Addis Ababa University (Since 1950)

Addis Ababa University

School of Graduate Studies

Faculty of Computer and Mathematical Sciences

Department of Mathematics

Project

On

BI-OBJECTIVE SHORTEST PATH PROBLEM

By: - ALEMU BEKELE FEYISSA

Advisor: -Dr. BERHANU GUTA (PhD)

A Project submitted to the Office of Graduate Programs of Addis Ababa University in Partial fulfillment of the requirements for the Degree of Master of Science in Mathematics

January, 2011

AA, Ethiopia

Addis Ababa University
school of Graduate Studies
Faculty of Computer and Mathematical Sciences
Department of Mathematics

Approved by the Board of Examiners:

_____	_____
Department Head	Signature
_____	_____
Examiner	Signature
_____	_____
Examiner	Signature

Declaration

I declare that this project has been composed by me and that no part of the project has formed the basis for the award of any Degree, Diploma, Associate ship, Fellowship or any other similar title to me.

Author's Signature

Permission

This is to certify that this project is compiled by Alemu Bekele in the Departement of Mathematics, Addis Ababa University, under my supervision. I hereby also confirm that the project can be submitted for evaluation by examiners and eventual defense.

Advisor's Signature

Acknowledgment

First of all I would like to praise my Lord, who gave me everything I need and brought me from nothing to here. I have no words to describe everything He has done/is doing for me, so I shall stop just by praising. I also like to express my gratitude to Dr. Berhanu Guta, my advisor for this master project, who devoted a great part of his valuable time and give me very important supports and suggestions for this work to be successfully done. Finally I'm very grateful to my spouse, w/ro Ejigayehu Bekele who made my studies possible by supporting me morally & financially.

Thank you very much

Alemu Bekele

Alemubekele5@gmail.com

January, 2011

Summary of the project report: In this paper I consider the bi-objective shortest path (BSP) problem as an extension of single objective shortest path (SSP) problem. The paper is focusing on solving the bi-objective shortest path problem which is an extension of the shortest path problem resulting from considering simultaneously two cost functions(criteria) for the arcs. And also I consider a Two phase method for computing the complete set of efficient solutions of the bi-objective minimum spanning tree problem. In this case the first phase computes the extreme efficient solutions by expanding the convex hull of the feasible space. The second phase repeatedly applies a k-best minimum spanning tree algorithm to find all non-extreme efficient solutions.

Key words: Bi-objective shortest path problem, Labeling algorithm,

Two phase method, Non-dominated path.

Table of Content

Title page

SECTION ONE

INTRODUCTION

Shortest path problems have been studied much in literature and among them the single objective shortest path (SSP) problem is the most widely studied. However, it is often not sufficient to restrict oneself to one objective. Different applications often indicate the necessity of taking two or more objectives in to account, resulting in bi-objective or multiple objective shortest path problems.

Combinatorial optimization problems with multiple objectives are natural extensions, practical as well as theoretical, of single objective problems. And BSP problem belongs to the class of multi objective combinatorial optimization (MOCO) problems. I will discuss how the bi-objective shortest path problems solution strategies can be generalized from the mono- objective shortest path problems solving techniques.

In BSP problems an efficient solution is the one such that there is no other solution which is better on both objectives. I consider in this paper the problem of finding all efficient solutions. In other words, without having any additional information about the relative importance of the different objectives, my task is to present the decision makers with all possible solutions and let them make the final selection. Other definitions of the optimality of a solution for BSP problems are based on lexicographic minimization. The lexicographic minimization requires the decision maker to order the criteria by their relative importance. This type of problem produces a single solution (in the objective space): it optimizes the first criteria and then the second. In this paper I use lexicographic minimization to select the permanent label from all temporary labels and not to decide efficient solutions.

Section 2 of the project report raises some points about shortest path problem and some of its applications and also contains the basic generic algorithms (labeling techniques) mainly focusing on single shortest path problems. **In section 3** basic concepts of the BSP problems are introduced, generalizing of the labeling algorithms to BSP is discussed and the correctness of the algorithm is proved. And also how the Two phase method can be used to solve the BSP problem will discussed in this section.

SECTION TWO

PRELIMINARY CONCEPTS

2.1SOME IMPORTANT DEFINITIONS

Definition 2.1.1 Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a directed graph

a)$\mathcal{G}$ is called a net if and only if

1- $\mathcal{G}$ a directed graph without circuit.

2- $\mathcal{G}$ Possess exactly one source and one sink.

b) Let $\mathcal{G}$ be a net and let $\psi : \mathcal{A} \rightarrow \mathbb{R}_+$ be a given function. Then the pair $(\mathcal{G}, \psi)$ is called a network.

Definition 2.1.2 A path is a sequence of arcs $P = \{(i_0, i_1), \ldots, (i_{p-1}, i_p)\}$ in which the initial node of each arc is the same as the terminal node of the preceding arc and each arc in the path is directed "away" from i_0 and "towards" i_p.

Definition 2.1.3 A circuit is a closed path.

Definition 2.1.4 A chain is similar structure to a path except that no all arcs are necessarily directed towards the terminal node.

Definition 2.1.5 A connected graph is a graph in which there exists a chain between every pair of nodes in$\mathcal{G}$.

Definition 2.1.6 A cycle is a closed chain.

Definition 2.1.7 A tree is a connected graph with no cycle.

Definition 2.1.8 A spanning tree is a tree that includes every nodes of a graph. i.e. it is spanning, connected sub graph with no cycle

2.2. WHAT IS THE SHORTEST PATH PROBLEM?

The shortest path problem is a classical and important combinatorial problem that arises in our day to day activities. We are given a directed graph ($\mathcal{N}$, A) with nodes numbered 1,...,$\mathcal{N}$. Each arc (i, j) $\in$ A has a cost or length C_{ij} associated with it. The length of forward path ($i_1, i_2, \ldots i_k$) is the length of its arcs

$$\sum_{n=1}^{k-1} C_{i_n i_{n+1}}$$

This path is said to be shortest if it has minimum length over all forward paths with the same origin and destination nodes. The length of a shortest path is also called the shortest distance. The shortest path problem deals with finding shortest distance between selected pairs of nodes.

2.3. WHAT APPLICATIONS DO THE SHORTEST PATH PROBLEMS HAVE?

The range of applications of the shortest path problem is very broad. Let us discuss a few representative applications.

1. Routing in Data Network

Date network communication involves the use of a network of computers (nodes) and communication links (arcs) that transfer packets (groups of bits) from their origin to their destinations. The most common method for selecting the path of travel (or route) of packets is based on a shortest path formulation. In particular, each communication link is assigned a positive scalar which is viewed as its length. A shortest path routing algorithm routes each packet along a minimum length (or shortest) path between the origin and destination nodes of the packet.

There are several possibilities for selecting the link lengths. The simplest is for each link to have unit length, in which case the shortest path is simply a path with minimum number of links. More generally the length of a link may depend on its transmission capacity and its projected traffic load. The idea here is that a shortest path should contain relatively few and uncongested links, and therefore be desirable for routing.

2. DYNAMIC PROGRAMMING

Here we have a discrete time dynamic system involving N stages. The state of the system at the start of the k_{th} stage is denoted by x_k and takes values in a given finite set, which may depend on the index k. The initial state of is given .During the k_{th} stage, the state of the system changes from x_k to x_{k+1} according to the equation of the form

$$x_{k+1} = f_k(x_k, u_k) \quad \text{---------- 2.1}$$

Where u_k is a control that takes values from a given finite set, which may depend on the index k. This transition involves a cost $g_k(x_k, u_k)$ .The final transition from x_{N-1} to

x_N , involves an additional terminal cost $G(x_N)$. Here, the functions f_k , g_k and G are given.

Given a control sequence $(u_0, \ldots, u_{N-1})$, the corresponding state sequence $(x_0, \ldots, x_N)$ is determined from the given initial state x_0 and the system **equation (2.1)** . The objective in dynamic programming is to find a control sequence and corresponding state sequence such that the total cost,

$$G(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k) \quad \text{is minimized}$$

We need to convert the dynamic programming problem to a shortest path problem and control sequences corresponds to paths originating at the initial state x_0 and terminating at one of the nodes corresponding to the final stage N. The optimal control sequence corresponds to a shortest path from node x_0 to node t.

3. Project management

Consider the planning of a project involving several activities, some of which must be completed before others can begin. The duration of each activity is known in advance. We want to find the time required to complete the project as well as the critical activities those that even if slightly delayed will result in a corresponding delay of completion of the overall project. In this problem nodes represent completion of some phase of the project and an arc (i ,j) represents an activity that starts ones phase i is completed and has known duration $t_{i,j} > 0$. A phase (node) j is completed when all activities or arcs (i ,j) that are incoming to j are completed. Two special nodes 1 and N represent the start and end of the project, respectively. Node 1 has no in coming arcs, while node N has no outgoing arcs. Furthermore, there is at least on path from node 1 to every other node. An important characteristic of an activity net work is that it is acyclic.

