BIOBJECTIVE MINIMUM COST FLOW PROBLEM

ADDIS ABABA UNIVERSITY
COLLEGE OF COMPUTATIONAL AND NATURAL SCIENCES
DEPARTMENT OF MATHEMATICS

"In partial fulfillment of the requirement of the degree of master of science in mathematics"

By: Bedilu Tsige Mekuria
Stream : Optimazation
Advisor: Berhanu Guta(PhD)

July 24, 2013

Abstract

In this project basically theory and algorithms for a single commodity flow problems simultaneously optimizing two objectives (Biobjective) are addressed. For both continuous and integer case methods like Biobjective network simplex method, two phase method and weighted metric methods are presented to compute the complete set of efficient solution in the objective space . In addition to these main ideas, the project also explains some points about multiobjective optimization problem and network flow problems.

Keywords: Network programming; Biobjective minimum cost flow problem; Efficient extreme points; Biobjective Network simplex method; Two phase method; weighted metric method.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor **Dr.Berhanu Guta** for his generous advice throughout the preparation of this project.

I also express my thanks to all **staff members of the department of mathematics and to all my friends** for their unmentioned cooperation on my project.

Finally I thanks my **family** for their endless love and support.

Contents

Abstract **i**

Acknowledgements **ii**

1 Introduction **1**

2 Preliminary **4**

2.1 Multi-objective optimization problem(MOOP) and Network flow problem 4

2.1.1 Multi-objective optimization problem(MOOP) 4

2.1.2 Methods to solve Multiobjective optimization problems(MOOP) . 6

2.1.3 Graphs and Network flows 11

3 Biobjective minimum cost flow problem **22**

3.1 Introduction 22

3.2 Methods to solve Biobjective minimum cost flow problems 23

3.2.1 Biobjective network simplex method 23

3.2.2 Two-phase method 27

3.2.3 Weighted metric methods([8]) 32

Reference **47**

Chapter 1

Introduction

In the selection of the best optimal solution on many network optimization problems more than one criterion can be considerd. For example, in Transportation problems, the criteria that can be used are the minimization of the cost for selected routes, the minimization of arrival times at the point of destination, the minimization of the deterioration of goods, the minimization of the load capacity that would not be used in the selected vehicles, the maximization of safety, reliability,etc. Thus leading to Multiobjective optimization problems (MOOP).In Multiobjective optimization problems (MOOP) there are more than one objective functions and there is no single optimal solution that simultaneously optimizes all the objective functions. In these case the decision makers are looking for the most preferred solution. In MOOP the concept of optimality is replaced with that of efficiency or Pareto optimality. The efficient (or Pareto optimal, non-dominated, non-inferior) solutions are the solutions that cannot be improved in one objective function without deteriorating their performance in at least one of the rest.

In this project we are going to deal with Biobjective minimum cost flow(BMCF) problem assuming that flows are both continuous and integer variables, taking advantage of the matrix property of the feasible region. Biobjective minimum cost flow(BMCF) problem is a particular case of the multiobjective linear programming(MOLP) and therefore BMCF problem can be solved by general MOLP techniques. However, the special feature of BMCF can be exploited to design more efficient algorithm. The aim in BMCF is to find efficient solutions. To solve these efficient solutions in the objective space, methods like Biobjective network simplex method, two phase method and weighted metric methods are presented.

Formulation of Biobjective minimum cost flow problem ([1],[3],[8],[9])

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

biobjective network flow(BNF) problem can be formalized in the following way:

$$\min f_1(x) = \sum_{i \in \mathbf{V}} \sum_{j \in \mathbf{suc(i)}} c_{ij}^1 x_{ij} \tag{1.1}$$

$$\min f_2(x) = \sum_{i \in \mathbf{V}} \sum_{j \in \mathbf{suc(i)}} c_{ij}^2 x_{ij} \tag{1.2}$$

$$\text{subject to : } \sum_{j \in \mathbf{suc(i)}} x_{ij} - \sum_{j \in \mathbf{pre(i)}} x_{ij} = b_i \qquad \forall i \in \mathbf{V} \tag{1.3}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in \mathbf{A} \tag{1.4}$$

Any vector x that satisfies equation(1.3) and (1.4) is called a feasible solution. The set of feasible solution or decision space is denoted by X.
If we define $f(x) = (f_1(x), f_2(x))$ then the image of X through f is
$f(X) = \{(f_1(x), f_2(x)) : x \in \mathbf{X}\}$. Thus $f(X)$ is called the objective space.

Basic concepts and results([8],[9])

For k = 1,2, let x^* be the solution of the problem

$$f_k^* = f_k^*(x^*) = min_{x \in X} \sum_{i \in V} \sum_{i \in suc(i)} c_{ij}^k x_{ij}; \qquad k = 1, 2 \tag{1.5}$$

So f_k^* is the optimum value for the single objective flow problem that results when only the k^{th} objective is taken into account.
In general, there is no feasible solution of the BNF problem that simultaneously minimizes f_1 and f_2. In other words, no optimum global solution exists. Therefore, the levels $f^* = (f_1^*, f_2^*)$ define the utopia or ideal point. Because of this difficulty, the solutions of the BNF problem are searched from among the set of efficient points, that is, points that satisfy the following definition.

Definition A feasible solution $x \in X$ of the BNF problem is efficient if and only if, there does not exist any feasible solution $\hat{x} \in X$ such that $f_k(\hat{x}) \leq f_k(x)$ for both k with $f_k(\hat{x}) \neq f_k(x)$ for at least one k, k = 1, 2.([6])

We will denote by $E[X]$ the set of efficient solutions of X and $E[f(X)] = \{f(x) \mid x \in E[X]\}$. We refer to $E[f(X)]$ as the set of efficient solutions of $f(X)$. At least one efficient solution exists among the solutions of problem above equation. Therefore, from now on we consider that x_1^*, $x_2^* \in E[X]$. The resolution of the BNF problem will be carried out by an adequate selection of one efficient solution of the sets $E[X]$ or $E[f(X)]$. However,finding all the points of these sets can be very expensive because the number of efficient solutions in the objective space can be very large. Nevertheless,given the characteristics of X and $f(X)$ in this case,$E[X]$ can be generated by a finite subset of its points: *the efficient extreme points*. Let $E_{ex}[X]$ be the set of the efficient extreme points of X and let $E_{ex}[f(X)]$ be the corresponding points of $f[X]$. Our intention is to identify $E_{ex}[f(X)]$, and by this set, to identify $E_{ex}[X]$. We denote by $\gamma(x^i, x^j)$ the edge of the compact polyhedral set X which connects the extreme points x^i and x^j.

Definition Let $\gamma(x^i, x^j)$ be an edge which connects two adjacent efficient extreme points x^i and x^j in X. Then $\gamma(x^i, x^j)$ is called an efficient edge.([8])

Generation of efficient solutions([10])

The following single objective problem $P(\lambda)$ can be used to obtain efficient solutions:

$$\min \quad \{\lambda f_1(x) + (1-\lambda) f_2(x)\}; \qquad s.t \qquad x \in X; \tag{1.6}$$

where $0 \leq \lambda \leq 1$. Let S_λ be the set of solutions of the $P(\lambda)$ problem. We define $f(S_\lambda) = \{f(x) \mid x \in S_\lambda\}$.

Proposition 1.0.1 **(i)** $\forall x \in E[X]$, $\exists \lambda \in [0,1]$ *such that* $x \in S_\lambda$

(ii) *If* $\lambda \in (0,1)$ *then* $S_\lambda \subseteq E[X]$. *If* $\lambda = 0$, *then* $x_2^* \in S_0$ *and if* $\lambda = 1$, $x_1^* \subseteq S_1$.*([10])*

The above proposition establishes a correspondence between the set $E[X]$ and the interval [0,1] through the $P(\lambda)$ problem. The following proposition allows us to state that S_λ contains a single efficient extreme point of $E[X]$ or an efficient edge of this polyhedron.

Proposition 1.0.2 *If* λ *is such that* $0 < \lambda < 1$, *then:*

(i) S_λ *is an unitary set or*

(ii) $\exists x_1; x_2 \in E[X]$ *with* $x_1 \neq x_2$, *such that* $S_\lambda = \{\lambda x_1 + (1-\lambda) x_2 \mid \lambda \in [0,1]\}$.*([10])*

The detail of the project and some technical words which are used in this chapter are presented on the next two consecutive chapters.

Chapter 2

Preliminary

2.1 Multi-objective optimization problem(MOOP) and Network flow problem

2.1.1 Multi-objective optimization problem(MOOP)

A multi-objective optimization problem has a number of objective functions which are to be minimized or maximized. As in the single objective optimization problem ,the multi-objective optimization usually has a number of constraints which any feasible solution(including the optimal solution)must satisfy.
The multi-objective optimization problem generally formulated as:([4],[5])

$$\text{minimize/maximize } f_m(x) \qquad \text{m=1,2,...,M;} \tag{2.1}$$

$$\text{subject to : } g_j(x) \geq 0, \qquad j = 1, 2, ..., J; \tag{2.2}$$

$$h_k(x) = 0, \qquad k = 1, 2, ..., K; \tag{2.3}$$

$$x_i^L \leq x_i \leq x_i^u, \qquad i = 1, 2, ..., n; \tag{2.4}$$

A variable x is a vector of n decision variables : $x = (x_1, x_2, ..., x_n)^T$. Equation (2.4) are called variables bound restricting each decision x_i to take a value with a lower x_i^L and an upper x_i^U bound. This bound constitute a decision variable space D or simply the decision space. The term g_i and h_k are constrained functions.
Any $x \in R^n$ that satisfies all constraints and variable bounds is known as a $feasible\ solution$. Otherwise it is called $infeasible\ solution$.
Multiobjective optimization is referred as Vector optimization because a vector objectives instead of a single objective is optimized.
If all objective functions and constraints are linear then the multiobjective optimization problem(MOOP) is called multiobjective linear programm(MOLP). Thus, the objective functions are

$$f_k(x) = c_k^T x \qquad \text{k= 1, . . . , p,}$$

where $c_k \in R^n$. The constraints $g_j(x) \leq 0$ are summarily written in matrix form and as equality constraints $Ax = b$.
As usual in linear programming we restrict the variables to the nonnegative orthant of $R^n : x \geq 0$.
A multiple objective linear program (MOLP) is given by

$$\min Cx$$

$$\text{subject to : } \; Ax = b$$

$$x \geq 0$$

with a $p \times n$ objective or criteria matrix C consisting of the rows c_k^T, k = 1, . . . , p. The feasible set in decision space is defined by the $m \times n$ constraint matrix A and the right hand side vector $b \in R^m$. ([2])
The feasible set in decision space is

$$X = \{x \in R^n : Ax = b, x \geq 0\}.$$

The feasible set in objective space is

$$Y = CX = \{Cx : x \in X\}.$$

Minimizing a vector-valued objective function needs some more explanation since there is no canonical ordering defined in R^p for $p \geq 2$. Throughout this article we use the Pareto concept of optimality for MOLPs which is based on the following two binary relations $<$ and $\leq$. Let y^1, $y^2 \in R^p$. Then

$$y^1 \leq y^2 :\Leftrightarrow, y_k^1 \leq y_k^2 \qquad \forall k = 1, ..., p \text{ and } y^1 \neq y^2$$
$$y^1 < y^2 :\Leftrightarrow, y_k^1 < y_k^2 \qquad \forall k = 1, ..., p$$

A point $y^2 \in R^p$ is called dominated by $y^1 \in R^p$ if $y^1 \leq y^2$.

Definition Let $\hat{x} \in X$ be a feasible solution of the MOLP and let $\hat{y} = C\hat{x}$.

1. $\hat{x}$ is called weakly efficient if there is no $x \in X$ such that $Cx < C\hat{x}$; $\hat{y} = C\hat{x}$ is called weakly nondominated.

2. $\hat{x}$ is called efficient if there is no $x \in X$ such that $Cx \leq C\hat{x}$; $\hat{y} = C\hat{x}$ is called nondominated.

3. $\hat{x}$ is called properly efficient if it is efficient and if there exists a real number $M > 0$ such that for all i and x with $c_i^T x < c_i^T \hat{x}$ there is an index j and $M > 0$ such that $c_j^T x > c_j^T \hat{x}$ and $\frac{c_i^T \hat{x} - c_i^T x}{c_j^T x - c_j^T \hat{x}} \leq M$.([6],[9])

Definition A subset X of R^n is said to be convex if $\alpha x^1 + (1 - \alpha)x^2 \in X$ for any two pair of $x^1, x^2 \in X$ and any $\lambda \in [0, 1]$. ([6],[4],[5])

Definition A function $f : R^n \to R$ is a convex function if for any two pair of $x^1, x^2 \in R^n$ the following condition is true $f(\lambda x^1 + (1-\lambda)x^2) \leq \lambda f(x^1) + (1-\lambda)f(x^2)$ for all $0 \leq \lambda \leq 1$

Definition A multiobjective optimization problem is convex if all objective functions are convex and the feasible region is convex set.([6],[4],[5])

Proposition 2.1.1 *The feasible sets X in decision space and Y in objective space of the MOLP are convex and closed.([6],[4],[5])*

2.1.2 Methods to solve Multiobjective optimization problems(MOOP)

Weighted sum method

The weighted sum method, as the name suggests, scalarizes a set of objectives in to a single objective by pre-multiplying each objective with a user supplied weight. This method is the simplest approach and is probably the most widely used classical approach. It is formulated as:([4],[5])

$$\text{minimize } F(X) = \sum_{m=1}^{n} w_n f_n(x) \tag{2.5}$$

$$\text{subject to} : g_j(x) \geq 0 \qquad j = 1, 2, ...J$$
$$h_k(x) = 0 \qquad k = 1, 2, ...K$$
$$x_i^L \leq x_i \leq x_i^u, \qquad i = 1, 2, ..., n;$$

where $w_n \in [0, 1]$ is the weight of the n^{th} objective function and $\sum_{m=1}^{n} w_n = 1$.

Theorem 2.1.2 *The solution of the problem represented by the equation (2.5) is pareto optimal if the weight is positive for all objectives.([4],[5])*

Proof Let x^* be a solution of the weighting problem with positive weighting coefficient. Let us suppose that it is not pareto optimal. This means that there is a solution x such that $f_i(x) \leq f_i(x^*)$ for all i = 1,2,...,k and $f_j(x) < f_j(x^*)$ for at least one j. Since $w_i > 0$ for all i = 1,2,...,k, we have

$$\sum_{i=1}^{k} w_i f_i(x) < \sum_{i=1}^{k} w_i f_i(x^*).$$

This contradicts the assumption that x^* is a solution of the weighting problem and, thus x^* must be pareto optimal.

Theorem 2.1.3 *If x^* is a pareto-optimal solution of a convex multiobjective optimization problem, then there exist a nonzero positive weight w such that x^* is a solution to the problem given by equation (2.5). ([4],[5])*

Advantages of weighted sum method

This is probably the simplest way to solve a Multiobjective optimization problem. The concept is intuitive and easy to use, for the problems having a convex pareto-optimal, this method guarantees finding solution on the entire pareto-optimal set.([4],[5])

Disadvantages of weighted sum method

Since most single objective optimization algorithms are designed to find a solution that satisfy the first optimality criterion, more test are needed to know whether the obtained solution is truly a minimum solution. This increases the computational complexity of the weighted sum approach. In interesting MOOPs,there may exists multiple minimum solution for a specific weighted vector. However each of this minimum solution may represent a different solution in the pareto-optimal set. Some of these solution can be weakly dominated to each other,thereby westing the search effort. Moreover,if the chosen single objective optimization algorithm can not find all minimum solutions for a weighted vector, some pareto-optimal solution can not be found.([4],[5])

ε-constraint method

In order to alleviate the difficulties faced by weighted sum approach in solving problems with non convex objective functions ,the ε-constraint method is used. This method keeps one of the objective and restricts the rest of the objectives within user specified values. The modified problem is as follows:([4],[5])

$$\begin{aligned} &\text{minimize } f_\mu(X), \\ \text{subject to : } &f_m(x) \leq \varepsilon_m \qquad m = 1, 2, ...M \; and \; m \neq \mu \\ &g_j(x) \geq 0 \qquad j = 1, 2, ...J \\ &h_k(x) = 0 \qquad k = 1, 2, ...K \\ &x_i^L \leq x_i \leq x_i^u, \qquad i = 1, 2, ..., n; \end{aligned} \tag{2.6}$$

In the above formulation the parameter ε_m represents upper bound of the value of f_m and need not necessarily mean a small value close to zero.

