甲-2807

SOME PROBLEMS ON NETWORK OPTIMIZATION

TETSUO ICHIMORI

20

SOME PROBLEMS
ON
NETWORK OPTIMIZATION

TETSUO ICHIMORI

CONTENTS

CHAPTER 1

INTRODUCTION

This thesis is concerned with combinatorial optimization problems on networks. Such combinatorial optimization problems arise in many fields, especially in operations research and computer science. Such problems were raised by Hitchcock, Kantorovitch, and Koopmans in the 1940's. Since then, vast amount of literatures have been published, and research works in these areas became more and more active in recent years.

Many optimization problems have been considered with respect to the shortest path, the minimum spanning tree and the maximum flow, since they are most fundamental and important for practical applications. We review these problems briefly and describe some notations necessary for the subsequent discussions.

A network or graph $G = (V, A)$ consists of vertex set V and arc set A. G is called directed (resp. undirected) if arcs are directed (resp. undirected). A path from vertex i to vertex j is defined to be a sequence of arcs

$$(v_1, v_2), (v_2, v_3), \dots , (v_{k-1}, v_k)$$

where $v_1 = i$ and $v_k = j$. The vertices v_1 and v_k are called end-vertices of the path. If $v_1 = v_k$, the path is called a cycle. If there is no repetition of vertices, then such a path is called simple. If there exists a path from vertex i to vertex j, we say vertex j is reachable from vertex i. A connected network is called a tree if it contains no cycle. A spanning tree is a tree that contains all vertices of G. If G contains a directed arc (i,j), then vertex i is called a parent of vertex j, by contrast vertex j a child of vertex i. We denote the set of all parents (resp. all children) of vertex i by A(i) (resp. B(i)). Let S be a subset of V, T be the complement of S in V, and let (S,T) denote the set of all arcs that lead from $i \in S$ to $j \in T$. The set (S,T) is called a cut.

Now we describe the three fundamental problems on networks.

(i) Shortest Path

Let $G = (V, A)$ be a directed network. Each arc (i,j) has length a_{ij}. The length of a path is defined to be the sum of lengths of the arcs which constitute the path. The shortest path problem is to find a path from a specified vertex to the other specified vertices with shortest length. For simplicity, all the other vertices are assumed to be reachable from the specified vertex.

(ii) Minimum Spanning Tree

Let $G = (V, A)$ be a connected and undirected network. Each arc (i,j) has weight w_{ij}. The weitht of a spanning tree is defined to be the sum of weights of the arcs which constitute the spanning tree. The minimum sapnning tree problem is to find a spanning tree with minimum weight.

(iii) Maximum Flow

Let $G=(V, A)$ be a directed network with source s and sink t. Namely, $A(s) = B(t) = \phi$. Each arc (i,j) has positive capacity $c(i,j)$. A function f defined on the arc set A is called a flow from s to t if it satisfies

$$\sum_{i\in B(s)} f(s,i) = v$$

$$\sum_{i\in A(j)} f(i,j) = \sum_{i\in B(j)} f(j,i) \qquad j\in V-\{s,t\}$$

$$\sum_{i\in A(t)} f(i,t) = v$$

$$0 \le f(i,j) \le c(i,j) \qquad (i,j)\in A.$$

The above number v is called the value of flow f. Each $f(i,j)$ is called an arc flow. The maximum flow problem is to find a flow of maximum value from s to t.

We will be concerned with the computational complexity (or time complexity) of algorithms. We say an algorithm has time complexity $O(Z)$ or an algorithm runs in time $O(Z)$ if the number of basic computer operations executed until termination is bounded from above by cZ for some constant c. Basic computer operations include additons, subtractions, multiplications, divisions, comparisons, memory accesses and pointer adjustments. An algorithm is called polynomially bounded if the above Z is a polynomial function in the input length of the problem.

Chapter 2 discusses the minimax flow problem. This is a variant of the maximum flow problem. The minimax flow problem is a problem of finding a maximum flow which minimizes the maximum value of any arc flow. Such a maximum flow is called a minimax flow. A minimax flow is desirable in the sense that the flow runs fairly along arcs of the network.

To solve this problem, we propose a solution procedure called the capacity expansion technique. This capacity expansion technique is very powerful in solving combinatorial optimization problems with minimax-type objectives (powerfullness of this technique is also shown in Chapter 3). We present a solution algorithm based on the capacity expansion technique. After that, we establish the validity and evaluate the complexity of the algorithm presented. Finally we summarize the results obtained and suggest further developments.

Chapter 3 discusses the weighted minimax flow problem. This problem is the same as the minimax flow problem in Chapter 2 except that each arc has a nonnegative weight. In other words, the weighted minimax flow problem is to find a maximum flow that minimizes the maximum value of any arc flow multiplied by its arc weight. There are two cases in this problem. One is the case that arc flows are required to be integers and the other is the case that they can be real numbers. We note that the optimal values for these two cases may be different. We present two algorithms for these two cases.

These two algorithms are distinct each other, different from the algorithm developed in Chapter 2. Both of the algorithms run in polynomial time.

As an application of weighted minimax flows, we consider a minimax sharing on a network with multiple sinks. A minimax sharing minimizes the maximum share that any sink receives. For this problem we give an efficient algorithm based on the capacity expansion technique.

Chapter 4 is concerned with some applications of shrtest paths. The first two are routing problems with the fuel limitation. We wish to route a vehicle through a traffic network with some vertices at which the vehicle can refuel completely. The vehicle can move only a limited distance without refueling. Let the limited distance be L. Here it should be noted that the vehicle is allowed to visit some refueling vertices for the caution of fuel exhaustion during the movement. The objective of the first routing problem is to determine the shortest route between the origin and the destination so that the vehicle can move without running out of fuel, where we assume that the value of L is fixed. In the second routing problem, let L be a variable. Here we refer to the origin as a depot. The objective thereof is to determine the minimum value of L so that the vehicle can travel to any vertex and return to the depot without running out of fuel. For these two routing problems we develop efficient algorithms.

The third is concerned with the staffing problem. At first sight this staffing problem seems to have no concern with the shortest path problem. However, we show that the staffing problem can be reduced to the shortest path problem.

The staffing problem is to find the minimum number of workers required to satisfy the weekly staffing requirements which may vary, depending upon the day of the week under the condition that each worker is allowed to be off his work for two consecutive days. A simple solution procedure is proposed for this staffing problem, in which the optimal solution can be obtained in closed form in terms of the values of requirements. The solution procedure is efficient in the sense that the computational effort is free from the values of requirements.

Finally, we treat the following location problem. Although this location problem also seems to have no concern with the shortest path problem at first sight, this location problem can be also reduced to the shortest path problem.

This location problem is called the multifacility minimax location problem with rectilinear distances. The rectilinear distance of two points in the plane is the sum of the differences of x and y coordinates of these two points. The travel cost between two facilities is defined to be the rectilinear distance, times a unit distance cost, plus a nonnegative fixed cost. When m old facilities are already located in the plane, n new facilities are to be located so that the maximum travel cost between any two facilities i.e. old and new, or new and new, may be minimized.

This problem is transformed into a parametric shortest path problem. This transformation plays a crucial role for the construction of an efficient algorithm with computational complexity $O(n \max(m \log m, n^3))$. For a special case though, the computational complexity is further reduced to $O(n \max(m, n^2))$.

Chapter 5 discusses a maximum flow problem with a special vertex called a check vertex and considers a minimum spanning tree problem with normal variates as weights. In the former problem, each unit of a flow must visit the check vertex. The objective thereof is to find a maximum flow subject to the check-vertex constraint. We will show that this flow problem can be reduced to the usual maximum flow problem. In the latter problem, the weight of each arc is assumed to be a random variable according to the normal distribution. We seek a spanning tree maximizing the probability that the sum of weights of arcs in the spanning tree is not greater than a given constant. An efficient algorithm is also given for this spanning tree problem.

CHAPTER 2

MINIMAX FLOWS

2.1 Introduction

This chapter considers a variant of the maximum flow problem. The problem considered here is to find a maximum flow which minimizes the maximum arc-flow value. Such a maximum flow is called a minimax flow. Minimax flows are desirable in the sense that flows run fairly along arcs of the networks.