For any path $p = \{(1, j_1), (j_1, j_2), \ldots, (j_k, i)\}$ from node 1 to node i, let D_p be the duration of the path defined as the sum of durations of its activities; that is,

$D_p = t_{ij_1} + t_{j_1 j_2} + \cdots\cdots + t_{j_k i}$ **then** the time T_i required to complete phase i is

$$T_i = \max_{\substack{\text{Paths } p \\ \text{From 1 to i}}} D_p$$

The maximum above is attained by some path because there can be only a finite number of paths from 1 to i, since the network is acyclic. Thus to find T_i, we should find the longest path from 1 to i. Because the graph is acyclic, this problem may also be viewed as a shortest path problem with the length of each arc (i ,j) being t_{ij}. In particular, finding the duration of the project is equivalent to finding the shortest path from 1 to N.

Bi-objective shortest path problem solves the more realistic problems than the mono-objective shortest path problem and have similar applications to mono-objective shortest path problem.

2.4. A generic shortest path Algorithm.

The shortest path problem can be posed in different ways. For example, finding shortest path from a single origin to a single destination or finding a shortest path from each of several origins to each of several destinations. Here I focus on problems with a single origin and many destinations. Fore concreteness, let us take origin node to be node 1, the arc lengths c_{ij} are given scalars.

Here let us develop a broad class of shortest path algorithms for the single origin /all destinations problem. These algorithms maintain and adjust a vector $(d_1,d_2,\ldots,d_N)$, where each d_j, called the label of node j, is either a scalar or ∞. Let us motivate the use of labels by simple optimality condition, given in the following proposition.

Proposition 2-1: Let $d_1 ,\ldots.d_N$ be scalars satisfying

$$d_j \le d_i + c_{ij}\ ,\ \forall (i\ ,j) \in \mathcal{A} \text{ ---------- 2.2}$$

And let p be a path starting at a node i_1, and ending at node i_k. If

$$d_j = d_i + c_{ij}, \text{ for all arcs (i , j) of p -------- 2.3}$$

Then p is a shortest path from i_1 to i_k

The conditions (2-2) & (2-3) are called the complementary slackness (CS) conditions for the shortest path problems.

Let us now describe a proto type shortest path method that contains several interesting algorithms as special cases. In this method, we start with some vector of labels $(d_1, \ldots, d_N)$, we successively select arcs (i , j) that violate the CS condition (2 .2), i.e. $d_j > d_i + c_{ij}$ And we set

$$d_j := d_i + c_{ij}$$

This is continued until the CS condition $d_j \leq d_i + c_{i,j}$ is satisfied for all arcs (i,j)

A key idea is that, in the course of the algorithm, d_i can be interpreted for all i as the length of some path p_i from 1 to i. In the case of the origin node 1, we will interpret the label d_1 as either the length of a cycle that starts and ends at 1, or in the case $d_1 = 0$, the length of the trivial "path" from 1 to itself. Therefore, if $d_j > d_i + c_{ij}$ for some arc (i , j) the path obtained by extending path p_i by arc (i , j) which has length $d_i + c_{ij}$ is a better path than the current path p_j , which has length d_j. Thus, the algorithm finds successively better paths from the origin to various destinations.

Instead of selecting arcs in arbitrary order to check violation of CS condition, it is usually most convenient and efficient to select nodes, one – at – a time according to some order, and simultaneously check violation of the CS condition for all of their outgoing arcs.

The corresponding algorithm, referred to as generic, maintains a list of nodes V, called candidate list, and a vector of labels$(d_1 ,d_2 ,\ldots d_N)$, where each of d_j is a real number or ∞

Initially,

V= {1}

$$\mathbf{d_1=0,} \quad d_i = \infty, \forall i \neq 1$$

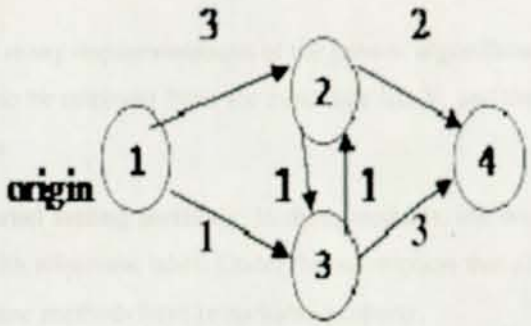
The algorithm proceeds in iterations and terminates when V is empty. The typical iteration is as follows:

Iteration of the Generic shortest path Algorithm

Remove a node i from the candidate list V. For each outgoing arc $(i , j) \in A$, if $d_j > d_i + c_{ij}$, set

$$d_j = d_i + c_{ij}$$

And add j to V if it does not already belongs to V



Iteration #	Candidate list V	Node labels	Node out of V
1	{1}	(0, ∞,∞,∞)	1
2	{2,3}	(0,3,1, ∞)	2
3	{3,4}	(0,3,1,5)	3
4	{2,4}	(0,2,1,4)	4
5	{2}	(0,2,1,4)	2
	∅	(0,2,1,4)	

Figure 2-1 illustrates the generic shortest path algorithm

The numbers next to the arcs are the arc lengths. If we remove 3 in the 2nd iteration each node would enter V only once. Thus the order in which nodes are removed from V is significant.

It can be seen that, in the course of the algorithm, the labels are monotonically non -increasing. Furthermore, we have $d_i < \infty \Leftrightarrow$ i has entered V at least once.

2.5 IMPLEMENTATIONS OF THE GENERIC ALGORITHM

There are many implementations of the generic algorithms. They differ in how they select the node to be removed from the candidate list V, and they are broadly divided into two categories.

a) **Label setting methods**: In these methods, the node i removed from V is a node with minimum label. Under the assumption that all arc lengths are non- negative, these methods have remarkable property.

   Each node will enter V at most once.

   Its label has its permanent or final value at the first time it is removed from V.

   The most time – consuming part of these methods is calculating the minimum label node in V at each iteration

Label setting (Dijkstra) methods is a special case of the generic algorithm where the node i removed from the candidate list V at each iteration has minimum label, that is

$$d_i = \min_{j \in v} d_j$$

Let use state this method explicitly.

Initially, we have

$$V = \{1\}$$

$$d_1 = 0 \quad d_i = \infty, \forall i \neq 1$$

The method proceeds in iteration and terminates when V is empty. The typical iteration is as follows.

Iteration of the label setting Method

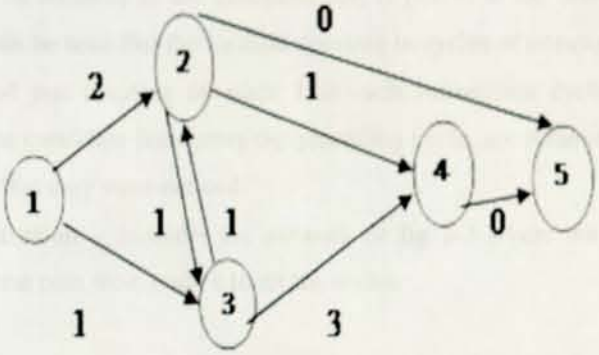
Remove from the candidate list V a node i such that

$$d_i = \min_{j \in V} d_j$$

For each out going arc $(i, j) \in A$, if $d_j > d_i + c_{ij}$.

Set $d_j := d_i + c_{ij}$

And add j to V if it does not already belong to V



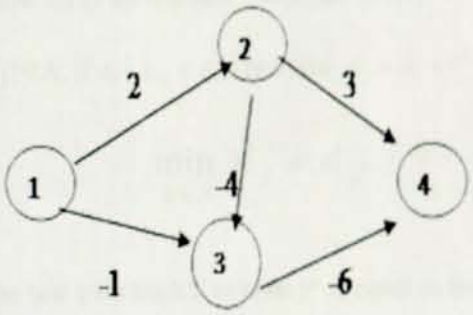
Iteration #	Candidate List V	Node labels	Node out of V
1	{1}	(0,∞,∞,∞,∞)	1
2	{2,3}	(0,2,1, ∞,∞)	3
3	{2,4}	(0,2,1,4, ∞)	2
4	{4,5}	(0,2,1,3,2)	5
5	{4} Ø	(0,2,1,3,2) (0,2,1,3,2)	4

Figure 2-2 example illustrating the label setting method. At each iteration, the node with the minimum label is removed from V. Each node enters V only once.

b) **Label correcting methods**. In these methods the choice of the node i removed from V is less sophisticated than in label setting methods, and requires less calculation. However, a node may enter V multiple times. An important advantage of label correcting methods is that they are more general, since they do not require non negativity of the arc lengths.

The Bellman – ford method is the simplest label correcting method that uses a first-in –first-out rule to update the queue that is used to store the candidate list V. In particular, a node is always removed from the top of the queue, and a node, up on entrance in the candidate list, is placed at the bottom of the queue. Thus, it can be seen that the method operates in cycles of iterations: the first cycle consists of just iterating on node 1 in each subsequent cycle, the nodes that entered the candidate list during the preceding cycle, are removed from the list in the order that they were entered.

Illustration: - consider the network of **fig 2-3** where we wish to find the shortest path from node 1 to all the nodes.



Iteration #	Candidate List V	Node labels	Node out of V
1	{1}	(0,∞,∞,∞,)	1
2	{2,3}	(0,2,-1, ∞)	2
3	{3,4}	(0,2,-2,5)	3
4	{4, } ∅	(0,2,-2,-8) (0,2,-2,-8)	4

Fig 2.3 example illustrating the label correcting method

The labeling algorithm for single objective shortest path problems is based on generic labeling algorithms.

