AN ALTERNATIVE APPROACH ON MINIMIZATION PROBLEM BY GRAPH THEORY

By

REHANA SULTANA BIPASHA

Student No. 100009003 P

Registration No. 0010243, Session: October-2000

MASTER OF PHILOSOPHY
IN
MATHEMATICS

Department of Mathematics

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA-1000

SEPTEMBER - 2006

The thesis titled

AN ALTERNATIVE APPROACH ON MINIMIZATION PROBLEM BY GRAPH THEORY

Submitted by

Rehana Sultana Bipasha

Roll No 100009003P, Registration No.0010243, Session: October-2000, a part-time M. Phil. student in Mathematics has been accepted as satisfactory in partial fulfillment for the degree of

MASTER OF PHILOSOPHY

in Mathematics

on September 19, 2006

Contents:

Abstract v

Declaration vi

Acknowledgement vii

Chapter-1

Introduction 01-17

Chapter-2

Linear And Nonlinear Programming on Graph 18-27

Chapter-3

Path Algorithms And Spanning Trees 28-46

Chapter-4

Shortest Paths Algorithm respect to Non-linear Complementarity Problem 47-61

References 62-64

Abstract

A randomized parallel linked residual network $G_f = (V, E_f)$ is presented. Algorithmically the residual network, which produces successive shortest path distances and the original graph $G = (V, E)$ is solved. This result is optimum with respect to both addition of flow and transferring of flow in path flow of the residual network. Depth First Search (DFS) techniques can calculate both the shortest path distances and time stamps. Thus non-linear complementarity problem can be reformulated as a shortest path distances.

Candidate's Declaration

I hereby declare that the work which is being presented in this dissertation entitled "An Alternative Approach on Minimization Problem by Graph Theory" submitted in partial fulfillment of the requirements for the award of the degree of Master of Philosophy in Mathematics in the Department of Mathematics, Bangladesh University of Engineering and technology (BUET), Dhaka-1000 is an authentic record of my own work.

It has not been submitted elsewhere (Universities or Institutions) for the award of any other degree.

RSultana
(Rehana Sultana Bipasha)

Acknowledgement

I am very much indebted to my supervisor Dr. Md. Elias, Associate Professor, Department of Mathematics, Bangladesh University of Engineering and Technology, Dhaka for suggesting me the research topic. I would also like to extend my heartiest gratitude and thanks to him for his continuous encouragement and valuable suggestions during the period of this research work of my M. Phil program.

I express my deep regards to my respectable teacher Professor Dr. Md. Mustafa Kamal Chowdhury, Head of the Department of Mathematics, Bangladesh University of Engineering and Technology, Dhaka for providing me with all necessary research facilities and helpful advice during my study in this department.

I also express my gratitude to my teacher Mr. Md. A. K. Hazra, Mr. Md. Obayedullah and Dr. Md. Abdul Maleque of the Department of Mathematics, Bangladesh University of Engineering & Technology for their kind cooperation and help during my M. Phil program.

Finally, I would like to express my special thanks to my husband Dr. Md. Abu Naim Sheikh, son Sheikh Ayman Shouki and daughter Sheikh Mridnla Koyshi for their constant encouragement, patience and moral support, which makes this work possible.

Chapter One

Introduction

1.1 Introduction

The problem of computing shortest paths is indisputably one of the most well studied problems in graph theory. Saunders et al. [18] introduced a faster algorithm for the shortest path problem and showed their various applications in graph theory. It is thoroughly surprising that in the setting of real-weighted graphs, many optimization problems have seen a great progress since the early work by Bellman-Ford, Floyd and others. Blasum et al. [3] introduced a network flow technique of optimization problem and solved it using shortest path distances. Korilis et al. [8] proposed that if flows sent from source vertex to the destination by shortest distances it would be efficient and optimum.

There are a lot of techniques to solve minimization problem using shortest distances. The shortest path problems, mainly, work between pairs of vertices in a weighted graph for the exploration of distances, time delay, cost and others that may be optimized. Christos et al. [5] have discussed the simplex method and the ellipsoid algorithm as well as other algorithms related to non-linear programming.

It is of our great interest to determine the shortest distance for a gradient based approach and it cannot yield the shortest distance only besides giving the time delay for general, real weighted inputs. Bertsekas [2] measured the path flow with two quantities.

a) The total cost with time delay and
b) Individually estimate each edges cost with respect to the minimal path flow i.e. the single source shortest path will minimize the total flow cost.

From this idea it can be modelled a residual network $G_f = (V, E')$ which is induced by flow 'f' in the NCP constraint graph $G = (V, E)$. The residual network is an optimization problem in view of non linear complementarity problem and defined as follows:

$$G_f = (V, E') .$$

Where

$E' = \{(u,v) \in V \times V; c_f(u,v) > 0\}$ and c_f is residual capacity,

$c_f = c(u,v) - f(u,v)$,

$c(u,v) > f(u,v)$,

and $f(v,u) > 0$

and G_f is a directed weighted graph in order to find the shortest path from the source s to the destination t. This network is modeled by weights with the addition of flow to any links and transferring flow from vertex to another vertex. For a comprehensive discussion of such problems we refer to Caccetta et al. [4] and Ahuja et al. [1].

The residual network, which is capable of executing almost parallel edges algorithms and deducted the shortest path in terms of algorithm defined in equation (4.2), is proposed. The interconnection between the vertices in one direction connected only to vertices in the previous one and the next vertex.

A minimum cost flow is found as they converge and solves the problem in equation (4.2), there after also solves the problem in equation (4.1) and this shortest path is derivative free and actually act as a merit function.

1.2 An overview

There are four chapters in this dissertation. In the first chapter, some definitions in graph theory have been given. In the second chapter, some optimization problems in linear and non-linear programming and a graphical model of some optimization problem have been discussed. In the third chapter, some search algorithms from finding distance from one vertex to all other vertex in the graph have been considered. Especially it has been found decomposition with depth first search method with examples. Also it has been solved the traveling salesman problem using closest insertion algorithm. In the final chapter, a residual network problem, which is a new approach for finding a minimum cost flow has been considered.

1.3 The Definitions

Some definitions and examples in graph theories that have been discussed for our study.

Definition 1.3.1 (Graph) :

A graph G consists of a pair $G = (V(G), E(G))$, where $V(G)$ is a non-empty finite set whose elements are called nodes or vertices and $E(G)$ is a set of unordered pairs of distinct elements of $V(G)$. The vertices $v_i, v_j \in V(G)$ associated with edge $e_{i,j} \in E(G)$ are called the end vertices of $e_{i,j}$.

Observe that this definition permits an edge to be associated with a vertex pair (v_i, v_j).

Definition 1.3.2 (Directed graph) :

A directed graph G is a pair (V, E), where V is a finite set and E is an associated relation of V. The set V is called the vertex set of G and its elements are called vertices. The set E is called the edge set of G and its elements are called edges. Figure 1.1(a) is a pictorial representation of a directed graph on the vertex set {1, 2, 3, 4}.

Definition 1.3.3 (Undirected graph) :

In an undirected graph $G = (V, E)$, the edge set E consists of unordered pairs of vertices. rather than ordered pairs. That is, an edge is a set (u, v), where u, $v \in V$ and $u \neq v$. (u, v) and (v, u) are considered to be the same edge in the undirected graph. Figure 1.1(b) is an undirected graph on the vertex set {1, 2, 3, 4}.

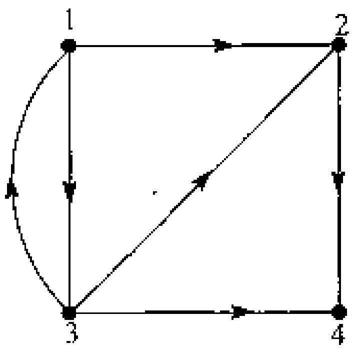


Figure 1.1(a). Directed graph

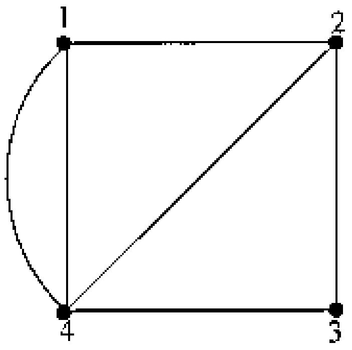


Figure 1.1(b). Undirected graph

Definition 1.3.4 (Walk) :

A walk in a graph G is a finite sequence $W = v_0 e_1 v_1 e_2 v_2 \cdots v_{k-1} e_k v_k$ whose terms are alternatively vertices and edges such that, for $1 \le i \le k$ the edge e_i has ends v_{i-1} and v_i. Thus each edge e_i is immediately preceded and succeeded by the two vertices with which it is incident. We say that the above W is a $v_0 - v_k$ walk or a walk from v_0 to v_k. The vertex v_0 is called the origin of the walk W, while v_k is called the terminus of W.

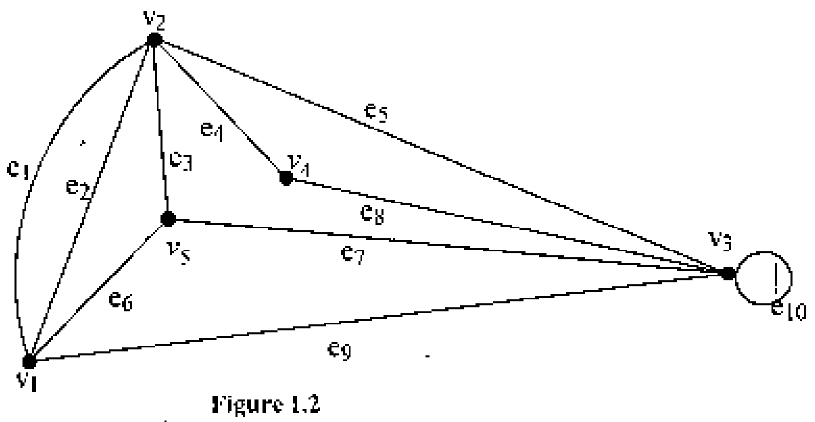


Figure 1.2

In figure 1.2, $W_1 = v_1 e_1 v_2 e_5 v_3 e_{10} v_3 e_5 v_2 e_3 v_5$ and $W_2 = v_1 e_1 v_2 e_1 v_1 e_1 v_2$ are both walks, of length 5 and 3 respectively, from v_1 to v_5 and from v_1 to v_2 respectively.

<u>Definition 1.3.5 (Trail) :</u>

If the edges $e_1, e_2, \cdots, e_k$ of the walk $W = v_0 e_1 v_1 e_2 v_2 \cdots e_k v_k$ are distinct then W is called a trail.

<u>Definition 1.3.6 (Complete Graph) :</u>

A simple graph G in which each pair of distinct vertices is joined by an edge is called a complete graph of G. Thus, a graph G with p vertices is complete if it has as many edges as possible provided that there are no loops and no parallel edges. If a complete graph G has p vertices $v_1, v_2, \cdots, v_p$, then

$$G = \{(v_i, v_j) : v_i \neq v_j ; i, j = 1, 2, \cdots, p\}.$$

The complete graph of n vertices is denoted by K_n.

