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

November 22, 2007

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 supported 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 some networks with 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 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 supported efficient solutions (efficient solutions which define extreme points of the convex hull of feasible solution vectors in objective space) are computed with a simplex-based algorithm. Other efficient solutions are computed in the second phase using a ranking algorithm on restricted areas of the objective space.

We test our algorithm on different problem instances generated with the well known network generator `NETGEN` and also on networks with a grid structure.

The rest of 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, that is the two phase method with a parametric network simplex approach in phase 1 and the k best flow algorithm in phase 2. 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 is introduced.

Let $G = (N, A)$ be a *directed network* with a set of nodes $N = \{1, \ldots, n\}$ and a set of arcs $A \subseteq N \times N$ with $a = (i, j) \in A$ and $|A| = m$. Two non-negative costs $c_a = (c_a^1, c_a^2) \in \mathbb{N} \times \mathbb{N}$ 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 $l_a \leqq u_a$ on every arc a. An integer numerical value b_i, the balance, is associated with each node. The 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:

$$\min \quad z(x) = \begin{cases} z_1(x) = \sum_{a \in A} c_a^1 x_a \\ z_2(x) = \sum_{a \in A} c_a^2 x_a \end{cases} \tag{1}$$

$$\text{s.t.} \quad \sum_{\{a:a=(i,j)\in A\}} x_a - \sum_{\{a:a=(j,i)\in A\}} x_a = b_i \quad \forall i \in N \tag{2}$$

$$l_a \leqq x_a \leqq u_a \quad \text{for all } a \in A \tag{3}$$

$$x_a \text{ integer} \quad \text{for all } a \in A. \tag{4}$$

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

We assume $l_a = 0$ in the following. 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).

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

$$\begin{aligned} y^1 \leqq y^2 \quad &\Leftrightarrow \quad y_k^1 \leqq y_k^2 \; k = 1, 2, \\ y^1 \leq y^2 \quad &\Leftrightarrow \quad y_k^1 \leqq y_k^2 \; k = 1, 2; \; y^1 \neq y^2, \text{ and} \\ y^1 < y^2 \quad &\Leftrightarrow \quad y_k^1 < y_k^2 \; k = 1, 2. \end{aligned}$$

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*

$$\min_{x \in X} \lambda^1 z_1(x) + \lambda^2 z_2(x) \tag{5}$$

  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 $conv(Z)$ of the feasible set in objective space.

- *Supported efficient solutions which define an extreme point of $conv(Z)$ are called* extreme *supported efficient solutions.*

- *The remaining efficient solutions in $X_{NE} := X_E \backslash X_{SE}$ 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 $conv(Z)$. The set of non-supported non-dominated points is denoted by Z_{NN}.*

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 $x \in X \backslash X_E$ are either dominated or equivalent to at least one $x \in X_E$.*

The presented solution approach computes a complete set 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 $l \in \{1,2\} \backslash \{k\}$. Then $z(\hat{x}) \leqq_{lex(k,l)} z(x')$ if either $z_k(\hat{x}) < z_k(x')$ or both $z_k(\hat{x}) = z_k(x')$ and $z_l(\hat{x}) \leqq z_l(x')$. We call $\hat{x}$ a $lex(k,l)$-best solution if $z(\hat{x}) \leqq_{lex(k,l)} z(x)$ for all $x \in X$. Let $x_{lex(k,l)}$ denote a $lex(k,l)$-best solution.*

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 the 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 solutions, consist of two phases, also known as the two phase method. In the first phase a complete set of supported efficient solutions, or at least the extreme ones, is computed. In the second phase all remaining 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 minimum cost flow problem can be used in phase 1 of BIMCF to find a complete set of extreme supported solutions, e.g. Lee and Pulat (1991); Pulat et al. (1992); Sedeño-Noda and González-Martín (2000, 2003). To solve the continuous problem it is sufficient to generate all extreme supported solutions. The algorithms presented by Lee and Pulat (1991); Pulat et al. (1992) may generate some non-extreme supported solutions, whereas the algorithms by Sedeño-Noda and González-Martín (2000, 2003) generate extreme supported solutions only.