Algorithm 2-1: single objective labeling Algorithm

Step 1: initialization: Assign set of un scanned nodes, X= {s} and

Label it as $d_s = 0$; $d_i = \infty, \forall i \in N-\{s\}$

Step 2 if $X \neq \emptyset$

Label some node $i \in X$ by d_i

And

Take the new set of un scanned nodes, $X = X \setminus \{i\}$

For all $(i, j) \in A$, if $d_i + c_{ij} < d_j$ Then take $d_j = d_i + C_{ij}$ and

$$\min_{j \in X} d_j = d_{j^*}$$

The best path from S to node j^* is equal to the best path from S to nod i merged to the path from i to j^*

$$\text{i. e.} \quad p_j^* = p_i^* \diamond \langle i, j \rangle$$

And the new set of un scanned nodes, $X = X \cup \{j\}$

(i.e. add node j as a member). And take $i=j^*$

Step 3: if $j^* = T$, terminal node, stop

Otherwise go to step 2

In few words, the algorithm scans the vertices of a subset $V \subseteq N$ in order to update the labels assigned to their successors. For $i \in V$, with label di the procedure computes

$d_i + c_{ij} < d_j$, then d_j is up dated and the vertex j joins the set V.

The label d_i corresponds to the value of the shortest path found, up to the present, from the source s to the vertex i. Hence, one strategy for selecting the vertex of V to be scanned is to pick the one with the less value for the label. This is named in the literature as label setting technique since if the optimality principle holds then each vertex is scanned at most one time.

Alternatively, one may select the vertex to scan just following, for instance a FIFO policy (first – in – first – out).

In this case, the same vertex may be selected more than once and, therefore, the procedure stops, only when the tree constituted by the shorts path from S to $i \in \mathcal{N}$ is obtained. Using this technique, known in the literature as the label correcting algorithm, one needs to compute the shortest tree rooted at s.

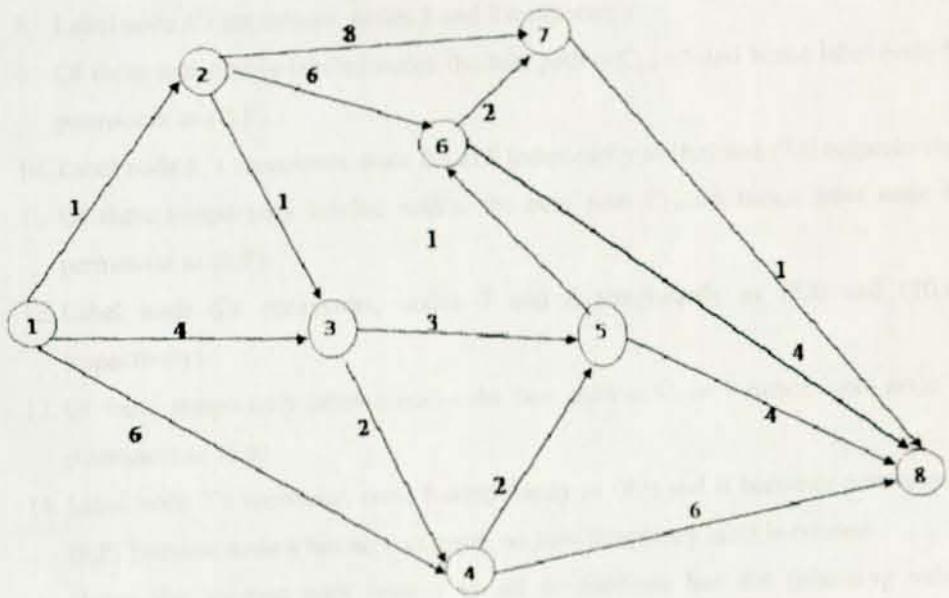


Fig 2.4 Example to illustrate the above algorithm

Solution:

1. Label the source S, temporarily as (0,t) and this 1st label is selected to be made permanent,(0,p)
2. Label nodes 2,3,4 temporarily as (1, t), (4, t) and (6,t) respectively
3. Of temporarily labeled nodes best path is $C_{1,2}$=1, hence label node 2 permanent as (1,P)
4. Label node 2's successors, nodes 3,6,7 temporarily as (2,t), (7, t) and (9,t) respectively
5. Of these temporary labeled nodes, the best path is $C_{1,3}$=2 label node 3 permanent as (2,P), which dominates the previous label of node 3
6. Label node 3's successors, nodes 4 and 5 temporarily as (4,t) and (5.t) respectively
7. Of these temporary labeled nodes the best path is $C_{1,4}$ =4 and hence label node 4 permanent as (4,p) which dominate the previous label of node 4
8. Label node 4's successors, nodes 5 and 8 temporarily
9. Of these temporarily labeled nodes the best path is $C_{1,5}$ =5 and hence label node 5 permanent as (5,P)
10. Label node 5 's successors, node 6 and 8 temporarily as (6,t) and (9,t) respectively
11. Of these temporarily labeled nodes, the best path $C_{1,6}$ =6 hence label node 6 permanent as (6,P)
12. Label node 6's successors, nodes 7 and 8 temporarily as (8,t) and (10,t) respectively)
13. Of these temporarily labeled nodes the best path is $C_{1,7}$= 8 hence label node 7 permanent as (8,p)
14. Label node 7's successor, node 8 temporarily as (9,t) and it becomes permanent, (9,P) Because node 8 has no successor, no new temporary label is created

    Hence the shortest path from s to all destinations has the following value

    [0 , 1 , 2 , 4 , 5 , 6 , 8 , 9]

SECTION 3

3. Bi-objective shortest path problems

3.1. Definitions, terminologies and basic results

Let $\mathcal{G}=(\mathcal{N},\mathcal{A})$ be a directed network with a set of nodes

$\mathcal{N}=\{1,\ldots,n\}$ and

$\mathcal{A} \subset \mathcal{N} \times \mathcal{N}$ is the set of oriented arcs

$C: \mathcal{A} \longrightarrow \mathbb{R}^2$

$(i,j) \rightarrow C(i,j) = C_{ij} = (C_{ij}^1, C_{ij}^2)$

A path p, from the vertex i to j, is an alternative sequence of nodes and arcs of the form $p = \langle v_0, a_1, v_1, \ldots, a_r, v_r \rangle$

Where:

$v_l \in \mathcal{N}, \forall l \in \{0,\ldots,r\}$

$v_0 = i$ and $v_r = j$

$a_l = (v_{l-1}, v_l) \in \mathcal{A} \ \forall l \in \{1,\ldots,r\}$

By convention $\langle v_0 \rangle$ is considered as a null path (r=0) .The set of all paths from i to j is denoted by $p_{i,j}$ and $p_{\mathcal{G}}$ represents the set of all paths in the network, that is,

$$p_{\mathcal{G}} = \bigcup_{i,j \in \mathcal{N}} p_{i,j}$$

Without loss of generality let us consider that $\mathcal{N}$ has an initial node s and a terminal node t such that $p_{s,t} \neq \emptyset$, $p_{s,i} \neq \emptyset$, and $p_{i,t} \neq \emptyset$, for any $i \in \mathcal{N}$- {s , t}, and to simplify the notation let us use p, instead of $p_{s,t}$.

Multiple arcs (arcs between the same pair of nodes) are not allowed. As a consequence, p can be denoted only by the sequence of its nodes, $\langle v_0, v_1, \ldots, v_r \rangle$

A bi-objective function f is defined over the set of all the paths on the network, as follows:

$$f : p_{\mathcal{G}} \longrightarrow \mathbb{R}^2$$

$$p \longmapsto f(p) = (f_1(p), f_2(p)) \quad \text{----------3.1}$$

$$\text{Where } f_l(p) = \sum_{(i,j)\in p} C_{i,j}^{l}, \forall\, l \in \{1, 2\}$$

The concatenation operator $\circ$ joins two paths $p = \langle v_0, \ldots, v_{r_p} \rangle$ and $q = \langle u_0, \ldots, u_{r_q} \rangle$ such that $v_{r_p} = u_0$ then, $p \circ q = \langle v_0, \ldots, v_{r_p} = u_0, \ldots, u_{r_q} \rangle$

Now, in order to solve the BSP problem, one looks for the set of non – dominated (ND) paths from s to t, mathematically described as follows.

Definition 3.1 let p and q be two paths on the network with the same initial and terminal node. We say p dominates q or q is dominated by p ($p <_D q$) if and only if

$$f(p) \neq f(q) \quad \text{and} \quad f(p) \leq_{\mathbb{R}^2} f(q)$$

$$\text{(i.e. } (f_l(p) \leq f_l(q)), l = 1, 2)$$

Definition3.2. Let p be a path in $p_{i,j}$ i, j $\in \mathcal{N}$. If there is no path $q \in p_{i,j}$ such that $q <_D p$ then p is called non -dominated, efficient or Pareto optimal path.

A Bi- objective combinatorial optimization (BOCO) problem can be described as:

$$\text{Min } (f_1(p), f_2(p)) \quad \text{----------3.2}$$
$$p \in X$$

Where X is a feasible set, p a feasible solution and $f_l(p)$ an objective function ($l \in \{1, 2\}$).