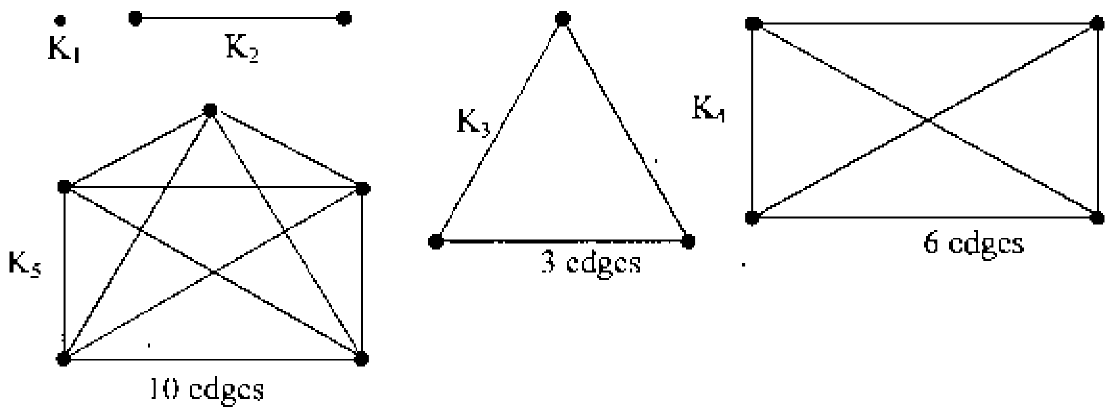


Figure 1.3 The complete graphs on at most 5 vertices

<u>Definition 1.3.7 (Null Graph)</u> :

A graph of order n and size zero is called a null graph of totally disconnected graph, and is denoted by $\overline{K_n}$. Thus $E(\overline{K_n}) = \phi$.

The following are the examples of null graph up to the order five:

(i) $\overline{K_1}$: (ii) $\overline{K_2}$: (iii) $\overline{K_3}$: (iv) $\overline{K_4}$: (v) $\overline{K_5}$:

Every vertex of a null graph is an isolated vertex. Further a graph of order n is a null graph if and only if it is a regular graph of regularity zero.

Definition 1.3.8 (Paths and lengths) :

A path of length k from a vertex u to a vertex u' in a graph $G = (V, E)$ is a sequence of vertices such that $u = v_0, u' = v_k$ and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \cdots, k$. The length of the path is the number of edges in the path. The path contains the vertices $v_0, v_1, \cdots, v_k$ and the edges $(v_0, v_1), (v_1, v_2), \cdots, (v_{k-1}, v_k)$. If there is a path p from u to u', we say that u' is reachable from u via p, which we sometimes write as $u \xrightarrow{p} u'$ if G is directed.

Definition 1.3.9 (Simple path) :

A path is simple if all vertices in the path are distinct. In figure 1.4, the path <1, 2, 3, 4> is a simple path of length 3.

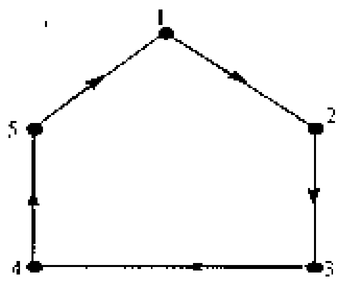


Figure 1.4

Definition 1.3.10 (Cycle) :

In a directed graph, a path $< v_0, v_1, \cdots, v_k >$ forms a cycle if $v_0 = v_k$ and the path contains at least one edge. In an undirected graph, a path $< v_0, v_1, \cdots\cdots, v_k >$ forms a cycle if $v_0 = v_k$ and $v_1, v_2, \cdots\cdots, v_k$ are distinct. For example in figure 1.5, the path <1 2 3 1> is a cycle.

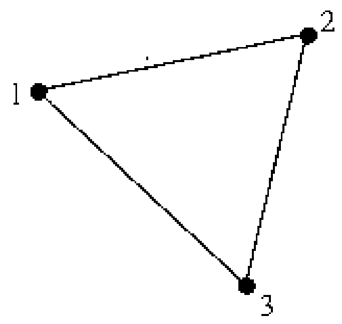


Figure 1.5

Definition 1.3.11 (Connected graph) :

An undirected graph is connected if a path connects every pair of vertices. The connected components of a graph are the equivalence classes of vertices under the "is reachable from" relation. The graph in figure 1.5 has connected components $\{1, 2, 3\}$.

Definition 1.3.12 (Strongly connected graph) :

A directed graph is strongly connected if every two vertices are reachable from each other. The graph in figure 1.6 has a strongly connected component $\{1, 2, 3, 4\}$.

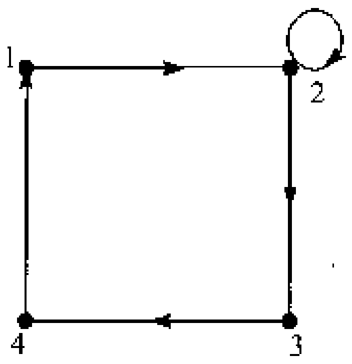


Figure 1.6

Definition 1.3.13 (Sub graph) :

A graph $G' = (V', E')$ is a sub graph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Given a set $V' \subseteq V$, the sub graph of G induced by V' is the graph $G' = (V', E')$ where

$$E' = \{(u, v) \in E : u, v \in V'\}.$$

The sub graph induced by the vertex set {1, 2, 4, 5} in figure 1.7(a) appears in figure 1.7(b) and has the edge set $\{(1,2),(2,4),(2,5),(1,5)\}$.

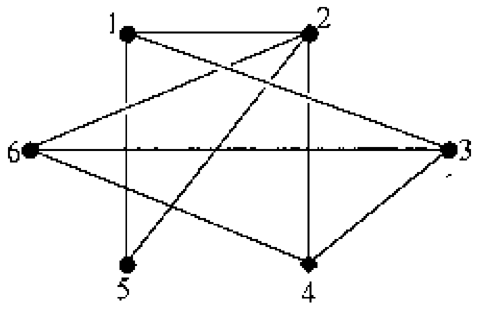


Figure 1.7(a)

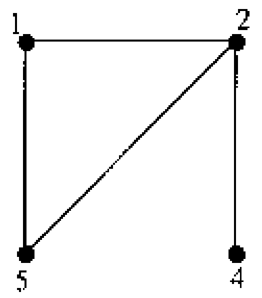


Figure 1.7(b)

Definition 1.3.14 (Parallel edges) :

Let G be a graph. If two (or more) edges of G have the same end vertices then these edges are called parallel. In the figure 1.8 the edges e_1 and e_2 are parallel.

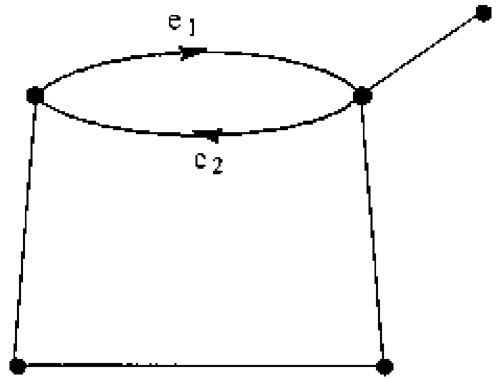


Figure 1.8

Definition 1.3.15 (Subgraph obtained by the removal of the vertex v_i):

Let G = (V, E) be a graph. Let $v_i \in V$. The subgraph of G obtained by removing the point and vertex v_i and all the lines incident with v_i is called the subgraph obtained by the removal of the vertex v_i and is denoted by $G - v_i$.

Thus if $G - v_i = (V_i, E_i)$, then $V_i = v - \{v_i\}$ and $E_i = \{e / e \in E$ and v_i is not incident with $e\}$, and $G - v_i$ is an induced subgraph of G.

Definition 1.3.16 (Cnmplement of a graph):

The complement $\overline{G}$ of a graph G is the graph with vertex set $V(G)$ such that any two vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G, G is said to be self complementary graph if $\overline{G}$ is isomorphic to G.

Definition 1.3.17 (The incidence matrix) :

The incidence matrix of a directed graph G = (V, E) is a $|V| \times |E|$ matrix $B = (b_{i,j})$ such that

$$b_{i\,j} = \begin{cases} -1 \text{ if edge } j \text{ leaves vertex } i \\ 1 \quad \text{if edge } j \text{ enters vertex } i \\ 0 \quad \text{otherwise} \end{cases}$$

Definition 1.3.18(Transpose of a graph) :

The transpose of a directed graph $G = (V, E)$ is the graph $G^t = (V, E^t)$ where $E^T = \{(v, u) \in V \times V : (u, v) \in E\}$. Thus, G^t is G with all its edges reversed.

Definition 1.3.19 (Neighborhood Sets) :

Two vertices joined by an edge are said to be adjacent or neighbors. The set of all neighbors of a flow vertex u of a graph G is called the neighborhood set of u and is defined by $N(u)$. The open neighborhood of u is

$$N(u) = \{v \in V : u.v \in E\}$$

and the closed neighborhood of u is

$$N[u] = \{u\} \cup N(u) .$$

Definition 1.3.20 (Adjaceocy -matrix representation) :

For the adjacency-matrix representation of a graph $G = (V, E)$. we assume that the vertices are numbered 1, 2, ... $|V|$ in some arbitrary manner. The adjacency-matrix representation of a graph G then consists of $|V| \times |V|$ matrix $A = (a_{i,j})$ such that

$$a_{i\,j} = \begin{cases} 1 & \text{if } (i,\, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Definition 1.3.21 (Single connected graph) :

A directed graph $G = (V, E)$ is singly connected if $u \longrightarrow v$ implies that there is at most one simple path from u to v for all vertices $u, v \in V$.

Definition 1.3.22 (Fusioo) :

Let u and v be distinct vertices of a graph G. We can construct a new graph G_1 by fusing (or identifying) the two vertices, namely by replacing them by a single new vertex x such that every edge that was incident with either u or v in G is now incident with x, *i e.,* the end u and the end v become end x.

Thus the new graph G_1 has one less vertex than G but the same number of edges as G and the degree of the new vertex x is the sum of the degrees of u and v.

We illustrate the process in figure 1.9 and 1.10

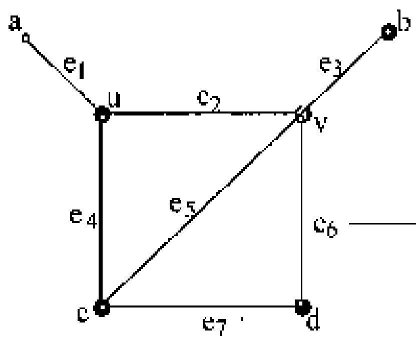


Figure 1.9

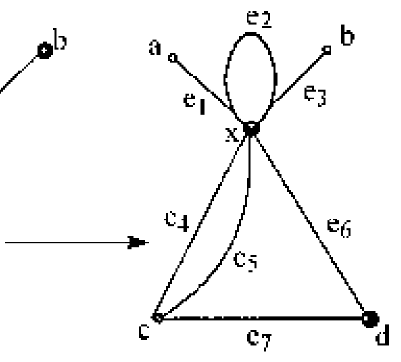


Figure 1.10

Definition 1.3.23 (Acyclic graph) :

A graph G is called acyclic if it has no cycle.

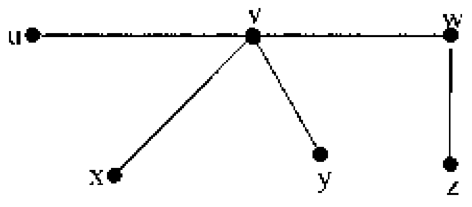


Figure-1.11

Definition 1.3.24(Tree of graph) :

Let G be a graph. If G is a connected acyclic graph. then it is called a tree.