Theorem 2.1.4 *The unique solution of the ε-constraint problem stated in equation(2.6) is pareto-optimal for any given upper bound vector $\varepsilon = (\varepsilon_1, ..., \varepsilon_{\mu-1}, \varepsilon_{\mu+1}..., \varepsilon_m)^T$. ([4],[5])*

Proof Let x^* be a unique solution of the ε-constraint problem. This mean that $f_l(x^*) < f_l(x)$ for all x in the decision space when $f_j(x) \leq \varepsilon_j$ for every $j = 1, 2, ..., k$, $j \neq l$. Let us assume that x^* is not pareto optimal. In this case, there is a vector x^0 in the decision space such that $f_i(x^0) \leq f_i(x^*)$ for all $i = 1, 2, ..., k$ and the inequality is strict for at least one index j. If $j = l$, this means that $f_l(x^0) < f_l(x^*)$ and $f_i(x^0) \leq f_i(x^*) \leq \varepsilon_j$ for all $i \neq l$. Here we have a contradiction with the fact x^* is a solution of the ε-constraint problem. On the other hand, if $j \neq l$, then $f_j(x^0) < f_j(x^*) \leq \varepsilon_j$, $f_i(x^0) \leq \varepsilon_j$ for all $i \neq l$ and $f_l(x^0) \leq f_l(x^*)$. This is a contradiction to x^* as a unique solution of the ε-constraint problem, and x^* has to be pareto optimal.

Advantages

Different pareto-optimal solution can be found by using different ε_m values. The same method can be used for the problems having convex or non convex objective spaces alike. In terms of the information needed from the user, this algorithm is similar to the weighted sum approach. In this approach a vector of ε values representing in the some sense, the location of the pareto-optimal solution is needed. However, the advantage of this method is that it can be used for any arbitrary problem with either convex or non convex objective space.([4],[5])

Disadvantages

The solution of the above problem largely depend on the chose of ε vector. It must be chosen so that it lies within the minimum or maximum value of the individual objective function. Moreover as the number of objective increases there exist more element in the ε vector, thereby requiring more information from the user.([4],[5])

Weighted metric methods

Instead of using a weighted sum of the objectives, other means of combining multiple objectives in to a single objective can also be used for this purpose weighted metrics such as l_p and l_∞ distance metrics are often used. For non-negative weights the weighted l_p distance measure of any solution x from the ideal solution z^* can be minimized as follows:([4],[5])

$$\begin{aligned} \text{minimize} \quad & l_p(X) = (\sum_{m=1}^{M} w_m |\ f_m(x) - z_m^*\ |^p)^{\frac{1}{p}} \\ \text{subject to}: \ & g_j(x) \geq 0 \qquad j = 1, 2, ...J \\ & h_k(x) = 0 \qquad k = 1, 2, ...K \\ & x_i^L \leq x_i \leq x_i^u, \qquad i = 1, 2, ..., n; \end{aligned} \tag{2.7}$$

The parameter p can take any value between 1 and ∞. when $p = 1$ is used, the resulting problem is equivalent to the weighted sum approach. When $p = 2$ is used, a weighted Euclidean distance of any point in the objective space from the ideal point is minimized. When a large p is used the above problem is reduced to a problem of minimizing the largest deviation $|\ f_m(x) - z_m^*\ |$. This problem has a spacial name - the weighted Tchebycheff problem:

$$\begin{aligned} \text{minimize} \ & l_\infty(X) = max_{m=1}^{M} w_m |\ f_m(x) - z_m^*\ | \\ \text{subject to}: \ & g_j(x) \geq 0 \qquad j = 1, 2, ...J \\ & h_k(x) = 0 \qquad k = 1, 2, ...K \\ & x_i^L \leq x_i \leq x_i^u, \qquad i = 1, 2, ..., n; \end{aligned} \tag{2.8}$$

however, the resulting optimal solution obtained by the chosen l_p depend on the parameter p. When the weighted Tchebycheff metric is used, any pareto-optimal solution can be found.([4],[5])

Theorem 2.1.5 *Let x^* be a pareto-optimal solution then there exist a positive weighting vector such that x^* is a solution of the weighted Tchebycheff problem shown in equation(2.8), where the reference point is the utopian objective vector Z^*.([4],[5])*

Proof Let x^* in decision space be pareto optimal. Let us assume that there does not exist a weighting vector $w > 0$ such that x^* is a solution of the weighted Tchebycheff problem. We know that$f_i(x) > z_i^*$ for all i = 1,2,...,k and for all x in the decision space. Now we choose $w_i = \frac{\beta}{f_i(x)-z_i^*}$ for all i = 1,2,...,k where $\beta > 0$ is same normalizing factor. If x^* is not a solution of the weighted Tchebycheff problem, there exists another point x^0 in decition space that is a solution of the weighted Tchebycheff problem , mean that

$$\begin{aligned}\max_i[w_i(f_i(x^0) - z_i^*)] &< \max_i[w_i(f_i(x^*) - z_i^{**})] \\ &= \max_i[\tfrac{\beta}{f_i(x)-z_i^*}(f_i(x^*) - z_i^*)] = \beta.\end{aligned}$$

Thus $w_i(f_i(x^0) - z_i^*) < \beta$ for all i = 1,2,...,k, this mean that

$$\tfrac{\beta}{f_i(x^0)-z_i^*}(f_i(x^*) - z_i^*) < \beta$$

and after simplifying the expression we have $f_i(x^0) < f_i(x^*)$ for all i = 1,2,...,k. Here we have a contradiction with the pareto optimality of x^* , which completes the proof.

Advantages

The weighted Tchebycheff metric guarantee finding each and every pareto-optimal solution when z^* is a utopian objective vector. Although in the above discussion only l_p metrics are suggested, other distance metrics are also used.([4],[5])

Disadvantages

Since different objectives may take values of different orders of magnitude, it is advisable to normalize the objective function. This requers a knowladge of minimum and maximum function values of each objective. moreover, this method also requers the ideal solution z^* therefore all M objectives need to be independently optimized before optimizing the l_p metrics.([4],[5])

Benson's method

This procedure is similar to the weighted metric approach, except that the reference solution is taken as a feasible non-pareto optimal solution. A solution z^0 is randomly chosen from the feasible region. Thereafter the non-negative difference $(z_m^0 - f_m(x))$ of each objective is calculated and their sum is maximized:([4],[5])

$$\text{maximize } (\sum_{m=1}^{M} max(0, (z_m^0 - f_m(x))) \tag{2.9}$$

$$\text{subject to : } f_m(x) \leq z_m^0 \qquad m = 1, 2, ...M$$

$$g_j(x) \geq 0 \qquad j = 1, 2, ...J$$
$$h_k(x) = 0 \qquad k = 1, 2, ...K$$
$$x_i^L \leq x_i \leq x_i^u, \qquad i = 1, 2, ..., n;$$

For any solution x, the above objective function has a value equal to half of the perimeter of a hyperbole having z^0 and $f(x)$ as the diagonal points. The only requirement is that the solution x must weakly dominate the solution at z^0. The maximization of the above objective is similar to finding a hypercube with the maximum perimeter. Since the pareto-optimal region lies at the extreme of the feasible search space, the optimum solution of the above optimization problem is a member of the pareto optimal.

Advantages

To avoid scaling problems individual difference can be normalized before the summation. To obtain different pareto optimal solution the difference can be weighted before summation. Thereafter by changing the weight vector, different pareto optimal solution can be obtained. In such a scenario, the use of the nadir point z^{nad}, as chosen point may be found as suitable. If z^0 is chosen appropriately this method can be used to find solutions in the non-convex pareto optimal region.([4],[5])

Disadvantages

The optimization problem formulated above has an additional number of constraints needs to restrict the search in the region dominating the chosen solution z^0. Moreover the objective function is non-differentiable, thereby cousing difficulties for gradiant based methods to solve the above problem.([4],[5])

Value function method

In the value function(or utility function) method the user provide a mathematical value function $U : R^m \to R$ relating all M objectives. The value function must be valid over the entire feasible search space. The task is then to maximize the value function as follow:([4],[5])

$$\text{maximize } U(f(x))) \tag{2.10}$$
$$\text{subject to : } g_j(x) \geq 0 \qquad j = 1, 2, ...J$$
$$h_k(x) = 0 \qquad k = 1, 2, ...K$$
$$x_i^L \leq x_i \leq x_i^u, \qquad i = 1, 2, ..., n;$$

Here, $f(x) = (f_1(x), f_2(x), ..., f_n(x))^T$. As seen from the above problem the value functions provides interactions among different objectives. Among the two solution i and j if $U(f(i)) > U(f(j))$ solution i is then preferred to solution j

Advantages

This idea is simple and ideal, if adequate value function information is available. The value function methods are mainly used in practice to multi-attribute decision analysis problem with a decreta set of feasible solutions, although the principle can also be used in continuous search space.([4],[5])

Disadvantages

As evident from the above discussion the obtained solution entirely depends on the chosen value function. It also requires users to came up with a value function, which is globally applicable over the entire search space. Thus there is a danger of using an over simplified function.([4],[5])

2.1.3 Graphs and Network flows

Notation and Definitions

In this section we give same basic definitions from graph theory and present some basic notation. It also state some elementary properties of graphs and begin by defining directed and undirected graphs.([7])

Directed Graphs and Networks : A directed graph $G = (N, A)$ consists of a set N of nodes and a set A of arcs whose elements are ordered pairs of distinct nodes. A directed network is a directed graph whose nodes and/or arcs have associated numerical values (typically,costs, capacities, and/or supplies and demands).

Undirected Graphs and Networks : We define an undirected graph in the same manner as we define a directed graph except that arcs are unordered pairs of distinct nodes. In an undirected graph, we can refer to an arc joining the node pair i and j as either (i, j) or (j, i). An undirected arc (i, j) can be regarded as a two-way street with flow permitted in both directions: either from node i to node j or from node j to node i. On the other hand, a directed arc (i,j) behaves like a one-way street and permits flow only from node i to node j.

Tails and Heads : A directed arc (i, j) has two endpoints i and j. We refer to node i as the tail of arc (i, j) and node j as its head. We say that the arc (i, j) emanates from node i and terminates at node j. An arc (i,j) is incident to nodes i and j. The arc (i,j) is an outgoing arc of node i and an incoming arc of node j. Whenever an arc $(i, j) \in A$, we say that node j is adjacent to node i.

Degrees : The indegree of a node is the number of incoming arcs of that node and its outdegree is the number of its outgoing arcs. The degree of a node is the sum of its indegree and outdegree. It is easy to see that the sum of indegrees of all nodes equals the sum of outdegrees of all nodes and both are equal to the number of arcs m in the network.

Adjacency List : The arc adjacency list A(i) of a node i is the set of arcs emanating from that node, that is, $A(i) = \{(i, j) \in A : j \in N\}$. The node adjacency list A(i) is the set of nodes adjacent to that node; in this case, $A(i) = \{j \in N : (i, j) \in A\}$. Often, we shall omit the terms "arc" and "node" and simply refer to the adjacency list; in all cases it will be clear from context whether we mean arc adjacency list or node adjacency list. We assume that arcs in the adjacency list A(i) are arranged so that the head nodes of arcs are in increasing order. Notice that $\mid A(i) \mid$ equals the outdegree of node i.

Since the sum of all node outdegrees equals m, we immediately obtain the following property:
Property : $\sum_{i \in N} \mid A(i) \mid = m$.

Multiarcs and Loops: Multiarcs are two or more arcs with the same tail and head nodes. A loop is an arc whose tail node is the same as its head node.

Subgraph : A graph $G' = (N', A')$ is a subgraph of $G = (N, A)$ if $N' \subseteq N$ and $A' \subseteq A$. We say that $G' = (N', A')$ is the subgraph of G induced byN' if A' contains each arc of A with both endpoints in N'. A graph $G' = (N', A')$ is a spanning subgraph of $G = (N, A)$ if $N' = N$ and $A' \subseteq A$.

Walk : A walk in a directed graph $G = (N, A)$ is a subgraph of G consisting of a sequence of nodes and arcs $i_1 - a_l - i_2 - a_2 - ... - i_{r-1} - a_{r-1} - i_r$ satisfying the property that for all $1 \leq k \leq r-1$, either $a_k = (i_k, i_{k+1}) \in A$ or $a_k = (i_{k+1}, i_k) \in A$. Alternatively, we shall sometimes refer to a walk as a set of (sequence of) arcs (or of nodes) without any explicit mention of the nodes (without explicit mention of arcs).

Directed Walk : A directed walk is an "oriented" version of a walk in the sense that for any two consecutive nodes i_k and i_{k+1} on the walk, $(i_k, i_{k+1}) \in A$.

Path : A path is a walk without any repetition of nodes. We can partition the arcs of a path into two groups: forward arcs and backward arcs. An arc (i, j) in the path is a forward arc if the path visits node i prior to visiting node j, and is a backward arc otherwise.

Directed Path : A directed path is a directed walk without any repetition of nodes. In other words, a directed path has no backward arcs. We can store a path (or a directed path) easily within a computer by defining a predecessor index pred(j) for every node j in the path. If i and j are two consecutive nodes on the path (along its orientation), pred(j) = i. (Frequently, we shall use the convention of setting the predecessor index of the initial node of a path equal to zero to indicate the beginning of the path.) Notice that we cannot use predecessor indices to store a walk since a walk may visit a node more than once, and a single predecessor index of a node cannot store the multiple predecessors of any node that a walk visits more than once.

Cycle : A cycle is a path $i_1 - i_2 - ... - i_r$ together with the arc $(i_r i_1)$ or $(i_1 i_r)$ We shall often refer to a cycle using the notation $i_1 - i_2 - ... - i_r - i_l$. Just as we did for paths, we can define forward and backward arcs in a cycle.

Directed Cycle : A directed cycle is a directed path $i_1 - i_2 - ... - i_r$ together with the arc (i_r, i_1).

Acyclic Graph : A graph is a acyclic if it contains no directed cycle.

Connectivity : We will say that two nodes i and j are connected if the graph contains at least one path from node i to node j. A graph is connected if every pair of its nodes is connected; otherwise, the graph is disconnected. We refer to the maximal connected subgraphs of a disconnected network as its components.

Strong Connectivity : A connected graph is strongly connected if it contains at least one directed path from every node to every other node.

Cut : A cut is a partition of the node set N into two parts, S and $\bar{S} = N - S$. Each cut defines a set of arcs consisting of those arcs that have one endpoint in S and another endpoint in $\bar{S}$. Therefore, we refer to this set of arcs as a cut and represent it by the notation $[S, \bar{S}]$.

s-t Cut : An s-t cut is defined with respect to two distinguished nodes s and t, and is a cut $[S, \bar{S}]$ satisfying the property that $s \in S$ and $t \in \bar{S}$.

Tree . A tree is a connected graph that contains no cycle.