Definition 3.3 two feasible solutions p and p' are called equivalent if $f(p) = f((p')$. A complete set X_E is a set of efficient solutions such that all $p \in X \setminus X_E$ are either dominated or equivalent to at least one $p \in X_E$

Another notion of optimality that is used in the context of bi-objective optimization is lexicographic minimization. Here, we choose among all optimal feasible solutions for the preferred component of the objective vector one that is optimal for the other component l

Definition 3.4 Let $k \in \{1, 2\}$ and $l \in \{1, 2\} / \{k\}$. Then $f(p) <_{lex(k,l)} f(p')$ if either $f_k(p) < f_k(p')$ or both $f_k(p) = f_k(p')$ And $f_l(p) < f_l(p')$ for $p, p' \in X$. we call p a lex (k, l)-prior preference if $f(p) \leq_{lex(k,l)} f(p')$

3.2. Generalizing the labeling algorithm for BSP

Now, let us remind that for the single objective problem case the shortest path problem (SPP) can be easily solved when the optimality principle holds, meaning that every shortest path is formed by shortest sub – paths. A necessary and sufficient condition for the SPP to verify the optimality principle is that the network has no cycles with negative cost.

To find paths a straight forward approach is to generalize single objective shortest path algorithms. Let us first assume that no negative cycles exist, i.e. for all cycles in $\mathcal{G}$, and for $l = 1, 2$; p paths in the cycle,

$$\sum_{(i,j)\in p} C_{ij}^{l} \geq 0 \text{ ----------3.3}$$

And that this inequality is strict for at least one l. under this assumption, an efficient path will be simple, i.e. never visit any node more than once, because including a cycle increase the total cost of the path for at least one objective, where as the other objective is not improved.

Proposition3.5. Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a graph that satisfies the assumption above. Let p be an efficient path from s to t, then p contains only efficient paths from s to intermediate nodes on the path.

proof (by contradiction) Let $p_{s,t}$ be an efficient path from s to t. Assume $p_{s,i}$ is a sub path from s to node v_i and this path is not efficient. Then there is a path

$$p'_{s,i} <_D p_{s,i}$$

i.e. $f(p'_{s,i}) \neq f(p_{s,i})$ And $f(p'_{s,i}) \leq_{\mathbb{R}^2} f(p_{s,i})$

Therefore $p'_{s,i}$ together with $p_{i,t}$ the sub path of p from i to t, is a path from s to t with

$f(p'_{s,i} \circ p_{i,t}) = f(p'_{s,i}) + f(p_{i,t}) < f(p_{s,i}) + f(p_{i,t}) = f(p_{s,t})$

Contradicting efficiency of $p_{s,t}$ ■

Actually, the proof of **proposition 3.5** still valid if the path $p_{s,i}$ is replaced by any sub path $p_{u,v}$ between nodes of the efficient path $p_{s,t}$. This is stated as a corollary

Corollary 3.6 under the assumption of **proposition 3.5** let $p_{s,t}$ be an efficient path from s to t. Then any sub path $p_{u,v}$ from u to v, where u and v are vertices on $p_{s,t}$ is an efficient path from u to v.

It is also important to note that although an efficient path is always composed of efficient sub paths between Vertices along the path it is in general not true that compositions of efficient paths yield efficient paths again. Illustrative example in the graph of **figure 3.1** paths P_{13}= (1, 3) and P_{34}= (3, 4) are efficient, but their composition is not, because it is dominated by the path (1 , 4) from node 1 to node 4

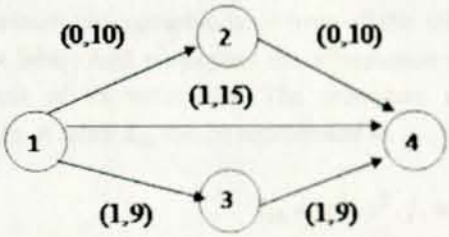


Fig 3-1 combining efficient paths

For BSP, the optimality principle corresponds to saying that every ND path is formed by ND sub paths. However, for optimality to be verified in this case, only the following sufficient condition is known for each criterion, the network has no cycle with negative cost.

When we generalize the labeling algorithm for BSP problem, some key aspect must be taken in to account.

Firstly, lets us remind that, now the concept of **best path** from s to t is not appropriate since we may have more than one "best" paths from s to any vertex. So, for each $i \in \mathcal{N}$, we define a set d_i containing all the paths from s to i for which, up to the moment, there is no other path dominating it. We call $d_i$the set of temporarily non – dominated path from s to i which has to be up dated whenever a new s-i path joins it.

A second key aspect is related to the possibility of having, contrarily to the single objective case, **more than one label** associated to the same vertex. Therefore, in labeling algorithm for BSP problem, the set of paths corresponding to temporary labels will consist of un scanned labels rather than nodes, as it were the case in single objective case. If we consider straight forward generalization of the labeling algorithm, we follow a label selection policy, that is, at each iteration, a single un scanned label is picked from temporary labels and the corresponding path from s to i $\in \mathcal{N}$ is expanded by adding up the arcs (i, j) $\in A$.

When we proceed to generalization of the **algorithm 2.1** for BSP case, the **stop condition** is another aspect that requires particular attention. In fact, since the number of ND paths from s to any destination $i \in \mathcal{N}$ is not known in advance, the algorithm stops only when the set of temporary labels of the entire nodes is empty. This means that the resolution of the BSP Problem, from a source s to a destination t, requires the computation of all ND paths from s to each vertex $i \in \mathcal{N}$

The **Algorithm 3.1** below shows a sketch of the bi -objective label setting algorithm. In BSP problem label setting algorithm case at each iteration for each vertex there are two different sets of labels; permanent and temporary labels. The algorithm selects the minimum lexicographic label from all the sets of temporary labels. Convert it to a permanent label. And propagates the information contained in this label to all the temporary labels of its successors. The procedure stops when there are no more temporary labels. A label l_{ik} can be represented as

$$l_{ik} = [z^1, z^2, j, m]$$

For i - node number

k-represent lexicographic preference rank of the label l_{ik} w.r.t. the entire labels on node i

Where $(z^1, z^2)$is the performance Vector. j is the adjacent vertex from which it was possible to label the vertex i. And m is the position of the label in the list of labels on vertex j.

ALGORITHM: 3.1 Label setting algorithm for BSP problem.

Require: $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, and c, the cost matrix for all arcs $(i, j) \in \mathcal{A}$

Ensure: All efficient paths from s to all vertices $i \in \mathcal{N} \setminus \{s\}$

l_i **:** is a label of vertex i

lt_i : is the entire list of temporary labels of vertex i

lp_i : is the entire list of permanent labels of vertex i

$z^p_{q,m}$: is the p^{th} performance of a permanent label of vertex q in position m

$<_D$:is the dominance relation (if $z <_D z'$, *then z dominates z'*)

Step 1: Initialization

lt_i , $lp_i \leftarrow \emptyset, \forall i \in \mathcal{N}$

$lt_s \leftarrow \{[0, 0, \wedge, \wedge]\}$.

Step 2: Main step

While $(\cup_{i \in \mathcal{N}} lt_i \neq \emptyset)$

Find the minimum lexicographic label in lt_i , $\forall i \in \mathcal{N}$

$l_q \leftarrow$ Min lex$\{\cup_{i \in \mathcal{N}} lt_i\}$

In case $l_q = l_y$ take in the topological order of the nodes (i.e. if q < y, first take l_q)

Then move the selected label from the 'temporary' list to the 'permanent' list

$lt_q \leftarrow lt_q \setminus \{l_q\}$: $lp_q \leftarrow lp_q \cup \{l_q\}$

Store the position of label l_q from list lp_q

$m \leftarrow$Card (lp_q)

For all $j \in \mathcal{N} | (q, j) \in \mathcal{A}$ do

Compute l_j ,the current label of vertex j

$l_j \leftarrow [z^1_{q,m} + c^1(q, j),\ z^2_{q,m} + c^2(q, m), q, m]$

Verify that there is no performance of vertex j label dominating perf(l_j)

If ($\nexists\, l'_j \in \{lt_j \cup lp_j\}$ such that perf (l'_j) $<_D$ perf (l_j)) then

Store the label l_j of vertex j as temporary

$lt_j \leftarrow lt_j \cup \{l_j\}$

Delete all temporary labels of vertex j dominated by l_j

$lt_j \leftarrow lt_j \setminus \{l'_j \in lt_j\}$ Such that perf(l_j) $<_D$ perf(l'_j)

End-if

End-for

End-while.

Each permanent label corresponds to a unique efficient path .To determine any of these paths, choose one permanent label on vertex q and extract the values corresponding to j *and* m for this label. So j is the vertex just before q in efficient path. To determine the label on vertex j that has produced this current path, the value of m is needed .This value indicates the m_{th} label on vertex j that has produced the current path. By moving backwards, the first vertex of the path(s) will be reached.

3.3 Determining the minimum Label in the set of paths corresponding to temporary labels

Let us recall that the label setting version of the algorithm is based upon the condition that the label selected at each iteration and which become permanent will become definitively non- dominated (ND). Next, we will show that this can be guaranteed if one considers an operator R defined over the set p_ζ and holding two properties.