Lee and Pulat (1991) remark that their procedure can be extended to generate all integer efficient solutions with image on the edges of $conv(Z)$, i.e. all supported solutions. Every efficient solution found by their algorithm corresponds to a basic tree and two solutions are called *adjacent* if the two corresponding trees differ in only two arcs. Whenever the flow changes by δ when moving from one efficient solution to an adjacent one, they propose to increase the flow

stepwise by $1, 2, \ldots, \delta - 1$ to obtain all *intermediate* solutions and claim to obtain all supported solutions this way. This is incorrect, as not all non-extreme supported 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, 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 supported 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) can not generate a complete efficient set.

Figueira (2002) present an approach where ε-constraint problems are repeatedly solved via branch-and-bound to obtain non-supported solutions.

In the following, we summarise a thesis and two recent reports that were not included in Hamacher et al. (2007). Do Castelo Batista Gouveia (2002) uses a k best flow algorithm to enumerate all solutions of biobjective network flow problems, including of course all efficient solutions. She analyses the number of feasible flows, of efficient solutions, and of non-dominated points in the problem.

Eusébio and Figueira (2006) give examples of networks, where for a supported extreme and supported non-extreme non-dominated point, basic and non-basic efficient solutions exist. It is known that there is always a basic 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, so that it may be impossible to obtain *all* efficient solutions when using a simplex-based method. Eusébio and Figueira (2006) also give a network in which supported solutions exist that can not be obtained as intermediate solutions between two extreme supported solutions.

In a more recent report, Eusébio and Figueira (2007) illustrate and prove that supported solutions are indeed connected via chains of zero-cost cycles in the incremental graph constructed from basic feasible solutions corresponding to extreme supported solutions. They use this relationship to obtain all supported solutions to a BIMCF problem. The same result can be obtained by considering a weighted sum objective (5) for which two neighbouring extreme supported solutions are optimal. The suppported points on the edge of $conv(Z)$ 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 algrithm is also based on cycles in the incremental graph. We

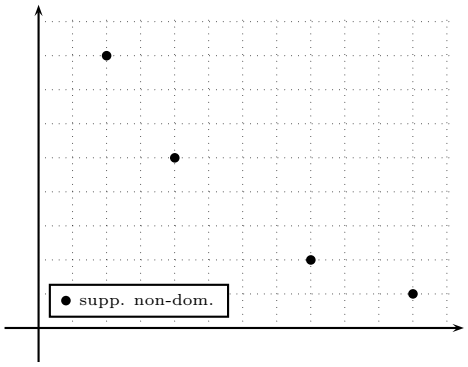

Figure 1: Supported non-dominated points.

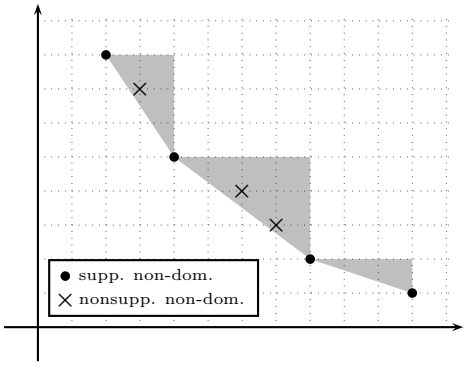

Figure 2: All non-dominated points.

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 formulation 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 supported efficient solutions are computed, possibly taking advantage of their property of being obtainable as solutions to the weighted sum problem (5), for an illustration see Figure 1. In phase 2 the remaining supported and non-supported efficient solutions 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.

4.1 Phase 1 – Parametric Simplex

In phase 1 of the two phase method, we compute a complete set of extreme supported 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 accordingly modified objective. From the initial solution, a network simplex algorithm is employed, always choosing a basis entering arcs with the least ratio of improvement of z_2 and worsening of z_1. If there is more than one arc with minimal ratio, one of them is chosen as entering arc, and the others are saved in a list of candidates. After the arc entered the basis, the

reduced costs of the remaining arcs in the candidate list are reevaluated. As long as there are remaining candidate arcs in the list violating optimality for the second objective, one of them is introduced into the basis, then the reduced costs of the remaining arcs are reevaluated until there is no more candidate arc in the list or all remaining candidate arcs are optimal with respect to the second objective. Now, an extreme supported solution is obtained and a new list of candidate arcs with minimal ratio is computed. The procedure generates extreme supported solutions moving in a left-to-right fashion. The parametric simplex algorithm finishes when no candidate arcs to enter the basis can be found, i.e. the $lex(2,1)$-best solution is obtained.