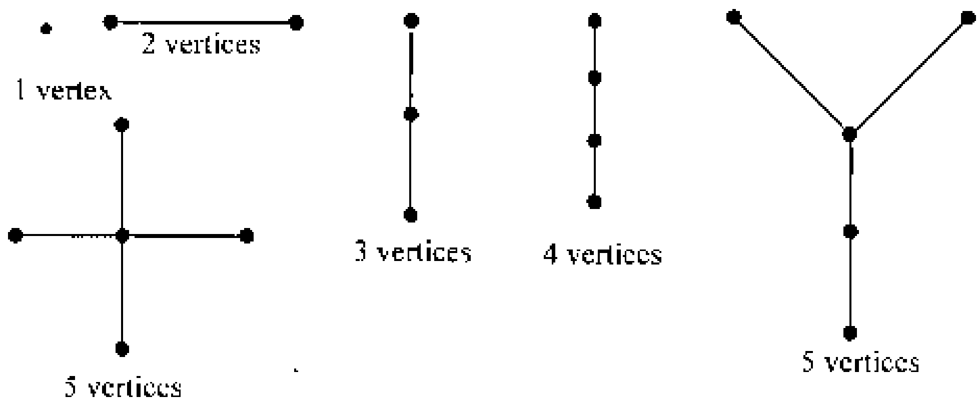


Figure 1.12: Tree with at most five vertices.

Definition 1.3.25 (Root) :

A distinguished vertex and for a branching the vertex from which every other is reachable is called the root of graph G. In the figure 1.13, s is the root of the tree.

Definition 1.3.26 (Network) :

A network N is a weakly connected simple digraph in which every link l of N has been assigned a nonnegative integer $c(l)$, called the capacity of l.
For a general network, a vertex s is called a source if it has in degree 0 while a vertex t of N is called a sink if it has out degree 0. Any other vertex of N is called an intermediate vertex.

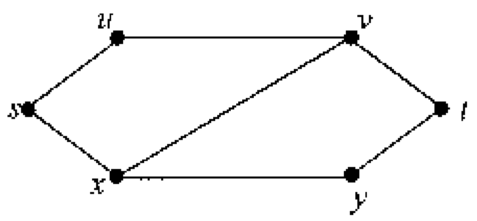


Figure 1.13. A Network

Definition 1.3.27 (Flow) :

A flow in a network N from the source s to the sink t is a function f which assigns a non-negative integer to each of the arcs l in N such that

1. (Capacity constraint) $f(l) \leq c(l)$ for each arc l,
2. The total flow into the sink t equals the total flow out of the source s and
3. (Flow conservation) for any intermediate vertex x, the total flow into x equals the total flow out of x.

Definition 1.3.28 (Saturated and Unsaturated Flow) :

The walk W is said to be f saturated if $i(W) = 0$. i denote the increment of flow, and if $i(W) > 0$, then the flow is called unsaturated flow.

Definition 1.3.29 (Labeling) :

Assignment of integers to vertices is called labeling of a graph.

Definition 1.3.30 (Arc) :

Directed edge (ordered pair of vertices) is called an arc.

Definition 1.3.31 (Convex set of graph) : The flows in a network form a convex set so that if f_1 and f_2 are flows, then so is $f_1 + (1-\alpha) f_2$ for all α in the range $0 \le \alpha \le 1$.

Definition 1.3.32 (Shortest-Paths Tree) :

A shortest -paths tree rooted at s is a directed subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$ such that

1. V' is the set of vertices reachable from s in G.
2. G' forms a rooted tree with roots and
3. for all $v \in V'$ the unique simple path from s to v in G' is a shortest path from s to v in G.

Definition 1.3.33 (Shortest Path Weight) :

The shortest path weight from u to v is defined by

$$\delta(u,v) = \begin{cases} \min\{w(p): \quad u \xrightarrow{P} v & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

Definition 1.3.34 (Ancestor) :

In a rooted tree, a vertex u along the path to the root is called an ancestor.

Definition 1.3.35 (Cost) :

The name of the objective function for many weighted minimization problem is called cost function.

Definition 1.3.36 (Connectivity of a graph) :

The connectivity k of a graph G is the minimum number of vertices whose removal results is disconnected or trivial graph. A graph G is said to be n connected if $k \geq n$. Two points u and v of a graph are said to be connected if there exists a $u-v$ path in G.

Definition 1.3.37 (Cut point and bridge) :

A cut point of a graph G is a point whose removal increases the number of components. A bridge of a graph G is a line whose removal increases the number of components.

Definition 1.3.38 (Minimal spanning tree) :

A spanning tree with the smallest weight in a weighted graph is called shortest spanning tree or shortest distance spanning tree or minimal spanning tree.

Definition 1.3.39(Cut-Sets) :

In a connected graph G, a cut-set is a set of edges whose removal from G leaves G disconnected, provided removal of no proper subset of these edges disconnects G. For instance, in fig 1.14 the set of edges $\{a, c, d, f\}$ is a cut - set.

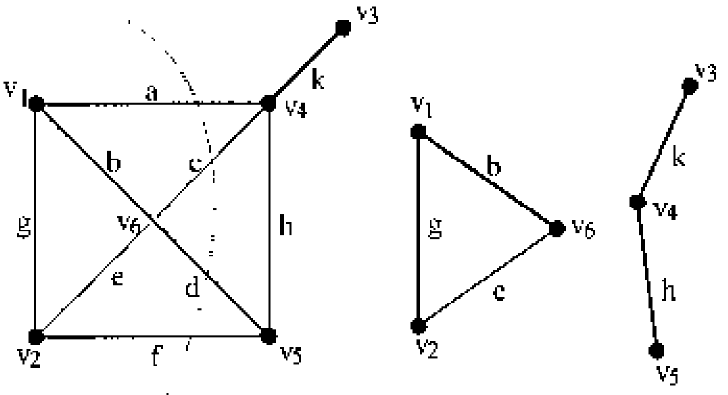


Figure 1.14

Definition 1.3.40 (Indegree and outdegree) :

Let v be a vertex in the digraph D. The indegree $id(v)$ or $IN(V)$ of v is the number of arcs of D that have v as its head, i.e., the number of arcs that "go to" v.

Similarly, the outdegree $od(v)$ or $OUT(v)$ of v is the number of arcs of D that have v as its tail i.e., that "go out" of v.

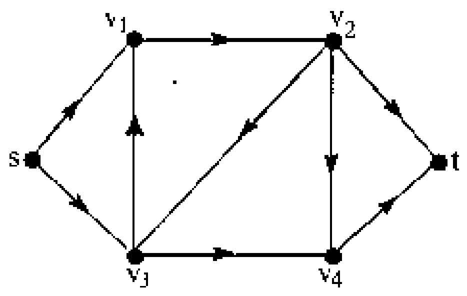


Figure 1.15

Thus in the digraph of figure 1.15, we have

$$id(s) = 0, id(v_1) = 2, id(v_2) = 1$$

$$id(v_3) = 2, id(v_4) = 2, id(t) = 2\,,$$

where

$$od(s) = 2, od(v_1) = 1, od(v_2) = 3$$
$$od(v_3) = 2, od(v_4) = 1, od(t) = 0$$

Definition 1.3.41 (Separating Sets) :

Let u and v be two distinct vertices of a graph G. A set S of vertices of G, containing neither u nor v, is said to be u–v separating if the vertex deleted subgraph G–S is disconnected with u and v lying in different components. In this case, S is also said to separate u and v.

Similarly, a set F of edges of G is said to be u–v separating if the edge deleted subgraph G–F is disconnected with u and v lying in different components. In this case, F is said to separate u and v.

Chapter Two

Linear And Nonlinear Programming On Graph

2.1 Introduction

In this chapter some relations on linear and nonlinear programming and reduced form of optimization problems into Graph model are given. Various models about these relations have been proposed in Cotle et al. [7], Mangasarian et al. [9], Newman [12], Saunders et al. [20] and Yamada et al. [22].

The general problem of linear programming is to optimize a linear function subject to linear equality and inequality constraints. In other words, it is need to determine the values of $x_1, x_2, \cdots\cdots, x_n$ that solves the program.

$$\text{minimize } z = \sum_{i=1}^{n} c_i x_i$$

$$\text{subject to } \sum_{j=1}^{n} a_{ij} x_j \le b_i \, .$$

Here c_i, b_i and a_{ij} are known as real numbers.

Definition 2.1.1

A vector $(x_1, x_2, \cdots\cdots, x_n)' \in R^n$ is called a feasible solution if the vectors satisfy the constraints. Here $(\cdots)'$ denotes the transpose.

Definition 2.1.2

A feasible solution is said to be an optimal solution if it gives the minimum value of the objective function provided the minimum value exists.

Lemma 2.1.3

The constraint set T is convex.

Proof:

Let f be a convex function on a constraint set $T \in R^n$. Then for every $k \in R$ we can define the sets.

$$T = \{x : x \in T,\ f(x) \le k\}$$

To show T is convex, let

$$x_1 \in T,\ x_2 \in T$$

Let T is convex. Then

$$\lambda x_1 + (1-\lambda)x_2 \in T \quad \text{for} \quad 0 \le \lambda \le 1$$

$$f(x_1) \le k,\ f(x_2) \le k$$

by the convexity property.

Now, using the convexity of f on T, we get

$$f(\lambda x_1 + (1-\lambda)x_2) \le \lambda f(x_1) + (1-\lambda)f(x_2) \le k$$

Hence,

$$\lambda x_1 + (1-\lambda)x_2 \in T \text{ for } 0 \le \lambda \le 1.$$

That is, T is a convex set. This completes the proof of Lemma 2.1.3.

Definition 2.1.4

A vector $x^* \in X$ is global minimum of f over the set X if it is no worse than other feasible vectors that is

$$f(x^*) \le f(x) \qquad x \in X.$$

A vector $x^* \in X$ is a local minimum of f over the set X if it is no worse than its feasible neighbors; that is, if there exists an $\varepsilon > 0$ such that

$$f(x^*) \le f(x) \quad x \in X$$

$$\text{with } \|x - x^*\| < \varepsilon.$$

If both f and X are convex, local minimum is also global.

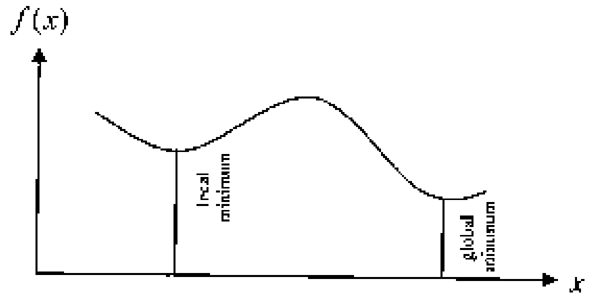


Figure 2.1 : Local and global minima of f over X

2. 2 Necessary and Sufficient Conditions for Optimality

At a local minimum x^*, the first order variation is considered $\nabla f(x^*)'\Delta x$ due to a small feasible variation Δx is nonnegative. Because X is convex, the feasible variations are of the form $\Delta x = x - x^*$, where $x \in X$. Thus the condition $\nabla f(x^*)'\Delta x \ge 0$ translates into the necessary condition

$$\nabla f(x^*)'(x - x^*) \ge 0$$

for all $x \in X$. When f is convex the following proposition hold.