Section 2 formulates this problem and proposes a solution algorithm. After the confirmation of validity and convergence of the algorithm in Section 3, an illustrative example is presented in Section 4. Finally Section 5 summarizes our results and suggests further developments.

2.2 Solution Procedure

Let $G = (V, A)$ denote a directed network, where V is the vertex set and A is the arc set. For convenience, arc $a = (i,j)$ may be denoted by 'a' instead of (i,j) and the capacity of arc a is denoted by $c(a)$. The minimax flow problem is to find a maximum flow f which

$$\text{minimize} \ \max_{a \in A} \ f(a),$$

where f(a) is an arc flow of arc a.

First we find a maximum flow and its corresponding minimum cut in G. Let v* denote the value of the maximum flow. Let the maximum arc-flow value in this minimum cut be c_1. Obviouly the value c_1 is the lower bound of the optimal value. If there is no arc having its arc-flow value more than c_1 in A, the optimal value is obtained. Otherwise, the capacities of the arcs with arc-flow values more than c_1 are reduced to c_1. And then, we again find a maximum flow and its minimum cut in the network with reduced capacities. Let the value of this maximum flow be v^1 and corresponding minimum cut be C^1. If $v^1 = v^*$, then c_1 is the optimal value for our problem. Otherwise, we define the new lower bound c_2 such that

$$v^* = \sum_{a \in C^1} \min(c_2, c(a)).$$

The same procedure is repeated until we obtain v*. We refer to this procedure as the capacity expansion technique.

Algorithm 2.1

Step 1. Set $i = 0$. Find a maximum flow f*, its value v* and the corresponding minimum cut C^0 in G. Find an arc having the maximum arc-flow value c_1 in the cut C^0.

Step 2. Set $i = i + 1$. Construct the associated network G_i whose vertices and arcs are the same as the original network G except that the capacity of each arc is changed to $c_i(a) = \min(c_i, c(a))$. Find a maximum flow f^i, its value v^i and the corresponding minimum cut C^i in the associated network G_i.

Step 3. If $v^i = v^*$, then go to Step 5. Otherwise, go to Step 4.

Step 4. Determine c_{i+1} by the equation

$$v^* = \sum_{a \in C^i} \min(c_{i+1}, c(a)). \tag{2.1}$$

Return to Step 2.

Step 5. Stop. (The current flow is optimal.)

Remark 1. The maximum flow f^{i-1} in G_{i-1} can be used as an initial flow to find a maximum flow f^i in G_i.

Remark 2. When capacities and arc-flow values are all confined to integers, each c_i is replaced by $\lceil c_i \rceil$ where $\lceil y \rceil$ is the least integer greater than or equal to y.

Remark 3. The following function F of c

$$F(c) = \sum_{a \in C} \min(c, c(a))$$

is a concave piecewise linear function with $|C|$ breaking points. The value c_{i+1} of (2.1) is determined as the intersection of the straight line $F(c) = v^*$ with the curve $F(c) = \sum_{a \in C} \min(c, c(a))$. (See Fig. 2.1.)

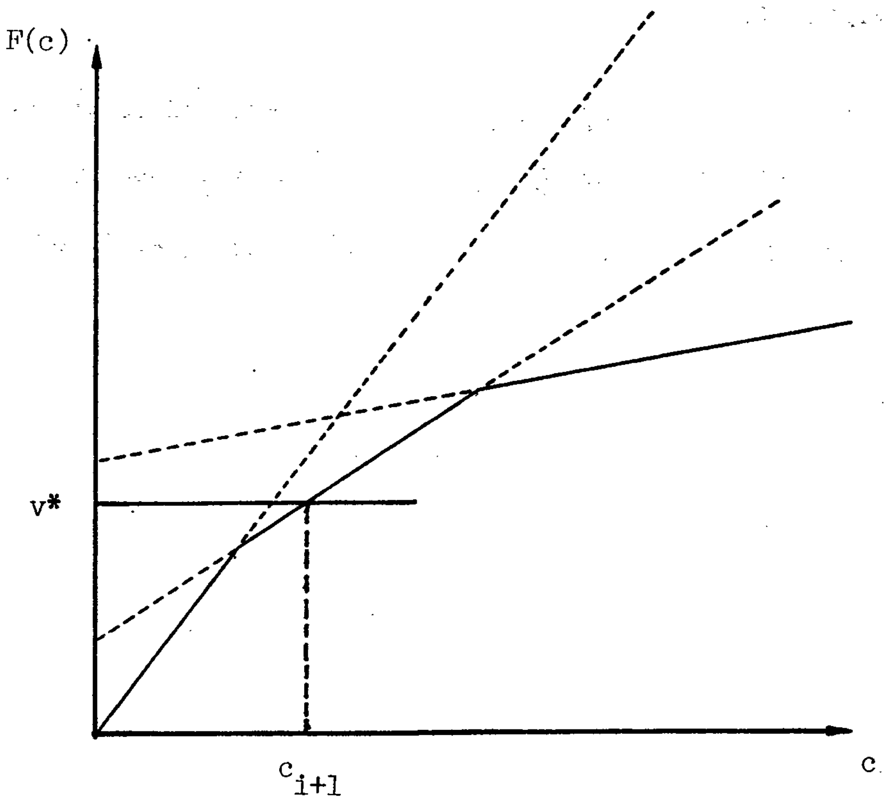


Fig. 2.1

2.3 Validity of Algorithm 2.1 and Time Complexity

Each arc a is called "active" or "dead" according to either $c_i(a) = c_i$ or $c_i(a) = c(a)$. Let m_i denote the number of active arcs in C^i for $i \geq 1$ and let m_0 denote the number of arcs in C^0.

Lemma 2.1 The relation

$$v^{i+1} - v^i \geq (v^* - v^i)m_{i+1}/m_i$$

holds for $i = 1, 2, \ldots, k-1$, where k is the number of iterations until termination.

Proof. Consider the minimum cut C^i for $i \leq k-1$. Since $v^i < v^*$, it would be desired to divide the difference $(v^* - v^i)$ equally among m_i active arcs. However there may be some arc a among m_i active arcs such that

$$c_i + (v^* - v^i)/m_i > c(a).$$

Hence we have

$$c_{i+1} \geq (v^* - v^i)/m_i + c_i.$$

In G_{i+1}, the definition of m_{i+1} implies that there are m_{i+1} arcs in C^{i+1} whose capacities are augmented by $c_{i+1} - c_i$. The above two observation imply

$$v^{i+1} - v^i \geq (c_{i+1} - c_i)m_{i+1} \geq (v^* - v^i)m_{i+1}/m_i. \quad \text{Q.E.D.}$$

Lemma 2.2 The inequalities

$$m_0 > m_1 > m_2 > \dots > m_{k-1} \geq m_k \qquad (k \geq 2)$$

hold.

Proof. It follows from Lemma 2.1 that $m_{i+1}/m_i \geq 1$ implies $v^{i+1} \geq v^*$. However, this is impossible except $i = k-1$. Hence we have $m_i > m_{i+1}$ for $i = 1, 2, \dots, k-2$ and $m_{k-1} \geq m_k$.

Suppose $m_0 \leq m_1$. From the definitions of m_0, m_1, c_1 and v_1, we have $m_0c_1 \geq v^*$ and $m_1c_1 \leq v^1$. Since $k \geq 2$, we have $v^1 < v^2 \leq v^*$. From the above, we have $m_0c_1 \leq m_1c_1 \leq v^1 < v^*$, i.e., $m_0c_1 < v^*$, contrary to the above relation $m_0c_1 \geq v^*$. Hence the result follows.

Q.E.D.

Lemma 2.3 Algorithm 2.1 terminates after at most m_0 iterations.

Proof. The integrality of m_i and Lemma 2.2 prove $k \leq m_0$. Q.E.D.

Theorem 2.4 The time complexity of Algorithm 2.1 is $O(m_0 n \max(n^2, m \log n))$[†], where $n = |V|$ and $m = |A|$.