So, given an operator R establishing a relation between paths linking the same pair of nodes, one may define an auxiliary function h_R: $p_G \rightarrow \mathbb{R}$ and the following binary relations:

$$\leq_R \subset p_{i,j} \times p_{i,j} \text{ , such that } p \leq_R q \Leftrightarrow h_R^{(p)} \leq h_R^{(q)}$$

$$<_R \subset p_{i,j} \times p_{i,j} \text{ ,such that } p <_R q \Leftrightarrow h_R^{(p)} < h_R^{(q)}$$

Theorem 3.7 stated below, proves the correctness of the label setting algorithm when $<_R$ and $\leq_R$ hold the following properties:

Property 1: [dominance] $p <_D q \Rightarrow p <_R q, \forall p, q \in p_{i,j}$

Property 2: [monotonic] $p \leq_D p \circ \langle j, l \rangle, \forall p \in p_{i,j}, \forall (j, l) \in A(j)$;

Where $A(j) := \{l | (j, l) \in A\}$

Theorem 3.7: Let $<_R$ and $\leq_R$ be the relations defined over $p_{i,j}$ (i, j $\in \mathcal{N}$) with an operator R, holding the dominance and monotonic properties. If $p \in X$ (the set of paths corresponding to un scanned labels) is path selected for scanning by the label setting such that $p \leq_R q, \forall q \in X$, (for X the set of arcs corresponding to temporary labels), then p is a (definitive) ND path.

Proof: (by contradiction) suppose that $p \in p_{s,i}$ is a dominated path then there is a path $w <_D p$. On the other hand, since $p \in X$, the path w only can be generated after the selection of p, for scanning. This means that w is obtained by an extension of one of the paths in X that is, $w = w_1 \circ w_2$, with $w_1 \in X$. Therefore,

$w_1 \leq_R w <_R p$ contradicting the hypothesis that $p \leq_R q, \forall q \in X$ ■

Let us illustrate the computation of **algorithm 3.1**using the small instance of BSP problem given in **figure 3.2**

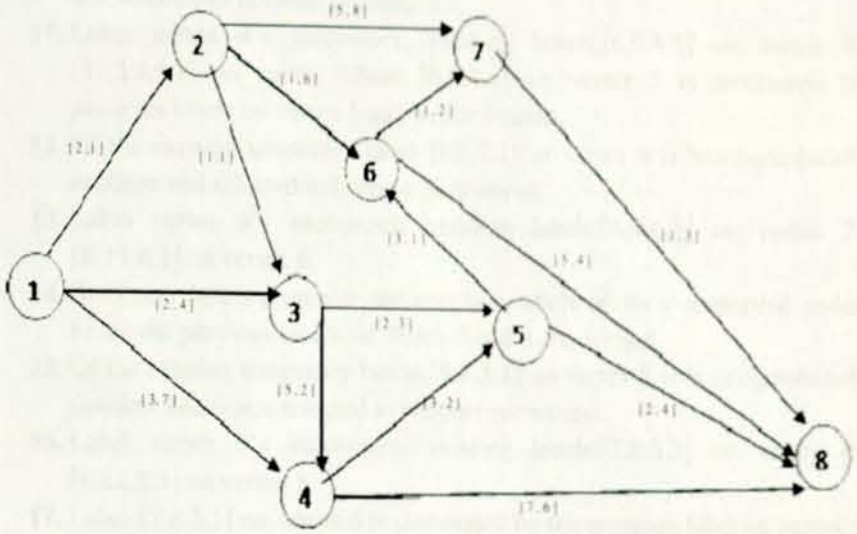


Fig3.2 Example to illustrate how to solve Bi-objective shortest path problem using the above algorithm

Solution

1. The label [0,0,∧,∧] is assigned to s, by the initialization step of the algorithm and hence the entire list of temporary label of vertex s is equal to {[0,0,∧,∧]} .
2. Due to (1) $\cup_{i \in \mathcal{N}} lt_i \neq \emptyset$ and also lexicographic min. label is l_s itself
3. l_s now become permanent and label its successors, yielding labels,[2,1,1,1] on vertex 2 ,[2,4,1,1] on vertex 3 and[3,7,1,1] on vertex 4
4. Of these labels,[2,1,1,1] on vertex 2 is lexicographically the smallest and hence selected to become permanent
5. Vertex 2's successors are in turn labeled; yielding temporary labels ,[3,2,2,1] on vertex 3, [3,7,2,1] on vertex 6, and [7,9,2,1] on vertex 7
6. Of the existing temporary labels, [2,4,1,1] on vertex 3 is lexicographically the smallest and hence selected to become permanent.
7. Label vertex 3's successors; yielding temporary labels,[4,7,3,1] on vertex 5, and [7,6,3,1] on vertex 4.
8. Of the existing temporary labels,[3,2,2,1] of vertex 3 is lexicographically the smallest and selected to become permanent.
9. Vertex 3's successors are again labeled; yielding temporary labels,[8,4,3,2] on vertex 4 and [5,5,3,2] on vertex 5.

10. Of the existing temporary labels,[3,7,1,1] on vertex 4 and [3,7,2,1] on vertex 6 are lexicographically the smallest and equal. Since for equal labels the algorithm selects in topological order of the node's number,[3,7,1,1] on vertex 4 is selected to become permanent.

11. Label vertex 4's successors; yielding labels,[6,9,4,1] on vertex 5 and [10,13,4,1] on vertex 8.here [6,9,4,1] on vertex 5 is dominated by the previous labels on vertex 5 and hence deleted.

12. Of the existing temporary label [3,7,2,1] of vertex 6 is lexicographically the smallest and selected to become permanent.

13. Label vertex 6's successors; yielding labels,[4,9,6,1] on vertex 7 and [8,11,6,1] on vertex 8.

14. Both new labels dominate the previous labels on their respective nodes and hence the previous labels on vertex 7 and 8 are deleted.

15. Of the existing temporary labels,[4,7,3,1] on vertex 5 is lexicographically the smallest and hence selected to become permanent.

16. Label vertex 5's successors; yielding labels,[7,8,5,1] on vertex 6 and [6,11,5,1] on vertex 8.

17. Label [7,8,5,1] on vertex 6 is dominated by the previous label on vertex 6 and hence deleted and the new label [6,11,5,1] on vertex 8 dominates the previous label on vertex 8, hence the previous label on vertex 8 deleted.

18. Of the existing temporary labels,[4,9,6,1] on vertex 7 is lexicographically the smallest and hence selected to become permanent.

19. Label vertex 7's successor; yielding label [5,12,7,1] on vertex 8.

20. Of the existing temporary labels,[5,5,3,2] on vertex 5 is lexicographically the smallest and hence selected to become permanent.

21. Label vertex 5's successors again ; yielding [8,6,5,2] on vertex 6 and [7,9,5,2] on vertex 8

22. Vertex 8 has no successor and hence no temporary label is created there. Therefore of all existing temporary labels [7,6,3,1] of vertex 4 is lexicographically the smallest and hence selected to become permanent.

23. Label vertex 4's successors; yielding labels [10,8,4,2] on vertex 5 and [14,12,4,2] on vertex 8, in this case both new labels are dominated by the previous labels on their respective nodes and hence deleted.

24. Of the existing temporary labels [8,4,3,2] on vertex 4 is lexicographically smallest and selected to become permanent.

25. Label vertex 4's successors again; yielding labels [11,6,4,3] on vertex 5 and [15,10,4,3] on vertex 8, in which case both are dominated by the previous labels of their respective nodes and hence deleted.

26. Of the existing temporary labels [8,6,5,2] on vertex 6 is lexicographically the smallest and selected to become permanent.

27. Label the successors of vertex 6 again; yielding [9,8,6,2] on vertex 7 and [13,10,6,2] on vertex 8.The one on vertex 8 is dominated by the previous labels and hence deleted.

28. Take the only reaming temporary label on vertex 7 [9,8,6,2] to make permanent.

29. The successor of vertex 7 is vertex 8, yielding [10,11,7,2] and this is dominated by the previous label [6,11,5,1] on vertex 8 and hence deleted.

30. Now no temporary label remains unfixed, hence to determine any of the efficient paths choose one permanent label on vertex q and by moving backwards, the first vertex of the path will be reached.

Therefore the **ND paths** from s (source node) to t (terminal node) are:

$P_1 : 1 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 8 \equiv [5,12]$,

$P_2 : 1 \rightarrow 3 \rightarrow 5 \rightarrow 8 \equiv [6,11]$,

$P_3 : 1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 8 \equiv [7,9]$.

The **Algorithm3.1**works not only for directed network but also for undirected graph. Let us see this using small instance of the following BSP problem.

