}} - (\tilde{e}_s^1)_{\text{old}}}{(\tilde{e}_a^2)_{\text{old}} - (\tilde{e}_s^2)_{\text{old}}} = \mu_j.
\]

A complete set of extreme efficient solutions \( E_{ex} \) is obtained by Algorithm 1: Let \( y \in Z_N \) be a non-dominated extreme point. There exists \( x \in X \) with \( z(x) = y \) and \( x \) is an optimal solution to (5) for some \( \lambda_1, \lambda_2 > 0 \). Hence, \( x \) is an optimal solution to (6) with \( \theta = \frac{\lambda_2}{\lambda_1} \). Hence, the algorithm did find a BFS \( x' \in X \) with \( z(x') = z(x') = y \).

We modify an implementation of the single-objective network simplex algorithm called MCF (Löbel 2004) for our purposes. Löbel’s network simplex implementation takes advantage of strongly feasible trees (Cunningham 1976) to prevent cycling.

### 4.2 Phase 2 – Ranking \( k \) Best Flows

In phase 2, (at least) a complete set of the remaining supported non-extreme efficient solutions and non-supported efficient solutions is computed. The objective vectors of those solutions can only be situated in the triangle defined by two consecutive extreme points as indicated in Figure 2. Let \( z^1, \ldots, z^k \), \( z^l \) are sorted by increasing \( z_1 \), be the non-dominated extreme points obtained in phase 1. For each pair of neighbouring extreme points \( z^i \) and \( z^{i+1} \), we define weighting factors by

\[
\lambda_1 = z_1(x^{i+1}) - z_1(x^i) \quad \text{and} \quad \lambda_2 = z_2(x^i) - z_2(x^{i+1}).
\]

Using \( \lambda_1 \) and \( \lambda_2 \) in (5), we obtain a single-objective flow problem which has optimal solutions \( x^i \), \( x^{i+1} \) (of course all convex combinations of \( x^i \) and \( x^{i+1} \) are

\[
\lambda_1 z_1(x^{i+1}) + \lambda_2 z_2(x^{i+1}) = \lambda_1 z_1(x^i) + \lambda_2 z_2(x^i),
\]

\[
\text{for} \quad i = 1, \ldots, k-1.
\]
optimal as well). We denote the weighted sum objective by \( c_\lambda = \lambda_1 z_1(x) + \lambda_2 z_2(x) \).

Applying a \( k \) best flow algorithm by Hamacher (1995) to the single objective problem \( \min_{x \in X} c_\lambda \), we can generate feasible network flows in order of their cost. The \( k \) best flow algorithm is used to generate all feasible integer flows in the current triangle until it can be guaranteed that all non-dominated points have been found. Before we continue with the algorithm for phase 2, we explain the \( k \) best flow algorithm.

### 4.2.1 The \( k \) Best Flow Algorithm

We give a summary of the \( k \) best flow algorithm here, the reader is referred to Hamacher (1995) for a more detailed description and proofs. First, we outline the \( k \) best flow algorithm for the single-objective minimum cost flow problem. Starting with an optimal solution \( x \) in the network \( N \), a so-called incremental graph \( G_x \) is constructed in which every arc represents an arc in \( N \) on which flow may be increased or decreased. A cycle in \( G_x \) represents a change of flow that leads from \( x \) to another feasible flow. Identifying a minimal cycle in the incremental graph leads to a second best flow solution in \( N \). Now, the problem is partitioned by modifying one lower and upper bound on an arc of \( N \) so that in one partition the original solution is optimal and the second best solution is infeasible and vice versa. By iterating this process, a ranking of the \( k \) best solutions can be obtained.

In Hamacher (1995) the algorithm is designed to solve problems in networks with the property that there cannot be two arcs between the same nodes \( i \) and \( j \), no matter if they have the same or opposite directions. When solving BIMCF problems, randomly generated networks generally do not satisfy this property. Also, real-world networks will most likely not satisfy this property (e.g. road networks). We outline a generalisation of the algorithm in the following. The only difficulty with multiple arcs between a pair of nodes is keeping track of which arc in the incremental graph belongs to which arc in the original network.