Proof. Computing the values of each v^i and v^* involves $O(n \max(n^2, m \log n))$, see [1], [2]. The other parts of Algorithm 2.1 do not need so much time. Hence the result follows. Q.E.D.

Proposition 2.5 At the termination, Algorithm 2.1 finds an optimal solution to the minimax flow problem.

Proof. Termination condition of Step 4 of Algorithm 2.1 itself proves the validity of Algorithm 2.1. Q.E.D.

2.4 Exampe

Consider the network of Fig. 2.2. The pair of the arc capaci-

[†] The logarithm is to the base 2 throughout this thesis.

ties and the arc flows are shown next to the arcs and the numbers in the circles are vertex numbers.

Algorithm 2.1 proceeds as follows.

(Initialization $i = 0$)

Step 1: $v^* = 15$. $C^0 = \{(5,6), (5,t), (5,7)\}$. $c_1 = 15$.

(1st iteration $i = 1$)

Step 2: An associated network G_1 is constructed and it is shown in Fig. 2.3, where of the pair of numbers written next to an arc, the first number is the capacity $c_i(a)$ and the second one is the arc-flow value. The maximum flow value $v^1 = 10$ and corresponding minimum cut $C^1 = \{(s,2), (s,4)\}$ are obtained.

Step 3: Since $v^i < v^*$, go to Step 4.

Step 4: By Remark 3, $F(c) = \min(c, 20) + \min(c, 30)$ and $F(c) = v^* = 15$ determine $c_2 = 7.5$. Moreover by Remark 2, $c_2 = 7.5$ is changed into $c_2 = 8$.

(2nd iteration $i = 2$)

Step 2: G_2 is constructed, $v^2 = 13$ and the minimum cut $C^2 = \{(2,5), (2,3), (s,4)\}$. (See Fig. 2.4.)

Step 3: Since $v^i < v^*$, go to Step 4.

Step 4:
$$\begin{aligned} v^* &= \min(c,3) + \min(c,2) + \min(c,30) \\ &= 3 + 2 + \min(c, 30) \\ &= 5 + \min(c, 30). \end{aligned}$$

Therefore $c_3 = 10$.

(3rd iteration $i = 3$)

Step 2: Fig. 2.5 shows G_3 and $v^3 = 15$.

Step 3: $v^3 = v^* = 15$. Go to Step 5.

Step 5: Terminate. The optimal flow pattern with maximum arc-flow value 10 (= f(s,4)) while an initial maximum arc-flow value is 15, is obtained and shown in Fig. 2.5.

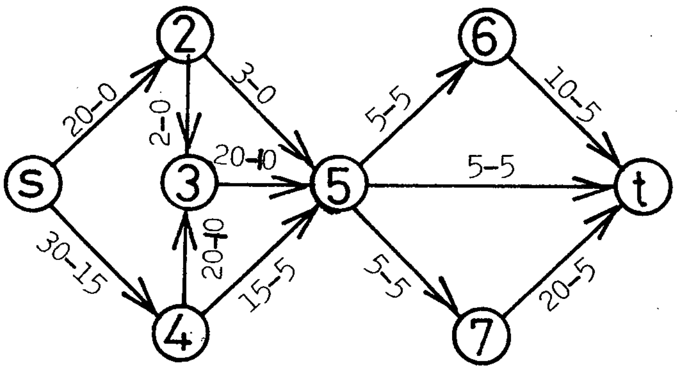


Fig. 2.2 The original network G. $v^* = 15$, $m_0 = 3$, max f(a) = 15.

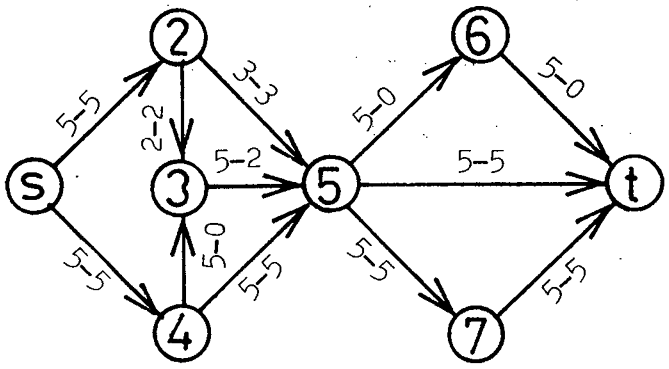


Fig. 2.3 The first associated network G_1. $v^1 = 10$, $m_1 = 2$, $c_2 = 8$.

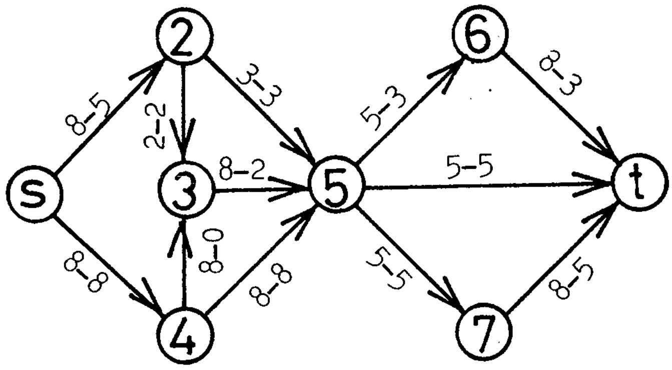


Fig. 2.4 The second associated network G_2. $v^2 = 13$, $m_2 = 1$, $c_3 = 10$.

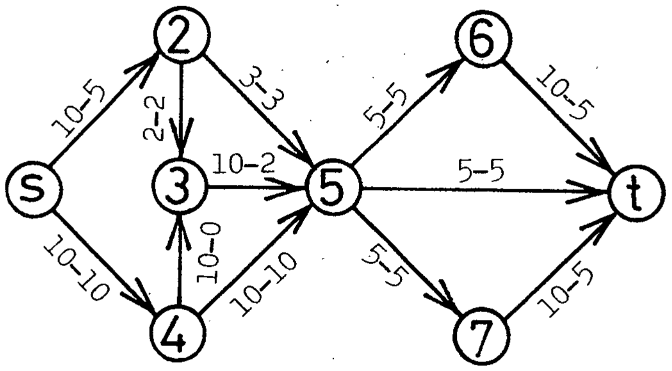


Fig. 2.5 The third associated network G_3. $v^3 = v^*$, $m_3 = m_2$, $c_4 = 10$, max $f(a) = 10$.

2.5 Conclusion

The problem considered in this chapter seems to be a new type of the maximum flow problem as far as the author knows. Algorithm 2.1 solves the minimax flow problem in at most $O(m_0 n \max(n^2, m \log n))$ time. It may be expected, however, the algorithm behaves more efficiently since m_i decreases rapidly as i and v^i increase. Moreover it may be also accelarated if v^{i-1} is used as an initial flow in G_i as stated in Remark 1.

Therefore the computational complexity of $O(m_0 n \max(n^2, m \log n))$ may be more improved. Furthermore our problem may be considered as a special case of so-called multipurpose network flow problem. These dirctions of research also are to be pursued.

References

[1] Karzanov, A.V.,"Determining the maximum flow in a network by the method of preflows", Soviet Mathematics Doklady 15 (1974) 434-437.

[2] Sleator, D.,"An O(nm log m) algorithm for maximum network flow," Ph. D. Dissertation, Stanford University, November, 1980.

CHAPTER 3

WEIGHTED MINIMAX FLOWS

3.1 Introduction

In this chapter, we consider two weighted versions of minimax flow problem. The first version is concerned with the intger-valued flows. Under the assumption that capacities and weights are all integers, we develop a polynomially bounded algorithm. The other version is concerned with the real-valued flows. For this version as well, we develop a polynomially bounded algorithm.

The weighted minimax flow problem is to find a maximum flow that minimizes the maximum value of respective arc flow, multiplied by its arc weight. Here it should be noted that the optimal value when flows are restricted to be integers is greater than or equal to the optimal value when flows can be real numbers.

After presenting two polynomially bounded algorithms, we consider the sharing problem which is an application of weighted minimax flows.