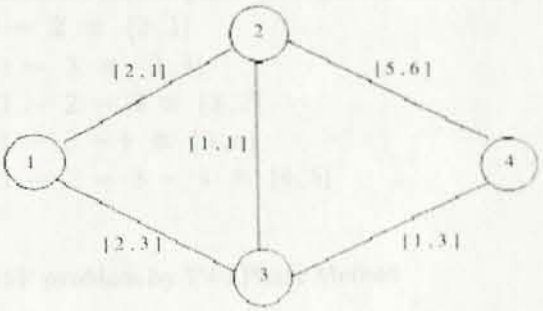


Fig 3.3 Example to illustrate Algorithm 3.1 for undirected graph

Let us find ND paths from node1to all other nodes

1. Using the initialization step take the entire list of temporary label on source node to be {[0 , 0 ,Λ ,Λ]}
2. The lexicographically minimum label to be selected to become permanent is [0 , 0 ,Λ ,Λ] on vertex 1
3. Label all reachable nodes from vertex 1;yielding labels [2 , 1 , 1 , 1] on vertex2 and [2 , 3 , 1 , 1] on vertex3

4. Of these temporary labels ,[2 , 1 ,1 , 1] on vertex 2 lexicographically smallest and hence selected to become permanent
5. Label all reachable nodes from vertex 2 ; yielding temporary labels, [3 , 2 ,2 , 1] on vertex 3 and [7 , 7 , 2 , 1] on vertex 4
6. Of the existing temporary labels, [2 , 3 , 1 , 1] on vertex 3 is lexicographically the smallest and hence selected to become permanent
7. Label all reachable nodes from vertex 3; yielding temporary labels,[3 , 4 , 3 , 1] on vertex 2 and [3 , 6 , 3 , 1] on vertex 4.
8. The label on vertex 2 is dominated by the previous label on vertex 2 and hence deleted and the previous label on vertex4 is dominated by the new label on vertex 4 and hence deleted.
9. Of the existing labels, [3 , 2 , 2 , 1] on vertex 3 is lexicographically smallest and hence selected to become permanent
10. Again label all reachable nodes from vertex 3;yielding temporary labels,[4 , 3 , 3 , 2] on vertex 2 and [4 , 5 , 3 , 2] on vertex 4
11. Of the existing temporary labels , [3 , 6 , 3 , 1] on vertex 4 is lexicographically the smallest and hence selected to become permanent
12. Labeling all reachable nodes from vertex 4, the new labels we get are dominated by the existing labels on their respective nodes and hence deleted.

Therefore the ND paths from node1 to other nodes in **Fig3.3** are:

$P_1 : 1 - 2 \equiv [2,1]$

$P_2 : 1 - 3 \equiv [2,3]$

$P_3 : 1 - 2 - 3 \equiv [3,2]$

$P_4 : 1 - 3 - 4 \equiv [3,6]$

$P_5 : 1 - 2 - 3 - 4 \equiv [4,5]$

3.4 Solving BSP problem by Two Phase Method

Let us consider the two phase method approach to solve shortest path problem in bi-criterion case of the minimum spanning tree (MST) problem. The minimum spanning tree problem is a classic and combinational optimization problem which has many direct and indirect applications .The most natural direct applications are network planning problem such as minimization of total length of pipeline or telephone wire or road to connect n-towns together. Some network design problems can be reduced to minimum spanning tree problem and hence solved in that way.

The MST problem is the following classic network optimization problem. Given an undirected, connected graph G = (V, E) with a single cost function $f : E \rightarrow \mathbb{R}^+$.

find a spanning tree T of G which minimizes $F(T) = \sum_{e \in T} f(e)$.

The bi-objective minimum spanning tree (BMST) problem is an extension of the single objective MST problem. It considers the more realistic case of more than one cost function influencing whether an edge is including in a tree. Here, without loss of generality, let us consider only integer costs. Given an un directed, connected graph G= (V , E) with two cost functions .

f_i: E $\longrightarrow$ $\mathbb{Z}^+$, $i \in \{1, 2\}$, a pair of numbers ($f_1(e)$, f_2 (e)) is the pair of cost associated with an edge e ∈ E . Let us denote the set of all spanning tree of G by ST(G). Each tree T∈ ST(G) has a pair of cost

$F_1(T) = \sum_{e \in T} f_1(e)$ And $F_2(T) = \sum_{e \in T} f_2(e)$ associated with it.

A solution T ∈ ST(G) is efficient if ($F_1(T)$, F_2 (T)) is not dominated by ($F_1(T')$, $F_2(T')$) for any $T' \in ST(G)$. The objective of the BMST problem is to find the set of efficient solutions.

A solution can be viewed as a tree T or as a cost pair (F_1(T) ,F_2 (T)) associated with it. Thus by saying that a pair of number (x, y) is an efficient solution we mean that there exist T ∈ $ST(G)$) with (F_1(T), F_2 (T)) = (x , y) and there is no $T' \in$ ST(G)

Such that ($F_1(T')$, $F_2(T')$) Dominates (x ,y) .

The following figure shows the structure of the feasible space

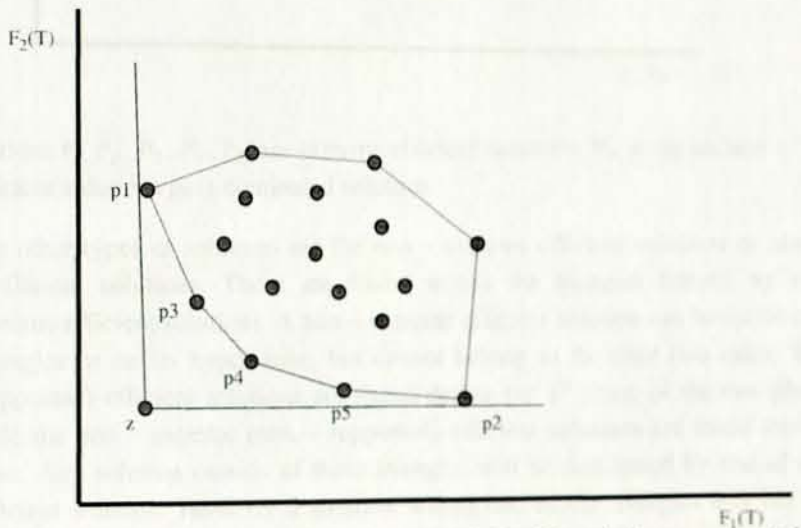


Fig 3.4. The polygonal region shows feasible space with feasible solutions represented by points. Solutions P_1, P2, P3, P4 and P5 are extreme efficient solutions and z is the ideal point.

The solutions (the cost pairs) can be viewed graphically as shown in **fig 3.4** we call the convex closure of the set of all cost pairs $(F_1(T), F_2(T))$, T∈ST (G), the feasible space F. There are two fixed points along the border of the feasible space, denoted by P_1 and P_2 in **fig 3.4** which is obtained by minimizing each criterion individually. The ideal point z is the point where both criteria are minimized. If this were a solution, then the efficient set would contain just this point as it would dominate every other solution. However, this case is unlikely to occur particularly with independent cost functions.

The solutions in the efficient set can be divided into two types. Some of the efficient solutions make a convex hull marking a boarder around the rest of the solutions, these are termed the extreme efficient solutions or supported efficient solutions. (**See fig 3.5**) below

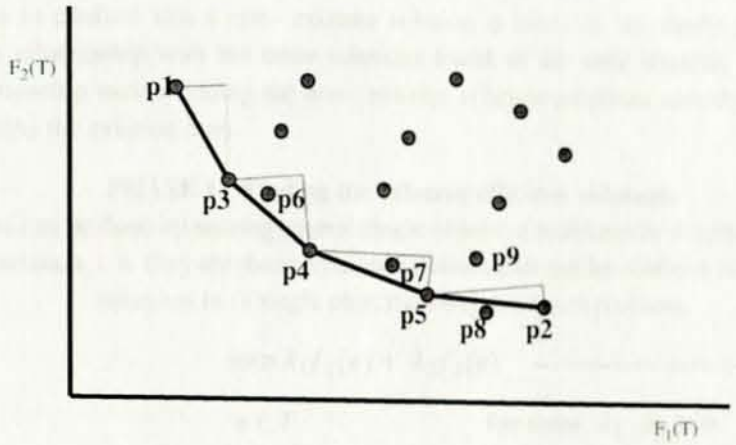


Fig 3.5 solutions P_1, P_2, P_3, P_4, P_5 are extreme efficient solutions. P_6, p_7, p_8 are non – extreme efficient solutions p_9 is dominated solution.

The other types of solutions are the non – extreme efficient solutions or non – supported efficient solutions. These are found within the triangles formed by two adjacent extreme efficient solutions. A non – extreme efficient solution can be inside one of these triangles or on its hypotenuse, but cannot belong to its other two sides. The extreme (supported) efficient solutions are found during the 1st phase of the two phase method while the non – extreme (non – supported) efficient solutions are found during the 2nd phase. Any solution outside of these triangles will be dominated by one of the extreme efficient solution. However a solution within one of the triangles will not be dominated by the extreme solutions, so it is potentially an efficient solution

In **fig (3.6)** below solution a is not dominated by either of the extreme efficient solutions P or q as it is within the triangle ,but there may be another solution b which dominates a.

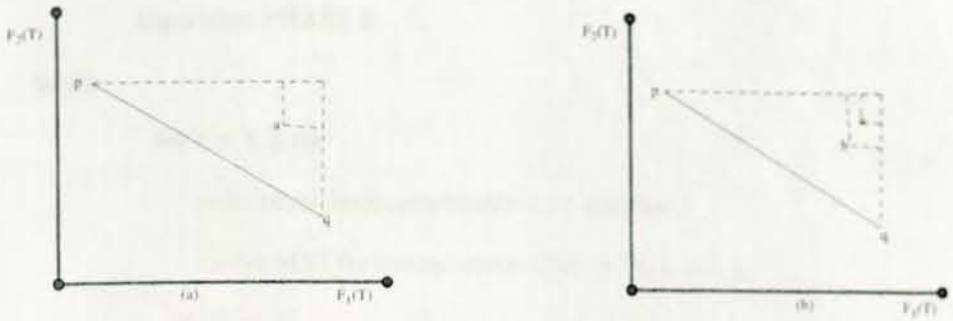


Fig3.6. Diagram (a) shows a potentially efficient solution a in the triangle defined by two consecutive extreme solutions p and q. Diagram (b) shows a point b which dominates a.