Proposition 2.2.1 (Optimality condition)

a) If x^* is a local minimum of f over X, then

$$\nabla f(x^*)'(x-x^*) \geq 0, \quad x \in X.$$

b) If f is convex over X, then the condition of part (a) is also sufficient for x^* to minimize f over X.

2.3 Descent directions and stepsize Rules of minimization problem

Consider the unconstrained minimization problem of a continuously differentiable function $f: R^n \to R$.

Most of the interesting algorithms for this problem rely on an important idea, called iterative descent that works as follows:
We start at some point x^0 (an initial guess) and successively generates $x^1, x^2, \cdots$ such that f is decreased at each iteration, that is,

$$f(x^{k+1}) < f(x^k), \; k = 0, 1, \cdots\cdots$$

In doing so, one can do successively improve solution and may decrease f all the way to its minimum. In this regards we define a general class algorithms based on iterative descent method. Mangasarian et al. [6] proposed an algorithm.

$$x^{k+1} = x^k + \alpha^k d^k, \; k = 0, 1, \cdots \text{.........}(2.3.1)$$

where, if $\nabla f(x^k)' \neq 0$, the direction d^k is chosen so that $\nabla f(x^k)' d^k < 0$ and the stepsize α^k are chosen to be positive. If $\nabla f(x^k) = 0$, then method stops, that is,

$$x^{k+1} = x^k$$

In this respect the improvement cost functions takes the form

$$f(x^k + \alpha^k d^k) < f(x^k), \; k = 0, 1, \cdots\cdots$$

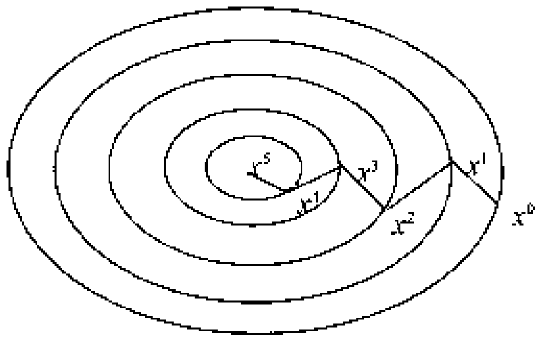


Figure 2.2 Iterative descent for minimizing a function f

Each vector in the generated sequence has a lower cost than its predecessor.

2.4 Systems of difference constraint into graph model

It is easy to interpret the systems of difference constraints of linear or non-linear programming from a graph theoretic viewpoint. By using Bellman-Ford idea in a system, $Ax \le b$ of difference constraints, the $n \times m$ linear-programming matrix A can be viewed as an incidence matrix for a graph with n vertices and m edges. Each vertex v_i in the graph, for $i = 1,2,\cdots,n$ corresponds to one of the n unknown variable x_i. Each directed edge in the graph corresponds to one of the m inequalities involving two unknowns.

In the following way Bellman-Ford converted the linear functions into graph method.

For a given system $Ax \le b$ of difference constraints, the corresponding constraint graph is a weighted, directed graph $G = (V, E)$ where

$$V = \{v_0, v_1, \cdots, v_n\}$$

and $$E = \{(v_i, v_j) : x_j - x_i \le b_k \text{ is a constraint}\}$$
$$\cup \{(v_0, v_1), (v_0, v_2), (v_0, v_3), \cdots, (v_0, v_n)\}.$$

Some important theorems, lemma & definitions are given in the following, which already have formulated as graph model from the linear and non-linear programming of minimization problem.

Theorem 2.4.1

Let $f : R^n \to R$ be a function defined over the graph G. Let $P = \{X, >>\}$ denote a partial order, where $X = \{x_1, \cdots, x_n\}$ is the set of variables. Then $f(A) \ge f(B)$, for all $A = \{a_1, \cdots, a_n\}$ majorizing $B = \{b_1, \cdots, b_n\}$ on P if and only if f is a function such that for every i, j with $x_i >> x_j$,

$$\frac{\partial f}{\partial x_i} \ge \frac{\partial f}{\partial x_j}$$

over all $X \in G$.

Lemma 2.4.2

If A is submajorized by B on P, there exists a set $C = \{c_1, c_2, \cdots, c_n\}$, where $c_i \ge 0$ for all i. such that $A + C$ is majorized by B on P.

Theorem 2.4.3

Let $f : R^n \to R$ be a monotone non-increasing function in each variable, with the Graph G. Denote by $P = \{X, >>\}$ a partial order on the set of variables $X = \{x_1, \ldots, x_n\}$. Then the following relations are equivalent

(i) for every $A, B \subseteq R^n$ such that B submajorizes A on P, we have

$$f(A) \ge f(B).$$

(ii) for every i, j such that $x_i >> x_j$ we have that

$$\frac{\partial f}{\partial x_i} \leq \frac{\partial f}{\partial x_j}$$

over all $X \in G$.

Proof.

To prove that (i) implies (ii), assume that $f(A) \geq f(B)$ for all A submajorized by B on P. Then, in particular,

$$f(A) \geq f(B),$$

or equivalently,

$$-f(A) \leq -f(B)$$

for all A submajorized by B on P. From Theorem 2.4.3, it follows that for all i, j such that $x_i >> x_j$ we have

$$\frac{\partial(-f)}{\partial x_i} \geq \frac{\partial(-f)}{\partial x_j},$$

or equivalently

$$\frac{\partial f}{\partial x_i} \leq \frac{\partial f}{\partial x_j}$$

for all $X \in G$.

In order to prove that (ii) implies (i), assume that (ii) holds and A is submajorized by B on P. From Lemma 2.4.2, there exists non-negative C such that $A + C$ is mojorized by B on P. From Theorem 2.4.3, it follows that

$$-f(A+C) \leq -f(B).$$

But f is monotone non-increasing for all x_i, so we can conclude that

$$f(A) \geq f(A+C) \geq f(B).$$

This completes the proof.

2.5 Optimal Routing of simplex method in a communication Network using graph model

Consider the case where the constraint set is a simplex

$$X = \left\{ x / x \geq o \sum_{i=1}^{n} x_i = r, \right.$$

where $r > 0$ is a given scalar.

There is a straightforward generalization of this problem in graph model which has been investigated by Bertsekas [8].

It is illustrated the graph representation of path flow of the simplex method with Figure 2.3.

Consider a directed graph, which is viewed as a model of a data communication network. It is also given a set ω of ordered node pairs $\omega = (i, j)$. The nodes i and j are referred to as the origin and the destination (OD) of ω respectively, and ω is referred to as an OD pair. For each ω, given a scalar r_ω referced to as the input data of ω. In the context of routing of data in a communication network, r_ω is the rate of data entering and leaving the network at the origin and the destination of ω. respectively. The communication configuration is to divide each r_ω into the paths from origin to destination in a way that the resulting total arc flow pattern minimizes a suitable cost function, we denote

D_ω : A given set of paths that starts at the origin and ends at the destination of ω. All arcs s on each of these paths are oriented in the destination from the origin to the destination.

v_p : The portion of r_ω assigned to path P, also the flow of path P.

The collection of all path flow

$$\{v_p \mid \omega \in W, p \in D_p\}$$

must satisfy the constraints

$$\sum_{p \in D_p} \sum v_p = r_w, \ \forall \omega \in W$$

The total flow f_{ij} of arc (i, j) is the sum of all path flows traversing the arc:

$$f_{ij} = \sum_{\substack{all\ path\ p \\ containing\ (i,j)}} v_p .$$

Consider a cost function of the form

$$\sum J_{ij}(f_{ij}).$$

The problem is to find a set of path flow $\{v_p\}$ that minimize this cost function subject to the constraints of the simplex. This is the minimization over a simplex.

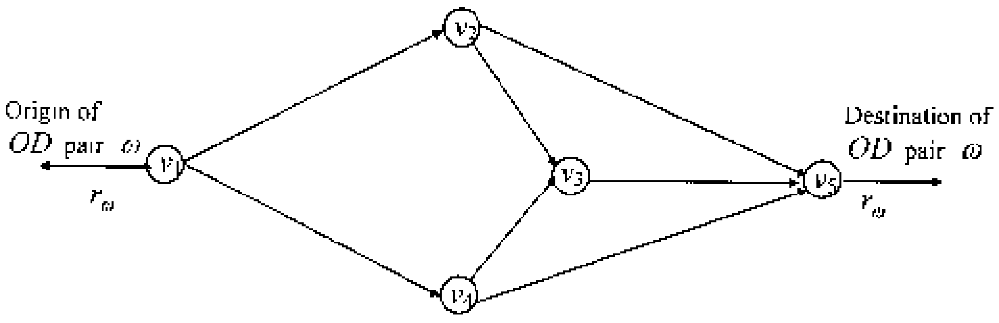


Figure 2.3 Constraints for the flows of an OD pair ω.

The path flows v_p of the paths $p \in D_\omega$ should be nonnegative and add up to the given input r_ω of the OD pair.

Definition 2.5.1

For $P = \{S, >>\}$ and $A = \{a_1, \cdots, a_n\}$, a set of weights for $S = \{s_1, \cdots, s_n\}$, choose $S_i, S_j \in S$ such that $S_i >> S_j$. Then a flow from a_i to a_j is a transformation from A to A', where

$$a_i' = a_i - \delta,$$
$$a_j' = a_j + \delta, \text{ for some } \delta > 0$$
$$a_k' = a_k, \quad \forall k \neq i, j.$$

The following corollary is important for network flow problem.

Corollary 2.5.2

Assume that P is a linearly ordered set. Then, A majorizes B on P if and only if there exists an $n \times n$ triangular matrix $M = [m_{ij}]$ such that $m_{ij} \geq 0$, $m_{ij} = 0$ for all $i < j$, $\sum_{j=1}^{n} m_{ij} = 1$ and $B = MA.$.

Matrix M can be viewed as a flow matrix where $m_{ij}a_i$ is the amount of flow from P_i to P_j.

Chapter Three

Path Algorithms And Spanning Trees

3.1 Introduction

In this chapter most fundamental graph algorithms concerning distances are those dealing with shortest path in a graph. In fact most algorithms involving distances and carrying flow over distance use basic search techniques of graph theory. Many typical routing algorithms send flows through shortest path without accounting for derivatives and thus bifurcating flows. We have discussed some standard shortest path algorithms using some methods. We have solved the traveling salesman problem using closest insertion algorithm and described elaborately the minimum spanning tree with an example. First of all we have defined shortest path.

3.2 Shortest path

Suppose every arc uv has been assigned a certain length $l(uv)$. Then the length of $u-v$ path with edge sequence $e_1, e_2, \cdots$ is the sum $l(e_1)+l(e_2)+\cdots+l(e_k)$ of the lengths of all edges of that path. If the source vertex s and v is any arbitrary vertex then we define the shortest path distance $\delta(s,v)$ from s to v as the minimum number of edges in any path from vertex s to vertex v, or else ∞, if there is no path from s to v. A path of length $\delta(s,v)$ from s to v is said to be a shortest path from s to v and there are many methods or techniques in graph theory to calculate or to obtain shortest paths. Before showing that search method actually computes shortest path distances, we investigate some important property of shortest path distances.