With respect to other network flow problems related to weighted minimax flows, we have the time transportation problem [2], [7], [8], [17] and the storage management problem [16]. However, the time transportation problem is discussed only in case of a bipartite graph and the storage management problem is discussed in case of a linear graph which can be drawn in a one-dimentional space. In addition, both objective functions are optimized only with respect to the arc weights. On the other hand, our weighted minimax flow problem is generalized from the following two viewpoints.

The first is the viewpoint where we consider the general network. The second is the viewpoint where we consider both arc weights and arc-flow values in our objective function.

3.2 Statement of the Problem

Let $G = (V, A)$ be a network where V is the vertex set and A is the arc set. Let s be the source and t be the sink. With each arc (i,j) we associate capacity $c(i,j) > 0$ and weight $w(i,j) \geq 0$. A flow is denoted by f. Given a flow f, we refer to $f(i,j)$ as the arc flow of arc (i,j). Then the weighted minimax flow problem is:

$$\min \left[\max_{(i,j)\in A} w(i,j)f(i,j) \right]$$

s.t.

$$\sum_{i\in A(j)} f(i,j) = \sum_{i\in B(j)} f(j,i) \qquad j \neq s,\ t$$

$$\sum_{i\in B(s)} f(s,i) = v^*$$

$$\sum_{i\in A(t)} f(i,t) = v^*$$

$$0 \le f(i,j) \le c(i,j) \qquad (i,j) \in A$$

where v^* denotes the value of a maximum flow from source s to sink t, and $A(j)$ (resp. $B(j)$) is the parent (resp. child) set of vertex j. The capacity of a cut (S,T) is denoted by $c(S,T)$ where S denotes a subset of V such that $s \in S$ and T denotes the complement of S in V such that $t \in T$. The value of $c(S,T)$ is defined as follows:

$$c(S,T) = \sum_{(i,j)\in(S,T)} c(i,j).$$

3.3 Solution Procedure for Weighted Minimax Integer-Valued Flows

In this section, we consider the weighted minimax flow problem where capacities, weights and flows are integers. For solving this problem, we use the capacity expansion technique presented in the preceding chapter and binary search (e.g. see [11]) instead of the minimax cost path in [9].

For a nonnegative parameter D, we define the new capacity $c'(i,j)$ for each arc (i,j) as follows:

$$c'(i,j) = \min(\lfloor D/w(i,j)\rfloor, c(i,j)) \qquad \text{if } w(i,j) \neq 0$$

$$c'(i,j) = c(i,j) \qquad \text{otherwise}$$

where $\lfloor x\rfloor$ denotes the largest integer y that satisfies $y \le x$. We define the new capacity of cut (S,T) by

$$c'(S,T) = \sum_{(i,j)\in(S,T)} c'(i,j).$$

For any cut (S,T) we have $c(S,T) \geq c'(S,T)$ obviously. It follows from the famous max-flow min-cut theorem [5] that $v^* = \min c(S,T)$. Let

$$v(D) = \min_{(S,T) \text{ in all cuts}} c'(S,T).$$

Note that $v(D)$ is a nondecreasing function of D. Let D^* be the minimum value of D such that $v(D) = v^*$. Then we have

$$f(i,j) \leq \min(\lfloor D^*/w(i,j)\rfloor, c(i,j)) \leq \lfloor D^*/w(i,j)\rfloor$$
$$\leq D^*/w(i,j),$$

i.e., $$f(i,j)w(i,j) \leq D^* \qquad \text{for any arc } (i,j).$$

Hence D^* is the optimal value for the weighted minimax integer-valued flow problem. We will find D^* by binary search.

Consider the network with capacity $c'(i,j)$ instead of $c(i,j)$ for each arc (i,j). The new network is denoted by $G(D)$. Note that if D is infinite, then $G(D)$ and G are identical. In addition, note that $v(D)$ is the value of a maximum flow in $G(D)$. If the value $v(D)$ is strictly less than v^*, then the value D is smaller than D^*. Otherwise, the value D is larger than or equal to D^*. Let c (resp. w) be the maximum value of $c(i,j)$ (resp. $w(i,j)$) of any arc (i,j). Then we have $D^* \leq cw$. For two real-valued D' and D'' such that $D' < D^* \leq$

D'', i.e., $v(D') < v^*$ and $v(D'') = v^*$, define the interval $I = D'' - D'$. If $I \le 1$, then we have $D^* = \lfloor D'' \rfloor$ obviously. A weighted minimax integer-valued flow is given by a maximum flow in $G(D^*)$. Let I_p denote the interval after applying binary search p times. Then we obtain $I_p \le cw/2^p$. If $p \ge \log(cw)$, then we have $I_p \le 1$. Define $P = \lceil \log cw \rceil$ where $\lceil x \rceil$ denotes the smallest integer y that satisfies $y \ge x$. After maximum flows are found P times, D^* is obtained. Since one maximum flow in $G(D)$ is found in time $c(n,m) = O(n \max(n^2, m \log n))$, see [10], [15], where $n = |V|$ and $m = |A|$, D^* is obtained in time $O(Pc(n,m))$.

If we redefine the value of P by

$$P = \lceil \log (\max_{(i,j)\in A} c(i,j)w(i,j) \rceil,$$

we can obtain D^* faster.

Algorithm 3.1

```
Step 1.  Compute $P = \lceil \log(\max_{(i,j)\in A} c(i,j)w(i,j) \rceil$, and $v^*$, the max-
         imum flow value in G.
         Set $D' = 0$, $D'' = P$, $I = P$ and $D = P/2$.
Step 2.  If $I \le 1$, then $D^* = \lfloor D'' \rfloor$ and stop. Otherwise, go to Step 3.
Step 3.  Construct $G(D)$ and compute $v(D)$.
Step 4.  If $v(D) < v^*$, then $D' = D$. Otherwise, set $D'' = D$.
Step 5.  Set $I = D'' - D'$, $D = (D' + D'')/2$. Return to Step 2.
```

Remark. The optimal flow which gives D^* is determined by a maximum flow in $G(D^*)$.

3.4 Solution Procedure for Weighted Real-Valued Flows

In this section, we consider the weighted minimax flow problem where capacities, weights and flows are real numbers. For this problem, we use the capacity expansion technique and the strategy for solving ratio minimization problems in [12].

For a nonnegative parameter D, we define the new capacity $c'(i,j)$ for each arc (i,j) as follows:

$$c'(i,j) = \min(D/w(i,j),\ c(i,j)) \qquad \text{if } w(i,j) \neq 0$$

$$c'(i,j) = c(i,j) \qquad \text{otherwise.}$$

Let $v(D)$, D^* and $G(D)$ be the same as in the preceding section.

We define $d(i,j) = c(i,j)w(i,j)$ for each arc (i,j), set D' to the maximum value of $d(i,j)$ such that $v(d(i,j)) < v^*$, and set D'' to the minimum value of $d(i,j)$ such that $v(d(i,j)) = v^*$. Then we have $D' < D^* \leq D''$.

The computation of a maximum flow involves operations (+, −, min) between two data. In general, data are integers or reals. However we consider the case where data are of linear form $aD + b$ (a and b are reals). Then for two data, $aD + b$ and $a'D + b'$ where $a \geq a'$, we have the following:

$$(aD + b) + (a'D + b') = (a + a')D + (b + b')$$

$$(aD + b) - (a'D + b') = (a - a')D + (b - b')$$

$$\min(aD + b,\ a'D + b') = \begin{cases} aD + b & D > B \\ a'D + b' & D \le B, \end{cases}$$

where B is an intersection point of aD + b and a'D + b', i.e.,

$$B = (b' - b)/(a - a') \qquad a \neq a'.$$

If D is restricted so that $D > B$ or $D \le B$, then these operations (+, -, min) between two data, aD + b and a'D + b' can be treated in a similar manner as for integers or reals.

Since data aD + b involve parameter D, we call them parametric. When "parametrically" is referred to in this thesis, we intend "by the operations between parametric data."