First, construct the incremental graph, a directed graph \( G_x = (V, A_x) \) with arc costs \( c_x \), corresponding to an optimal flow \( x \) in \( N = (G, c, l, u) \) with

\[
    a^+ \in A_x^+ \quad \text{for all } a \in A \text{ and } x_a < u_a,
\]
\[
    a^- \in A_x^- \quad \text{for all } a \in A \text{ and } x_a > l_a,
\]

and let \( A_x = A_x^+ \cup A_x^- \). The arcs in \( A_x \) have the following relationship to arcs in \( A \): For each \( a^+ \in A_x \) we have \( t(a^+) = t(a) \), \( h(a^+) = h(a) \), and \( c_{a^+} = c_a \). For each \( a^- \in A_x \) we have \( t(a^-) = h(a) \), \( h(a^-) = t(a) \), and \( c_{a^-} = -c_a \). Note that in the following \( G_x \) is always constructed from the network \( N \) in which \( x \) is a best solution.

If for an \( a \in A \) both \( a^+ \in A_x \) and \( a^- \in A_x \) (which is the case only if \( l_a < x_a < u_a \)), we call \( a^+ \) and \( a^- \) a pair of symmetric arcs, otherwise we call an arc non-symmetric. A proper minimum cost cycle is a minimum cost cycle in \( G_x \), excluding all cycles that consist of one or multiple pairs of symmetric arcs.
\( a^+, a^-: \)

\[
\mathcal{C}(G_x) = \{ C : C \text{ cycle in } G_x \text{ with at least one arc } a^+ \in C \text{ and } a^- \notin C \\
or C \text{ cycle in } G_x \text{ with at least one arc } a^- \in C \text{ and } a^+ \notin C \}. \quad (9)
\]

From \( \mathcal{C}(G_x) \) a proper minimum cost cycle \( C \in \arg\min\{c^x(C) : C \in \mathcal{C}(G_x)\} \) is obtained. By increasing the flow by one unit along \( C \), a second-best flow \( \hat{x} \) is obtained in the original network \( N \): In \( N \), increasing the flow on arc \( a^+ \in A_x \) corresponds to increasing the flow on arc \( a \in A \), and increasing the flow on \( a^- \in A_x \) corresponds to decreasing the flow on arc \( a \in A \), see Procedure 3. This yields a second best flow \( \hat{x} \) with \( c(x) \leq c(\hat{x}) \), where \( c(\hat{x}) = c(x) + c^x(C) \).

**Procedure 3** compute_second_best_flow\([N, x, C]\)

1. **input:** \( N = (G, c, l, u) \), best solution \( x \), proper minimum cost cycle \( C \)
2. \( \hat{x}_a = \begin{cases} x_a + 1 & \text{if } a^+ \in C \cap A_x^+ \\ x_a - 1 & \text{if } a^- \in C \cap A_x^- \\ x_a & \text{otherwise} \end{cases} \)
3. **output:** second best flow \( \hat{x} \)

Now the network \( N \) is modified, by modifying one lower and upper bound of \( N \), so that \( x \) remains optimal in the network \( (G, c, l, u') \) with modified upper bound \( u' \) and \( \hat{x} \) is infeasible in this network. Also, new lower bounds \( l'' \) are derived, so that \( \hat{x} \) becomes optimal and \( x \) infeasible in \( (G, c, l'', u) \). In order to do this, the bounds of only one of the arcs \( a \) where flow was increased by one unit are modified as in Procedure 4. Note that this arc always exists as \( c(C) \geq 0 \) implies \( C \cap A^+ \neq \emptyset \).