Lemma 3.2.1

Let $G=(V,E)$ be a directed or undirected graph and let $s \in V$ be an arbitrary vertex, then for any edge $(u,v) \in E$, $\delta(s,v) \le \delta(s,u) + w(u,v)$, where w is a weight function.

Proof. If u is reachable from s, then so is v. In this case the shortest path from s to v cannot be longer than the shortest path from s to u followed by the weighted edge (u,v) and thus the inequality holds.

3.3 Weighted shortest path

In a shortest path problem, we are given a weighted directed graph $G=(V,E)$ with weight function $w: E \to R$ mapping edges to real valued weights. The weight of path $P = < v_0, v_1, \cdots, v_k >$ is the sum of the weights of its constituent edges:

$$w(P) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

By definition, we define the shortest path weight from u to v by

$$\delta(u,v) = \min \begin{cases} \infty & : \text{if there is no path from } u \text{ to } v \\ w(P) : u \xrightarrow{P} v & \text{if there is a path from } u \text{ to } v \\ 0 & : \text{otherwise} \end{cases}$$

A shortest path from vertex u to vertex v is then defined as any path P with weight and written as

$$w(P) = \delta(u,v).$$

Edge weights can be interpreted as matrices other than distances. They are often used to represent time, cost, penalties, or another quantity that accumulates linearly along a path, which is shortest distance.

The following lemma and corollary states the optimal-substructure property of shortest paths more precisely.

Lemma 3.3.1

Sub paths of shortest paths are shortest paths.

Proof. If we decompose path P into $v_1 \xrightarrow{P_{1i}} v_i \xrightarrow{P_{ij}} v_j \xrightarrow{P_{jk}} v_k$, then $w(P) = w(P_{1i}) + w(P_{ij}) + w(P_{jk})$. Now, assume that there is a path P_{ij} from v_i to v_j with weight $w(P'_{ij}) < w(P_{ij})$. Then $v_1 \xrightarrow{P_{1i}} v_i \xrightarrow{P'_{ij}} v_j \xrightarrow{P_{jk}} v_k$ is a path from v_1 to v_k whose weight $w(P_{1i}) + w(P'_{ij}) + w(P_{jk})$ is less than $w(P)$ which contradicts the premise that P is a shortest path from v_1 to v_k.

Corollary 3.3.2

Let $G = (V,E)$ be a weighted, directed graph with weight function $w : E \to R$. Suppose that a shortest path P from a source s to a vertex v can be decomposed into $s \xrightarrow{P'} u \to v$ for some vertex v and path P'. Then the weight of a shortest path from s to v is $\delta(s,v) = \delta(s,u) + w(u,v)$.

Proof. By Lemma 3.3.1, it can be proved that sub path P' is a shortest path from source s to vertex u.

Thus

$$\begin{aligned}\delta(s,v) &= w(P) \\ &= w(p') + w(u,v) \\ &= \delta(s,u) + w(u,v).\end{aligned}$$

3.4 The structure of a shortest path with examples

Shortest paths are not necessarily unique. Using adjacency matrix $W=(w_{ij})$ we characterize the structure of an optimal solution. A natural combinatorial optimization problem is the shortest path problem and it has an input directed graph, each edge with a given length. Now to construct a shortest path P from vertex i to vertex j and suppose that P contains at most m edges. Assuming that there are no negative-weight cycles, m is finite, If $i=j$, then P has weight O and no edges. If vertices i and j are distinct, then we can decompose path P into $i \xrightarrow{P'} k \to j$, where path P' now contains at most $m-1$ edges. P' is a shortest path from i to k and from lemma 3.3.1 and Corollary 3.3.2, we have

$$\delta(i,j) = \delta(i,k) + w_{kj}.$$

Figure 3.1 illustrates a directed shortest path with positive weights and computing not only shortest path weights, but the vertices on the shortest paths as well.

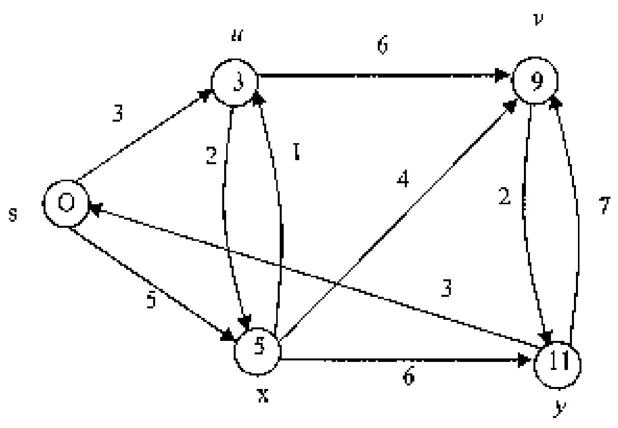
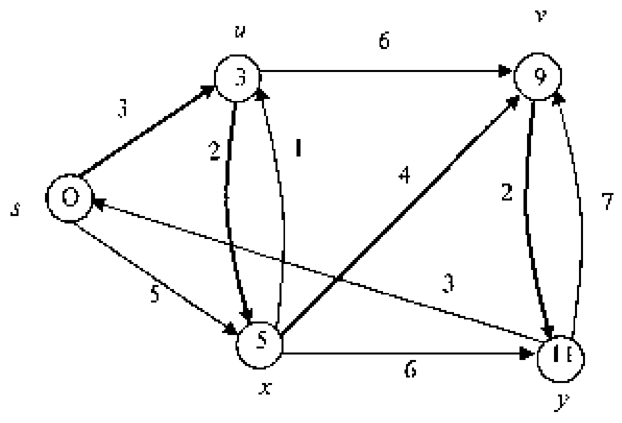


Figure 3.1 (a). A weighted, directed graph with shortest path weights from source s



3.1 (b). The shaded edges form shortest paths distances

Calculation of the shortest path weight of figure 3.1 :

From the definition of weighed shortest path, we get

$$\delta(s,u) = w(s,u) = 3$$

and $\delta(s,x) = \delta(s,u) + w(u,x) = 3 + 2 = 5$ (using corollary 3.3.2)

Thus $\delta(s,v) = \delta(s,x) + w(x,v) = 5 + 4 = 9$

and $\delta(s,y) = \delta(s,v) + w(v,y) = 9 + 2 = 11$

Thus the shortest path from s to y is $<s,u,x,v,y>$ with weight 11.

In most cases when describing algorithms we shall assume that adjacency lists are used to describe any graph as an input to an algorithm. When using such an algorithm, if the graph is not already labeled, simply label the nodes and record the adjacency list for input.

An example of a graph and its adjacency lists is given below in the figure 3.2 and the table.

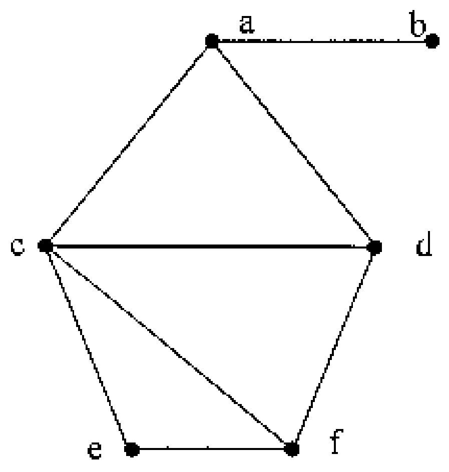


Figure 3.2

A graph and its adjacency lists

Node		Adjacency list
a		b. c, d
b		a
c		a, d, e, f
d		a, c, f
e		c, f
f		c, d, e

3.5 Breadth-First Search (BFS)

Thomas et al. [21] discovered breadth-first search in the context of finding paths through mazes. *BFS* is a fundamental search technique that traces a rooted spanning tree in a connected graph so that the distance from the root to each node in the tree corresponds to its distance in the original graph. The basic idea is to begin at the root and find its neighbors and then neighbors, and so on, until one has spanned throughout the graph and reached all nodes. Since in most applications where *BFS* is used, one wants to know the distance from the root to each other node, we present a form of the algorithm. which records the distances. The input to the algorithm is the list of nodes, their adjacency lists, and the label of the root. Assume that the distance $\delta(s,v_k)$, from the root s to node v_k is stored in array $d[v]$. Let $N[v]$ denote the adjacency list of node v.

3.6 The Breadth First Search (BFS) Algorithm

Let G be a graph and let s,t be two specified vertices of G. We will now describe a method of finding a path from s to t, if there is any, which uses the least number of edges. Such a path, if it exists, is called a shortest path from s to t.

The method assigns labels 0, 1, 2, 3, ... to vertices of G and is called the Breadth First Search technique. In Clark et al. [6], the following algorithm is given below:

Step 1. Label vertex s with 0. Set $i=0$.

Step 2. Find all unlabelled vertices in G, which are adjacent to vertices labeled i. If there are no such vertices then t is not connected to s (by a path). If there are such vertices, label them $i+1$.

Step 3. If t is labeled, go to step 4. If not, increase i to $i+1$ then go to step 2.

Step 4. The length of a shortest path from s to t is $i+1$ stop.

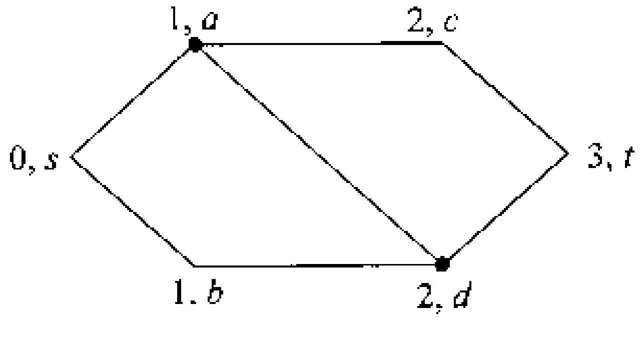


Figure 3.3

First s is labeled 0. The a and b are labeled 1. Then c,d are labeled 2 and t is labeled 3, the level of a shortest path from s to t is 3.

3.7 Complexity of BFS

The complexity of BFS on $G=(m,n)$ graph, where $m=|V|$ and $n=|E|$ is $O(m+n)$. The BFS runs in time linear in the size of the adjacency list representation of G.

We now discuss the Depth first search and its properties. DFS is a powerful technique to obtain shortest path.

Depth first search

Depth first search (DFS) begins at the root and trace out a path from the root until one can go no farther without revisiting a node. Then backtrack along the path until reaching the first node with an alternate route available and proceed forward again. Repeat this until one can go no farther. Like BFS, a depth-first search traces a spanning tree in a connected graph, but in a different manner. One might think of DFS as the way an intelligent but determined mouse might find its way through a search.

The following DFS algorithm is the basic depth-first search algorithm.

$DFS(G)$ Algorithm

```
Procedure $DFS(v)$
begin
        $l(v) = 1$        (tracks the order in which nodes are visited)
        $i = i + 1$
while $N(v) \neq \phi$ do
                for $u \varepsilon N(v)$ do
                begin
                        $T = TU\{u, v\}$
                        remove $u$ from all adjacency lists
                        $DFS(u)$
end $DFS$
begin        (driver algorithm)
        input adjacency lists and root
        $T = \phi$ (stores edges of the tree as they are selected)
        for $v \varepsilon G$ do
                $l(v) = 0$
        $l(\text{root}) = 1$
        $i = 2$
        while there exist some $u$ for which $l(u) = 0$ do
        for highest labeled node $v$ with $N(v) \neq \phi$ do
        begin remove $v$ from all adjacency lists
                $DFS(v)$
        end
```

The backtracking along a path in the tree each time control and is returned to the driver program and one looks for the highest labeled node that has unvisited neighbors. Note as with BFS, removing a node W from all adjacency lists in this algorithm is easy because $N(w)$ tells which lists are involved. Algorithm can easily be modified to determine whether G is connected simply by testing for nodes v that still have $l(v)=0$ when the algorithm terminates.