We modify an implementation of the network simplex algorithm called MCF (Löbel 2003) for our purposes. The network simplex implementation takes advantage of strongly feasible trees (Cunningham 1976) to prevent cycling.

We could derive some non-extreme supported solutions here, by considering intermediate solutions whenever the flow between two solutions changes by $\delta > 1$. We would obtain $\delta - 1$ non-extreme supported solutions. We do not implement this – an explanation follows at the end of Section 4.2.

4.2 Phase 2 – Ranking k Best Flows

In phase 2, a complete set of the remaining supported non-extreme solutions and non-supported solutions is computed. The objective vectors of those solutions can only be situated in the triangle defined by two consecutive supported extreme points as indicated in Figure 2. Let $z^1, \ldots, z^s$, where $z^i = (z_1(x^i), z_2(x^i))$ and z^i are sorted by increasing z_1, be the extreme supported points obtained in phase 1. For each pair of neighbouring extreme supported points z^i and z^{i+1}, we define weighting factors by

$$\lambda^1 = z_1(x^{i+1}) - z_1(x^i) \text{ and } \lambda^2 = z_2(x^i) - z_2(x^{i+1}). \tag{6}$$

Using λ_1 and λ_2 in (5), we obtain a single-objective flow problem which has optimal solutions z^i, z^{i+1} (of course all supported solutions between z^i and z^{i+1} are optimal as well). Applying a k best flow algorithm by Hamacher (1995) to problem (5), we can generate feasible network flows in order of their cost. The k best flow algorithm is used to generate all feasible 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 briefly outline the k best flow algorithm for the single-objective minimum cost flow problem. Starting with an optimal solution x in the network G, a so-called *incremental graph* G_x is constructed in which every arc represents an arc in G 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 G. Now, the problem is partitioned by modifying bounds on arcs of G 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 can not 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). 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. We thus enumerate arcs in the original network, for notation we use the unique arc identifier $a \in A$.

First, construct the incremental graph $G_x = (N, A_x)$ corresponding to an *optimal flow* x in G with

$$\begin{aligned} (i,j)_a \in A_x^+ \text{ with cost } \tilde{c}_{ij_a} = c_a \quad &\text{if} \quad a = (i,j) \in A \text{ and } x_a < u_a, \\ (j,i)_a \in A_x^- \text{ with cost } \tilde{c}_{ji_a} = -c_a \quad &\text{if} \quad a = (i,j) \in A \text{ and } x_a > l_a = 0, \\ A_x \quad &= \quad A_x^+ \cup A_x^-. \end{aligned}$$

If, for an arc $a = (i,j) \in A$, both $(i,j)_a$ and $(j,i)_a \in A_x$, we call $(i,j)_a$ and $(j,i)_a$ *symmetric* arcs, otherwise an arc is called *non-symmetric*.

Next, a *proper minimum cost cycle* (operation $properMinimalCycle$ in Algorithm 1) in the incremental graph G_x can be obtained by

- For all symmetric arcs $(i,j)_a$:
  find minimum cost cycle C_{ija} in $G_x = (N, A_x \backslash \{(j,i)_a\})$.
- For all non-symmetric arcs $(i,j)_a$:
  find minimum cost cycle C_{ija} in $G_x = (N, A_x)$.
- Choose proper minimum cost cycle $C \in argmin\{c(C_{ija}) : (i,j)_a \in A_x\}$

Sending one unit of flow along the proper minimal cycle C, we obtain a second best flow in the original network G. In $G = (N, A)$, increasing the flow on arc $(i,j)_a \in A_x^+$ corresponds to increasing the flow on arc $a \in A$, and increasing the flow on $(j,i)_a \in A_x^-$ corresponds to decreasing the flow on arc $a \in A$. This yields a second best flow $\hat{x}$.