Thus to confirm that a non– extreme solution is efficient, we would first have to know its relationship with the other solutions found in the same triangle. Discovering that relationship makes finding the non– extreme efficient solutions considerably harder than finding the extreme ones.

PHASE 1 : Finding the extreme efficient solutions

This can be done by solving several single objective problems in weighted sum formulation, i. e. they are those efficient solutions that can be obtained as optimal solutions to (a single objective) weighted sum problem.

$$\min \lambda_1 f_1(e) + \lambda_2 f_2(e) \quad \text{----------3.4}$$

$$e \in T \qquad \text{for some } \lambda_1, \lambda_2 > 0$$

To find extreme efficient solutions, we use the geometric method. This method is based on a procedure which computes for a given pair of extreme efficient solutions S' and S'' another extreme efficient solution between S' and S'' (if there exist any)

The initial part of algorithm PHASE 1 finds the two solutions S_1 and S_2 obtained by solving the single objective minimum spanning tree for each criterion separately, taking the other criterion into account only to break ties .These solutions S_1, S_2 must be extreme efficient solution. If they are the same solution, then this is the unique minimum and the problem is solved (the efficient set consists of just this one cost pair). Otherwise it is called the procedure Border search (S_1, S_2)

Algorithm PHASE 1

begin

for $i = 1, 2$ **do**

order edges lexicographically w.r.t. criterion i

solve MST for this ordering- solution : $S_i = (x_i, y_i)$

if $S_1 = S_2$

then return $\{S_1\}$

else

L:= Border search (S_1, S_2)

return concatenation of $\{S_1\}, L, \{S_2\}$

end

Procedure Border search (S', S'')

begin

compute new edge costs $f_1(e)(y' - y'') + f_2(e)(x'' - x')$

order edges according to new costs

solve MST for this ordering – solution: $S = (x, y)$

if $S = S'$ *or* $S = S''$

then return empty list

else

L' :=Border search(S', S)

L'' :=Border search(S, S'')

return concatenation of $L', \{S\}, L''$

end

The recursive Border search (S', S'') computes and returns a list of the consecutive extreme solutions on the part of the convex hull of F between the extreme efficient Solution $S' = (x', y')$, and the extreme efficient solution $S'' = (x'', y''), x' < x''$. The procedure first finds the extreme efficient solutions $S = (x, y)$ which minimizes the following function.

$$\frac{y-y''}{y'-y''} - \frac{x''-x}{x''-x'} \quad \text{----------3.5}$$

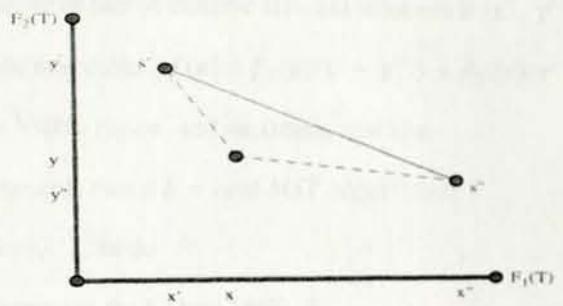


Fig 3 .7 Graphical representation of border search for phase 1.

This solution is the extreme efficient solution between S' and S''which is furthest away from the line joining S' and S'' (see **Fig 3.7**), and it is the cost pair of the minimum spanning tree with respect to the following edge costs:

$$f(e)=f_1(e)\,(y'-y'') + f_2(e)(x''-x') \text{ ---------- (3.6)}$$

If the computed solution S is the same as one of the solutions S' *or* S'' then there is no extreme efficient solution between S' and S'' . Otherwise, solution S is our next extreme efficient solution and the computation continues by calling recursively border search (S' , S) and Border search (S ,S''). The list of the computed extreme solutions consists of the extreme efficient solutions returned by the first recursive call, followed by S, and followed by the extreme efficient solutions returned by the second recursive call.

PHASE 2: Finding the non – extreme efficient solutions

Each non- extreme efficient solutions lies within a triangle, an initial viable region, formed by two adjacent extreme efficient solutions as shown in **fig 3.5 (a)**. A solution point a found with in this region is not dominated by the extreme points, and (whether itself is efficient or not) dominates any solution found in the rectangle formed in

fig 3.6(a). Having such a solution a, we can redefine the viable region by excluding that rectangle. However point a is not necessarily efficient as there may be potentially another solution point within the region that might dominate it. (See **fig 3.6(b)**). Algorithm phase 2 searches for non extreme efficient solutions separately in each triangle formed by two consecutive extreme efficient solutions. The search in one triangle starts from its hypotenuse and proceeds towards its right angle, find
ing solutions in the increasing order of their distances to the hypotenuse. Each time a new solution is found in the viable regions, the viable region is modified.

Algorithm PHASE 2

begin

for each consecutive pair of extreme efficient solutions $\boldsymbol{p}(\boldsymbol{x}', \boldsymbol{y}')$, $\boldsymbol{q}(\boldsymbol{x}'', \boldsymbol{y}'')$ **do**

compute new costs $\boldsymbol{f}(\boldsymbol{e}) = \boldsymbol{f}_1(\boldsymbol{e})(\boldsymbol{y}' - \boldsymbol{y}'') + \boldsymbol{f}_2(\boldsymbol{e})(\boldsymbol{x}'' - \boldsymbol{x}')$

define Viable region and maximum cost line

$\{comment: run\ a\ k - best\ MST\ algorithm\}$

for k=1,….., ∞ do

determine the k_{th} best MST T

if no more solution then break

else

let $S = (x, y)$ be the cost pair of T

if S within the viable region then

add S to list of non – extreme efficient solutions

update the viable region and maximum cost line according to S.

else if S on or past maximum cost line then break.

end- for

end-for

end

Since the search proceeds systematically from the hypotenuse, each time a solution is found in the current viable region, it must be efficient: none of the previously found

solution is dominate it, and none of the solution found subsequently could dominate it. The search is performed using a k- best minimum spanning tree algorithm, which computes for the single cost minimum spanning tree problem the first k spanning trees in a non decreasing order of their costs.

The search in a triangle supported by two extreme efficient solutions $P = (x', y')$ *and* $q = (x'', y'')$ starts by setting the edge cost $f(e)$ according to **equation 3.6** .The main part of the computation is an application of a k- best MST algorithm to single objective problem defined by the edge cost $f(e)$. The minimum spanning trees with respect to these edge costs are the spanning trees T with cost pairs $(F_1(T), F_2(T))$ lying on the line joining P and q. Thus these trees will be the first trees found by the "k- best" algorithm. For each new spanning tree found, we check if its cost pair (x, y) belongs to the (current) viable region .This check is done by a simple linear search through the list of the consecutive corner points defining the viable regions. If (x, y) belongs to the viable region, then it is the next non –extreme efficient solution found and the viable region is updated.

This process carries on until we either run out of solutions or reach the maximum cost line: the line parallel to the $P - q$ line passing through the corner point of the current viable region which is furthest away from the P - q line. **Fig 3-8** shows the maximum cost line at the beginning of the computation and after the first update of the viable region.

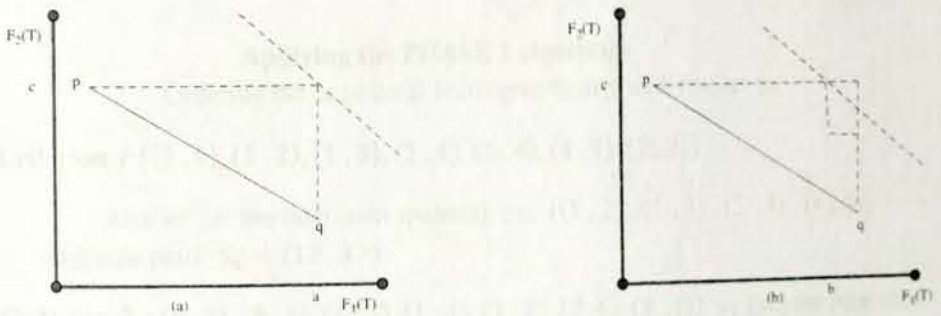


Fig 3-8 Diagram (a) shows the initial maximum cost line through (a, c) diagram (b) shows the new maximum cost through (b, c) after the viable region has been changed.