Property
(a) A tree on n nodes contains exactly n - 1 arcs.
(b) A tree has at least two leaf nodes (i.e., nodes with degree O).
(c) Every two nodes of a tree are connected by a unique path.

Forest : A graph that contains no cycle is a forest. Alternatively, a forest is a collection of trees.

Subtree : A connected subgraph of a tree is a subtree.

Rooted Tree : A rooted tree is a tree with a specially designated node, called its root; we regard a rooted tree as though it were hanging from its root. Each node i(except the root node) has a unique predecessor, which is the next node on the unique path in the tree from that node to the root; we store the predecessor of node i using a predecessor index pred(i). If j = pred(i), we say that node j is the predecessor of node i and node i is a successor of node j. These predecessor indices uniquely define a rooted tree and also allow us to trace out the unique path from any node back to the root.

DirectedOut Tree : A tree is a directed out-tree routed at node s if the unique path in the tree from node s to every other node is a directed path.Observe that every node in the directed out tree (except node 1) has indegree 1.

Directed-In-Tree : A tree is a directed in-tree routed at node s if the unique path in the tree from any node to node s is a directed path.Observe that every node in the directed in-tree (except node 1) has outdegree 1.

Spanning Tree : A tree T is a spanning tree of G if T is a spanning subgraph of G. Every spanning tree of a connected n-node graph G has (n - 1) arcs. We refer to the arcs belonging to a spanning tree T as tree arcs and arcs not belonging to T as nontree arcs.

Fundamental Cycles : Let T be a spanning tree of the graph G. The addition of any nontree arc to the spanning tree T creates exactly one cycle. We refer to any such cycle as a fundamental cycle of G with respect to the tree T. Since the network contains m - n + 1 nontree arcs, it has m - n + 1 fundamental cycles. Observe that if we delete any arc in a fundamental cycle, we again obtain a spanning tree.

Fundamental Cuts : Let T be a spanning tree of the graph G. The deletion of any tree arc of the spanning tree T produces a disconnected graph containing two subtrees T_l and T_2 Arcs whose endpoints belong to the different subtrees constitute a cut. We refer to any such cut as a fundamental cut of G with respect to the tree T. Since a spanning tree contains n - 1 arcs, the network has n - 1 fundamental cuts with respect to any tree. Observe that when we add any arc in the fundamental cut to the two subtrees T_l and T_2 , we again obtain a spanning tree.

Node-Arc Incidence Matrix

The node-arc incidence matrix representation, or simply the incidence matrix representation, represents a network as the constraint matrix of the minimum cost flow problem . This representation stores the network as an $n \times m$ matrix N which contains one row for each node of the network and one column for each arc. The column corresponding to arc (i , j) has only two nonzero elements: It has a +1 in the row corresponding to node i and a -1 in the row corresponding to node. The node-arc incidence matrix has a very special structure: Only 2m out of its nm entries are nonzero, all of its nonzero entries are $+1$ or -1, and each column has exactly one $+1$ and one -1. Furthermore, the number of $+1$'s in a row equals the outdegree of the corresponding node and the number of -1 's in the row equals the indegree of the node. Because the node-arc incidence matrix N contains so few nonzero coefficients, the incidence matrix representation of a network is not space efficient.

Definition An $m \times n$ matrix is said to be unimodular if for every submatrix of size $m \times m$ the value of its determinant is 1, 0, or -1. A matrix, A, is said to be totally unimodular if the determinant of every square submatrix is 1, 0 or -1. ([7])

Lemma 2.1.6 *The node-arc incidence matrices of network flow problems are totally unimodular. ([7])*

Proof Let A be an $m \times n$ node-arc incidence matrix. The entries of A are -1, 0, or 1. A has exactly two non-zero entries(-1 and 1) per column.
Consider a $k \times k$ submatrix B of A,
if B has a zero column, it determinant is zero. If all columns of B have two non zero entries then $1^T B = 0$, $detB = 0$. Otherwise B has a column, say column j, with One nonzero entries B_{ij}, so
$detB = (-1)^{i+j} B_{ij} detC$.
where C is square of order k-1, obtained by deleting row i and column j of B. Hence, can show by indaction on k that all the determinant of a square submatrix of A are -1, 1 or 0.

Consider the general linear programming problem with integer coefficients in the constraints,right-hand sides and objective function:
(LP)

$$max\ cx$$
$$subject\ to: \qquad Ax \leq b$$

Recall that a basic solution of this problem represents an extreme point on the polytope defined by the constraints (i.e. cannot be expressed as a convex combination of any two other feasible solutions). Also, if there is an optimal solution for LP then there is an optimal, basic solution. If in addition A is totally unimodular we have the following result.

Lemma 2.1.7 *If the constraint matrix A in LP is totally unimodular, then there exists an integer optimal solution. In fact every basic or extreme point solution is integral. ([7])*

Proof If A is $m \times n$, $m < n$, a basic solution is defined by a nonsingular,$m \times m$ square submatrix of A, $B = (b_{ij})$, called the basis, and a corresponding subset of the decision variables, x_B, called the basic variables, such that

$$x_B = B^{-1} b$$

By Cramers rule, we know

$$B^{-1} = \frac{C^T}{det(B)}$$

where $C = (c_{ij})$ is an $m \times m$ matrix, called the cofactor matrix of B. Each c_{ij} is the determinant of a submatrix formed by by deleting row i and column j from B; this determinant is then multiplied by an appropriate sign coefficient (1 or -1) depending on whether $i + j$ is even or odd. That is,

$$C_{ij} = (-1)^{i+j}.det \begin{pmatrix} b_{1,1} & ... & b_{1,j-1} & X & b_{1,j+1} & ... & b_{1,m} \\ ... & ... & ... & ... & ... & ... & ... \\ b_{i-1,1} & ... & b_{i-1,j-1} & X & b_{i-1,j+1} & ... & b_{i-1,m} \\ X & X & X & X & X & X & X \\ b_{i+1,1} & ... & b_{i+1,j-1} & X & b_{i+1,j+1} & ... & b_{i+1,m} \\ ... & ... & ... & ... & ... & ... & ... \\ b_{m,1} & ... & b_{m,j-1} & X & b_{m,j+1} & ... & b_{m,m} \end{pmatrix}$$

Thus, since the determinant is a sum of terms each of which is a product of entries of the submatrix,then all entries in C are integer. In addition, A is totally unimodular and B is a basis (so $|det(B)| = 1$). Therefore, all of the elements of B^{-1} are integers. So we conclude that any basic solution $x_B = B^{-1}b$ is integral.

The set of totally unimodular integer problems includes network flow problems. The rows (or columns) of a network flow constraint matrix contain exactly one 1 and -1, with all the remaining entries 0. A square matrix A is said to have the network property if every column of A has at most one entry of value 1 and 1, and all other entries 0.

Minimum cost flow problem

Minimum cost flow (MCF) problem is a general form of the network flow problem whose aim is to find the least cost of shipment of a commodity through a capacitated network in order to satisfy demands at certain nodes from available supplies at other nodes. Because it represents a general form of network flow, the result from the study of MCF problem can be applied to many other network problems such as: transportation, maximum flow, assignment, shortest path, and transshipment problems. The MCF problem is also very practical, it has been used to solve several real-world application problems such as: multistage production,inventory planning, Mold allocation, nurse scheduling, project assignment, faculty course assignment, and automobile routing .
Given a connected and directed graph $G = (V, A)$ a linear cost function $C : A \mapsto Z_+$, and node balances $b : V \mapsto Z$ such that $\sum_{i \in V} b_i = 0$. A node i is called a supply node when $b_i > 0$, a demand node when $b_i < 0$, and a transshipment node else. The minimum cost flow problem is to find a vector $x^* : A \mapsto Z$ such that x^* is an optimal solution of the linear program

$$min \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{2.11}$$

$$\text{subject to :} \sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = b_i \quad \forall i \in V \tag{2.12}$$

$$l_{ij} \leq x_{ij} \leq u_{ij}, \quad \forall (i,j) \in A \tag{2.13}$$

The equation(2.12) is called flow conservation constraint while the equation(2.13) is called flow bound constraint on x. A flow x is called a feasible flow , if it satisfies (2.12) and (2.13). In matrix notation

$$\min\{c^T x | Ex = b, l \leq x \leq u\} \tag{2.14}$$

where E is the node-arc incidence matrix of the digraph G. Consider $\pi : V \mapsto Q$(the so-called node potentials) and $\delta : A \mapsto Q$ as the dual multipliers for the equations(2.12) and (2.13) respectively. The dual problem of equation (2.14) is

$$\max\{\pi^T b - \delta^T u | \pi^T E - \delta^T \leq c^T, \delta \geq 0\}$$

which is

$$\max \sum_{i \in V} \pi_i b_i - \sum_{(i,j) \in A} \delta_{ij} u_{ij}$$

$$\text{subject to} : \pi_i - \pi_j - \delta_{ij} \leq c_{ij} \quad \forall (i,j) \in A$$
$$\delta_{ij} \geq 0 \quad \forall (i,j) \in A$$

The MCF formulation is particularly broad and can be used as a template upon which a number of network problems may be modelled. The following are some examples.

Shortest Path problem

The shortest path problem is to find the directed paths of shortest length from a given root node to all other nodes. We assume, without loss of generality that the root node is 1, and set the parameters of MCF as follows: $b(1) = n-1$, $b(i) = -1$ for all other nodes, c_{ij} is the length of arc (i, j), $l_{ij} = 0$ and $u_{ij} = n-1$ for all arcs. The LP formulation is:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\text{subject to} : \sum_{j:(1,j)\in A} x_{1j} - \sum_{j:(j,1)\in A} x_{j1} = n-1$$
$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = -1 \quad \text{for} i = 2,3,...,n$$
$$0 \leq x_{ij} \leq n-1, \quad \forall (i,j) \in A$$

The optimum solution to this problem sends unit flow from the root to every other node along a shortest path.

Maximum Flow problem

The maximum flow problem is to send the maximum possible flow from a specified source node (node 1) to a specified sink node (node n). The arc (n, 1) is added with $c_{n1} = -1$, $l_{n1} = 0$ and $u_{n1} = \infty$. The supply at each node is set to zero, i.e. $b(i) = 0$ for all $i \in V$, as is the cost of each arc $(i,j) \in A$. Finally, u_{ij} represents the upper bound on flow in arc (i, j) to give the following LP formulation:

$$\max x_{n1}$$

$$\text{subject to} : \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = 0 \quad \forall i \in V$$
$$0 \leq x_{ij} \leq u_{ij}, \quad \forall (i,j) \in A$$
$$0 \leq x_{n1} \leq \infty, \quad \forall (i,j) \in A$$

The Assignment Problem

The graph $G = (V, A)$ is composed as follows: $V = V_1 \bigcup V_2$ where V_1 and V_2 are two disjoint sets of equal size. The problem is to assign each element in V_1 to one element in V_2 (e.g. people to machines), at minimum cost. If element $i \in V_1$ is assigned to element $j \in V_2$, then $x_{ij} = 1$. Otherwise $x_{ij} = 0$. The cost of assigning $i \in V_1$ to $j \in V_2$ is represented by c_{ij} . For each $i \in V_1$; $b(i) = 1$, and for each $j \in V_2$, $b(j) = -1$. All lower bounds are 0, and all upper bounds are 1. The LP formulation is:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\text{subject to} : \sum_{j:(i,j)\in A} x_{ij} = 1 \quad \forall i \in V_1$$

$$\sum_{i:(i,j)\in A} x_{ij} = -1 \quad \forall i \in V_2$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A$$

The Transportation Problem

As for the Assignment Problem, the Transportation Problem is based on a bi-partite graph, where V_1 is the set of source nodes, and V_2 is the set of sink or destination nodes, such that $V = V_1 \bigcup V_2$, and the set of arcs is defined by $A = \{(i, j) \mid i \in V_1; j \in V_2\}$. The objective is to find the least cost shipping plan from the sources of supply (V_1) to the destinations (V_2), where x_{ij} is the number of units shipped from source i to destination j, and c_{ij} is the cost of shipping one unit from i to j. The supply and demands at sources and destinations respectively are denoted by b(i). There are no upper bounds on flows. (A problem with no upper flow bounds is said to be uncapacitated). In LP form, the problem is stated:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\text{subject to} : \sum_{j:(i,j)\in A} x_{ij} = b(i) \quad \forall i \in V_1$$

$$\sum_{i:(i,j)\in A} x_{ij} = -b(i) \quad \forall i \in V_2$$

$$x_{ij} \geq 0, \quad \forall (i, j) \in A$$

The Transshipment Problem

The Transshipment problem is a generalisation of the above with the addition of intermediate, zero-supply nodes called transshipment nodes. We divide the nodes into three subsets: V_1, a set of supply nodes, V_2, a set of demand nodes and V_3, a set of transshipment nodes. The problem is equivalent to the transportation problem in all other

respects. The objective is to minimise shipping costs and the arcs are uncapacitated. The LP formulation is:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\text{subject to : } \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b_i \quad \forall i \in V_1, V_2$$

$$\sum_{i:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = 0 \quad \forall j \in V_3$$

$$x_{ij} \geq 0, \quad \forall (i,j) \in A$$

The only difference between the transshipment formulation and that of the general MCF is that the latter includes upper and lower bounds on the flows in the arcs.

Solution Methods of Minimum cost flow problem

There are a variety of solution methods for the MCF problem . These may be categorised into a number of principal approaches, namely *primal*, *dual*, *primaldual* ,*scaling alogorithms* and *Network Simplex Method* . For example, negative cycle and primal simplex (as its name suggests) are both primal algorithms, while dual methods include successive shortest paths and the dual simplex method for LP problems. Fords Out-of-Kilter algorithm is perhaps the best known primal-dual method. Finally, there exist a family of relaxation algorithms based on either right-hand-side or cost scaling. We briefly review each of these methods as follows.

Negative Cycle Algorithm

This algorithm is based on the observation that an optimal solution to the MCF problem admits no negative cost augmenting cycle. In outline form the algorithm involves finding an initial feasible flow through the network, and subsequently identifying any negative cost augmenting cycles. Such cycles are eliminated by augmenting flow in all arcs of the cycle by the maximum amount possible. The method used to identify a negative cycle is based on a label-correcting shortest path algorithm. This algorithm also yields a method, based on the negative incremental cost of the incoming cycle at each iteration, for determining an upper bound on the difference in cost between the current and optimal solutions. This gives the option of algorithm termination once a solution is reached which is close to optimal.

Successive Shortest Path (SSP) Algorithm

The SSP algorithm maintains dual feasibility and strives to attain primal feasibility. The flow bound constraints are always satisfied while the supply/demand constraints are initially violated. The algorithm generally proceeds as follows: At each stage, a node i with excess supply, and a node j with unfulfilled demand are selected. A shortest path (i.e. path of minimum per unit cost with excess capacity) from i to j is found, and flow

is sent along this path from i to j. The algorithm terminates when the current solution satisfies all the supply and demand constraints.

Primal-Dual and Out-of-Kilter Algorithms

The primal dual algorithm is similar to the successive shortest path algorithm but uses many paths to augment the flow at each iteration.The MCF is transformed into a single source, single sink problem, generally by the addition of a super-source, s and super-sink, t. At each iteration, the node potentials are updated by finding a tree of shortest paths with the node s as its root. A maximum flow problem is then solved with the object of sending flow from s to t using only arcs with zero reduced cost. Thus the excess supply of some node strictly decreases as does the node potential of the sink. The Out-of-Kilter algorithm maintains a solution satisfying the mass balance constraints but violating the dual feasibility and flow bound conditions. The idea behind the algorithm is to adjust the flow in an arc until it is consistent with the reduced cost of that arc with regard to the dual-derived optimality conditions i.e if $\bar{c}_{ij} > 0$ drive x_{ij} to zero, if $\bar{c}_{ij} > 0$ drive x_{ij} to u_{ij}, else allow any flow $0 < x_{ij} < u_{ij}$. The kilter number of an arc, k_{ij} , is defined as the minimum increase or decrease in the flow necessary so that the arc satisfies its flow bound and dual feasibility constraints. The algorithm iteratively reduces the kilter numbers of the arcs by augmenting flow in cycles, until all arcs are in kilter and the MCF is solved to optimality.