Now one upper bound on G is modified, so that x remains optimal in G with modified bounds l, u' and $\hat{x}$ is infeasible in this network. Also, another set of bounds l', u is derived, so that $\hat{x}$ becomes optimal, x infeasible. The bounds of one of the arcs $\tilde{a}$ where flow was increased by one unit are modified. The increased flow on this arc is $\hat{x}_{\tilde{a}} = x_{\tilde{a}} + 1$ and we derive bounds:

$$u'_a = \begin{cases} x_a \text{ if } a = \tilde{a} \\ u_a \text{ otherwise} \end{cases} \tag{7}$$

$$l'_a = \begin{cases} x_a + 1 \text{ if } a = \tilde{a} \\ l_a \text{ otherwise} \end{cases} \tag{8}$$

Algorithm 1 K best flows

input: Network $G = (N, A)$ with cost c, lower bounds $l = (0, \ldots, 0)$, upper bounds u, optimal solution x, max number of flows K
$Partitions = \{\}$ /* list of partitions, ordered by cost of second best flows in the partition. Every element contains (x, l, u, C) where C is the minimal cost cycle from which the second best flow can be derived */
Find *properMinimalCycle* C in G_x derived from x, c, l, u
$Partitions = \{(x, l, u, C)\}$
$k = 2$
while $Partitions$ is nonempty and $k < K$ **do**
    $(x_p, l_p, u_p, C_p) = Partitions[1]$ /* element with least cost second best flow */
    derivePartitions x_p, l_p, u'_p and $\hat{x}_p, l'_p, u_p$
    Find *properMinimalCycle* C in G_{x_p} (incremental graph for x_p, c, l_p, u'_p)
    Find *properMinimalCycle* $\hat{C}$ in $G_{\hat{x}_p}$ (incremental graph for $\hat{x}_p, c, l'_p, u_p$)
    Insert (x_p, l_p, u'_p, C) and $(\hat{x}_p, l'_p, u_p, \hat{C})$ into $Partitions$ /* so that the order of $Partitions$ is maintained */
    save k^{th} best flow $\hat{x}_p$
    k=k+1
end while
output: $2^{\text{nd}}, 3^{\text{rd}}, \ldots, k^{\text{th}}$ best flow and $k \leqq K$

In Algorithm 1 we call this operation *derivePartitions*.

In each of the networks with modified bounds l, u' and l', u, 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, which is again partitioned and resolved, etc. A pseudo-code is shown in Algorithm 1.

4.2.2 Adaptation of the k Best Flow Algorithm in Phase 2

When solving phase 2, we can not specify a value of K a priori. Instead, we continue until it is guaranteed that all efficient points between z^i and z^{i+1} have been found.

We call $z^i_{lN} = (z_1(x^{i+1}), z_2(x^i))$ the *local nadir point* of the current triangle. The "worst" solution we are interested in, is the one that is one unit of cost better than z^i_{lN} 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

$$\lambda^1 z_1(x) + \lambda^2 z_2(x) \leqq u_\lambda \text{ with } u_\lambda = \lambda^1(z_1(x^{i+1}) - 1) + \lambda^2(z_2(x^i) - 1). \quad (9)$$

Whenever an efficient solution with cost vector within the triangle is found, it is saved and the upper bound can be improved, as the new point dominates parts of the triangle. For a detailed description of how the upper bound is updated, please refer to Przybylski et al. (2008) or Raith and Ehrgott (2007). The phase 2 algorithm is described in Algorithm 2.

Algorithm 2 Phase 2 BIMCF

input: Network (N, A) with cost $z = (c^1, c^2)$, lower bound $l = (0, \ldots, 0)$, upper bound u, list of extreme supported solutions $z^1, \ldots, z^s$
$i = 1$
while $i < s$ **do**
  Compute λ_1, λ_2 (6), the upper bound u_λ (9), and $c = \lambda_1 c^1 + \lambda_2 c^2$.
  Find $properMinimalCycle$ C in $G_{x^{i+1}}$ derived from x^{i+1}, c, l, u
  $Partitions = \{(x, l, u, C)\}$
  while ($Partitions$ is nonempty) and (cost of second best flow in $Partitions[1]$: $c(x_p) + c(C_p) \leqq u_\lambda$) **do**
    Steps 7-11 in Algorithm 1 /* Execute one iteration of k best flow */
    if $z(\hat{x}_p)$ in current triangle and not domianted by any point in the triangle found so far **then**
      Insert $\hat{x}_p$ into list of efficient solutions, and eleminate other solutions that are now dominated.
      Update u_λ if possible.
    end if
  end while
  $i = i + 1$
end while
output: Complete set of non-extreme efficient solutions

Unfortunately the k best flow algorithm will generate solutions with objective vector outside the current triangle which cannot be removed 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 Δ, we could save this solution and use it to compute a better upper bound u^*_λ in Δ. This will, however, not speed up the algorithm, as we still have to rank flows in Δ starting from the least cost flow. There are two possibilities:

- Ranking flows and updating the upper bound in Δ stops the algorithm before the solution x^* is enumerated, or
- Ranking flows in Δ generates the solution x^* again, now the bound is updated to u^*_λ (or an even better value than that) anyway.