**Procedure 4** derive_partition\([N, x, C]\)

1. **input:** \( N = (G, c, l, u) \), best solution \( x \), proper minimum cost cycle \( C \)
2. Select one arc \( \tilde{a} \) with \( \tilde{a}^+ \in C \cap A_x^+ 
3. \( u' = \begin{cases} u'_a = x_a & \text{if } a = \tilde{a} \\ u'_a = u_a & \text{otherwise} \end{cases} \) and \( l'' = \begin{cases} l''_a = x_a + 1 & \text{if } a = \tilde{a} \\ l''_a = l_a & \text{otherwise} \end{cases} \)
4. **output:** Network \( N' = (G, c, l, u') \) and network \( N'' = (G, c, l'', u) \)

In each of the two networks with modified bounds \( l'' \) and \( u' \), respectively, we can again compute a second best flow. Out of the two second best solutions, the flow with smaller cost is selected, this is the third best solution in the original network \( N \). The partition in which the third best flow was obtained is again partitioned and both new partitions resolved, etc. A pseudo-code for the \( k \) best flow algorithm is shown in Algorithm 2.

### 4.2.2 Adaptation of the \( k \) Best Flow Algorithm in Phase 2

When solving phase 2, we cannot specify a value of \( K \) a priori. Instead, we continue until it is guaranteed that all non-dominated points between \( z^i \) and \( z^{i+1} \) have been found.

We call \( z^i_{IN} = (z_1(x^{i+1}), z_2(x^i)) \) the local nadir point of the current triangle \( T_i \) given by the three points \( (z_1(x^i), z_2(x^i)), (z_1(x^{i+1}), z_2(x^{i+1})) \), and
Algorithm 2 $K$ best flows

1: **input:** $N = (G, c, l, u)$, best solution $x$, $K$
   /* For a network $N$ with best solution $x$, we denote the incremental graph by $G_x = (V, A_x)$ and it has costs $c^x$ */
2: $C \in \text{argmin}\{c^x(C) : C \in C(G_x)\}$ /* See (9) for $C(G_x)$ */
3: $\hat{x} = \text{compute_second_best_flow}[N, x, C]$
4: $P = \{(x, \hat{x}, N, C)\}$
5: $k = 2$
6: **while** $P \neq \emptyset$ and $k < K$ **do**
7: Choose $(x^p, \hat{x}^p, N^p, C^p)$ with $c(\hat{x}^p) = \min\{c(\hat{x}) : (x^q, \hat{x}^q, N^q, C^q) \in P\}$
8: $P = P \setminus \{(x^p, \hat{x}^p, N^p, C^p)\}$
9: $\{N', N''\} = \text{derive_partition}[N^p, x^p, C^p]$ /* $x^p, \hat{x}^p$ is best solution in $N', N''$, respectively */
10: **if** $C(G_{x^p}) \neq \emptyset$ **then**
11: $C' \in \text{argmin}\{c^{x_p}(C) : C \in C(G_{x^p})\}$ and $x' = \text{compute_second_best_flow}[N', x^p, C']$ /* See (9) for $C(G_{x^p})$ */
12: $P = P \cup \{(x^p, x', N', C')\}$
13: **end if**
14: **if** $C(G_{x^p}) \neq \emptyset$ **then**
15: $C'' \in \text{argmin}\{c^{\hat{x}_p}(C) : C \in C(G_{\hat{x}_p})\}$ and $x'' = \text{compute_second_best_flow}[N'', \hat{x}^p, C'']$ /* See (9) for $C(G_{\hat{x}_p})$ */
16: $P = P \cup \{(\hat{x}^p, x'', N'', C'')\}$
17: **end if**
18: save $k^{th}$ best flow $\hat{x}^p$
19: $k = k + 1$
20: **end while**
21: **output:** $2^{nd}, 3^{rd}, \ldots, k^{th}$ best flow and $k \leq K$

($z_1(x^{i+1})$, $z_2(x^i)$). For any solution $x$ that lies within the triangle $T_i$, $z_1(x^i) \leq z_1(x^{i+1})$ and $z_2(x^{i+1}) \leq z_2(x) \leq z_2(x^i)$ holds. The “worst” solution we are interested in, is the one that is one unit of cost better than $z_{i,N}$ in each objective. Its weighted objective value is an upper bound to the weighted sum of the two costs of any efficient feasible flow in the current triangle. Thus, initially, we enumerate $k$ best flows $x$ while

$$c_3(x) \leq u_\lambda \text{ with } u_\lambda = \lambda_1(z_1(x^{i+1}) - 1) + \lambda_2(z_2(x^i) - 1). \quad (10)$$