Besides creating a depth first tree DFS also timestamps for each vertex. Each vertex v has two timestamps: the first timestamp $d[v]$ records when v is first discovered and the second timestamp $f[v]$ records when the search finishes examining v's adjacency list.

Corollary 3.7.1 (nesting of descendants' intervals)

Vertex v is a proper descendant of vertex u in the depth-first forest for a (directed or undirected) graph G if and only if $d(u)<d(v)<f(v)<f(u)$.

The proof has been illustrated in Thomas et al.[21].

The DFS algorithm is illustrated by an example in Figure 3.5. In the given graph G of five vertices and seven edges, the starting vertex s is specified. The order in which the edges are explored is given in Figure 3.5 (b) and for this order of traversal $\vec{G}$ is given in figure 3.5 (c)

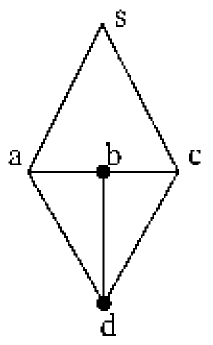


Figure 3.5(a) Graph G before DFS

(s, a) : branch

(a, b) : branch

(c, s) : frond

(d, b) : frond

(c, d) : branch

(b, c) : branch

(d, a) : frond

Figure 3.5(b) Order in which edge were scanned

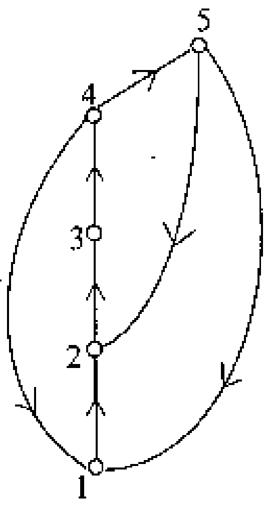


Figure 3.5(c) Graph $\ddot{G}$ after DFS

We now describe the closest insertion algorithm. The description gives the idea of the distance of a vertex v from a walk W. This is defined to be

$d(v,W) = \min\ \{d(v,u) : u \text{ is a vertex of } W\}$

The vertex v, not in W, is then said to be closest to W

if $d(v,W) \leq d(x,W)$ for any other vertex x not in W.

We describe the way in which a new vertex is inserted into a cycle by using bold type. We also solve the Traveling salesman problem for the complete weighted graph using closest insertion algorithm.

The closest Insertion Algorithm:

Step 1. Choose any vertex v_1 as a starting vertex.

Step 2. From among the $n-1$ vertices chosen so far, find one, say v_2, which is closest to v_1. Let W_2 denote the walk $v_1v_2v_1$.

Step 3 From among the $n-2$ vertices not chosen so far, find one, say v_3, which is closest to the walk $W_2 = v_1v_2v_1$. Let W_3 be the walk $v_1v_2v_3v_1$.

Step 4 From among the $n-3$ vertices not chosen so far, find one, say v_4, which is closest to the walk W_3. Determine which of the walks (cycles) $v_1v_2v_3v_4v_1$, $v_1v_2v_4v_3v_1$, $v_1v_4v_2v_3v_1$ is the shortest. Let W_4 denote the shortest one and relabeled it, if necessary, as $v_1v_2v_3v_4v_1$.

Step 5 From among the $n-4$ vertices not chosen so far, find one, say v_5 which is closest to the walk W_4.

Determine which of the cycles $v_1v_2v_3v_4v_5v_1$, $v_1v_2v_3v_5v_4v_1$, $v_1v_2v_5v_3v_4v_1$, $v_1v_5v_2v_3v_4v_1$ is shortest.

Let W_5 denote the shortest one and relabeled it, if necessary, as $v_1v_2v_3v_4v_5v_1$ continue in this way to eventually arrive at.

Step n Denote the remaining unclosen vertex by v_n and determine which of the cycles $v_1v_2\cdots v_{n-1}v_nv_1$, $v_1v_n\cdots v_{n-1}v_1$ is shortest.

Let W_n denote the shortest of these cycles.

Since our target is to solve the Traveling Salesman Problem of the complete weighted graph. For this purpose we first define the Traveling salesman problem. Details on TSP have been described in Narsingh [11].

Traveling Salesman Problem *(TSP)*

We can represent the salesman's territory by a weighted graph $G(V,E)$. The vertices correspond to the towns and two vertices are joined by a weight edge if there is a road connecting the corresponding towns, which does not pass through any of the other towns, the edges weight representing the lengths of the road between the towns. The sales man finds his optimal distance from a starting vertex and visits each vertex and again comeback to his first starting vertex. He continues this process until every vertex visited. And this system is known as traveling salesman problem.

We illustrate Traveling Salesman problem with an example and solve it using closest insertion Algorithm.

Let a traveling salesman's territory includes five cities A, B, C, D and E which corresponding to the vertices. Each road connecting a pairs of vertices with weights w. His job requires him to visit each city. So, it is required for him to plan a round trip by car enabling him to visit each of the cities exactly once and minimize the total distance traveled. We wish to find an ordering of the five cities starting with vertex A so that if he visits five vertices only once in this optimal order, then the total distance is smallest possible.

The following figure 3.5 is a complete weighted graph and A denotes the starting vertex.

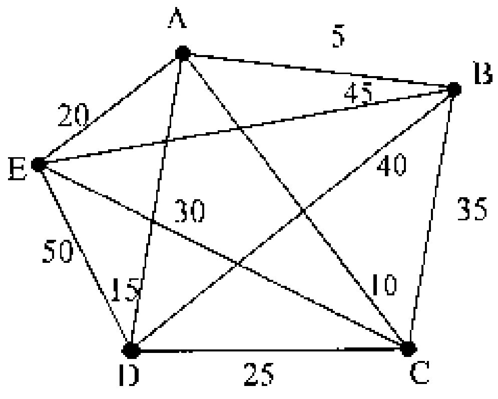


Figure: 3.5 A weighted graph

Algorithm (using closest Insertion Algorithm)

Step 1. Let $v_1 = A$

Step 2. $v_2 = B$ is closest to A so $W_2 = v_1v_2v_1 = ABA$

Step 3. $v_3 = C$ is closest to W_2 (It has distance 10 from A) W_3 is the cycle $v_1v_2v_3v_1 = ABCA$ and has length 5+35+10=50.

Step 4. $v_4 = D$ is closest to W_3 (It has distance 15 from A) to W_3.

We find the length of the cycles $v_1v_2v_3v_4v_1$, $v_1v_2v_4v_3v_1$, $v_1v_4v_3v_2v_1$:

$v_1v_2v_3v_4v_1 = ABCDA$ has length $5+35+25+15=80$

$v_1v_2v_4v_3v_1 = ABDCA$ has length $5+40+25+10=80$

$v_1v_4v_3v_2v_1 = ADCBA$ has length $15+25+35+5=80$

We choose $W_4 = ABCDA$ as the shortest one.

Step 5. $V_5 = E$ is closest to W_4 (It has distance 20 from A)

We find the lengths of the cycles $v_1v_2v_3v_4v_5v_1$, $v_1v_2v_3v_5v_4v_1$, $v_1v_2v_5v_3v_4v_1$ and $v_1v_5v_2v_3v_4v_1$

$v_1v_2v_3v_4v_5v_1 = ABCDEA$ has length $5+35+25+50+20=135$

$v_1v_2v_3v_5v_4v_1 = ABCEDA$ has length $5+35+30+50+15=135$

$v_1v_2v_5v_3v_4v_1 = ABECDA$ has length $5+45+30+25+15=120$

$v_1v_5v_2v_3v_4v_1 = AEBCDA$ has length $20+45+35+25+15=140$

We can therefore conclude that a reasonably efficient shortest path distance is $v_1v_2v_5v_3v_4v_1 = 120$.

3.8 Minimum spanning tree

Minimum spanning tree is important in Graph theory. Finding a spanning tree in a weighted graph such that the sum of the weights of the edges in the tree is minimum and solves a wide variety of problem.

Definition:

A minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

Consider a connected graph each edge of which has a positive length. A connected subset M of edges is called a tree if there is no cycle in this subset. A tree is a spanning tree if it connects all the nodes, i.e. there is a path between any pair of nodes. It is required to find a spanning tree sum of whose edge lengths is the minimum; such a spanning tree is called a minimum spanning tree. Hence forth, we assume that the edge lengths are distinct.

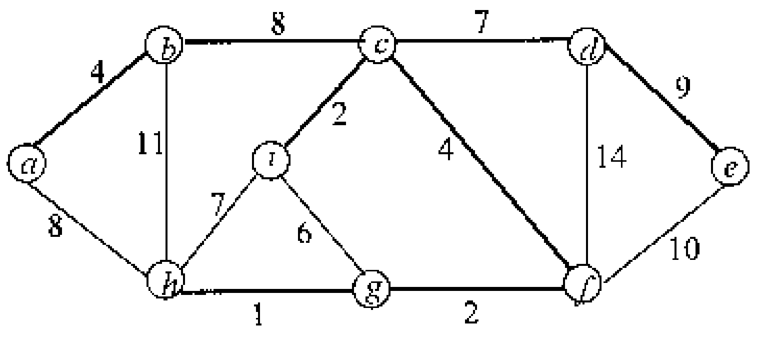


Figure 3.6 Example graph for minimum spanning tree

A minimum spanning tree for a connected graph and the weights on edges are shown, and the edges in a minimum spanning tree are shaded. The total weight of the tree is 37. The tree is not unique: removing the edge (*b*, *c*) and replacing it with the edge (*a*, *h*) yield another spanning tree with weight 37.

Properties of Minimum Spanning Tree

Let M be a minimum spanning tree.

- (P_1) There are $n-1$ edges in a spanning tree.
- (P_2) There is exactly one path connecting a pair of nodes in a spanning tree
- (P_3) Adding an edge to a spanning tree creates a cycle.
- (P_4) Removing any edge from a cycle as in (P_3) creates a spanning tree.
- (P_5) Let e be any edge outside the minimum spanning tree. The edges on the cycle created by adding e have lower lengths than that of e.

Prim's Algorithm

Prim's algorithm is more efficient to construct a minimum spanning tree. Using prim's algorithm we construct a minimum spanning tree.

Due to Clark et al. [6], we have the following Prim's Algorithm :

Step 1. Choose any vertex v_1 of G.

Step 2. Choose an edge $e_1 = v_1v_2$ of G such that $v_2 \neq v_1$ and e_1 has smallest weight among the edges of G incident with v_1.

Step 3. If edges $e_1, e_2, \cdots, e_i$ have been chosen involving end points $v_1, v_2, \cdots, v_{i+1}$, choose an edge $e_{i+1} = v_jv_k$ with $v_j \in \{v_1, \cdots, v_{i+1}\}$ and $v_k \notin \{v_1, \cdots, v_{i+1}\}$ such that e_{i+1} has smallest weight among the edges of G with precisely one end in $\{v_1, \cdots, v_{i+1}\}$.