Thus, saving solutions in other triangles cannot improve the run-time of phase 2. We remarked at the end of Section 4.1 that we could consider intermediate solutions whenever the flow between two adjacent solutions obtained in phase 1 changes by $\delta > 1$. Due to the nature of the phase 2 algorithm, including intermediate solutions from phase 1 and thus obtaining $\delta - 1$ smaller triangles instead of the one defined by the two extreme solutions does not present an advantage. The ranking algorithm would generate the same rankings $\delta - 1$ times as we can not 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 solutions including the intermediate solutions), and after that the upper bound will be as good as it would be in the smaller triangles.

Table 1: Test Instances: NETGEN

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

All efficient solutions are found by this phase 2 approach. In our implementation, however, only a complete set of solutions is saved.

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: $mincost = 0$, $maxcost = 100$, $\%highcost = 0$, $\%capacitated = 100$, $mincap = 0$, and $maxcap = 50$. 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}, max capacity u_{max}, and sum of supply $\sum_{i \in N: b_i > 0} b_i$. 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 $\sum_{i \in N: b_i > 0} b_i$. 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.

Table 2: Test Instances: Grid

Name	h	w	n	m	c_{max}	u_{max}	$\sum_{i \in N: b_i > 0} b_i$
G01	4	5	20	62	100	50	100
G02	5	8	40	134	100	50	100
G03	6	10	60	208	100	50	100
G04	8	10	80	284	100	50	100
G05	6	10	60	208	100	75	100
G06	6	10	60	208	100	100	100
G07	6	10	60	208	25	50	100
G08	6	10	60	208	50	50	100
G09	8	10	80	284	100	75	100
G10	8	10	80	284	100	100	100
G11	8	10	80	284	25	50	100
G12	8	10	80	284	50	50	100

Table 3: Results for problems N01 – N12

	$\lvert Z_N \rvert$			$\lvert Z_{SN} \rvert / \lvert Z_{NN} \rvert$	time		
Name	average	min	max	average	average	min	max
N01	168.13	15	392	0.28	0.40	0.01	1.45
N02	271.13	66	852	0.22	0.76	0.09	3.17
N03	375.43	126	702	0.18	1.40	0.27	3.78
N04	455.10	137	879	0.15	7.09	1.67	26.36
N05	660.63	252	1801	0.14	11.84	3.16	36.95
N06	948.30	266	2280	0.12	22.58	5.05	74.91
N07	867.80	410	1399	0.11	42.21	11.48	94.32
N08	1510.37	531	2834	0.09	90.88	27.11	245.20
N09	1553.47	808	2448	0.09	112.62	32.77	238.82
N10	1138.77	552	1901	0.10	125.42	46.44	372.95
N11	2036.20	989	4109	0.08	289.05	69.97	559.34
N12	2480.70	1287	3921	0.07	397.94	138.38	813.76

5.1 Numerical Results

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.

We make the following observations:

- When fixing the number of nodes n in a network but increasing the number of arcs m the number of efficient solutions increases, this is illustrated by instances N01-N12 and F01-F12.

- For all presented instances, we can observe that the more efficient solutions there are in a problem, the longer the run-time of the algorithm. Despite the instances being fairly small, they have a lot of solutions.