Next we seek a maximum flow parametrically in G(D), $D' < D \le D''$. Note that at the start of this computation, data are D/w(i,j), $D' < D \le D''$ or c(i,j). During the computation, the min operations may demand the restriction of D, i.e., $D' < D \le B$ or $B < D \le D''$, in order to determine min(aD + b, a'D + b') = aD + b or a'D + b'. In this case, we put pause to the parametric computation of the maximum flow in G(D), $D' < D \le D''$. Instead of it, we compute v(B). Note that in this computation all data are constant but not parametric. If

$v(B) < v^*$, then we have $B < D^* \le D''$; otherwise $D' < D^* \le B$. Hence we replace D' or D'' by B according to the value of $v(B)$ and resume the preceding parametric computation. At the end of this process we get the value of the maximum flow in $G(D)$, which is of form $a^*D + b^*$. If $a^* \neq 0$, then $D^* = (v^* - b^*)/a^*$; otherwise $D^* = D''$.

Algorithm 3.2

Step 1. Set $D' = \max\{d(i,j) \mid v(d(i,j)) < v^*\}$.

Set $D'' = \min\{d(i,j) \mid v(d(i,j)) = v^*\}$.

Step 2. Compute a maximum flow parametrically in $G(D)$, $D' < D \le D''$, from the start or the recent point of restriction of D. If there are no more restrictions of D, then we have the value of the maximum flow (denoted by $a^*D + b^*$) and go to Step 4; otherwise let the parametric computation of the maximum flow make a pause at the point of restriction of D and go to Step 3.

Step 3. Let B denote an intersection point involved by the current restriction of D where $D' < B \le D''$. Compute $v(B)$. If $v(B) < v^*$, then set D' to B; otherwise set D'' to B, and go back to Step 2 to resume the parametric computation of the maximum flow.

Step 4. If $a^* \neq 0$, compute $D^* = (v^* - b^*)/a^*$; otherwise $D^* = D''$. Stop.

Remark. The optimal flow which gives D* is determined by a maximum flow in G(D*).

Time Complexity of Algorithm 3.2

It takes $O(n^3 \log n)$ time to obtain D' and D" in Step 1 by binary search, which is not so important here since the time bound is negligible as compared with that in Step 3. The maximum flow problem is solvable in time $c(n,m) = O(n \max(n^2, m \log n))$. Therefore Step 2 takes c(n,m) time though it has many interruptions. (Note that the computation of finding a maximum flow in G(D) parametrically is done exactly once.) Since it is obvious that the number of interruptions in Step 2 is bounded by the number of comparisons of parametric data and since all of the comparisons must be included in the c(n,m) operations for finding the maximum flow in Step 2. Step 3 is iterated at most c(n,m) times, where each iteration needs the computation of v(B) whose time complexity is c(n,m). Hence Step 3 needs a total of $(c(n,m))^2$ time. Therefore the overall time complexity of Algorithm 3.2 is $(c(n,m))^2$.

3.5 Illustration of Algorithm 3.2

Consider the network G = (V, A) depicted in Fig. 3.1, where the first number of the label on each arc is its capacity and the second number is its weight. The value of the maximum flow in G = (V, A), v* is 7. A description of the algorithm is given below.

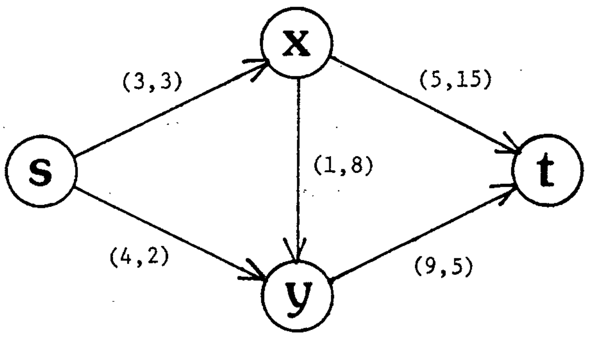


Fig. 3.1 G= (V, A). The label (c(i,j), w(i,j)) denotes (capacity, weight) of arc (i,j).

Step 1: Arc (s,x) gives D' = 9 and arc (y,t) gives D" = 45 since $v(9) = 12/5 < 7$ (see Fig. 3.2) and $v(45) = 7$ (see Fig. 3.3), respectively.

Step 2: We compute the maximum flow parametrically in G(D), $9 < D \leq 45$, (see Fig. 3.4). We choose the path s-y-t and send the largest possible flow through this path. This involves the computation min(4, D/5). Since

$$\min(4, D/5) = \begin{cases} D/5 & 9 < D \leq 20 \\ 4 & 20 < D \leq 45, \end{cases}$$

the restriction of D is demanded. We put pause to this computation with respect to G(D) and go to Step 3 for determining whether min (4, D/5) = 4 or D/5.

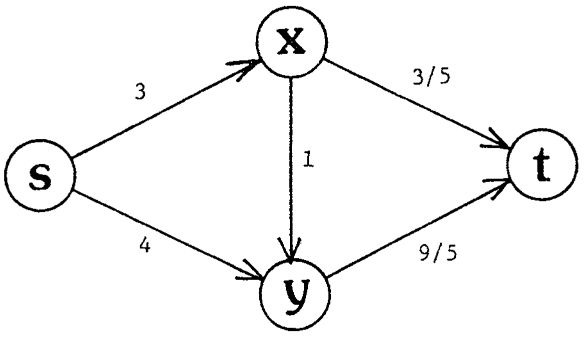


Fig. 3.2 Network G(9). The number on each arc is its capacity, $c'(i,j)$ which is given by the following equation: $c'(i,j) = \min(9/w(i,j), c(i,j))$. Arcs (x,t) and (y,t) determine the value of v(9) as:

$v(9) = 3/5 + 9/5 = 12/5$.

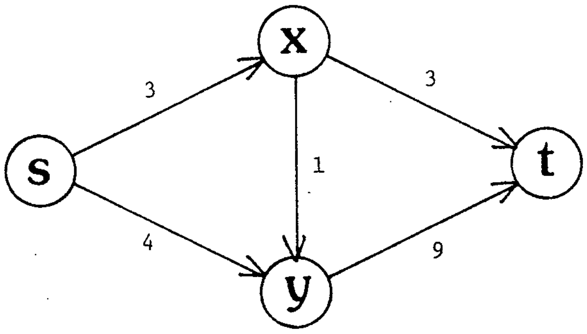


Fig. 3.3 Network N(45). Each capacity: c'(i,j) = min(45/w(i,j), c(i,j)). Arcs (s,x) and (s,y) give the value of v(45) as v(45) = 3 + 4 = 7.

Step 3: We get B = 20 and v(20) = 16/3. See Fig. 3.5 where c'(i,j) = min(20/w(i,j), c(i,j)). Since v(20) < 7, we have D' = 20. Note $20 < D^* \leq 45$.

Step 2: Step 3 above yields min(4, D/5) = 4, $20 < D \leq 45$. Let f(i,j) denote the arc flow on arc (i,j). Then f(s,y) = f(y,t) = 4. Next we choose the path s-x-y-t from G(D). Then the following computation is necessary: min(3, 1, D/5 - 4). Since

$$\min(1, D/5 - 4) = \begin{cases} D/5 - 4 & 20 < D \le 25 \\ 1 & 25 < D \le 45, \end{cases}$$

again we pause to this computation and go to Step 3.

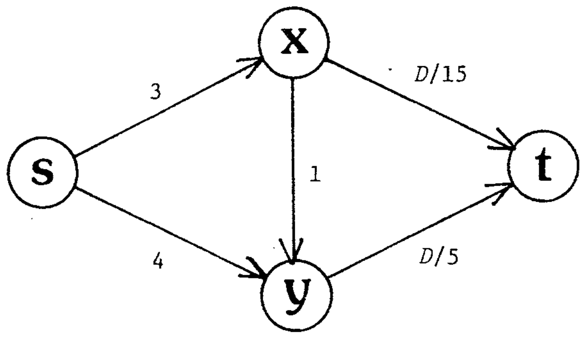


Fig. 3.4 G(D) where the number on each arc is its capacity: c'(i,j) = min(D/w(i,j), c(i,j)), 9 < D ≤ 45.