Scaling Algorithms

The basic idea underlying the scaling algorithms is that they commence with a feasible flow for a transformed or altered set of constraints, and iteratively, in successively smaller steps, move towards an optimal solution. Scaling algorithms are generally subdivided into Right-Hand and Cost scaling algorithms. Right-Hand scaling is derived from the successive shortest path algorithm.

Relaxation Algorithms

The approach of these relaxation algorithms is to maintain a pseudo-flow satisfying the flow bound constraints and to move towards satisfaction of the flow balance constraints. This is done through an iterative descent of the dual functional. The dual price vector, π, and the corresponding flow vector, x, which together satisfy complementary slackness, are maintained, and the algorithm iterates using procedures based on flow augmentation and price adjustment.

Network Simplex Method

The Network Simplex Method is an adaption of the bounded variable primal simplex algorithm, specifically for the MCF problem. The basis is represented as a rooted spanning tree of the underlying network, in which variables are represented by arcs, and the simplex multipliers by node potentials. At each iteration, an entering variable is selected

by some pricing strategy, based on the dual multipliers (node potentials), and forms a cycle with the arcs of the tree. The leaving variable is the arc of the cycle with the least augmenting flow. The substitution of entering for leaving arc, and the reconstruction of the tree is called a pivot. When no non-basic arc remains eligible to enter, the optimal solution has been reached.

Chapter 3

Biobjective minimum cost flow problem

3.1 Introduction

In the fist chapter we have introduce concepts and formulation of the biobjective minimum cost flow problem. Now in this chapter we are basically looking to the method that solve the biobjective minimum cost flow problem. To remind the formulation of Biobjective minimum cost flow problem we have:
Given a directed network G=(V,A), let V={1,2,...,n} be the set of nodes and A be the set of arcs. Let suc(i) be the set $\{j \in \mathbf{V} : (i,j) \in \mathbf{A}\}$ and pred(i) be the set $\{j \in \mathbf{V} : (j,i) \in \mathbf{A}\}$. Let an integer b_i be the supply/demand rate of node i. Each arc $(i,j) \in \mathbf{A}$ has associated the following values: u_{ij} the upper bound on flow through arc (i,j); l_{ij} the lower bound on flow through arc(i,j); c_{ij}^k the cost per unit of flow on arc (i,j) for k=1,2. ([8],[9],[10])
The biobjective network flow (BNF) problem can be formalized in the following way:

$$min f_1(x) = \sum_{i \in \mathbf{V}} \sum_{j \in \mathbf{suc(i)}} c_{ij}^1 x_{ij} \tag{3.1}$$

$$min f_2(x) = \sum_{i \in \mathbf{V}} \sum_{j \in \mathbf{suc(i)}} c_{ij}^2 x_{ij} \tag{3.2}$$

$$subjectto : \sum_{j \in \mathbf{suc(i)}} x_{ij} - \sum_{j \in \mathbf{pre(i)}} x_{ij} = b_i \qquad \forall i \in \mathbf{V} \tag{3.3}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in \mathbf{A} \tag{3.4}$$

To solve this problem we will apply different methods.

3.2 Methods to solve Biobjective minimum cost flow problems

In this section, we propose an approach to solve the biobjective minimum cost flow problem. An algorithm to obtain all efficient (integer or real) solutions of this problem is introduced. These method is characterized by the use of the classic resolution tools of network flow problems, such as the network simplex method. Furthermore, the methods calculate dominated and non- dominated solutions.

3.2.1 Biobjective network simplex method

This method starts with the efficient extreme points in the objective space $f(x^0)$ that results when the $P(\lambda^0)$ problem which is defined at equation(1.6) is solved with $\lambda^0 = 1$. Then, it finds the remaining efficient extreme points in the same space by a finite sequence of pivots.([3],[8])
To compute x^0 we only optimize the first objective. In this way, we obtain the strongly feasible spanning tree (B^0, L^0, U^0) and the node potentials π^1 with respect to the first objective . Let $\bar{c}^1_{ij} = c^1_{ij} - \pi^1_i + \pi^1_j$ for all $(i,j) \in A$. Then (B^0, L^0, U^0) is optimum and satisfies the following optimality conditions with respect to the first objective:

$$\begin{gathered} l_{ij} \leq x^0_{ij} \leq u_{ij} \text{ and } \bar{c}^1_{ij} = 0 \ \forall (i,j) \in B^0 \ ; \\ x^0_{ij} = l_{ij} \text{ and } \bar{c}^1_{ij} \geq 0 \ \forall (i,j) \in L^0; \\ x^0_{ij} = u_{ij} \text{ and } \bar{c}^1_{ij} \leq 0 \ \forall (i,j) \in U^0 \end{gathered}$$

Once (B^0, L^0, U^0) and π^1 are obtained, we can obtain the node potentials π^2 with respect to the second objective by solving the system of equations $\bar{c}^2_{ij} = c^2_{ij} - \pi^2_i + \pi^2_j = 0$, $\forall (i,j) \in B^0$: As x^0 is an efficient extreme point of the decision space and $f(x^0)$ is an efficient extreme point of objective space, the next step consists of finding an efficient extreme point of the objective space that is adjacent to $f(x^0)$. Before we give a definition of adjacent efficient extreme points in the objective space, we recall some known theorems and definitions for the Multiobjective Linear Programming problem.

Proposition 3.2.1 *Let $x^0, x^1, ..., x^i$ the set of all efficient extreme points of the BNF problem with $i \geq 2$. Then, by starting with any $x^j (0 \leq j \leq i)$ and moving to adjacent efficient extreme point, the set of all efficient extreme points can be obtained. ([8])*

In this way, finding an efficient extreme point adjacent to another in the decision space is equal to finding the efficient edge which connects them, that is, choosing an adequate non-basic variable to enter the current basis. In absence of degeneracy, this efficient edge can be found by checking the reduced costs $\bar{c}^1_{ij}$ and $\bar{c}^2_{ij}$ associated with the current extreme point for all non-basic variables. If the current solution is degenerate, that is, it has more than one basis associated with it, it is interesting to find the efficient one. A definition of the efficient basis is the following.

Definition Let B be a basis corresponding to an efficient extreme point x^i and let x^j be an adjacent efficient extreme point. If, in relating to B, we can obtain x^j through x^i by one pivot, then B is called efficient basis.([8])

We know that $f(E_{ex}[X]) \subseteq E_{ex}[f(X)]$, furthermore, the contents could be strict. However, the adjacency between efficient extreme points are not preserved through f. Therefore, it is necessary to introduce the concept of adjacent efficient extreme points in the objective space.

Definition Given the adjacent efficient extreme points $x^i, x^{i+1}, ..., x^j$ in the decision space. Then $f(x^i)$ and $f(x^j)$ are adjacent efficient extreme points if in the sequence $f(x^i), f(x^{i+1}, ..., f(x^j)$, the only extreme points of $f(X)$ are $f(x^i)$ and $f(x^j)$.([8])

Proposition 3.2.2 *Let $f(x^i)$ and $f(x^j)$ be two efficient extreme points of the objective space. Then, starting from x^i, we can obtain x^j.([8])*

Proof The following two cases must be considered:
(i) If x^i and x^j are adjacent in the decision space, by the efficient edge $\gamma(x^i, x^j)$.
(ii) If not adjacent, by the sequence of efficient edges $\gamma(x^i, x^{i+1})$, . . . , $\gamma(x^{j-1}, x^j)$.
Given an efficient extreme point of the objective space, the maximum number of adjacent efficient extreme points to this one is two. In absence of degeneracy, given an efficient extreme point in the objective space, an adjacent efficient extreme point can be obtained by introducing into the basis of the first point a set of non-basic variables. When the solution is degenerate in $x^i \in X$, that is, more than one basis corresponding to this point exists, all of these bases correspond to the same point $f(x^i)$. In the process of finding an efficient extreme point of the objective space, we need to calculate the efficient basis among the degenerate bases.

The previous ideas allow us to develop the algorithm that we now comment on in detail. The proposed method uses the tree indices Pred, Depth and Thread to improve the operations of updates in the pivot process. The pivot process is a finite sequence of single-objective Network Simplex method pivots. As we have already indicated, the arcs that do not fulfill the optimality conditions with respect to the second objective are the candidate arcs to be entered into the basis, that is, the (i, j) non-basic arcs such that: $\bar{c}^2_{ij} < 0$ for $(i; j) \in L$ or $\bar{c}^2_{ij} > 0$ for $(i, j) \in U$. The arcs associated with an efficient edge in the objective space are chosen among these arcs. The algorithm and the procedures in the algorithm have the following scheme.

Biobjective Network Simplex (BNS) algorithm ([3],[10])

begin
Solve the $P(\lambda)$ problem with $\lambda = 1$ obtaining x^0, π^1, π^2 and (B, L, U);
Let Pred, Depth and Thread be the spanning tree indices;
Store x^0 as an efficient extreme point in the objective space;
Set t = 1;
Compute-Entering-Arcs $(L, U, c, \pi^1, \pi^2, S, \theta^t)$;
while $S <> \emptyset$ **do**
begin
$x^t := x^{t-1}$;

Compute-New-point $(x^t, B, L, U, \pi^1, \pi^2, Pred, Depth, Thread, S)$;
if $x^t <> x^{t-1}$ **then**
begin
Store x^t as an efficient extreme point in the objective space;
$\lambda^t = \frac{|\theta^t|}{1+|\theta^t|}$, $t := t+1$
end;
Compute - Entering- Arcs $(L, U, c, \pi^1, \pi^2, S, \theta^t)$
end
end
procedure Compute- Entering- Arcs $(L, U, c, \pi^1, \pi^2, S, \theta^t)$;
begin
$\bar{c}^k_{ij} = c^k_{ij} - \pi^k_i + \pi^k_j \ \forall (i,j) \in A$, k = 1, 2;
$S = \emptyset$;
Set $\theta^t := min\{\frac{\bar{c}^2_{ij}}{\bar{c}^1_{ij}} : \bar{c}^2_{ij} < 0 \ \forall (i,j) \in L, \ \frac{\bar{c}^2_{ij}}{\bar{c}^1_{ij}} : \bar{c}^2_{ij} > 0 \ \forall (i,j) \in U\}$;
Let S be the set of arcs where the last minimum is reached
end;
procedure Compute- New- point $(x, B, L, U, \pi^1, \pi^2, Pred, Depth, Thread, S)$;
begin
while $S <> \emptyset$ **do**
begin
Let (i,j) be the first arc of S ;
$S := S - (i,j)$;
if (i,j) does not fulfill the optimality conditions with respect to the second objective **then**
begin
Make a pivot with the entering arc (i,j) ;
Update x, B, L, U, π^1, π^2, $Pred$, $Depth$, $Thread$
end
end
end;
The algorithm starts from the efficient extreme point of the objective space that results when only the first objective is considered. The Compute-Entering-Arcs procedure consists of computing the arcs that do not fulfill the optimality conditions with respect to the second objective. These arcs make up the sequence of pivots to reach the adjacent efficient extreme point of the objective space. The Compute-New-point procedure carries out these pivots updating the spanning tree structure, the tree indices and the potentials of the nodes with respect to the two objectives. If in this process the flow x^{t-1} changes, then we have computed the point x^t. That is, the efficient extreme point $f(x^t)$ is adjacent to $f(x^{t-1})$ in the objective space. This new point is optimal for the $P(\lambda)$ problem with $\lambda^t \leq \lambda \leq \lambda^{t-1}$. At the end of the method, the set of the all efficient extreme points in the objective space is obtained.
We already mentioned that once the efficient extreme points in the objective space are obtained, it is possible to determine all efficient points in the decision space. For that, it is only necessary that the algorithm stores the pair (x, S) corresponding to each efficient

extreme point in the objective. So, in a second phase, each efficient extreme point in the decision space can be generated by making one pivot with each non-basic arc in the set S with respect to the flow x, and so on.

Example Consider the example given in the Fig. below where $b_1 = 10$, $b_6 = -10$ and $b_i = 0$, with i = 2, . . . , 5. This example has eight efficient extreme points in the decision space, from which four correspond to efficient extreme points of the objective space (x^1, x^2, x^3 and x^8). These points can be seen in Table below. For each efficient solution, the basic arcs are distinguished in bold and the efficient solutions that are extreme points of the objective space appear underlined.

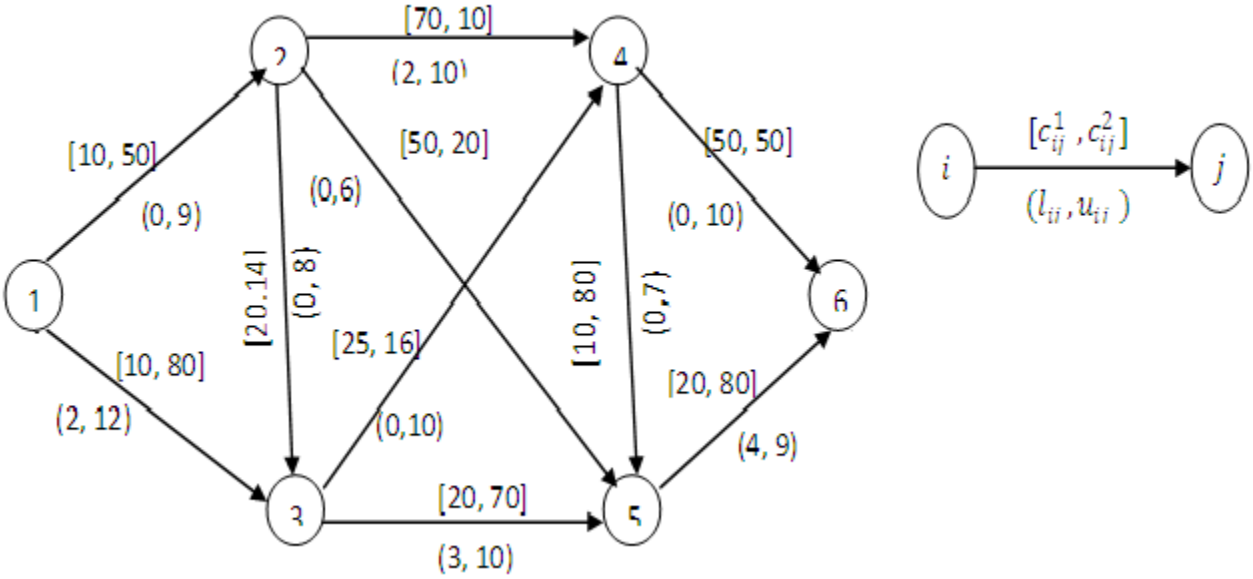


Table: Efficient extreme solutions in the decision space

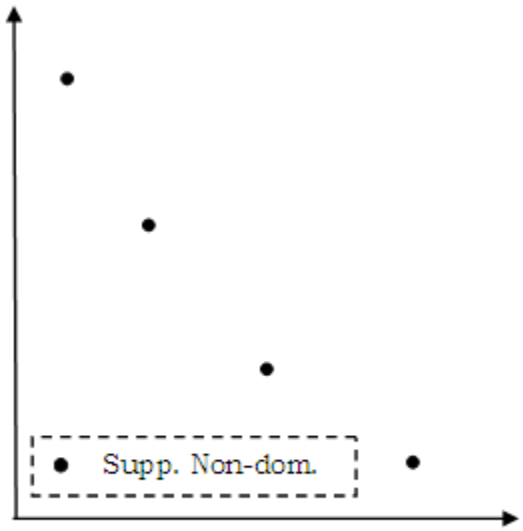
	x_ij										f_1	f_2
	(1,2)	(1,3)	(2,3)	(2,4)	(2,5)	(3,4)	(3,5)	(4,5)	(4,6)	(5,6)		
$\underline{x}^1$	2	8	0	2	0	0	8	1	1	9	640	2170
$\underline{x}^2$	2	8	0	2	0	0	8	0	2	8	660	2060
$\underline{x}^3$	7	3	0	2	5	0	3	0	2	8	810	1660
$\underline{x}^4$	8	2	1	2	5	0	3	0	2	8	830	1644
$\underline{x}^5$	3	7	0	2	1	4	3	0	6	4	830	1644
$\underline{x}^6$	7	3	0	6	1	0	3	0	6	4	1010	1500
$\underline{x}^7$	8	2	5	2	1	4	3	0	6	4	930	1564
$\underline{x}^8$	8	2	1	6	1	0	3	0	6	4	1030	1484