Whenever a solution with cost vector within the triangle is found that is not dominated by a solution found previously (and also not equivalent to a solution found previously), it is saved and the upper bound can be improved, as the new point dominates parts of the triangle. Let $\mathcal{E}_i = \{x^{i,0}, x^{i,1}, \ldots, x^{i,r}, x^{i,r+1}\}$ be a set of feasible non-equivalent solutions whose objective vectors are not dominating each other, and with image in the triangle $T_i$ defined by the supported efficient solutions $x^{i,0} = x^i$ and $x^{i,r+1} = x^{i+1}$. Furthermore, let the elements of $\mathcal{E}_i$ be ordered by increasing $z_1$-value, so that $z_1(x^{i,0}) < z_1(x^{i+1}) < \ldots < z_1(x^{i,r}) < z_1(x^{i,r+1})$. This yields the upper bound $\Delta$ such that $c_3(x) \leq \Delta$ for
all efficient solutions $x$ with image in $T_i$:

$$
\Delta = \max \{ \lambda_1(z_1(x^{i,j+1}) - 1) + \lambda_2(z_2(x^{i,j}) - 1), j = 0, \ldots, r \}.  \tag{11}
$$

For a detailed description of this bound, refer to Przybylski et al. (2008) or Raith and Ehrgott (2008). The phase 2 algorithm incorporating the upper bound and parts of the $k$ best flow algorithm is described in Algorithm 3.

**Algorithm 3 Phase 2 BIMCF**

1: input: Network $(G, c, l, u)$ with $c = (c^1, c^2)$, list of extreme efficient solutions $x^1, \ldots, x^s$
2: $i = 1$
3: while $i < s$ do
4: $E_i = \{x^i, x^{i+1}\}$ /* $x^{i,0} = x^i$ and $x^{i,r+1} = x^{i+1}$ */
5: Compute $\lambda_1$, $\lambda_2$ and $c_\lambda = \lambda_1 c^1 + \lambda_2 c^2$. /* See (8) for $\lambda_1$, $\lambda_2$ */
6: $\Delta = u_\lambda$ /* Initial value for $\Delta$. See (10) for $u_\lambda$ */
7: $C = \arg\min\{c_\lambda(C) : C \in C(G_x)\}$ /* See (9) for $C(G_x)$ */
8: $\hat{x} = \text{compute\_second\_best\_flow}[N, x^{i+1}, C]$
9: $P = \{(x^{i+1}, \hat{x}, N, C)\}$
10: while $P \neq \emptyset$ and $\min\{c_\lambda(\hat{x}^p) : (x^p, \hat{x}^p, N^p, C^p) \in P\} \leq \Delta$ do
11: Steps 7-16 in Algorithm 2 /* Execute one iteration of $k$ best flow */
12: if $z(\hat{x}^p)$ in current triangle and not dominated by the objective vector of any element of $E_i$ and $\hat{x}^p$ not equivalent to any $x \in E_i$ then
13: Insert $\hat{x}^p$ into $E_i$
14: Update $\Delta$ /* See (11) for $\Delta$ */
15: end if
16: end while
17: $i = i + 1$
18: end while
19: output: Complete set $E = \bigcup_{i=1, \ldots, s-1} E_i$ of efficient solutions

**Theorem 2** The set $E = \bigcup_{i=1, \ldots, s-1} E_i$ generated by Algorithm 3 is a complete set of efficient solutions of BIMCF.

**Proof** Without loss of generality, choose $i$ with $1 \leq i \leq s - 1$.

All solutions contained in $E_i$ obtained by Algorithm 3 are efficient: Whenever a solution $\hat{x}^p$ is first inserted into $E_i$, its objective vector lies within $T_i$ and is not dominated by the objective vector of any $x \in E_i$. and $\hat{x}^p$ is not equivalent to any $x \in E_i$. Suppose there exists $x \in X$ dominating $\hat{x}^p$, which implies $z(x) \leq z(\hat{x}^p)$ and therefore $c_\lambda(x) < c_\lambda(\hat{x}^p)$. Thus $x$ would have been found before $\hat{x}^p$ by the ranking procedure.