Step 4. Stop after $n-1$ edges have been chosen otherwise repeat Step 3.

Chapter Four

Shortest Paths Algorithm respect to Non-Linear Complementarity Problem

4.1 Introduction

This chapter deals with nonlinear programming (minimization problem) with the graph model. The nonlinear complementarity problem (NCP) is to find a point $x \in X$ such that

$$x^T F(x) = 0$$
$$x \geq 0, F(x) \geq 0, \text{ ....(4.1)}$$

where T denotes the transpose and $F : R^n \to R^n$ is a given function (Cotle et al. [7], Mangasarian et al. [9] and Yamada et al. [22]).

Recently this problem has been reformulated as a minimization problem in order to apply well-developed optimization methods (Cotle et al. [7] and Yamada et al. [22]). We have shown the equation (4.1) can be formulated as a graph model and reformulated as a residual flow problem. In residual network both the cost and price are efficient (Mangasarian et al. [9] and Thomas et al. [21]).

4.2 Flow properties in Network

Let $G = (V, E)$ be a flow network, where flow based edges are connected by weight function W. A weight function defined as

$$W_{ij} = \begin{cases} 0 & \text{if} \quad i = j \\ \text{the weight of directed edge } (i, j) \in E \\ \infty & \text{if} \quad i \neq j \text{ and } (i, j) \notin E \end{cases}$$

In a given network, a flow f is an assignment of a nonnegative number $f_{i,j}$ to every directed edge (i,j) such that the following conditions are satisfied.

1. For every directed edge $(i,j) \in G$

$$f_{i,j} \le c_{i,j}, \text{ where } c_{ij} \text{ is the capacity of edges } i,j \in G$$

2. There is a specified vertex s in G called the source for which,

$$\sum_i f_{si} - \sum f_{is} = d \cdot$$

Where the summations are taken over all vertices in G. Quantity d is called the value of the flow.

3. There is another specified vertex t in G, called the destination for which

$$\sum_i f_{ti} - \sum_i f_{it} = -d .$$

4. All other vertices are called intermediate vertices. For each intermediate vertex j

$$\sum_i f_{ji} - \sum_i f_{ij} = 0 .$$

We are giving a general statement about optimal routes with regard to network topology.

4.3 The optimality principle

The optimality principle states that if router I is on the optimal path from s to t. then the optimal path from any J to K in between s and t also falls along the same route.

Graph representation of the equation (4.1) :

Consider the variables that the flows through each of edges in $G(V,E)$. W is a weight function. Let the flow pattern be denoted by a column vector f and $f' = \sum_{l \in I} f_l' = w_l$ denote the variable vector, where f' is the derivative of f. Also let d denote the row vector. Then the problem is to find a flow that minimize $d \ f'$.

$$\text{Subject to the constraints } A' \cdot f' = 0$$

$$f \le c, \text{ for } c > 0$$

$$\text{and } f \ge 0, \ \ldots\ldots\ldots\ldots\ldots (4.2)$$

where A' is the incidence matrix obtained by adding an edge from t to s (Narsingh [11]). Clearly there exists a parallel edge in the graph model in equation (4.2). Also the edges in f, c and A' must appear in the same order (Clark et al. [6]).

We shall show that the equation (4.2) can be reformulated as a residual network to obtain a minimal flow cost. The author in Yamada et al. [22] has shown that residual function solves the nonlinear complementarity problem.

To construct a residual network as an optimization problem we give some preliminaries on optimal routing residual network.

4.4 Model and Preliminaries of Residual Network

We consider a flow constraint network $G = (V, E)$, where V is a finite set of vertices and $E = l(u,v), E \in V \times V$ is a set of directed links. A set

$I = \{1, 2, \cdots, I\}$ of users which is used as vertices that share the network in the way that all users ship flow from a common source s to the common destination t. For each user $i \in I$ there has some throughput demand w_i in the network referred to as the input traffic of W ($W = \sum w_i$). In the context of routing of data in communication network $w_i, i \in I$, is the arrival rate of traffic entering and existing the network from the origin to the destination of w. Routing objective is to divide each $w_i, i \in I$, among the many paths from origin to destination so as to optimize its performance objective. Without loss of generality we assume that $w_1 > w_2 > \cdots > w_I$ which do users control (Blasum et al. [24]). We denote the terms of user flows f_l^i where $l \in E$, user routing strategy f^i, user strategy space F^i and system flow configuration f. such that strategy space F^i should account for the conservation property of flow at all nodes, that is, for all $l \in E$, we get

$$\sum_{l \in out(v)} f_l^i = \sum_{l \in in(v)} f_l^i + w_v, \quad v \in V$$

where

$$w_t^i = -w^i, \quad w_s^i = w^i \text{ and } w_v^i = 0 \text{ for } v \neq s, t.$$

The system of flow configuration aims to find a strategy $f^i \in F^i$, which minimizes its cost. Suppose that associated with each edges of the residual network there exists a number d_l^i, $l \in E$ and $i \in I$ which quantified as a cost variable in the network. The minimum cost depends on the routing decisions of the other users, described by the strategy profile

$$f^i = (f^1, \cdots f^{i-1}, \cdots, f^I)$$

and d^i is a function of the system of flow configuration f. By the constraint set property, the collection of all path flow f_l^i must satisfy the constraint set

$$\sum f_l^i = w_i, \quad f_l^i \geq 0, \; i \in I. \text{......................} (4.3)$$

Thus nonlinear complementarity problem using the equation (4.2) is to find a strategy profile such that

$$f^i \in \arg\min d^i(g^i f^i), \qquad g^i \in F^i \text{, a real number............}(4.4)$$

Definition 4.1

User are said to be identical if their demands are equal, *i.e.*, $w^i = w^j,\ i, j \in I$.

Definition 4.2

A user is said to be simple if its flows are routed through links (or paths) of minimal delay.

4.5 Minimization model of residual network

Let G_f is the residual network. Consider the cost function in the form

$$d_l^i f_l^i \text{..........................}(4.5)$$

associated with edge $l = (u, v)$ in the residual network G_f. It is desired to construct a flow pattern sending a specified value w_i, $i \in I$ from source s to sink t, satisfying the flow constraints defined in (4.2) which minimize the total flow cost

$$\sum d_l^i f_l^i, \quad \forall i \in I, \quad l \in E$$

in overall flows that send W units from s to t.

The author in Bertsekas [2] showed that the problem is to find a set of path flows $\{f_l^i\}$ that minimizes this cost function subject to the constraints equation (4.2). We assume that d_l^i is a convex and continuously differentiable function.

Now by expressing the total flow in terms of the path flows in the cost function in equation (4.5) with constraint sets, the minimization problem can be formulated in terms of the path flow variables $\{f_l^i\}, i \in I, l \in E$ as

$$
\begin{aligned}
&\text{minimize } D(f) \\
&\text{subject to } \sum f_l^i = w_l, \forall l \in E \\
&\qquad\qquad f_l^i > 0, \\
&\qquad\qquad c_l^i > 0,
\end{aligned}
$$

where c_l^i is the residual capacity and

$$D(f) = \sum d_l^i \left(\sum f_l^i\right).$$

4.6 Structure of the residual network

Consider the residual network G_f in a system of parallel links with capacity configuration C. A number of intuitive monotonicity properties of this network have been established in Korilis et al. [8] and are summarized in the following:

Lemma 4.3

Let f be the unique flow in the network of parallel links with capacity configuration C. Then

1. The expected flow of any user $i \in I$ decreases in the link number, i.e., $f_1^i \geq f_2^i \geq \cdots \geq f_E^i$. In particular, for $f_l^i > 0$, we have $f_l^i = f_m^i$ if and only if $c_l = c_m$.
2. For any link $l \in E$, the flows decrease in the user number, i.e., $f_l^1 \geq f_l^2 \geq \cdots \geq f_l^I$. In particular for $f_l^I > 0$, we have $f_l^i = f_l^j$ if and only if $w^i = w^j$.
3. The residual capacity is decreasing in the link number, i.e., $c_1 - f_1 \geq c_2 - f_2 \geq \cdots \geq c_E - f_E$. In particular $f_l = f_m$ if and only if $c_l = c_m$.
4. For every user $i \in I$, the residual capacity $c_l^i = c_l - f_l^i$, seen by the user on link l is decreasing in the link number, i.e., $c_1^i \geq c_2^i \geq \cdots \geq c_E^i$. In particular, $c_l^i = c_m^i$ if and only if $c_l = c_m$.

In a gradient method (for minimization) each vector in the generated sequence has a lower cost than its predecessors. Similar to the gradient method, we decompose the input graph $G = (V, E)$ using *DFS* technique.

4.7 Acyclic decomposition of the graph $G = (V, E)$

We have decomposed the input graph $G = (V, E)$ similar to the gradient method. Thomas et al. [21] have considered the gradient method such as edge set E into T and $\overline{T}$, where

$$T = \{(v_i, v_j) \in E \cdot i < j\}$$

and

$$\overline{T} = \{(v_i, v_j) \in E \cdot i > j\}$$

and defined $G_f = (V, T)$ and $G_b = (V, \overline{T})$ in the residual network and showed that G_f and G_b are acyclic. In our decomposition we assumed that the set of increment of flow, $\nabla f(x)$, correspond to the addition of flow in the forward edges and the set of flow direction *d* corresponds to the transferring of flow in the backward edges.

To reduced this decomposition into NCP form, consider a node $v \in V$ such that $IN(v) = \{l(u, v)\}$ and $OUT(v) = \{l(v, w)\}$ of an instance requesting a spanning u, w path via v in G. We now show that the following lemma is acyclic.

Lemma 4.4

The decomposition of the graph $G = (V, E)$ is acyclic.

Proof:

Consider an instance that requests a u, w path via v in P. Form G by splitting each vertex x into a path x^-, x^0, x^+, with x^- inheriting all edges with heads at x and x' inheriting all edges with tails at x. A spanning u,w path in P becomes a u,w path in G by replacing each x by the sequence x^-,x^0,x^+ (Narsingh [11]). Conversely, since a spanning u,w path in G must visit each x^0, it must visit traverse all sequences of the form x^-,x^0,x^+ forwards or backwards. Since no vertices of the same sing are adjacent, these traversals must all be in the same direction and then they collapse to the desired u,w path in P which is acyclic.

In general an acyclic decomposition consists of a set of trigger vertices T and a set of vertices $\overline{T} = V - T$ that forms a subgraph constituting a large acyclic region with G. This different acyclic decomposition by trigger vertices is equivalent to that of the feedback vertex set, that is, the graph is decomposed into a feedback set T and its associated acyclic region $\overline{T}$. The successive shortest path algorithms are able to reduce the cost function. The cost $d_i^j(f_i^j)$ of an edge $u \to v$ in P is defined as the cost of the shortest path of the form $u \to v'$ for $u \in T$ in the forward edges and $v \to w$ in P when $w \in \overline{T}$ in the backward edges. In path P, let $f'_i{}^j$ be the minimum of all differences $[c(u,v) - f(u,v)]$ in forward edges and also let $f''_i{}^j$ be the minimum of flows in backward edges. Let $f_i^j = \min f_i^j$. Then the flow in the network G can be increased by increasing the flow in each forward edges and decreasing the flow in each backward edges by an amount f_i^j, which satisfy the optimality conditions defined in Korilis et al. [8] and Thomas et al. [21], i.e., for a flow vector $f_i^j > 0$, we find $\min C_f(P)$

which reduced the cost $d'_l(f'_l)$ with respect to non-linear complementarity problem when the path is shortest.