Step 3: Since B = 25 and v(25) = 20/3 < 7, we set D' = 25. Note 25 < D* ≤ 45.

Step 2: Since we get min(3, 1, D/5 - 4) = 1, 25 < D ≤ 45, we have f(s,x) = f(x,y) = 1 and f(y,t) = 4 + 1 = 5. Next we choose the path s-x-t from G(D), which needs the computation min(3 - 1, D/15). Here the following relation holds:

$$\min(2, D/15) = \begin{cases} D/15 & 25 < D \le 30 \\ 2 & 30 < D \le 45. \end{cases}$$

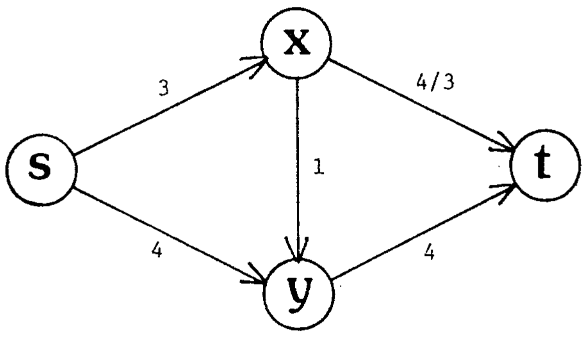


Fig. 3.5 Network G(20). $v(20) = 4/3 + 4 = 16/3$ since arcs (x,t) and (y,t) are saturated.

Step 3: Since $B = 30$ and $v(30) = 7$, we set $D'' = 30$. Note $25 < D^* \le 30$.

Step 2: Since $\min(2, D/15) = D/15$, $25 < D \le 30$, we have $f(s,x) = 1 + D/15$, $f(x,t) = D/15$. Now the maximum flow is obtained since there is no path capable of sending a flow, (see Fig. 3.6). The maximum flow value, $aD^* + b^*$ is $D/15 + 5$.

Step 4: $D^* = 15(7 - 5) = 30$, (see Fig. 3.7).

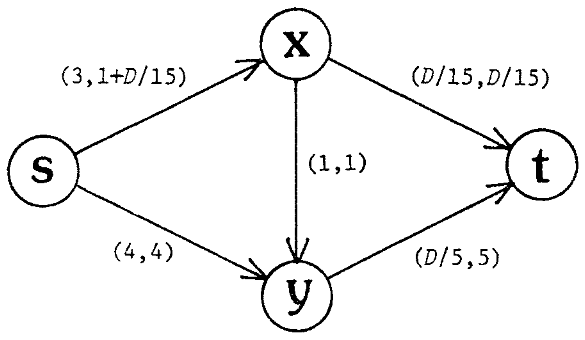


Fig. 3.6 Network G(D). First label is arc capacity and second label is arc flow. Arcs (s,y), (x,y) and (x,t) are saturated.

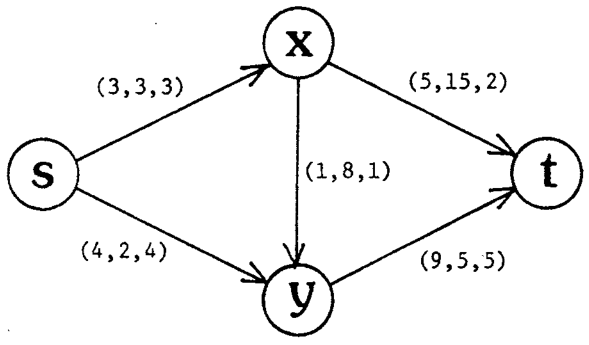


Fig. 3.7 Optimal flow with D* = 30. The label on each arc is (capacity, weight, flow).

3.6 Sharing Problems

We discuss sharing problems as an application of weighted minimax flows.

It is very important to distribute a given quantity of resources equitably in many situations. However the distribution system may not allow the perfect equitableness when capacity constraints are present. Brown [1] has considered the problem which has a maximin objective function. However, it is also important to consider the problem which has a minimax objective function, since it is often not easy to determine which objective function realizes the equitableness much more.

For convenience, let Brown's problem be referred to as maximin sharing and the problem posed above as minimax sharing. We consider optimal sharing which is not only maximin sharing but also minimax sharing.

In Sections 3.7 and 3.8, we discuss these two sharings in detail. As an illustration, consider the following coal-strike problem, supplied by Brown.

During a prolonged coal strike, some "non-union" mines could be producing. In this case, an important problem would be to distribute equitably the limited coal supply produced by the "non-union" mines among the power companies. Since power companies vary in size, it would not be a good idea to give each power company the same amount of coal.

Thus our distribution scheme must consider the relative size of the companies. Let T represent the set of power companies and let w(t) represent the amount of coal normally used by power company t during a given period of time.

If f(t) represents the amount of coal which our distribution scheme assigns to power company t, then f(t)/w(t) is the proportion of the amount of demand supplied to power company t by the distribution scheme. An equitable distribution scheme might attempt to

(1) equalize these proportions (perfect sharing),

(2) maximize the smallest proportion (maximin sharing),

(3) minimize the largest proportion (minimax sharing),

(4) not only maximize the smallest proportion but also minimize the largest proportion (optimal sharing), or

(5) maximize lexicographically the $|T|$-tuple of the numbers f(t)/w(t), $t \in T$ arranged in order of increasing magnitude (lexicographically optimal, or lex-optimal sharing).

Note that Brown considers the preceding problem only in the context of maximin sharing (2).

Lex-optimal sharing (5) was first proposed by Megiddo [12] and then solved efficiently in [13] in time $O(|T|n^2m)$ where $|T|$, n and m are respectively the numbers of sinks (corresponding to power companies in the coal-strike problem), vertices and arcs in the distribution network.

Recently Fujishige [6] has reduced the time bound for (5) to $O(|T|c(n,m))$, where $c(n,m)$ is the time bound for the maximum flow problem.

In the following section, we propose a new sharing i.e., minimax sharing (3), and combine it with maximin sharing (2) considered by Brown to produce optimal sharing (4). Algorithms given here run in time $O(|T|c(n,m))$ which is the same as Fujishige's.

We note that lex-optimal sharing (5) is always optimal sharing (4). Therefore judging from the time complexity obtained by Fujishige, our results seem to be covered by those due to him.

However our strategy used here (i.e., the application of capacity expansion) is essentially new and has the advantage of being applicable to almost all combinatorial problems with minimax-type objectives, including storage management, minimax flows, bottleneck assignments, minimax matchings, etc. (e.g. see [11]).

Finally, we note that each of problems (1)-(4) has a very straightforward linear programming formulation: hence it can be solved by the simplex method as well.

3.7 Minimax Sharing

Let $G = (V, A)$ be a distribution network where V is the vertex set and A is the arc set. Let s be a supply vertex, $T \subset V$ be the set of sinks, and let the other vertices be intermediate. Each arc (i,j) has capacity $c(i,j) > 0$. Let $f(i,j)$ be a flow on arc (i,j), $w(t)$ be a positive weight associated with sink t, and $f(t)$ be the total flow into sink t.

The minimax sharing problem is:

$$\min[\max_{t \in T} f(t)/w(t)]$$

s.t.

$$\sum_{i \in A(j)} f(i,j) = \sum_{i \in B(j)} f(j,i) \qquad j \in V - T - \{s\}$$

$$\sum_{i \in B(s)} f(s,i) = v^*$$

$$\sum_{i \in A(t)} f(i,t) = f(t) \qquad t \in T$$

$$0 \leq f(i,j) \leq c(i,j) \qquad (i,j) \in A$$

where v^* denotes a supply value at s.

For simplicity we assume that v^* is the value of the maximum flow from s to T. Otherwise, it is sufficient to introduce the super source σ and the arc (σ,s) with capacity $c(\sigma,s) = v^*$. Here we present an algorithm for the above problem.

Algorithm 3.3

Step 0. Add an arc to G from every sink t to a super sink u.