When Algorithm 3 stops ranking solutions in a triangle $T_i$ (i.e. when the upper bound $\Delta$ is exceeded), the set $E_i$ contains a complete set of efficient solutions within the triangle $T_i$: The ranking procedure enumerates all solutions $x$ with $c_\lambda(x) \leq \Delta$. Among all enumerated solutions, a complete set of efficient solutions $E_i$ is identified.
Assume that, after the ranking procedure stops, there exists an efficient solution \( x^e \notin \mathcal{E}_i \) within triangle \( T_i \) that is not equivalent to some \( x \in \mathcal{E}_i \). Ranking stops as soon as \( \min\{c_\lambda(x^p) : (x^p, \hat{x}^p, N^p, C^p) \in \mathcal{P} \} > \Delta \). As the solution \( x^e \notin \mathcal{E}_i \) was not obtained during the ranking process before it was stopped (otherwise \( x^e \) or an equivalent solution would be in \( \mathcal{E}_i \)), it follows that \( c_\lambda(x^e) \geq \min\{c_\lambda(\hat{x}^p) : (x^p, \hat{x}^p, N^p, C^p) \in \mathcal{P} \} > \Delta \).

We always have \( |\mathcal{E}_i| \geq 2 \), therefore \( z_1(x^{i,j}) < z_1(x^e) < z_1(x^{i,j+1}) \) and \( z_2(x^{i,j}) > z_2(x^e) > z_2(x^{i,j+1}) \) for some \( j \in \{j' : j' = 0, \ldots, r \} \) and \( z_1(x^{i,j+1}) - z_1(x^{i,j}) \geq 2 \) and \( z_2(x^{i,j}) - z_2(x^{i,j+1}) \geq 2 \). In particular, \( z_1(x^e) \leq z_1(x^{i,j+1}) - 1 \) and \( z_2(x^e) \leq z_2(x^{i,j}) - 1 \). We can derive the following contradiction:

\[
\lambda_1 z_1(x^e) + \lambda_2 z_2(x^e) \leq \lambda_1(z_1(x^{i,j+1}) - 1) + \lambda_2(z_2(x^{i,j}) - 1) \text{ or } c_\lambda(x^e) \leq \Delta.
\]

This contradicts the existence of an efficient solution \( x^e \notin \mathcal{E}_i \) within \( T_i \) that is not equivalent to some solution in \( \mathcal{E}_i \) and that was not obtained during the ranking procedure.

\[ \square \]

### 4.3 Remarks

If there are multiple equivalent efficient solutions the ranking algorithm will obtain them all. In the phase 2 approach described in Algorithm 3, however, only one of them is inserted into the set \( \mathcal{E}_i \). All efficient solutions can be found by slightly altering the phase 2 approach to keep equivalent solutions. Also, another component needs to be considered when calculating the upper bound to ensure that ranking is not terminated before all equivalent solutions have been obtained. The value of \( \Delta \) is still computed as in (11), and we add a component \( \gamma \) as follows:

\[
\begin{align*}
\gamma &= \max_{j = 1, \ldots, r} \{ \lambda_1 z_1(x^{i,j}) + \lambda_2 z_2(x^{i,j}) \} , \\
\Delta' &= \max\{ \Delta, \gamma \} .
\end{align*}
\]

As we aim at obtaining a complete set of efficient solutions, in our implementation we only keep one of the equivalent solutions for each non-dominated point in objective space. The algorithm could be slightly modified by changing \( \Delta \) to \( \Delta' \) as in (12) and modifying Step 12 in Algorithm 3 to keep every efficient solution \( \hat{x}^p \) including those that are equivalent to a previously obtained efficient solution.