From the following algorithm we solve the successive shortest path distances from a source s to destination t using DFS technique.

4.8 Successive shortest paths Algorithm

```
Step 1. Initialize pre flow $(G,s)$
Step 2. A linked list, $L \leftarrow V[G] - \{s,t\}$ in any order
Step 3. Let $s \in T$ be the set of links $uv$ in $G_f$ such that
        $c(u,v) > f(u,v)$. ($T$ consists of forward links).
        Let $t \in \overline{T}$ be the set of links $uw$ such that $f(v,w) > 0$, ($\overline{T}$ consists
        of reverse links).
Step 4. For such
        $u \in IN(v)$ with excess $e(u) > 0$
        do $c_f = c(u,v) - f(u,v)$ in $G_f$
        do $f'_l = \min c_f, i \in V, l \in E$.
Step 5. Using a backtracking procedure, identifying the incrementing
            path from $s$ to $t$ in $G_f$.
Step 6. Pull $(w \in \overline{T})$
        transfer $f'_l = \min f''_l$ in the backwards direction.
        do $\delta(s,t) = \delta_l(u,v) + \delta_l(v,w)$ with reduced cost $d'_l(f'_l)$.
```

Step 7. do $f[u,v] \leftarrow f[u,v] + f''_i$.

Step 8. Repeat until

$$c_f(u,v) = \min c_f(u,v),\ \forall f_i' > 0.$$

Theorem 4.5

Given a system $Ax \le b$ of difference constraints, let $G = (V, E)$ be the corresponding constraint graph. If G contains no negative-weight cycles. then

$$x = (\delta(v_0, v_1), \delta(v_0, v_2), \delta(v_0, v_3), \cdots\cdots, \delta(v_0, v_n))$$

is a feasible solution for the system. If G contains a negative-weight cycle, then there is no feasible solution for the system.

Before going proof we will introduce a constraint graph with examples.

In a system of the difference constraint matrix A where each row of A contains one 1 and one −1 and all others entries are 0. Thus the constraints given by $Ax \le b$ are a set of m difference constraints involving n unknowns, in which each constraint is a simple linear inequality of the form

$$x_j - x_i \le b_k,$$

where $1 \le i,\ \ j \le n$ and $1 \le k \le m$.

Now consider the problem of 5-vector $x = (x_i)$ that satisfies

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \le \begin{pmatrix} 4 \\ -6 \\ 1 \\ 3 \\ 5 \\ 10 \\ -4 \end{pmatrix} \text{.......................................} (1)$$

Now writing the above matrix in the inequalities form

$$x_1 - x_2 \le 4$$

$$x_2 - x_4 \le -6$$

$$x_3 - x_2 \le 1$$

$$x_4 - x_1 \le 3 \quad \text{...} (2)$$

$$x_4 - x_3 \le 5$$

$$x_4 - x_5 \le 10$$

$$x_5 - x_3 \le -4 .$$

One solution to the difference constraint in equation (2) is $x = (8,4,5,10,0)$, where we have assumed that $x_5 = 0$.

We set up a constraint graph with the inequalities (2). The value of $\delta(v_0, v_i)$ is shown in each vertex v_i, $i = 1,2,3,4,5$, and $\delta(v_0, v_i)$ denotes the shortest distances between pairs of vertices.

Figure 1 shows the constraint graph for the system (2) of difference constraint (using Bellman-Ford idea)

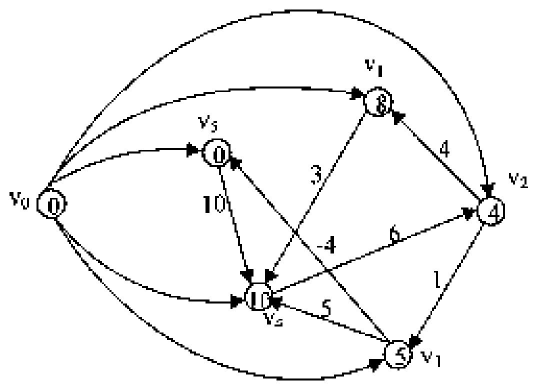


Figure 1. A feasible solution to the system is $x = (8, 4, 5, 10, 0)$

Proof (first part of the theorem 4.5)

We know

$$\delta(v_0, v_j) - \delta(v_0, v_i) \le w(v_i, v_j)$$

Now let, $\quad \delta(v_0, v_j) = x_j$

and $\quad \delta(v_0, v_i) = x_i$.

So $\quad x_j - x_i \le w(v_i, v_j)$ .... (3)

which corresponds to the constraint matrix $Ax \le b$ from the above example in equation (1) and (2) and also corresponds to the edge (v_i, v_j). Thus $\delta(v_0, v_k)$, $k = 1, 2, \cdots\cdots$ is a feasible solution of the theorem 4.5. Now we show that (second part of the theorem) for the negative weight cycle of the constraint graph.

For the purpose of negative weight cycles, let the negative weight cycle be $C = (v_1, v_2, v_3 \cdots\cdots v_k)$ where $v_1 = v_k$.

Proof (second part of the theorem 4.5)

Again we consider the equation (3)

$$\begin{aligned}
x_2 - x_1 &\le w(v_1, v_2) \\
x_3 - x_2 &\le w(v_2, v_3) \\
&\vdots \\
&\vdots \\
x_k - x_{k-1} &\le w(v_{k-1}, v_k) \\
x_1 - x_k &\le w(v_k, v_1) \qquad [\text{since } v_k = v_1].
\end{aligned}$$

Now we sum the inequalities we get

$$0 \le w(c).$$

But we have considered the weight cycle is negative i.e.

$$w(c) \le 0,$$

Thus solution for the x must satisfy

$$0 \le w(c) \le 0,$$

which is impossible. This completes the proof of theorem 4.5.

4.9 Mathematical form of shortest path distance by Nonlinear complementarity problem

In Bertsekas [2] the author have been illustrated that optimal routing directs traffic exclusively along paths that are shortest with respect to arc lengths that depend on the flows carried by the arcs. In contrast to the equation (4.2), we shall show that,

NCP can formulated to the shortest path distances which solves the residual networks.

Poof: From equation (2.3.1) we get

$$d^k = \frac{1}{\alpha^k}\left[x^{k+1} - x^k\right] \quad \forall \alpha \ge 0 \quad \text{and} \quad k \ge 1, 2, 3, \cdots$$

Using Theorem 4.5, letting

$$x^k = \delta(v_0, v_i) \quad \text{and} \quad x^{k+1} = \delta(v_0, v_j)$$

which satisfies the difference constraint that corresponds to edge (v_i, v_j) in the descent direction of NCP.

Thus

$$\delta(v_0, v_j) - \delta(v_0, v_i) = \alpha^k d^k$$

$$\Rightarrow d^k = \frac{1}{\alpha^k}\left[\delta(v_0, v_j) - \delta(v_0, v_i)\right]$$

$$= \frac{1}{\alpha^k}\left[w(v_i, v_j)\right] = \frac{1}{\alpha^k}\left[w(P)\right].$$

Here we have used Corollary 3.2.3 and Chapter 3(section 3.3), which proves that the weighted shortest path is monotonically decreasing when $\alpha \to \infty$ and has a lower bound. This shows that the shortest path is monotonically decreasing.

References

[1] R. K. Ahuja, T. L. Magnanti and J. H. Orlin. Network Flows: Theory, Algorithms and Applications, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] D. P. Bertsekas, Nonlinear Programming, Athena Scientific-1995.

[3] U. Blasum, W. Hochstattler, C. Moll and H. Rieger, Using Network-Flow Techniqnes To Solve An Optimization Problem From Surface-Physics, J. Phy. A: Math. Gen. Vol. 29 (1996), pp-459-463.

[4] L. Caccetta and M. Kraetzl, Blocking Probabilities Of Certain Classes Of Channel Groups, Vishwa International Publications, H. No. 10-3/79, N. V. Layout Gulbagra 585103, India, pp-69.

[5] H. P. Christos and S. Kenneth, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall. 1982.

[6] J. Clark and D. A. Holton. A First Look at Graph Theory, Allied Ltd., 1995.

[7] R. W. Cotle, J. S. Pang and R. E. Stone, The Linear Complementarity Problem, Academic Press, New York, 1992.

[8] Y. A. Korilis. A. A. Lazar, and A. Orda, Capacity Allocation Under Non-Cooperative Routing. IEEE Transactions on Automatic Control. Vol. 42, no.3 (1997), pp-309-325.

[9] O. L. Mangasarian and M. V. Solodou. A Linearly Convergent Descent Method For Strongly Monotone Complementarily Problems. Computational optimization and applications, Vol. 14, PP. 5-16, 1999.

[10] M. Molloy and B. Reed, Random Struct And Algorithms 6. 161(1995): Combinatorics. Prob. Comput. 7, 295(1998).

[11] D. Narsingh. Graph Theory with applications to engineering and computer science, Prentice Hall of India, New Delhi. 2001.

[12] M. E. J. Newman, Scientific Collaboration Networks, II. Shortest Paths, Weighted Networks And Centrality, Cornell University, Rhodes Hall. Ithaca, New York 14853, Published 28 Jnne 2001.

[13] S. Pettie and Ramachandran, A Shortest Path Algorithm For Real-Weighted Undirected Graphs, SIAM J. COMPUT. Vol. 34, no. 6(2005), pp-1398-1431.

[14] S. Pettie, A New Approach To All-Pairs Shortest Paths On Real-Weighted Graphs. Theoretical Computer Science 312, 1 (2004), 47-74.

[15] S. Pettie and Ramachandran. An Optimal Minimum Spanning Tree Algorithm, J. of the ACM, Vol. 49, no.1 (2002), pp-16-34.

[16] S. Pettie and Ramachandran, A Randomized Timework Optimal Parallel Algorithm For Finding A Miuimum Spanning Forest, SIAM J. COMPUT. Vol. 31, No. 6 (2002), pp. 1879–1895.

[17] S. Saunders and T. Takaoka, Efficient Algorithms For Solving Shortest Paths On Nearly Acyclic Directed Graphs, Proceedings of the 2005 Australasian symposium on Theory of computing, Vol. 41. pp 127 – 131.

[18] S. Sauuders and T. Takaoka, Improved Shortest Path Algorithms For Nearly Acyclic Graphs. Theoretical Computer Science 293, 3 (2003), 535-556.

[19] T. Takaoka, A Faster Algorithm For The All-Pairs Shortest Path Problem and Its Application. In Proc. COCOON (2004), Vol. 3106 of Lecture Notes in Computer Science, pp-278-289.

[20] T. Takaoka, Shortest Path Algorithms For Nearly Acyclic Directed Graphs, Theoretical Computer Science 203, 1 (1998), 143-150.

[21] H. C. Thomas, E. L. Charles and L. R. Ronald, Introduction to Algorithms. Prentice-Hall of India Private Limited, New Delhi-11001, 2001.

[22] K. Yamada, N. Yamashita and M. Fukushima, A New Derivative-Free Descent Method For The Nonlinear Complimentarily Problem, Nonlinear Optimization and Related Topics, Edited by G. Di Pillo and F. Giannessi, Kluwer Academic Publishers, Boston, Massachusetts (2000), pp-463-487.