Set $i = 0$. Compute $D_1 = v^*/(\sum_{t \in T} w(t))$.

Step 1. Set $i = i + 1$. Giving each added arc (t,u) capacity $c_i(t,u) = D_i w(t)$, find a maximum flow f_i from s to u, its value v_i and its corresponding minimum cut $(X_i, \overline{X}_i)$. (Note, for $i \geq 2$, we use f_{i-1} as an initial solution.)

Step 2. If $v_i < v^*$, then go to Step 3. Otherwise stop; the current D_i is the optimal value of the minimax sharing.

Step 3. Compute $D_{i+1} = (v^* - v_i)/(\sum_{t \in T_i} w(t)) + D_i$ where $T_i = T \cap X_i$. Return to Step 1.

Validity and Complexity

Given a nonnegative parameter D, define the new network $G^*(D) = (V^*, A^*)$, which is used in Algorithm 3.3 from the original distribution network $G = (V, A)$:

$$V^* = V \cup \{u\},$$

$$A^* = A \cup \{(t,u) \mid t \in T\}.$$

The capacity $c^*(i,j)$ of arc (i,j) in the set A^* is defined as follows:

$$c^*(i,j) = c(i,j) \qquad \text{for arc } (i,j) \text{ in the set } A \qquad (3,1)$$

which is referred to as constant capacity,

$$c^*(i,j) = Dw(t) \qquad \text{for arc } (t,u),\ t \in T \qquad (3.2)$$

which is referred to as parametric capacity. We may write $c^*(i,j;D) = c^*(i,j)$ to express the value of D explicitly.

Define the capacity of cut $(X,\overline{X})$, $s \in X$ and $t \in \overline{X}$, in $G^*(D) = (V^*, A^*)$ as follows:

$$c^*(X,\overline{X};D) = \sum_{(i,j) \in (X,\overline{X})} c^*(i,j).$$

It follows from (3.1) and (3.2) that

$$c^*(X,\overline{X};D) = \sum_{(i,j) \in (X,\overline{X}) \cap A} c(i,j) + D(\sum_{t \in T \cap X} w(t)) \qquad (3.3)$$

since $(X,\overline{X}) - A = \{(t,u) \mid t \in T \cap X\}$.
The function (3.3) is linear in D whose slope is $\sum_{t \in T \cap X} w(t) \geq 0$, where if $T \cap X = \phi$, then define $\sum_{t \in T \cap X} w(t) = 0$.

Let $v(D)$ be the value of the maximum flow from s to u in $G^*(D)$. That is,

$$v(D) = \min c^*(X,\overline{X};D), \qquad (3.4)$$

where the min operation is taken over all cuts $(X,\overline{X})$, $s \in X$ and $t \in \overline{X}$, in $G^*(D)$. Then, we have easily

Lemma 3.1 If $D_i < D'$, $T_i \neq \phi$ and $(X,\overline{X})$ is any cut in $G^*(D)$, then

$$c^*(X_i,\overline{X}_i;D_i) < c^*(X_i,\overline{X}_i;D'), \qquad (3.5)$$

$$v_i = v(D_i) = c^*(X_i,\overline{X}_i;D_i), \qquad (3.6)$$

$$c^*(X_i,\overline{X}_i;D_i) \le c^*(X,\overline{X};D_i), \tag{3.7}$$

$$c^*(X_i,\overline{X}_i;D_{i+1}) = v^*, \tag{3.8}$$

where D_i, D_{i+1}, X_i, X_{i+1}, $\overline{X}_i$, $\overline{X}_{i+1}$, T_i and v_i are defined in Algorithm 3.3.

Let $f^*(i,j)$ be a flow on arc (i,j) in $G^*(D)$. For flow f^*, define flow f and $f(t)$, $t \in T$ in $G = (V, A)$ as follows:

$$f(i,j) = f^*(i,j) \qquad \text{for } (i,j) \in A \tag{3.9}$$

$$f(t) = f^*(t,u) \qquad \text{for } t \in T. \tag{3.10}$$

Let $\overline{D}$ be the minimum value of D satisfying $v(D) = v^*$. That is,

$$v(D) = v^* \qquad \text{for } D \ge \overline{D}, \tag{3.11}$$

$$v(D) < v^* \qquad \text{for } D < \overline{D}. \tag{3.12}$$

Since we have $\overline{D} = c^*(t,u;\overline{D})/w(t)$ for all $t \in T$ from definition (3.2) and since we have $f(t) = f^*(t,u) \le c^*(t,u;\overline{D})$ from (3.10) and the fact that $f^*(t,u)$ are flows on arcs (t,u), we have $f(t)/w(t) \le \overline{D}$ for all $t \in T$. This means that $\overline{D}$ is the optimal value of the minimax sharing problem.

Lemma 3.2 If $v_i < v^*$, then $T_i \ne \phi$.

Proof. Suppose $T_i \neq \phi$. Then $c^*(X_i, \overline{X}_i; D)$ is constant, i.e.

$$v_i = v(D_i) = c^*(X_i, \overline{X}_i; D_i) = c^*(X_i, \overline{X}_i; D). \quad (3.13)$$

Since the capacity of any cut in $G = (V, A)$ is not less than v^*, (3.13) yields $v_i \geq v^*$ in contradiction to the assumption $v_i < v^*$. Hence $T_i \neq \phi$. Q.E.D.

Theorem 3.3 For $\overline{D}$, D_i and D_{i+1} defined in Algorithm 3.3, $v(D_i) < v^*$ implies $\overline{D} \geq D_{i+1} > D_i$.

Proof. For minimum cut $(X_i, \overline{X}_i)$ of $G^*(D_i)$ consider the function $c^*(X_i, \overline{X}_i; D)$. Since $v(D_i) < v^*$ and $v_i = v(D_i)$, we have $T_i \neq \phi$ from Lemmma 3.2, which implies $\sum_{t \in T_i} w(t) > 0$. Hence from the definition of D_{i+1} we have $D_{i+1} > D_i$.

Suppose $\overline{D} < D_{i+1}$. Noticing that $T_i \neq \phi$, we have from (3.5) $c^*(X_i, \overline{X}_i; \overline{D}) < c^*(X_i, \overline{X}_i; D_{i+1})$. From (3.8) we obtain $c^*(X_i, \overline{X}_i; \overline{D}) < v^*$. Since $v(\overline{D}) = v^*$, there must exist a flow of value v^* in $G^*(D)$, which induces a contradiction. Therefore $\overline{D} > D_{i+1}$. Q.E.D.

Corollary 3.4. $D_1 \leq \overline{D}$.

Theorem 3.5 If $v_{i+1} < v^*$, then $|T_{i+1}| < |T_i|$ for all $i \geq 1$.

Proof. Note that the values of $c_i(t,u)$ increase with respect to i where $c_i(t,u) = c^*(t,u;D_i)$. Let f_i be a flow of value v_i in the network $G^*(D_i)$. At $(i + 1)$st iteration, since the capacities of arcs in $(X_i,\overline{X}_i) \cap A$ do not change, the values of flows on arcs (t,u), $t \in T - T_i$ are the same as at i-th iteration, so long as we use flow f_i as the initial flow to get flow f_{i+1}, where we note that flows on arcs (t,u), $t \in T - T_i$ go through arcs $(X_i,\overline{X}_i) \cap A$ with constant capacities. Accordingly if a sink t is contained in the set $\overline{X}_i$, then the sink t is contained in the set $\overline{X}_j$ for $j > i$. That is, $T_i \supseteq T_{i+1}$.

Suppose $T_i = T_{i+1}$. Consider two minimum cuts $(X_i,\overline{X}_i)$ in $G^*(D_i)$ and $(X_{i+1},\overline{X}_{i+1})$ in $G^*(D_{i+1})$. For thses two cuts, consider the functions $c^*(X_i,\overline{X}_i;D)$ and $c^*(X_{i+1},\overline{X}_{i+1};D)$. Since $T_i = T_{i+1}$, thses two linear functions have the same slope. However from (3.6) and (3.7) we have

$$v_i = c^*(X_i,\overline{X}_i;D_i) \le c^*(X_{i+1},\overline{X}_{i+1};D_i). \tag{3.14}$$

And, from (3.6), (3.7), (3.8) and assumption $v_{i+1} < v^*$, we have

$$v_{i+1} = c^*(X_{i+1},\overline{X}_{i+1};D_{i+1}) < c^*(X_i,\overline{X}_i;D_{i+1}) = v^*. \tag{3.15}$$

Thus, (3.14) and (3.15) imply a contradiction. Hence $T_i \supset T_{i+1}$.