Unfortunately the \( k \) best flow algorithm will generate solutions with objective vector outside the current triangle which cannot be removed from \( \mathcal{P} \) as those solutions might later lead to other solutions within the triangle. Whenever a solution \( x^* \) with cost outside the current triangle lies within another triangle \( T \), we could save this solution and use it to compute a better upper bound \( \Delta' \) in \( T \). This will, however, not speed up the algorithm, as flows in \( T \) still have to be ranked starting from the least cost flow. When ranking flows in \( T \), one of the following two cases occurs:

- Ranking flows and updating the upper bound in \( T \) stops the algorithm before the solution \( x^* \) is enumerated, or
Table 1: Test Instances: NETGEN

<table>
<thead>
<tr>
<th>Name</th>
<th>n</th>
<th>m</th>
<th>sources</th>
<th>sinks</th>
<th>(\sum_{i \in V: b_i &gt; 0} b_i)</th>
<th>transshipment sources</th>
<th>transshipment sinks</th>
</tr>
</thead>
<tbody>
<tr>
<td>N01 / F01</td>
<td>20</td>
<td>60</td>
<td>9</td>
<td>7</td>
<td>90 / 100</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>N02 / F02</td>
<td>20</td>
<td>80</td>
<td>9</td>
<td>7</td>
<td>90 / 100</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>N03 / F03</td>
<td>20</td>
<td>100</td>
<td>9</td>
<td>7</td>
<td>90 / 100</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>N04 / F04</td>
<td>40</td>
<td>120</td>
<td>18</td>
<td>14</td>
<td>180 / 100</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>N05 / F05</td>
<td>40</td>
<td>160</td>
<td>18</td>
<td>14</td>
<td>180 / 100</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>N06 / F06</td>
<td>40</td>
<td>200</td>
<td>18</td>
<td>14</td>
<td>180 / 100</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>N07 / F07</td>
<td>60</td>
<td>180</td>
<td>27</td>
<td>21</td>
<td>270 / 100</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>N08 / F08</td>
<td>60</td>
<td>240</td>
<td>27</td>
<td>21</td>
<td>270 / 100</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>N09 / F09</td>
<td>60</td>
<td>300</td>
<td>27</td>
<td>21</td>
<td>270 / 100</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>N10 / F10</td>
<td>80</td>
<td>240</td>
<td>35</td>
<td>38</td>
<td>350 / 100</td>
<td>17</td>
<td>14</td>
</tr>
<tr>
<td>N11 / F11</td>
<td>80</td>
<td>320</td>
<td>35</td>
<td>38</td>
<td>350 / 100</td>
<td>17</td>
<td>14</td>
</tr>
<tr>
<td>N12 / F12</td>
<td>80</td>
<td>400</td>
<td>35</td>
<td>38</td>
<td>350 / 100</td>
<td>17</td>
<td>14</td>
</tr>
</tbody>
</table>

- Ranking flows in \(T\) generates the solution \(x^*\) again, now the bound is updated to \(\Delta^*\) (or a better upper bound \(\Delta < \Delta^*\)).

Thus, saving solutions in other triangles cannot improve the run-time of phase 2.

We can use smaller triangles than those given by extreme efficient solutions: We can consider intermediate solutions whenever the flow between two adjacent solutions obtained in phase 1 changes by \(\delta > 1\) along the cycle connecting the two solutions. Intermediate solutions can be used to construct \(\delta - 1\) smaller triangles. Due to the nature of the phase 2 algorithm, including those smaller triangles instead of the one defined by the two extreme efficient solutions does not present an advantage. The ranking algorithm would generate the same rankings \(\delta - 1\) times as we cannot restrict the ranking to the current triangle. There is also no advantage in a better upper bound, as the ranking algorithm will first generate all alternative optimal solutions (i.e. the non-extreme supported efficient solutions including the intermediate solutions), and after that the upper bound will be as good as it would be in the smaller triangles.

5 Numerical Results

We investigate the performance of our solution method with networks generated by NETGEN (Klingman et al. 1974), which is slightly modified to include a second objective function. We generate two sets of test instances, with the following parameters fixed for all problems: \(\text{mincost} = 0\), \(\text{maxcost} = 100\), \(\%\text{highcost} = 0\), \(\%\text{capacitated} = 100\), \(\text{mincap} = 0\), and \(\text{maxcap} = 50\). Furthermore, we vary parameters as in Table 1. We generate 30 problems for each set of parameters. We generate problems N01-N12 with varying sum of supply \(\sum_{i \in N: b_i > 0} b_i\) and problems F01-F12 with fixed total sum of supply, as we observe that increasing the sum of supply with the network size significantly complicates the problem. All NETGEN instances are listed in Table 1.