In this example, the optimum solution with $\lambda = 1$ is x^1 (optimal for $0.84615386 \leq \lambda \leq 1$). The lowest value of θ is obtained for the non-basic arc (5,6) with $\theta = -5.5$. Now the entering arc is (5,6), the leaving arc is (4,5) and the extreme point x^2 is obtained. This

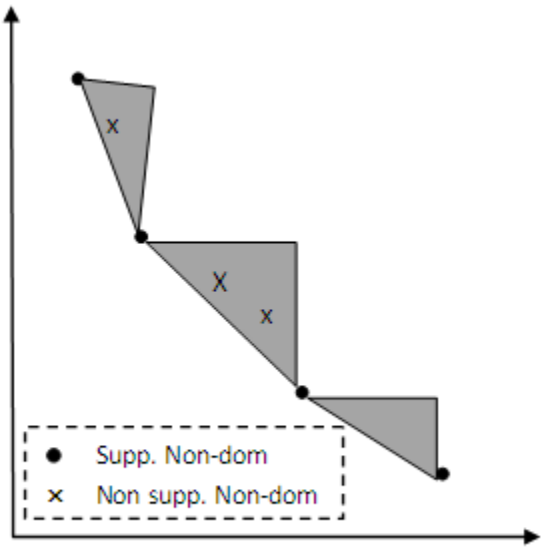
point is optimal for $0.72222222 \leq \lambda \leq 0.8461538$. Now, $\theta = -2.666666$ for the arc (2,5). The entering arc is (2,5), the leaving arc is (3,5) and the extreme point x^3 is obtained. This is optimal for $0.4444444 \leq \lambda \leq 0.72222222$. In x^3, the entering arcs are $\{(2,3),(3,4),(2,4)\}$ with $\theta = -0.8$. The entrance of these into the basis implies the leaving of the arcs $\{(1,3),(5,6),(3,4)\}$. The obtained extreme point is x^8, this point remains optimal for $0 \leq \lambda \leq 0.4444444$. At this stage, all non-basic arcs fulfill the optimality conditions with respect to the second objective.

3.2.2 Two-phase method

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 for illustration see figure A, below. In phase 2 the remaining supported and non-supported efficient solutions are computed with an enumerative approach, as there is no theoretical characterization 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 below B. 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.([1],[9],[11])



A. Supported nondominated points



B. All nondominated points

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 biob-

jective continuous minimum cost flow problem can be used here.We use a parametric simplex method . 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 f_2 and worsening of f_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.([1],[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 B above. Let $f^1, ..., f^s$, where $f^i = (f_1(x^i), f_2(x^i))$ and f^i are sorted by increasing f_1, be the extreme supported points obtained in phase 1. For each pair of neighboring extreme supported points f^i and f^{i+1},we define weighting factors by $\lambda_1 = f_1(x^{i+1}) - f_1(x^i)$ and $\lambda_2 = f_2(x^i) - f_2(x^{i+1})$ Using λ_1 andλ_2 :min $\lambda_1 f_1(x) + \lambda_2 f_2(x)\}$ subject to $x \in X$ we obtain a single-objective flow problem which has optimal solutions f^i, $f^{(i+1)}$ (of course all supported solutions between f^i and f^{i+1},are optimal as well). Applying a k best flow algorithm to problem min $\lambda_1 f_1(x) + \lambda_2 f_2(x)$ subject to $x \in X$ }, 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.([1],[2]) Before we continue with the algorithm for phase 2, we explain the k best flow algorithm.

The k-Best Flow Algorithm ([1],[2])

We give a summary of the k best flow algorithm here. 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. 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 BMCF 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

$$(i,j)_a \in A_x^+ \text{with cost} \hat{c}_{(i,j)_a} = c_a \text{if} a = (i,j) \in A \text{and} x_a < u_a \tag{3.5}$$

$$(j,i)_a \in A_x^- \text{with cost} \hat{c}_{(i,j)_a} = -c_a \text{if} a = (i,j) \in A and x_a > l_a = 0, \tag{3.6}$$

$$A_x = A_x^+ \bigcup A_x^-$$

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 proper Minimal Cycle in Algorithm 1) in the incremental graph G_x can be obtained by

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

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

$$u_a^{'} = \begin{cases} x_a & if a = \bar{a} \\ u_a & otherwise \end{cases} \tag{3.7}$$

$$l_a^{'} = \begin{cases} x_a + 1 & if a = \bar{a} \\ l_a & otherwise \end{cases} \tag{3.8}$$

Algorithm 1 K best flows ([1],[2])

1. **input**: Network $G = (N, A)$ with cost c, lower bounds $l = (0, ..., 0)$, upper bounds u, optimal solution x, max number of flows K
2. 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 */
3. Find proper Minimal Cycle C in G_x derived from x, c, l, u
4. Partitions = {(x,l,u,C)}
5. k = 2
6. **while** Partitions is nonempty and $k < K$ do
7. (x_p, l_p, u_p, C_p) = Partitions[1] /* element with least cost second best flow*/
8. derive Partitions $x_p, l_p, u_p^{'}$ and $\bar{x}_p, l_p^{'}, u_p$
9. Find proper Minimal Cycle C in $G_{(x_p)}$ (incremental graph for $x_p, c, l_p, u_p^{'}$)
10. Find proper Minimal Cycle $\bar{C}$ in $G_{(\bar{x}_p)}$ (incremental graph for $\bar{x}_p, c, l_p^{'}, u_p$)
11. Insert $(x_p, l_p, u_p^{'}, C)$ and $(\bar{x}_p, l_p^{'}, u_p, \bar{C})$ into Partitions /* so that the order of Partitions is maintained */
12. save k^{th} best flow $\bar{x}_p$
13. k = k+1
14. **end while**
15. **output**: 2^{nd} 3^{rd}, ..., k^{th} best flow and $k < K$

In Algorithm 1 we call this operation derive Partitions.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.

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 f^i and f^{i+1} have been found. We call $f_{lN}^i = (f_1(x^{i+1}), f_2(x^i))$ 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 f_{lN}^i 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 $\lambda_1 f_1(x) + \lambda_2 f_2(x) \leq u_\lambda$ with $u_\lambda = \lambda_1(f_1(x^{i+1}) - 1) + \lambda_2(f_2(x^i) - 1)$ 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, The phase 2 algorithm is described in Algorithm 2. ([1],[2])

Algorithm 2 Phase 2 BMCF

1. **input**: Network (N,A) with cost $f = (c^1, c^2)$, lower bound $l = (0, ..., 0)$, upper bound u, list of extreme supported solutions f^1,...,f^s,
2. i= 1
3. **while** $i < s$ **do**
4. Compute $\lambda_1 = f_1(x^{i+1}) - f_1(x^i)$, $\lambda_2 = f_2(x^i) - f_2(x^{i+1})$, the upper bound $u_\lambda = \lambda_1(f_1(x^{i+1}) - 1) + \lambda_2(f_2(x^i) - 1)$, and $c = \lambda_1 c^1 + \lambda_2 c^2$.
5. Find proper Minimal Cycle C in $G_(x^{i+1})$ derived from x^{i+1}, c, l, u
6. Partitions = $\{(x, l, u, C)\}$
7. **while**(Partitions is nonempty) and (cost of second best flow in Partitions[1]: $c(x_p)$+ $c(C_p) \leq u_\lambda$) **do**
8. Steps 7-11 in Algorithm 1 /* Execute one iteration of k best flow */
9. **if** $f(\bar{x}_p)$ in current triangle and not dominated by any point in the triangle found so far **then**
10. Insert $\bar{x}_p$ into list of efficient solutions, and eliminate other Solutions that are now dominated.
11. Update u_λ if possible.
12. **end if**
13. **end while**
14. **i= i+ 1**
15. **end while**
16. **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 phase2. We remarked at the end of this section that we could consider intermediate solutions whenever the flow between two adjacent solutions obtained in phase1 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 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 solutions including the intermediate solutions), and after that the upper bound will be as good as it would be in the smaller triangles.([1],[2])

3.2.3 Weighted metric methods([8])

To obtain efficient solution the following single objective problem can be used:

$$mind_\pi(f^*, f(x)) \tag{3.9}$$

$$\text{subject to} \qquad x \in X$$

Where X is the decision space of BNF problem and d_π is an appropriate distance defined by a parameter given as π. Lets call the above problem as (P_π) problem. which is defined by the weighted maximum distance (Tchebychev metrics);
$d_\pi(f^*, f(x)) = max_{(k=1,2)}\{\pi_k(f_k(x) - f^*_k)\}$
Where the weight π are built in the following way: For each objective the decision-maker(DM) introduces an aspiration level , $z_k > f^*_k$, k= 1,2, and then

$$\pi_k = \frac{\frac{1}{z_k - f^*_k}}{\sum_{i=1,2} \frac{1}{z_i - f^*_i}}$$

The problem in equation (3.9) with metric d_π and weights π , has a solution in the closest efficient point, in accordance with the search direction given by the ideal points and the aspiration level vector z .

Proposition 3.2.3

- *The solution of (P_π) is an efficient point of BNF problems on the intersection of $E[f(X)]$ with the line defined by f^* and the aspiration level vector z.*
- *There exist a one-to-one correspondence between efficient solution of BNF problem and weight π generated using the previous form. ([8])*

Solving the (P_π) problem

In our case the (P_π) problem can be expressed in the following way:

$$\bar{y} = min \;\; y \tag{3.10}$$

$$subject \;\; to : \pi_k(\sum_{i \in \mathbf{V}} \sum_{j \in \mathbf{suc(i)}} c_{ij}^k x_{ij} - f_k^*) \leq y, \qquad k = 1, 2 \tag{3.11}$$

$$\sum_{j \in \mathbf{suc(i)}} x_{ij} - \sum_{j \in \mathbf{pre(i)}} x_{ij} = b_i \qquad \forall i \in \mathbf{V} \tag{3.12}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in \mathbf{A} \tag{3.13}$$

The constraints (3.12) and (3.13) are the classic constraints of the minimum cost flow problem in network. The restriction in constraint (3.11) is related to the aspiration level for each objectives, incorporated into the problem by the weight . The objective function (3.10) involves minimizing the great deviation of the values reached by the different objective and the corresponding ideal points. The restriction (3.11) breaks the matrixs unimodularity property of the minimum cost flow problem. However, this difficulty can be over came if constraint (3.11) is incorporated in to the objective function by the lagrangian relaxation ([8])

$$min \;\; (y + \sum_{k=1}^{2} w_k(\pi_k(\sum_{i \in \mathbf{V}} \sum_{j \in \mathbf{suc(i)}} c_{ij}^k x_{ij} - f_k^*) - y) \tag{3.14}$$

$$subject \; to : \sum_{j \in \mathbf{suc(i)}} x_{ij} - \sum_{j \in \mathbf{pre(i)}} x_{ij} = b_i \qquad \forall i \in \mathbf{V} \tag{3.15}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in \mathbf{A} \tag{3.16}$$

$$w_k \geq 0 \qquad k = 1, 2 \tag{3.17}$$

Where w is the lagrangian multiplier vector. By reordering the terms of the objective functions the following objective function is obtained:

$$y(1 - \sum_{k=1}^{2} w_k) - \sum_{k=1}^{2} w_k \pi_k f_k^* + \sum_{k=1}^{2} w_k \pi_k \sum_{i \in \mathbf{V}} \sum_{j \in \mathbf{suc(i)}} c_{ij}^k x_{ij}$$

It may be noted that the first two terms of the above expression do not depend on x. In this case, once w is fixed, the single objective minimum cost flow problem to solve is:

$$L_1(w) = min\ (\sum_{i\in\mathbf{V}} \sum_{j\in\mathbf{suc(i)}} \sum_{k=1}^{2} w_k \pi_k c_{ij}^k x_{ij}) \tag{3.18}$$

$$subject\ to: \sum_{j\in\mathbf{suc(i)}} x_{ij} - \sum_{j\in\mathbf{pre(i)}} x_{ij} = b_i \qquad \forall i \in \mathbf{V} \tag{3.19}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in \mathbf{A} \tag{3.20}$$

$$w_k \geq 0 \qquad k = 1,2 \tag{3.21}$$

We define:

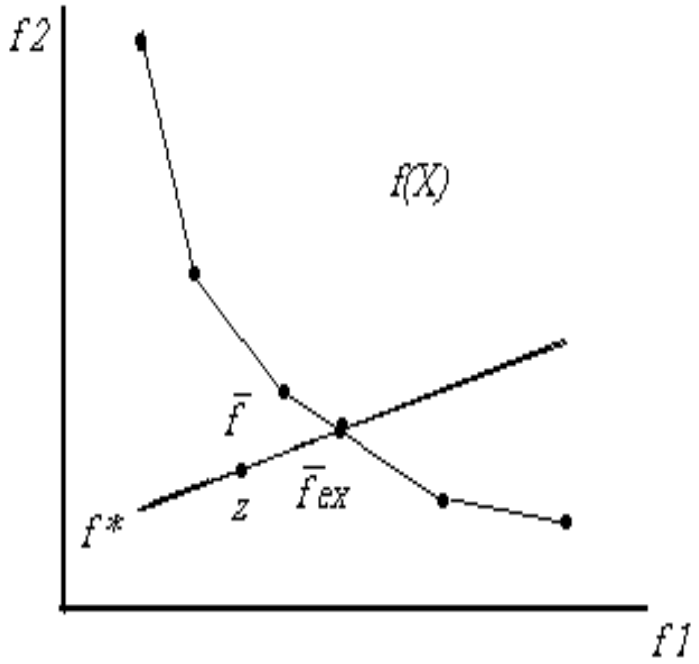
$$L(w) = L_1(w) - \sum_{k=1}^{2} w_k \pi_k f_k^* + y(1 - \sum_{k=1}^{2} w_k) \tag{3.22}$$

Now, we solve the lagrangian multiplier (LM) problem: $\bar{L} = max_{w\geq 0} L(w)$. If $\bar{y}$ is the optimal solution of (P_π),for any choice of the lagrangian multiplier vector w, and any feasible solution x of (P_π),then $L(w) \leq \bar{L} \leq \bar{y} \leq y$. Furthermore, if w is a vector of lagrangian multipliers and x is a feasible solution for the problem (P_π) satisfying $L(w) = y$, then w is an optimum solution for LM problem and x is an optimum solution for (P_π).