- For problem type F10, the number of efficient solutions is lower, on average, than that of problems F01, F04, and F07 although they all have the same ratio n/m. This happens, because the value of $\sum_{i \in N: b_i > 0} b_i$ is

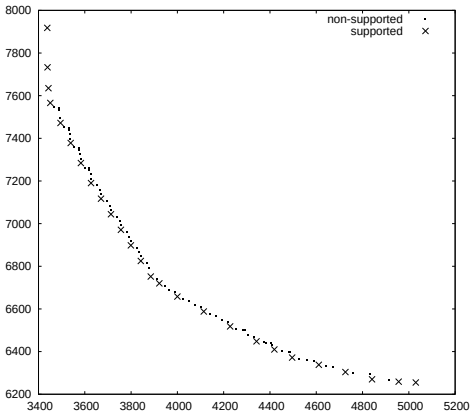

Figure 3: All non-dominated points of one instance of class F01.

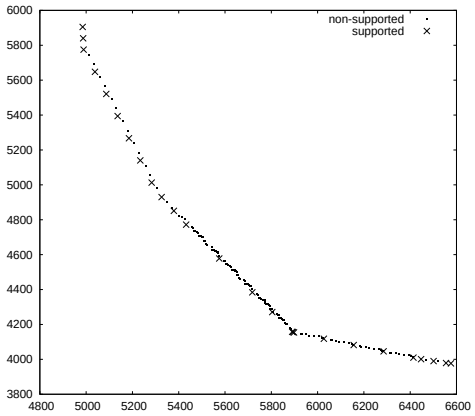

Figure 4: All non-dominated points of one instance of class N01.

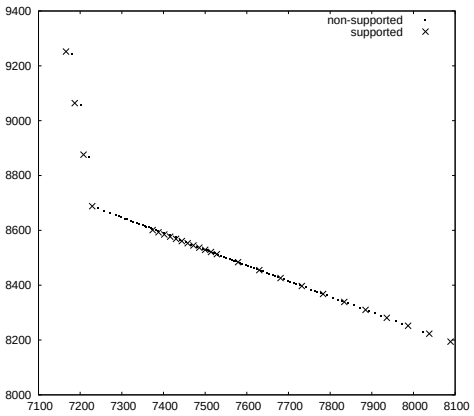

Figure 5: All non-dominated points of one instance of class G01.

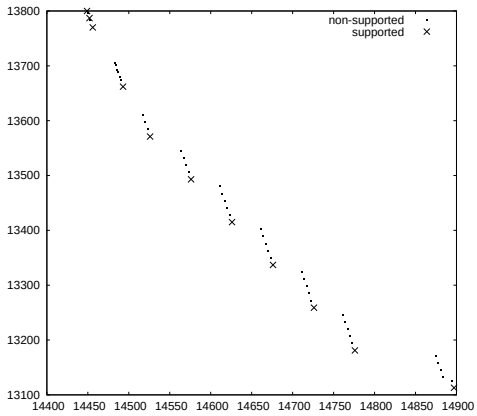

Figure 6: All non-dominated points of one instance of class G02.

Table 4: Results for problems F01 – F12

Name	$\lvert Z_N \rvert$ average	min	max	$\lvert Z_{SN} \rvert / \lvert Z_{NN} \rvert$ average	time 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	203.97	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	$\lvert Z_N \rvert$ average	min	max	$\lvert Z_{SN} \rvert / \lvert Z_{NN} \rvert$ average	time 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	420.6	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	23.61	1.33	55.53
G12	326.80	109	645	0.20	21.27	5.62	70.89

fixed, in problem F10 there are only 100 units of flow shipped through the network consisting of 80 nodes.

- The sum of supply significantly increases the number of efficient solutions, 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 $\sum_{i \in N: b_i > 0} b_i$ while increasing the network size.

- We generate grid network instances G01-G04 similar to instances F01-F12 and N01-N12 generated by `NETGEN`. Comparing the number of solutions of G01-G04 to those of N01-N12 we observe that there are (on average) always fewer solutions in the grid networks. This is not the case when comparing the average number of solutions of G03 and G04 to those of F07 and F10/F11, respectively.

- When decreasing c_{max} in grid instances G07/G08 and G11/G12, we observe that smaller c_{max} leads to fewer efficient solutions and thus to a faster run-time. When increasing u_{max} in G05/G06, the number of solutions increases and so does the run-time. But increasing u_{max} to 100 in G10 leads to less solutions that increasing u_{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. 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 respectively 20% and 30% of all solutions are supported. Thus, the majority of solutions is 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 $conv(Z)$. The given figures are just three examples, but we observe a similar behaviour in most of the problem instances. By obtaining only the supported solutions of a problem, a fairly good approximation of the set of efficient solutions 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 $conv(Z)$.