We also generate networks with a grid structure. Nodes are arranged in a rectangular grid with given height and width. Every node has at most four outgoing arcs (up, down, left, and right), to its immediate neighbours. Only
nodes on the boundary of the grid have fewer outgoing arcs. A grid is defined by the parameters height $h$, width $w$, maximum cost $c_{max}$, maximum capacity $u_{max}$, and sum of supply $\sum_{i \in V} b_i > 0 b_i$. Nodes are randomly chosen to be demand-, supply-, or transshipment nodes with probabilities 0.4, 0.4, and 0.2, respectively. It is, however, possible that some demand- or supply-nodes are assigned a balance of 0. Instances G01-G04 are created with the same number of nodes as instances N01-N12 and the same $\sum_{i \in V} b_i > 0 b_i$. In instances G05/G06 and G09/G10 we increase $u_{max}$ of G03 and G04, respectively. In instances G07/G08 and G11/G12 we decrease $c_{max}$ of G03 and G04, respectively. Again, we generate 30 problems for each set of parameters. All grid instances are listed in Table 2.

All numerical tests are performed on a Linux (Ubuntu 7.04) computer with 2.80GHz Intel Pentium D processor and 1GB RAM. We use the gcc compiler (version 4.1) with compile option -O3. The methods are implemented in C. When measuring run-time, we disregard the time it takes to read the problem from a problem file. Run-time does include the generation of all non-dominated points together with the efficient flows. Run-time is measured with a precision of 0.01 seconds.

In Table 3 - Table 5 we show the average, minimum, and maximum $|Z_N|$, average $|Z_{SN}|/|Z_{NN}|$, and average, minimum, and maximum CPU time for the two different NETGEN instances (N and F), and the grid instances (G), respec-
Table 4: Results for problems F01 – F12

| Name | $|Z_N|/|Z_{NN}|$ average | time in seconds average | average | min | max | average | min | max |
|------|-----------------|-----------------|-----------------|------|------|-----------------|------|------|
| F01  | 181.13          | 24              | 491             | 0.27 | 0.52 | 0.04            | 2.81 |
| F02  | 260.53          | 15              | 685             | 0.24 | 0.99 | 0.02            | 4.58 |
| F03  | 353.77          | 158             | 788             | 0.20 | 1.54 | 0.28            | 6.41 |
| F04  | 213.87          | 65              | 380             | 0.20 | 2.44 | 0.58            | 5.58 |
| F05  | 354.10          | 144             | 701             | 0.15 | 5.19 | 1.86            | 11.77|
| F06  | 478.87          | 176             | 714             | 0.13 | 9.20 | 2.53            | 33.65|
| F07  | 263.97          | 48              | 410             | 0.16 | 7.17 | 0.87            | 22.40|
| F08  | 343.23          | 165             | 860             | 0.14 | 13.48| 5.31            | 41.27|
| F09  | 454.17          | 230             | 950             | 0.12 | 21.35| 8.18            | 47.9 |
| F10  | 146.43          | 72              | 277             | 0.18 | 8.80 | 2.75            | 17.27|
| F11  | 277.90          | 131             | 680             | 0.15 | 19.64| 8.38            | 54.04|
| F12  | 414.50          | 234             | 693             | 0.12 | 34.03| 12.57           | 66.47|