Proposition 3.2.4 *The optimum value of problem LM is equal to the optimum value of* (P_π)*, that is* $\bar{L} = \bar{y}$*. ([8])*

Proof Since the solution of (P_π) is the intersection point between the set of efficient points E[f(X)] with the line defined by f^* and the aspiration level vector z (proposition 1), then $y = \pi_k(\sum_{i\in\mathbf{V}} \sum_{j\in\mathbf{suc(i)}} c_{ij}^k x_{ij} - f_k^*) \qquad \forall k = 1,2$ Therefor, $\bar{y} = min\ (y + \sum_{k=1}^{2} w_k(\pi_k(\sum_{i\in\mathbf{V}} \sum_{j\in\mathbf{suc(i)}} c_{ij}^k x_{ij} - f_k^*) - y)$ and $\bar{L} = \bar{y}$

Furthermore, the resolution of equation(3.10) can be carried out by adequate solving of equation(3.18) and LM problems. We gave initial value the multiplier vector w (for example equal to 1) and we modify in each iteration the value of w, according to the scheme of the subgradiant optimization technique, until $\bar{L}$ is calculated. In this process, we use the network simplex method to solve problem (3.18). In the end we will obtain the efficient extreme point closest to the efficient solution of (P_π). Let $\bar{f} \in f(X)$ be an efficient solution of (P_π) and $\bar{f}_{ex} \in f(X)$ be the efficient extreme point closest to $\bar{f}$ according to the weighted maximum distance d_π(see figure below)



One problem that can arise is that the network simplex method, when applied to solve equation (3.18), is not able to calculate $\bar{f}$ if this point is not an extreme point of $f(X)$. In order to avoid this situation upper bound of $\bar{y}$ are obtained. For this reason, it is necessary to identify same extreme points of the $f(X)$ polyhedron. that is to determine the lower frontier of the closest extreme points to the solution efficient point.
Let H_p be a line segment defined by two extreme points (not necessarily adjacent) in the objective space. Then, we define $\bar{f}$ as the intersection point between the line $t = f^* + \alpha(z - f^*)$ and H_p. We can assign to y the value of d_π that corresponds to $\bar{f} = f^* + \alpha(z - f^*)$. In this sense, at the end of our approach $\bar{f}$ will be equal to $\bar{f}$ since H_p coincide with the line segment defined by the two adjacent efficient extreme points, which are closest to the solution point. ([8])
In the biobjective case, the efficient solution $\bar{f}$ is always obtained solving (P_π). This is because at the end of resolution of (P_π) Segment, H_p coincides with an efficient edge of polyhedron $f(X)$. By proposition 1 we have that,$\bar{y} = \pi_1(\bar{f}_1 - f_1^*) = \pi_2(\bar{f}_2 - f_2^*)$. Consider this result in $L(w)$,we obtain that , $L(w) = L_1(w) - \sum_1^2 w_k\pi_k f_k^* + \bar{y}(1 - \sum_{k=1}^2 w_k)$, where the two last terms are known. The subgradiant optimization technique allows us to obtain $\bar{f}$ and therefore $\bar{y}$. we store in a list the two closest extreme points in non-decreasing order of the values of $d_\pi(f^*, f(x)) = max_{k=1,2}\{\pi_k(f_k(x) - f_k^*)\}$. ([8])
At the beginning of the procedure, the segment H_p is defined by the extreme points associated with the ideal level of each objectives,therefore H_p is a list that contain these extreme points. Let $\bar{f}_{ex}$ be the current best efficient extreme point. At the end of the algorithm this point will be the solution point. Let $\bar{f}_{ex}$ be the new extreme point that is calculated in each iteration, the algorithm starts with $w = 1$ and $\lambda = 2$. In each iteration $L_1(w)$ is solved and an extreme point $\bar{f}_{ex}$ is obtained, if the value of

$d_\pi(f^*, f(x)) = max_{k=1,2}\{\pi_k(\bar{f}_{ex} - f_k^*)\}$ for $\bar{f}_{ex}$ is less than the value of d_π for any other extreme point in H_p, then $\bar{f}_{ex}$ is added to H_p and the extreme point in the segment with the greatest value of d_π is eliminated from H_p. the new segment must have non-null intersection with the line defined by f^* and z . This will give a new point point $\bar{f}$ and a new upper bound $\bar{y}$ which improves the previous value. This value is used to calculate the step size and the values of the multipliers of the following iteration. If the value of $L(w)$ has not increased with respect to the value obtained in the r previous iteration, (in our performance $r = 3$), the scale factor λ become equal to $\frac{\lambda}{2}$. The step size θ is calculated according to Newtons method for solving systems of non-linear equations. At the optimum,$\bar{y}$ must coincide with $\bar{L}$, we can use the stop rule:$\hat{y} = L(w)$. ([8])

Algorithm SP_π(solving P_π) ([8])

H_p is a list containing the extreme points associated to the ideal levels of each objective(line segment);
$\bar{f}$ is the point obtained from the intersection between H_p and the search direction;
$\bar{y} = \pi_k(\bar{f}_k - f_k^*)$ k=1,2
$\bar{f}_{ex}$:= first point of listH_p
$q := 1$; {iteration}; $\lambda := 2$;{scale factor to update the step size};
$w_k^q := 1$; k = 1,2 { initialization of the lagrangian multiplier }
While ($\bar{y} <> L(w_k^q)$) **do**
begin
solve $L_1(w_k^q)$ using the network simplex method and obtain $\bar{f}_{exnt}$;
If $\bar{f}_{ex}$ is closer than other extreme point in H_p **then**
begin
Interchange $\bar{f}_{ex}$ with the extreme point of H_p in such a way that the intersection with the search direction is nun-null;
$\bar{f}$ is the point obtained from the intersection between H_p and the search direction;
$\bar{y} = \pi_k(\bar{f}_k - f_k^*), k = 1, 2$
$\bar{f}_{ex}$:= first point of list H_p
$L(w_k^q) = L_1(w) - \sum_1^2 w_k \pi_k f_k^* + \bar{y}(1 - \sum_{k=1}^2 w_k)$
end;
if $L(w)$ has not improved after r iteration **then** λ:= $\frac{\lambda}{2}$;
$\theta := \lambda(\frac{\bar{y} - L(w)}{\|\pi_k(\bar{f}_{ex(k)} - \bar{f}_k)\|^2})$ k=1,2;{step size}

$$w_k^q := \begin{cases} w_k^q + \theta(\pi_k(\bar{f}_{ex(k)} - \bar{f}_k)) & if \quad w_k^q + \theta(\pi_k(\bar{f}_{ex(k)} - \bar{f}_k)) \geq 0 \ \ k = 1, 2 \ \ \{wnext\}; \\ w_k^q & otherwise \end{cases}$$

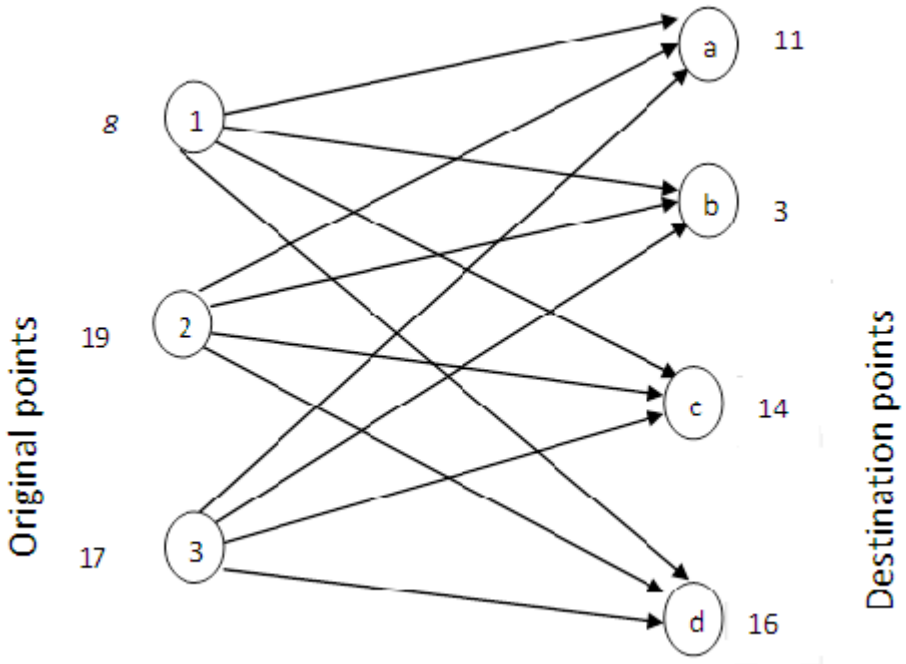
q:= q+1;
End

Example This example presented by Aneja and Nair (1979) and used by Ringuest and Rinks (1987). It is a biobjective transportation problem that can be converted easily into a biobjective minimum cost flow problem. The data of the problem is as follow:

Destination points

Original points	A	B	C	d	a_i
1	(1,4)	(2,4)	(7,3)	(7,4)	8
2	(1,5)	(9,8)	(3,9)	(4,10)	19
3	(8,6)	(9,2)	(4,5)	(6,1)	17
b_i	11	3	14	16	

Where the value that appears in each cell of the table is (c_{ij}^1, c_{ij}^2). Graphically this problem is represented as follow;



Solution

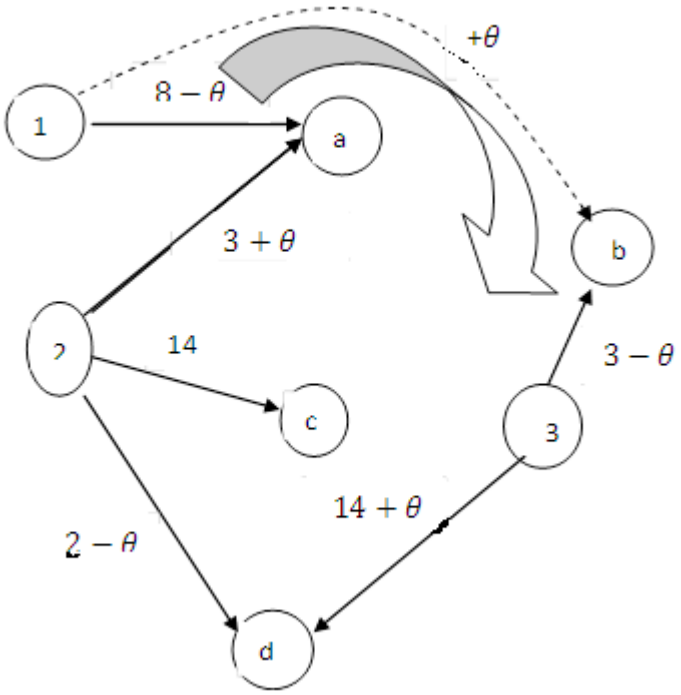
In order to solve this problem by the above algorithm fist we have to determine the extreme. First let as solve the extreme point of each objective function side by side. This will be done as follow:

In order to get the first initial feasible tree lets apply the least cost method

For the fist objective function

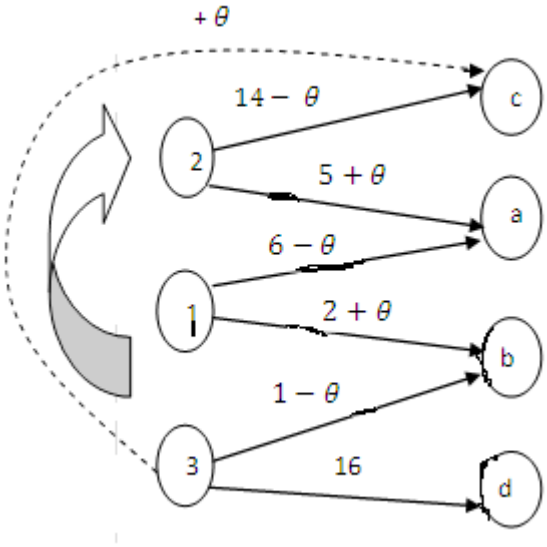
	a	b	C	D	a_i
1	1 [8]	2	7	7	8
2	1 [3]	9	3 [14]	4 [2]	19
3	8	9 [3]	4	6 [14]	17
b_i	11	3	14	16	

Therefore the first feasible tree is given by:

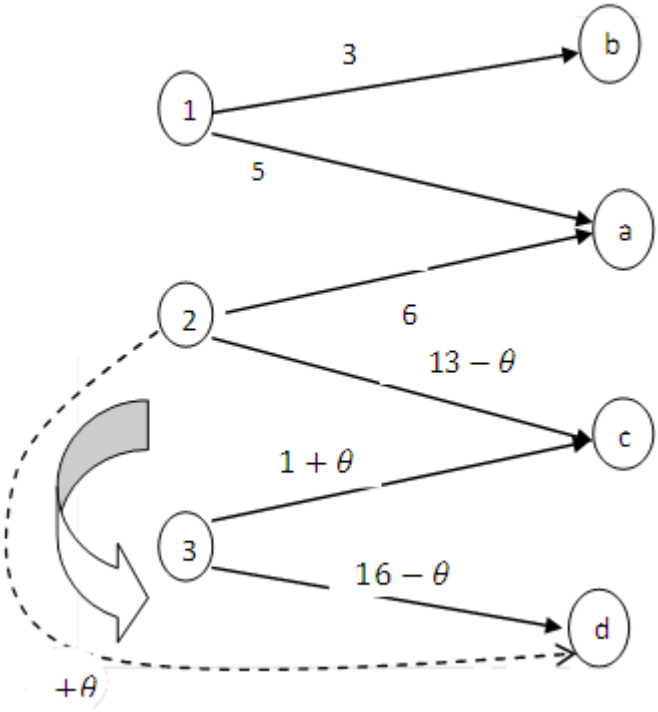


we can get the optimal value of the first objective function as follow:
1^{s}t step: find the potential node for basic variable:
$\pi_1^1 - \pi_a^1 = c_{1a}^1 \Rightarrow \pi_1^1 - \pi_a^1 = 1 \Rightarrow$ if $\pi_1^1 = 0$ then $\pi_a^1 = -1$
$\pi_2^1 - \pi_a^1 = c_{2a}^1 \Rightarrow \pi_2^1 - \pi_a^1 = 1 \Rightarrow$ since $\pi_a^1 = -1$ then $\pi_2^1 = 0$
$\pi_2^1 - \pi_c^1 = c_{2c}^1 \Rightarrow \pi_2^1 - \pi_c^1 = 3 \Rightarrow$ since $\pi_2^1 = 0$ then $\pi_c^1 = -3$
$\pi_2^1 - \pi_d^1 = c_{2d}^1 \Rightarrow \pi_2^1 - \pi_d^1 = 4 \Rightarrow$ since $\pi_2^1 = 0$ then $\pi_d^1 = -4$
$\pi_3^1 - \pi_d^1 = c_{3d}^1 \Rightarrow \pi_3^1 - \pi_d^1 = 6 \Rightarrow$ since $\pi_d^1 = -4$ then $\pi_3^1 = 2$
$\pi_3^1 - \pi_b^1 = c_{3b}^1 \Rightarrow \pi_3^1 - \pi_b^1 = 9 \Rightarrow$ since $\pi_3^1 = 2$ then $\pi_b^1 = -7$
The reduced cost for non basic arcs is given by: $\bar{c}_{ij}^1 = c_{ij}^1 - \pi_i^1 + \pi_j^1$ therefore:
$\bar{c}_{1b}^1 = c_{1b}^1 - \pi_1^1 + \pi_b^1 \Rightarrow \bar{c}_{1b}^1 = 2 - 0 - 7 \Rightarrow \bar{c}_{1b}^1 = -7$