Let us illustrate the computation of algorithms PHASE 1 and **PHASE 2** using the small instance of the BMST problem given in **fig 3.9**

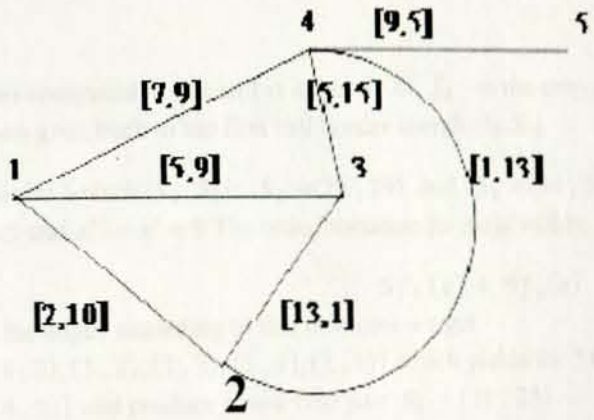


Fig 3-9 example instance of BMST problem

Applying the PHASE 1 algorithm

Ordering the edge costs lexicographically with respect to:

Criterion 1:$\{(2,4),(1,2),(1,3),(3,4),(1,4),(4,5),(2,3)\}$

And we get the minimum spanning tree $\{(1,2),(1,3),(2,4),(4,5)\}$ with cost pairs $S_1 = (17,37)$

Criterion 2: $\{(2,3),(4,5),(1,3),(1,4),(1,2),(2,4),(3,4)\}$ we find the minimum spanning tree $\{(1,3),(1,4),(2,3),(4,5)\}$ with cost Paris of $S_2 = (34,24)$.

$$S_1 \neq S_2$$

Border search (S_1, S_2) which will return a list of all extreme efficient solutions

Since $y' - y'' = 37\text{-}24 = 13$ and $x'' - x' = 34\text{-}17 = 17$

The transformation formula for this call is

$$13f_1(e) + 17f_2(e) \quad and$$

ordering the edges according to new cost we get
$\{(2,4),(1,2),(1,3),(4,5),(2,3),(1,4)\}$ and the MST for this order is the tree
$\{(1,2),(2,3),(2,4),(4,5)\}$ with cost pair $S_3 = (25, 29)$

Since $S_3 \neq S_1, S_2$ we call recursively border search(S_1, S_3) and $y' - y'' = 8$
and $x'' - x' = 8$, The transformation formula will be

$$8f_1(e) + 8f_2(e)$$

the solution computed in this call is either S_1 or S_2 so the empty list is returned and the computation goes back to the first call border search (S_1, S_2).

To call Border Search (S_3, S_2); $S_3 = (25, 29)$ and $S_2 = (34, 24)$
$y' - y'' = 5$ and $x'' - x' = 9$ The transformation formula will be

$$\mathbf{5}f_1(e) + \mathbf{9}f_2(e)$$

Ordering the edges according to this new cost we get
$\{(2,3),(4,5),(1,2),(1,3),(1,4),(2,4)\}$ which yields the MST tree $\{(1,2),(1,4),(2,3),(4,5)\}$ and produce a new cost pair $S_4 = (31, 25)$.

$$S_4 \neq S_3, S_2$$

Then consider recursive borders search ; $Border\ search(S_3, S_4)$ and Border search(S_4, S_2) .
To call Border search(S_3, S_4); $S_3 = (25, 29)$ and $S_4 = (31, 25)$. The transformation formula will be

$$4f_1(e) + 6f_2(e)$$

Ordering edges according to this new cost we get $\{(2,3),(4,5),(1,2),(2,4),(1,4)\}$ which yields the MST$\{(2,3),(4,5),(1,2),(2,4)\}$ and now $S = (25, 29) = S_3$
Therefore there does not exist efficient solution between S_3 and S_4 so return empty list.
Consider again recursive border search(S_4, S_2); for $S_4 = (31, 25)$
and $S_2 = (34, 24)$.The transformation formula will be $f_1(e) + 3f_2(e)$ .Ordering edges according to the new cost we get $\{(2,3),(4,5),(1,2),(1,4)\}$ in which $S = (31, 25) = S_4$. Therefore there does not exist efficient solution between S_4 and S_2

Thus the lists of solutions found in Phase 1 are;

$S_1 = (17, 37)$ $S_3 = (25, 29)$
$S_4 = (31, 25)$ $S_2 = (34, 24)$

And these points are shown in the **fig 3.10**

Applying PHASE 2 algorithm

The first pair of consecutive extreme solutions considered by algorithm PHASE 2 are S_1 and S_3 . The costs of edges are set of $8f_1(e) + 8f_2(e)$, the initial viable region is the

triangle defined by points (17 ,37) , (25 , 29) , (25,37) and the initial maximum cost line passes through (25 , 37) and has gradient -1. The sequence of solutions produced by the K- best algorithm is given in the following table.

K	TREE	Single cost	cost pair	comment
1.	{(1,2),(1,3),(2,4), (4,5)}	432	(17,37)	extreme efficient solution , S_1
2.	{(1,2),(2,3),(2,4), (4,5)}	432	(25,29)	extreme efficient solution ,S_3
3.	{(1,3),(2,3),(2,4), (4,5)}	448	(28,28)	not within the viable region
4.	{(1,2),(2,3),(1,4), (4,5)}	448	(31,25)	not within the viable region
5.	{(1,2),(1,3),(1,4), (4,5)}	448	(23,33)	non extreme efficient solution; a
6.	{(1,3),(1,4),(2,3), (2,3)(4,5)}	464	(34,24)	not within the viable region
7.	{(1,4),(2,3),(2,4), (4,5)}	464	(30,28)	not within the viable region
8.	{(1,3),(1,4),(2,4), (4,5)}	464	(22,36)	non extreme efficient solution; b
9.	{(1,2),(2,3),(2,4), (4,5)}	480	(29,31)	not within the viable region

The last solution found (29, 31) is beyond the maximum last line and so stop this part of the algorithm. Thus the solutions found in phase 2 for first part is

(23, 33), (22, 36)

We can see that the solution (28, 28) encountered twice, in the first part when the extreme solutions S_1 and S_3 is considered. But it is ignored since it is outside of the viable region and second time when the pair of extreme solutions S_3 and S_4 is considered. Now it is in the viable region and hence a non-extreme efficient solution. We have no solution in the third part. Thus the solutions found in phase 2 are

(23, 33), (22, 36), (28, 28)

In our example, the k- best algorithm applied to the triangles based on the segment $[S_1, S_3]$ discovers all efficient solutions in the triangle based on the segments $[S_3, S_4]$ as well as those in the triangle based on the segment $[S_4, S_2]$. As the cost line of the first triangle sweeps over the viable region of this triangle, it also sweeps over the entire viable regions of the other two triangles (see **fig3.10**). Not that if only some but not all efficient solutions in a sub sequent triangle are discovered, then we would have to apply the *k* best algorithm to that subsequent triangle, re- discovering some efficient solutions which are already known.

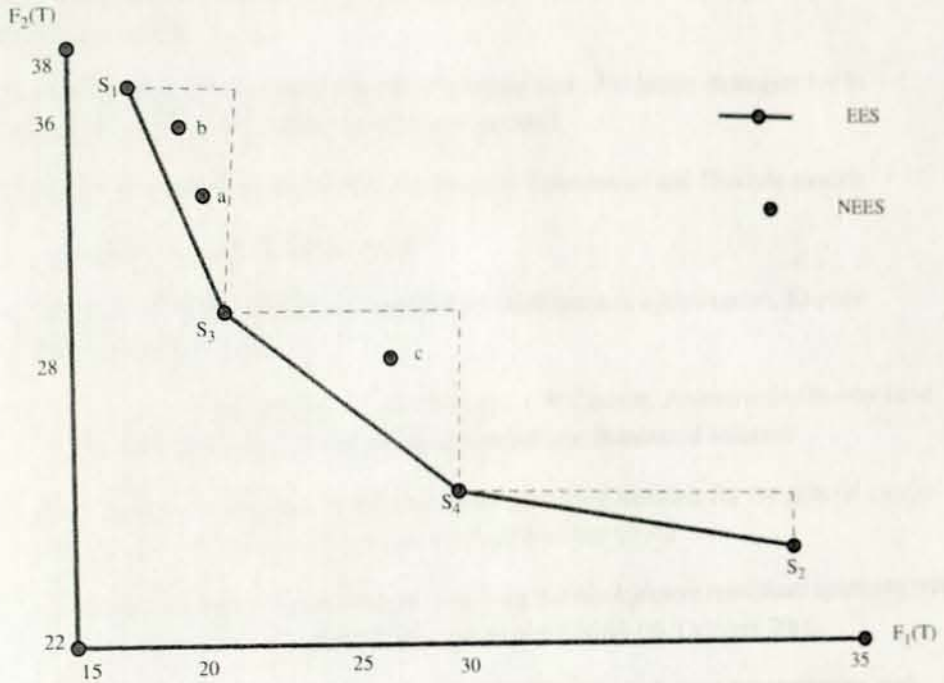


Fig 3-10 Graph of solutions for the problem

Hence for this example the efficient solution is

$$\{(17, 37), (25, 29), (31, 25), (34, 24), (23, 33), (22, 36), (28, 28)\}$$

References

[1] Matthias Ehrgott: Multicriteria optimization second edition, Springer Berlin, Heidelberg 2005

[2] Matthias Ehrgott and Andra Raith: A comparison of solution strategies for bi-objective shortest path problems, February 16, 2007.

[3] Dimitri P .Bertsekas: Network optimization: Continuous and Discrete models

Athena scientific, Belmont 1998

[4] Matthias Ehrgott and Xavier Gandibleux: Multicriteria optimization, Kluwer academic publisher 2002

[5] J.M. Coutinho Rodrigues, J.C.N. Climaco, J. R.Current; Aninteractive bi-objective shortest path approach searching for unsupported non dominated solution.

[6] Jose Manuel Paixao and Jose Luis Santos -Labeling methods for the general case of multi-objective shortest path problem, preprint Number 07-42

[7] Sarah Steiner and Tomasz Radzik – Solving the bi-objective minimum spanning tree problem using k-best algorithm. Technical Report TR-03-06, October 2003.

[8] Mokhtar S.Bazara, John J. Jarvis and Hanif D. sherali: Linear programming and Network flows second edition, John Wiley & sons, Inc, 1990