Table 5: Results for problems G01 – G12

| Name | $|Z_N|/|Z_{NN}|$ average | time in seconds average | average | min | max | average | min | max |
|------|-----------------|-----------------|-----------------|------|------|-----------------|------|------|
| G01  | 74.13           | 5               | 276             | 0.52 | 0.11 | 0.00            | 0.79 |
| G02  | 211.23          | 37              | 817             | 0.27 | 1.99 | 0.09            | 10.54|
| G03  | 256.07          | 86              | 592             | 0.22 | 8.72 | 2.22            | 33.23|
| G04  | 354.20          | 58              | 1092            | 0.20 | 21.20| 2.40            | 99.01|
| G05  | 319.67          | 64              | 1034            | 0.21 | 8.90 | 1.45            | 23.48|
| G06  | 420.60          | 106             | 955             | 0.19 | 12.17| 2.66            | 37.72|
| G07  | 194.63          | 39              | 433             | 0.30 | 6.78 | 0.44            | 25.18|
| G08  | 235.33          | 25              | 477             | 0.27 | 8.00 | 0.61            | 40.42|
| G09  | 477.33          | 176             | 1094            | 0.17 | 34.38| 6.00            | 293.53|
| G10  | 397.77          | 113             | 1069            | 0.19 | 21.54| 2.04            | 65.64|
| G11  | 265.93          | 35              | 541             | 0.27 | 25.61| 1.33            | 55.53|
| G12  | 326.80          | 109             | 645             | 0.20 | 21.27| 5.62            | 70.89|

respectively. We make the following observations (see Tables 3 to 5):

- When fixing the number of nodes $n$ in a network but increasing the number of arcs $m$ the number of non-dominated points $|Z_N|$ increases. This is illustrated by instances N01-N12 and F01-F12.
- For all presented instances, we can observe that the more non-dominated points there are in a problem, the longer the run-time of the algorithm. Despite the instances being fairly small, they have a lot of non-dominated points.
- The sum of supply significantly increases the number of non-dominated points, which can be seen by comparing the results for problems F01-F12 with the corresponding results of problems N01-N12. It is, however, more realistic to increase $\sum_{i\in V} b_i > 0 b_i$ while increasing the network size.
- We generate grid network instances G01-G04 with similar numbers of nodes and arcs as instances F01-F12 and N01-N12 generated by NETGEN. Comparing the number of non-dominated points of G01-G04 to those of (corresponding similar sized networks) N01-N12 we observe that there are (on average) always fewer non-dominated points in the grid networks. This is not the case when comparing the average number of
non-dominated points of G03 and G04 to those of F07 and F10/F11, respectively.

- When decreasing $c_{\text{max}}$ in grid instances G07/G08 and G11/G12, we observe that smaller $c_{\text{max}}$ leads to fewer non-dominated points and thus to a faster run-time. When increasing $u_{\text{max}}$ in G05/G06, the number of solutions increases and so does the run-time. But increasing $u_{\text{max}}$ to 100 in G10 leads to less solutions than increasing $u_{\text{max}}$ to 75 in G09.

- $|Z_{SN}|/|Z_{NN}|$, the ratio of supported and non-supported non-dominated points, is decreasing when the total number of solutions is increasing for NETGEN instances, on average, an observation also made by Sedeño-Noda and González-Martín (2001). For grid instances there seems to be the same trend, but the total number of solutions does not increase as much. In most NETGEN and grid instances, less than 20% and 30% of all solutions are supported, respectively. Thus, the majority of non-dominated points are non-supported.

- In Figures 3 - 5, the non-dominated points of one instance of each of the classes F01, N01, and G01 are shown. This illustrates that most non-supported points are in fact very close to the boundary of $\text{conv}(Z)$. The given figures are just three examples, but we observe a similar behaviour in most of the problem instances. By obtaining only the supported non-dominated points of a problem, a fairly good approximation of the set of non-dominated points can be obtained. There are, however, exceptions such as the example in Figure 6, where there are a lot of non-supported points far from the boundary of $\text{conv}(Z)$.
6 Conclusion

The presented two phase algorithm solves the BIMCF problem, but the problems solved within reasonable run-time are fairly small. It is therefore worth investigating how to increase the performance of the presented algorithm to make it possible to solve bigger problems. Future research could address the extension of the the two phase algorithm for BIMCF to the MIMCF problem with more than two objectives. This can be done along the lines of Przybylski et al. (2007), where a two phase method for multiobjective integer programming is presented together with an example of the application to the assignment problem with three objectives.

References