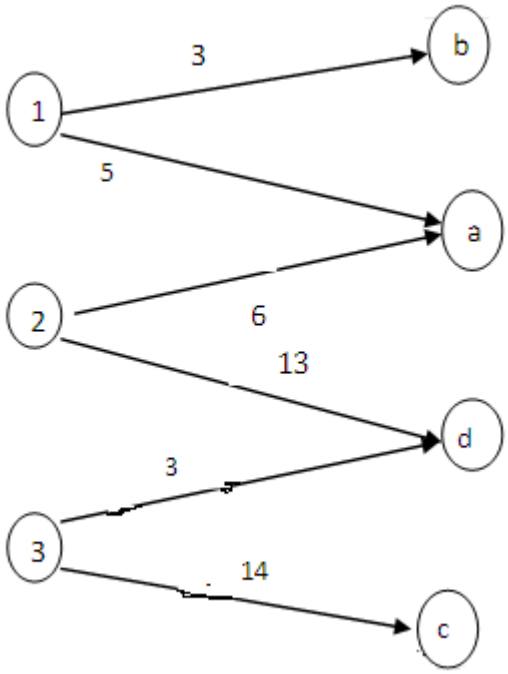
$\bar{c}^1_{1c} = c^1_{1c} - \pi^1_1 + \pi^1_c \Rightarrow \bar{c}^1_{1c} = 7 - 0 - 3 \Rightarrow \bar{c}^1_{1c} = 4$
$\bar{c}^1_{1d} = c^1_{1d} - \pi^1_1 + \pi^1_d \Rightarrow \bar{c}^1_{1d} = 7 - 0 - 4 \Rightarrow \bar{c}^1_{1d} = 3$
$\bar{c}^1_{2b} = c^1_{2b} - \pi^1_2 + \pi^1_b \Rightarrow \bar{c}^1_{2b} = 9 - 0 - 7 \Rightarrow \bar{c}^1_{2b} = 2$
$\bar{c}^1_{3a} = c^1_{3a} - \pi^1_3 + \pi^1_a \Rightarrow \bar{c}^1_{3a} = 8 - 2 - 1 \Rightarrow \bar{c}^1_{3a} = 5$
$\bar{c}^1_{3c} = c^1_{3c} - \pi^1_3 + \pi^1_c \Rightarrow \bar{c}^1_{3c}) = 4 - 2 - 3 \Rightarrow \bar{c}^1_{3c} = -1$
Arc (1,b) and arc(3,c) violet the optimality criteria the above flow is not optimal. The most negative reduced cost is $\bar{c}^1_{1b} = -7$, therefore arc (1,b) is inserted in the tree above, then the tree form a cycle by augmentation we will get $\theta = 2$. Therefore the arc (2,d) have zero flow and leaves the tree , then we will get



Again find the potential node of the above tree, it will be :
$\pi^1_1 - \pi^1_a = c^1_{1a} \Rightarrow \pi^1_1 - \pi^1_a = 1 \Rightarrow$ if $\pi^1_1 = 0$ then $\pi^1_a = -1$
$\pi^1_1 - \pi^1_b = c^1_{1b} \Rightarrow \pi^1_1 - \pi^1_b = 2 \Rightarrow$ since $\pi^1_1 = 0$ then $\pi^1_b = -2$
$\pi^1_2 - \pi^1_a = c^1_{2a} \Rightarrow \pi^1_2 - \pi^1_a = 1 \Rightarrow$ since $\pi^1_a = -1$ then $\pi^1_2 = 0$
$\pi^1_2 - \pi^1_c = c^1_{2c} \Rightarrow \pi^1_2 - \pi^1_c = 3 \Rightarrow$ since $\pi^1_2 = 0$ then $\pi^1_c = -3$
$\pi^1_3 - pi^1_b = c^1_{3b} \Rightarrow \pi^1_3 - \pi^1_b = 9 \Rightarrow$ since $\pi^1_b = -2$ then $\pi^1_3 = 7$
$\pi^1_3 - \pi^1_d = c^1_{3d} \Rightarrow \pi^1_3 - \pi^1_d = 6 \Rightarrow$ since $\pi^1_3 = 7$ then $\pi^1_d = 1$
The reduced cost for non basic arcs be came
$\bar{c}^1_{1c} = c^1_{1c} - \pi^1_1 + \pi^1_c \Rightarrow \bar{c}^1_{1c} = 7 - 0 - 3 \Rightarrow \bar{c}^1_{1c} = 4$
$\bar{c}^1_{1d} = c^1_{1d} - \pi^1_1 + \pi^1_d \Rightarrow \bar{c}^1_{1d} = 7 - 0 + 1 \Rightarrow \bar{c}^1_{1d} = 8$
$\bar{c}^1_{2b} = c^1_{2b} - \pi^1_2 + \pi^1_b \Rightarrow \bar{c}^1_{2b} = 9 - 0 - 2 \Rightarrow \bar{c}^1_{2b} = 7$
$\bar{c}^1_{2d} = c^1_{2d} - \pi^1_2 + \pi^1_d \Rightarrow \bar{c}^1_{2d} = 4 - 0 + 1 \Rightarrow \bar{c}^1_{2d} = 5$
$\bar{c}^1_{3a} = c^1_{3a} - \pi^1_3 + \pi^1_a \Rightarrow \bar{c}^1_{3a} = 8 - 7 - 1 \Rightarrow \bar{c}^1_{3a} = 0$
$\bar{c}^1_{3c} = c^1_{3c} - \pi^1_3 + \pi^1_c \Rightarrow \bar{c}^1_{3c}) = 4 - 7 - 3 \Rightarrow \bar{c}^1_{3c} = -6$
Since arc (3,c) violet the optimality criteria the above flow is not optimal. therefore arc (3,c) is inserted in the tree above, then the tree form a cycle by augmentation we will get $\theta = 1$. Therefore the arc (3,b) have zero flow and leaves the tree , then we get another feasible tree which is shown below



Again find the potential node of the above tree, it will be :
$\pi_1^1 - \pi_a^1 = c_{1a}^1 \Rightarrow \pi_1^1 - \pi_a^1 = 1 \Rightarrow$ if $\pi_1^1 = 0$ then $\pi_a^1 = -1$
$\pi_1^1 - \pi_b^1 = c_{1b}^1 \Rightarrow \pi_1^1 - \pi_b^1 = 2 \Rightarrow$ since $\pi_1^1 = 0$ then $\pi_b^1 = -2$
$\pi_2^1 - \pi_a^1 = c_{2a}^1 \Rightarrow \pi_2^1 - \pi_a^1 = 1 \Rightarrow$ since $\pi_a^1 = -1$ then $\pi_2^1 = 0$
$\pi_2^1 - \pi_c^1 = c_{2c}^1 \Rightarrow \pi_2^1 - \pi_c^1 = 3 \Rightarrow$ since $\pi_2^1 = 0$ then $\pi_c^1 = -3$
$\pi_3^1 - pi_c^1 = c_{3c}^1 \Rightarrow \pi_3^1 - \pi_c^1 = 4 \Rightarrow$ since $\pi_c^1 = -3$ then $\pi_3^1 = 1$
$\pi_3^1 - \pi_d^1 = c_{3d}^1 \Rightarrow \pi_3^1 - \pi_d^1 = 6 \Rightarrow$ since $\pi_3^1 = 1$ then $\pi_d^1 = -5$
The reduced cost for non basic arcs be came
$\bar{c}_{1c}^1 = c_{1c}^1 - \pi_1^1 + \pi_c^1 \Rightarrow \bar{c}_{1c}^1 = 7 - 0 - 3 \Rightarrow \bar{c}_{1c}^1 = 4$
$\bar{c}_{1d}^1 = c_{1d}^1 - \pi_1^1 + \pi_d^1 \Rightarrow \bar{c}_{1d}^1 = 7 - 0 - 5 \Rightarrow \bar{c}_{1d}^1 = 2$
$\bar{c}_{2b}^1 = c_{2b}^1 - \pi_2^1 + \pi_b^1 \Rightarrow \bar{c}_{2b}^1 = 9 - 0 - 2 \Rightarrow \bar{c}_{2b}^1 = 7$
$\bar{c}_{2d}^1 = c_{2d}^1 - \pi_2^1 + \pi_d^1 \Rightarrow \bar{c}_{2d}^1 = 4 - 0 - 5 \Rightarrow \bar{c}_{2d}^1 = -1$
$\bar{c}_{3a}^1 = c_{3a}^1 - \pi_3^1 + \pi_a^1 \Rightarrow \bar{c}_{3a}^1 = 8 - 1 - 1 \Rightarrow \bar{c}_{3a}^1 = 6$
$\bar{c}_{3b}^1 = c_{3b}^1 - \pi_3^1 + \pi_b^1 \Rightarrow \bar{c}_{3b}^1) = 9 - 1 - 2 \Rightarrow \bar{c}_{3b}^1 = -6$
Since arc (2,d) violet the optimality criteria the above flow is not optimal. therefore arc (2,d) is inserted in the tree above, then the tree form a cycle by augmentation we will get $\theta = 13$. Therefore the arc (2,c) have zero flow and leaves the tree , then we get another feasible tree which is shown below:



Again find the potential node of the above tree, it will be :
$\pi_1^1 - \pi_a^1 = c_{1a}^1 \Rightarrow \pi_1^1 - \pi_a^1 = 1 \Rightarrow$ if $\pi_1^1 = 0$ then $\pi_a^1 = -1$
$\pi_1^1 - \pi_b^1 = c_{1b}^1 \Rightarrow \pi_1^1 - \pi_b^1 = 2 \Rightarrow$ since $\pi_1^1 = 0$ then $\pi_b^1 = -2$
$\pi_2^1 - \pi_a^1 = c_{2a}^1 \Rightarrow \pi_2^1 - \pi_a^1 = 1 \Rightarrow$ since $\pi_a^1 = -1$ then $\pi_2^1 = 0$
$\pi_2^1 - \pi_d^1 = c_{2d}^1 \Rightarrow \pi_2^1 - \pi_d^1 = 4 \Rightarrow$ since $\pi_2^1 = 0$ then $\pi_d^1 = -4$
$\pi_3^1 - pi_d^1 = c_{3d}^1 \Rightarrow \pi_3^1 - \pi_d^1 = 6 \Rightarrow$ since $\pi_d^1 = -4$ then $\pi_3^1 = 2$
$\pi_3^1 - \pi_c^1 = c_{3c}^1 \Rightarrow \pi_3^1 - \pi_c^1 = 4 \Rightarrow$ since $\pi_3^1 = 2$ then $\pi_c^1 = -2$
The reduced cost for non basic arcs be came
$\bar{c}_{1c}^1 = c_{1c}^1 - \pi_1^1 + \pi_c^1 \Rightarrow \bar{c}_{1c}^1 = 7 - 0 - 2 \Rightarrow \bar{c}_{1c}^1 = 5$
$\bar{c}_{1d}^1 = c_{1d}^1 - \pi_1^1 + \pi_d^1 \Rightarrow \bar{c}_{1d}^1 = 7 - 0 - 4 \Rightarrow \bar{c}_{1d}^1 = 3$
$\bar{c}_{2b}^1 = c_{2b}^1 - \pi_2^1 + \pi_b^1 \Rightarrow \bar{c}_{2b}^1 = 9 - 0 - 2 \Rightarrow \bar{c}_{2b}^1 = 7$
$\bar{c}_{2c}^1 = c_{2c}^1 - \pi_2^1 + \pi_c^1 \Rightarrow \bar{c}_{2c}^1 = 3 - 0 - 2 \Rightarrow \bar{c}_{2c}^1 = 1$
$\bar{c}_{3a}^1 = c_{3a}^1 - \pi_3^1 + \pi_a^1 \Rightarrow \bar{c}_{3a}^1 = 8 - 2 - 1 \Rightarrow \bar{c}_{3a}^1 = 5$
$\bar{c}_{3b}^1 = c_{3b}^1 - \pi_3^1 + \pi_b^1 \Rightarrow \bar{c}_{3b}^1) = 9 - 2 - 2 \Rightarrow \bar{c}_{3b}^1 = 5$
Since all non basic arcs have a positive reduced cost the above flow is optimal therefore the total minimum cost of the first objective is:

$$min f_1(x) = \sum_{i \in \mathbf{V}} \sum_{j \in \mathbf{suc(i)}} c_{ij}^1 x_{ij}$$

$f_1(x) = c_{1a}^1 x_{1a} + c_{1b}^1 x_{1b} + c_{2a}^1 x_{2a} + c_{2d}^1 x_{2d} + c_{3c}^1 x_{3c} + c_{3d}^1 x_{3d}$
$=(1 \times 5) + (2 \times 3) + (1 \times 6) + (4 \times 13) + (4 \times 14) + (6 \times 3)$
$=143$
by applying Biobjective network simplex method as we did before in section 2.3.1 we can find all extremal points of this problem.

Therefore, : $f^1 = (143, 265)$, $f^2 = (165, 200)$,$f^3 = (176, 175)$, $f^4 = (186, 171)$, $f^5 = (208, 167)$
are the extreme points of the given problem.

The ideal point is $f^* = (143, 167)$. if the aspiration level vector is z=(160 ,180) then the corresponding weights which is given by :

$$\pi_k = \frac{\frac{1}{z_k - f_k^*}}{\sum_{i=1,2} \frac{1}{z_i - f_i^*}}$$

$$\pi_1 = \frac{\frac{1}{z_1 - f_1^*}}{\sum_{i=1,2} \frac{1}{z_i - f_i^*}} = \frac{\frac{1}{z_1 - f_1^*}}{\frac{1}{z_1 - f_1^*} + \frac{1}{z_2 - f_2^*}} = \frac{\frac{1}{160-143}}{\frac{1}{160-143} + \frac{1}{180-167}} = 0.433333$$

$$\pi_2 = \frac{\frac{1}{z_2 - f_2^*}}{\sum_{i=1,2} \frac{1}{z_i - f_i^*}} = \frac{\frac{1}{z_2 - f_2^*}}{\frac{1}{z_1 - f_1^*} + \frac{1}{z_2 - f_2^*}} = \frac{\frac{1}{180-167}}{\frac{1}{160-143} + \frac{1}{180-167}} = 0.566667$$

Therefore we must obtain the closest extreme point to the solution of problem P_π .
The process is shown in the following table:

q	H_p	f	$\bar{y}$	f_{ex}	w_k^q	f_{exnt}	$L_1(w_k^q)$	$L(w_k^q)$	θ
1	$\{f^1, f^5\}$	(186.126,199.979)	18.688	f^5	(1,1)	f^3	174.433	6.672	0.175
2	$\{f^1, f^3\}$	(171.064,188.461)	12.161	f^3	(1.374,1)	f^3	203.967	7.472	0.149
3	$\{f^1, f^3\}$	(171.064,188.461)	12.161	f^3	(1.694,1)	f^2	227.829	10.30	0.006
4	$\{f^2, f^3\}$	(167.445,188.461)	10.593	f^3	(1.662,1.053)	f^3	231.104	10.37	0.008
5	$\{f^2, f^3\}$	(167.445,185.693)	10.593	f^3	(1.694,1)	f^2	227.829	10.30	0.006
6	$\{f^2, f^3\}$	(167.445,185.693)	10.593	f^3	(1.662,1.053)	f^3	231.104	10.374	0.004
7	$\{f^2, f^3\}$	(167.445,185.693)	10.593	f^3	(1.678,1.026)	f^3	229.723	10.593	0

Table: Efficient(extreme) solution of P_π problem
In this table :- q is iteration;
H_p is the current search segment defined by the extreme points enclosed between brace;
$\bar{f}$ is the best known solution ;
$\bar{y}$ is its objective value associated with problem P_π ;
$\bar{f}_{ex}$ is the actual best efficient extreme point;
w_k^q is the lagrangian multiplier vector obtained in the iteration q;
$\bar{f}_{exnt}$ is the extreme point obtained in resolution of $L_1(w_k^q)$;
θ is the step size for the lagrangian multipliers in the next iteration.
The above table shows that the optimal solution of problem p_π is known from iteration 3 because the efficient extreme points f^2 and f^3 have already been obtained. The extreme points calculated in subsequent iteration will be f^2 and f^3 respectively. At the end of the algorithm $f = \bar{f}$ is the efficient solution of the problem and $\bar{f}_{ex} = f^3$ is the closest extreme point to the solution point in the given direction.
In the resolution of P_π the calculated extreme point belong to the cone that is defined by extreme points in the segment H_p and the ideal point. The extreme point f^4 is not calculated since it need not be computed to obtain the optimal solution of problem P_π . In each iteration, only an extreme point which belongs to this cone is calculated. If

this extreme point fulfills the condition given in the previous method, the new cone will be smaller than the previous cone. Furthermore, the point in R^2 belonging to the new cone have values for y lower than the points in the previous cones. In figure below the successive cones for the example of the above table are shown. Once P_π is solved, we can construct a method in order to obtain all the efficient extreme points in the objective space of BNF problems. Furthermore we can design a method of obtain the efficient extreme points belonging to a special region of the objective space given by the decision maker.