6 Conclusion

The presented two phase algorithm works well to solve 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 multi-objective integer programming is presented together with an example of the application to the assignment problem with three objectives.

References

R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

W.H. Cunningham. A network simplex method. *Mathematical Programming*, 11:105–116, 1976.

M. do Castelo Batista Gouveia. The bi-criteria network flow problem: A study of an algorithm for searching efficient solutions (in Portuguese). Master's thesis, Faculdade de Economia da Universidade de Coimbra, 2002.

A. Eusébio and J.R. Figueira. Why is it difficult to find non-dominated solutions in bi-criteria network flow problems when using simplex based methods? Technical report, Instituto Superior Técnico, Portugal, 2006. submitted.

A. Eusébio and J.R. Figueira. A note on the computation of supported non-dominated solutions for bi-criteria network flow problems. Technical Report Working Paper N3/2007, Instituto Superior Técnico, Portugal, 2007.

J.R. Figueira. On the integer bi-criteria network flow problem: A branch-and-bound approach. Technical report, Université Paris-Dauphine, 2002. Cahier du LAMSADE no 191.

H.W. Hamacher. A note on k best network flows. *Annals of Operations Research*, 57:65–72, 1995.

H.W. Hamacher, C.R. Pedersen, and S. Ruzika. Multiple objective minimum cost flow problems: A review. *European Journal of Operational Research*, 176:1404–1422, 2007.

F. Huarng, P. Pulat, and Ravindran A. An algorithm for bicriteria integer network flow problem. In *Proceedings of the 10th International Conference on Multiple Criteria Decision Making, Taipei, Taiwan*, volume 3, pages 305–318, 1992.

D. Klingman, A. Napier, and J. Stutz. NETGEN: A program for generating large scale assignment, transportation, and minimum cost flow problems. *Management Science*, 20:814–821, 1974.

H. Lee and P.S. Pulat. Bicriteria network flow problems. *European Journal of Operational Research*, 51:1190126, 1991.

H. Lee and P.S. Pulat. Bicriteria network flow problems: Integer case. *European Journal of Operational Research*, 66(1):148–157, 1993.

A. Löbel. MCF, version 1.3. `http://www.zib.de/Optimization/Software/Mcf/`, 2003.

A. Przybylski, X. Gandibleux, and M. Ehrgott. The biobjective integer minimum cost flow problem–incorrectness of Sedeño-Noda and González-Martín's algorithm. *Computers & Operations Research*, 33(5):1459–1463, 2006.

A. Przybylski, X. Gandibleux, and M. Ehrgott. A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. Technical report, LINA, Université de Nantes, 2007.

A. Przybylski, X. Gandibleux, and M. Ehrgott. Two-phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research*, 185(2):509–533, 2008.

P.S. Pulat, F. Huarng, and H Lee. Efficient solutions for the bicriteria network flow problem. *European Journal of Operational Research*, 19(7):649–655, 1992.

A. Raith and M. Ehrgott. A comparison of solution strategies for biobjective shortest path problems. Technical Report 648, Department of Engineering Science, The University of Auckland, 2007. `http://www.esc.auckland.ac.nz/research/tech/tech.html`.

G. Ruhe. Complexity results for multicriteria and parametric network flows using a pathological graph of Zadeh. *Zeitschrift für Operations Research*, 32: 59–27, 1988.

A. Sedeño-Noda and C. González-Martín. The biobjective minimum cost flow problem. *European Journal of Operational Research*, 124:591–600, 2000.

A. Sedeño-Noda and C. González-Martín. An algorithm for the biobjective integer minimum cost flow problem. *Computers & Operations Research*, 28 (2):139–156, 2001.

A. Sedeño-Noda and C. González-Martín. Am alternative method to solve the biobjective minimum cost flow problem. *Asia-Pacific Journal of Operational Research*, 20:241–260, 2003.

P. Serafini. Some considerations about computational complexity for multi-objective combinatorial problems. In J. Jahn and W. Krabs, editors, *Recent advances and historical development of vector optimization*, volume 294 of *Lecture Notes in Economics and Mathematical Systems*, pages 222–232. Springer Verlag, Berlin, 1986.

E. L. Ulungu and J. Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1995.