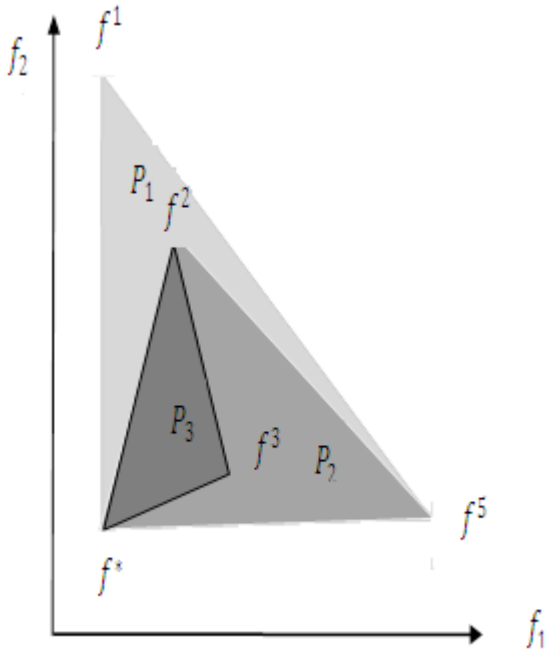


Figure which shows The successive cone

A method to obtain All the efficient extreme points of BMCF problem

The method to calculate all the efficient extreme points is based on the previous ideas. Given a search direction, specified by an aspiration level vector z and the ideal point, we can find the closest efficient extreme point in accordance with the direction given by using the previous algorithm. ([8])

In the course of solving problem P_π we can take advantage of the fact that other efficient extreme points are calculated before the optimal solution is obtained. We can use this advantage to design a method to compute all efficient extreme points.

At the beginning we only know the efficient extreme points that minimize each objective separately, and the segment H_p is defined by these points. Then, given an adequate weight π, the problem P_π is solved in the following way

- If an extreme point not previously calculated is found(though it may not be the optimal for P_π , two segments are then built. These new segments are the result of the gradual substitution of the extreme point previously found by each one of the points of the segment.

- If a new extreme point is not found however, it means that for the cone defined by the points in the segment and by the local ideal point no other extreme point exists, as such, this segment is net worthy of further consideration.

The method uses a queue(Q) that store the segment that determine the cones to be examined. For each one of them, an arbitrary aspiration level vector z is given, which belongs to the examined cone; this determines the weight . The corresponding P_π problem is then solved.

Once the current segment has been examined, we continue with the following segment in the queue. The method ends when the queue is empty. ([8])
The proposed method has the following scheme:

Algorithm EEP(efficient extreme points) ([8])

$H_{p-current}$:= segment defined by the extreme points in the minimization for each objective;
P_{ex}:= the point of $H_{p-current}$;
Int-queue(Q);
Put-in-queue($H_{p-current}$)
While($Q <> Null$) **do**
begin
$H_{p-current}$:= first element(Q);
f_L^*:= local ideal point associated with $H_{p-current}$;
New extreme point:=false;
π:= build an adequate weight;
SP_π($H_{p-current}$, f_L^*, π, new extreme point, f_{ext})(stop solution of P_π if a new efficient extreme point is obtained f_{ext} may not be the optimum for P_pi)
If new extreme point **then**
begin
$P_{ex} := P_{ex} + f_{ext}$;
Add to the queue Q the two segments which result from the interchange of f_{ext} with each one of the points in $H_{p-current}$.
end
end

The extreme points calculated by the algorithm are stored in P_{ex}.
Let $H_{(p-current)}$ be the search segment that is currently examined by the algorithm and that, together with the local ideal point f_L^*, defines the search cone given a direction and the search cone, the associated P_π problem will then be solved. In the resolution of this

problem, if a new extreme point(in the algorithm f_{ext}) is obtained, the process will be terminated.

If this occurs, f_{ext} is introduced in P_{ex}, and the new created segment are added to the queue Q. The vector z in each examined cone can be chosen in the following way. Let f^r and f^s be the extreme points that along with the local ideal point f_L^*,define the examined cone, then each z_k is equal to $z_k = \frac{|f_k^r - f_k^s|}{2} + f_{L_k}^*$,k=1,2. There is no problem if vector z lies inside or outside the efficient boundary due to proposition 1(z determines one weight π).

Figure below illustrates the idea of the method used for the example previously introduced. In figure , the first segment is defined by the f^1 and f^5 because these are points that minimize each objective separately. In this case, the local ideal points coincides with the ideal level point f^*. We suppose that the search direction is as indicated in the figure, and that by solving P_π, f^2 is obtained(though the extreme point solution is f^3). In figure b , once f^2 is found, the generated segments became visible. We now have two problems similar to the original, and as such, we give a direction for each cone and we solve the P_π problem associated with each of them.

In figure c, the two new segment(H_{p3} and H_{p4}) can be observed when the point f^3 has been found. We should therefore observe that in the cone defined by H_{p1}, no other extreme point has been found, so the method does not examined more than that region.

In figure d two new segment H_{p5} and H_{p6} are shown when f^4 is found. No other extreme point is found in H_{p3} and therefore, no new segments are generated. There are no extreme point are found in the cone H_{p5} and H_{p6} and so, the method ends Having identified all the efficient extreme points.

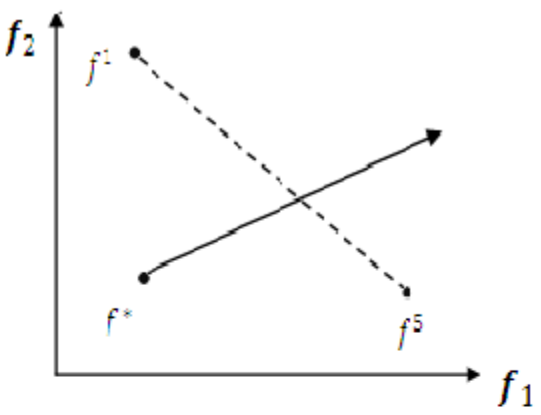


Figure a

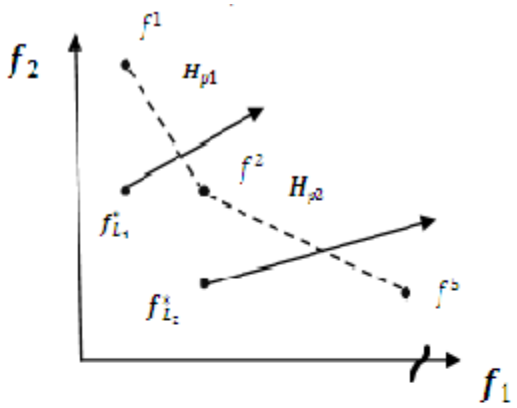


Figure b

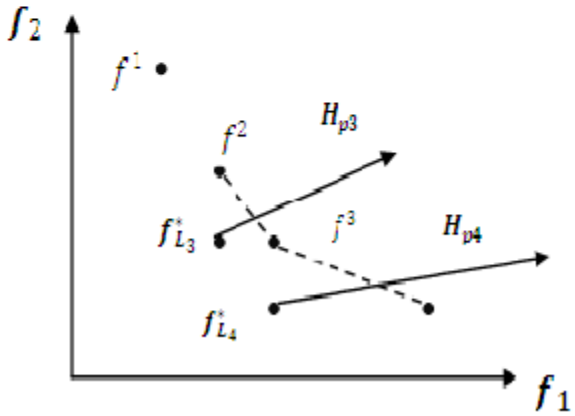


Figure c

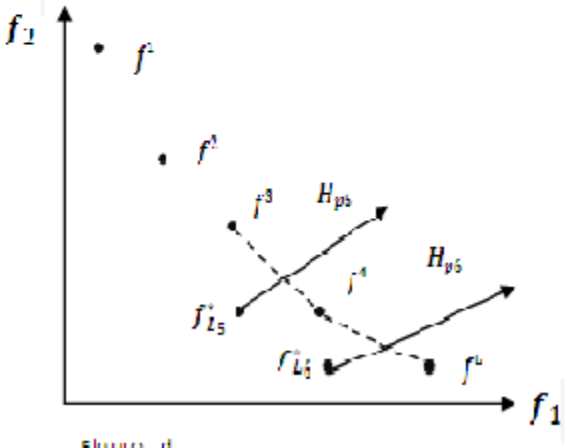
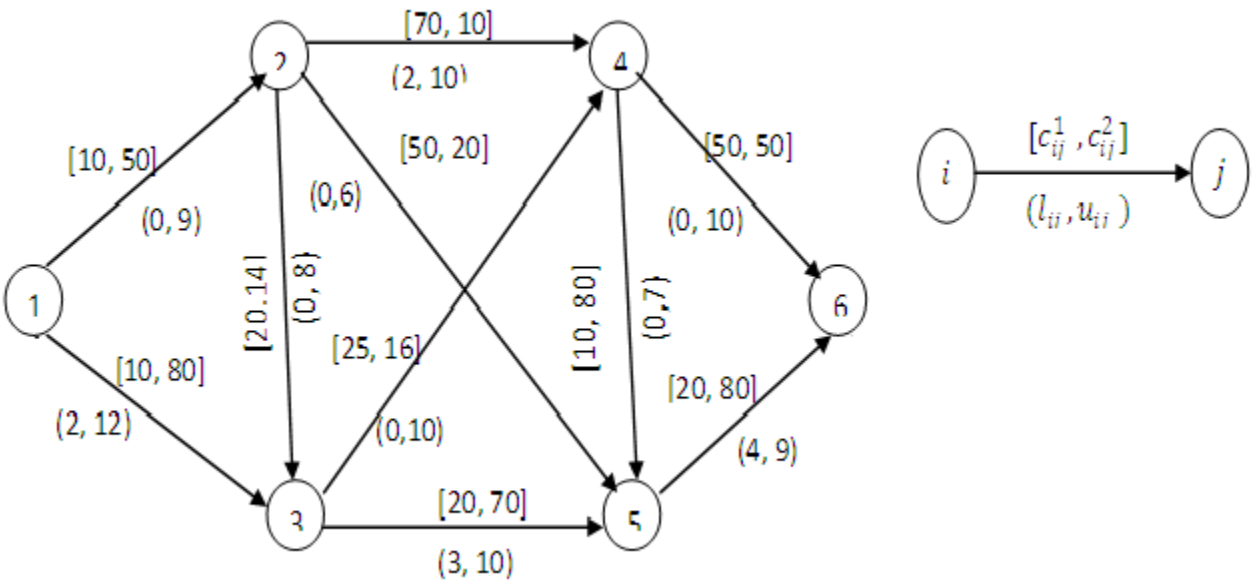


Figure d

Example Consider the example given in the Fig. below where $b_1 = 10$, $b_6 = -10$ and $b_i = 0$, with i = 2, . . . , 5. This example has eight efficient extreme points in the decision space, from which four correspond to efficient extreme points of the objective space (x^1, x^2, x^3 and x^8).



These four extreme points in the objective space are :
$f^1 = (640, 2170), f^2 = (660, 2060), f^3 = (810, 1660), f^4 = (1030, 1484)$
The EEP algorithm identify them when the extreme point f^1 and f^4 are known. In table below, iter represent the iteration, $H_{p-current}$ is the current search segment defined by the extreme points enclosed between braces,π stores the weight that define the search direction, Q is the queue where all the segments are to be examined are stored, including the current one. f_{ext} is the new extreme point and P_{ex} is the list where the extreme points are stored. Table below shows that in iteration 2,4 and 5 no extreme point is found because in the search cone $H_{p-current}$ there are no extreme points. In iteration 3, all the extreme points are known.

Table: EEP of the objective space

Iter	$H_{p-current}$	π	Q	f_{ext}	p_{ext}
1	$\{f^1, f^4\}$	(0.6375,0.3625)	$\{\{f^1, f^4\}\}$	f^3	$\{f^1, f^4, f^3\}$
2	$\{f^3, f^4\}$	(0.2391,0.7609)	$\{\{f^3, f^4\}, \{f^1, f^3\}\}$		$\{f^1, f^4, f^3\}$
3	$\{f^1, f^3\}$	(0.7500,0.2500)	$\{\{f^1, f^3\}\}$	f^2	$\{f^1, f^4, f^3, f^2\}$
4	$\{f^1, f^2\}$	(0.9844,0.0156)	$\{\{f^1, f^2\}, \{f^1, f^2\}\}$		$\{f^1, f^4, f^3, f^2\}$
5	$\{f^2, f^3\}$	(0.8337,0.1663)	$\{\{f^2, f^3\}\}$		$\{f^1, f^4, f^3, f^2\}$

Reference

1. Andrea Raith and Matthias Ehrgott, :A Two-Phase Algorithm for the Biobjective Integer Minimum Cost Flow Problem(November 22, 2007).

2. Horst W. Hamacher, Christian Roed Pederseny, Stefan Ruzika: Multiple Objective Minimum Cost Flow A review ,18th February 2005.

3. H.Lee and P.S. Pulat: Bicriteria network flow problems: Continuous case. European Journal of Operational Research, 51:119 - 126, 1991.

4. Kalyanmoy Deb : Multi-objective optimization using Evolutionary Algorithm-Wiley and Sons, inc, NewYork, NY, USA(2001).

5. Kaisa Miettinen : Non-linear Multiobjective Optimization. Kluwer Acadamic publishers,1999. International series in operations research and management science, Volume 12.

6. Matthias Ehrgott: Multicriteria Optimization - springer(2005).

7. Ravidra K. Ahuja, Thomas L.Magnanti, James B. Orlin : Network Flows, Theory, algorithms and applications-Prentice Hall(1993).

8. Sedeno-Noda A. and Gonzalez-Martin C. : An alternative method to solve the biobjective minimum cost flow problem. Asia-Pacific Journal of OperationalResearch, 20(2003)241 - 260.

9. Sedeno-Noda A. and Gonzalez-Martin C. : An algorithm for the biobjective integer minimum cost flow problem. Computers and Operations Research, 28(2001)139-156.

10. Sedeno-Noda A. and Gonzalez-Martin C. The biobjective minimum cost flow problem. European Journal of Operational Research, 124(2000) 591-600.

11. Sarah Steiner and Tomasz Radzik : Solving the Biobjective Minimum Spanning Tree problem using a k-best algorithm(October 2003).

ADDIS ABABA UNIVERSITY DEPARTMENT OF MATHEMATICS

The undersigned hereby certify that they have read and recommend to the department of mathematics for acceptance a project entitled "Biobjective minimum cost flow problem" by Bedilu Tsige in partial fulfillment of the requirements for the degree of Master of Mathematics.

Date: _____

Advisor Signature

Dr. Berhanu Guta _____

By Signature

Bedilu Tsige Mekuriya _____