Q.E.D.

From Theorem 3.5 the maximum number of iterations of Algorithm 3.3 (the last value of i) is $|T|$. If $T_i \neq \phi$, then we have $v_i = v^*$ from Lemma 3.2, since it is impossible that $v_i > v^*$. Hence Theorems 3.3 , 3.3 and Corollary 3.4 show the validity of Algorithm 3.3.

Remark. Though Algorithm 3.3 starts with $D_1 = v^*/(\sum_{t \in T} w(t))$, it is clear from Theorem 3.3 that it may start with any value of D such that $v(D) < v^*$.

Let c(n,m) denote the upper bound of the number of operations to find a maximum flow f_i at Step 1. Then from Theorem 3.5 it follows that Algorithm 3.3 runs in time $O(|T|c(n,m))$.

The best values of c(n,m) are $O(n^3)$ for dense networks (see Karzanov [10]) and O(nm log n) for sparse networks (see Sleator [15]), where $n = |V|$ and $m = |A|$. Since in general the distribution networks are sparse and $|T| << n$, we may safely say that the computational complexity of $O(|T|nm \log n)$ is very good.

The maximin sharing problem of Brown is solved in time $O(|T|n^5)$ by using the results of Edmonds and Karp (see [4]). However, its time complexity can be also reduced to $O(|T|nm \log n)$, since Brown's algorithm also runs in time $O(|T|c(n,m))$.

3.8 Optimal Sharing

Comparing the maximin and minimax sharing problems, the former is equivalent to the network flow problem with arcs (t,u) having lower bounds on the flows (see Christofides [3, p. 299] and Brown [1]), while the latter is equivalent to the problem with arcs (t,u) having upper bounds on the flows (i.e., capacities).

Though it is often not easy to determine which criteron is more suitable to realize equitableness, maximin or minimax, it is clear that the sharing which meets both criteria (i.e., optimal sharing) is better.

Let $\underline{D}$ and $\overline{D}$ denote the optimal values of the maximin and the minimax sharing problems, respectively. Then an optimal sharing is such that $\underline{D} \le f(t)/w(t) \le \overline{D}$ for all $t \in T$.

Proposition 3.6 $\underline{D} \le \overline{D}$.

Proof. Given a maximin sharing $(\underline{f}(i,j), \underline{f}(t))$ where each $\underline{f}(i,j)$ is a flow on arc (i,j) in $G = (V, A)$ and given a minimax sharing $\overline{f}(t)$, we have by definition $\underline{D} \le \underline{f}(t)/w(t)$ and $\overline{D} \ge \overline{f}(t)/w(t)$ for all $t \in T$. Since $v^* = \sum_{t \in T} \underline{f}(t) = \sum_{t \in T} \overline{f}(t)$, $\underline{D} \le \overline{D}$ holds. Q.E.D.

Remark. If $\underline{D} = \overline{D}$, then either sharing is a perfect sharing.

Consider the problem of finding a feasible flow $\hat{f}$ in $G^*(\infty)$ such that

$$\sum_{i \in A(j)} \hat{f}(i,j) = \sum_{i \in B(j)} \hat{f}(j,i) \qquad j \in V^* - \{s,u\},$$

$$\sum_{i \in B(s)} \hat{f}(s,i) = \sum_{i \in A(u)} \hat{f}(i,u),$$

$$0 \le \hat{f}(i,j) \le c(i,j), \qquad (i,j) \in A$$

$$\underline{D}w(t) \le \hat{f}(t,u) < \infty, \qquad t \in T$$

where A(j) and B(j) are defined for G*(D).

In fact a maximin sharing ($\underline{f}(i,j)$, $\underline{f}(t)$) gives a feasible flow $\hat{f}$ in G*(∞) where $\hat{f}(i,j) = \underline{f}(i,j)$, $(i,j) \in A$ and $\hat{f}(t,u) = \underline{f}(t)$, $t \in T$, which implies the following Proposition 3.7.

Proposition 3.7 There exists a feasible flow $\hat{f}^*$ in G*(∞) such that $\hat{f}^*(t,u) = \underline{D}w(t)$ for all $t \in T$.

Proof. See [3, pp. 299-300]. Q.E.D.

From Proposition 3.7 the flow $\hat{f}^*$ is feasible in $G^*(\underline{D})$ as well. We consider the case $\underline{D} < \overline{D}$. In this case, from (3.12) $v(\underline{D}) < v^*$. Hence by Remark of preceding section, even if we replace D_1 by $\underline{D}$ in Algorithm 3.3, Lemmas 3.1 and 3.2 and Theorem 3.3 are still valid. If we use $\hat{f}^*(i,j)$ as an initial solution for arc (i,j) in the set A* when finding a maximum flow f_1 in Algorithm 3.3, it follows that from the trivial property of the maximum flow algorithm that the values of f(t) on arc (t,u), where each t is adjacent to super sink u, do not decrease with respect to i. Hence at any iteration we have $f(t)/w(t) \ge \underline{D}$ for all $t \in T$.

Since Algorithm 3.3 can find a minimax sharing such that $f(t)/w(t) \leq \overline{D}$, $t \in T$, which also satisfies $f(t)/w(t) \geq \underline{D}$, $t \in T$, the resulting minimax sharing is an optimal sharing.

Algorithm 3.4

Step 0. Compute $\underline{D}$ (and $\underline{f}(i,j)$ and $\underline{f}(t)$). If $\underline{f}(t)$ gives a perfect sharing, then stop. Otherwise continue.

Step 1. Find the flow $\hat{f}^*$ in $G^*(\infty)$. Set $D_1 = \underline{D}$, $i = 0$.
Go to Step 1 of Algorithm 3.3. (Use $\hat{f}^*$ as an initial solution when computing f_1).

Algorithm 3.4 has the same order of complexity as Algorithm 3.3 since Algorithm 3.4 employs Algorithm 3.3 at most once and Brown's algorithm once and $\hat{f}^*$ is found in time $c(n,m)$.

References

[1] Brown, J.R.,"The sharing problem", Operations Research 27 (1979) 324-340.

[2] Burkard, R.E.,"A general Hungarian method for the algebraic transportation problem", Discrete Mathematics 22 (1978) 219-232.

[3] Christofides, N., Graph Theory: an algorithmic approach, Academinc Press, New York, 1975.

[4] Edmonds, J. and R. Karp,"Theoretical improvements in algorithmic efficiency for network flow problems," Journal of the Association for Computing Machinery 19 (1972) 248-264.

[5] Ford, L.R., Jr. and D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, 1962.

[6] Fujishige, S.,"Lexicographically optimal base of a polymatroid with respect to a weight vector," Mathematics of Operations Research 5 (1980) 186-196.

[7] Garfinkel, R.S. and M.R. Rao,"The bottleneck trasportation problem," Naval Research Logistics Quarterly 18 (1971) 465-472.

[8] Hammer, P.L.,"Time minimizing transportation problems," Naval Research Logistics Quarterly 16 (1960) 345-357.

[9] Ichimori, T., M. Murata, H. Ishii and T. Nishida,"Minimax cost flow problem," Technology Reports of the Osaka University 30 (1980) 39-44.

[10] Karzanov, A.V.,"Determining the maximum flow in a network by the method of preflows," Soviet Mathematics Doklaky 15 (1974) 434-437.

[11] Lawler, E.L., Combinatorial Optimization: networks and matroids, Holt, Rinehart and Winston, New York, 1976.

[12] Megiddo, N.,"Optimal flows in networks with multiple sources and sinks," Mathematical Programming 7 (1974) 97-107.

[13] Megiddo, N.,"A good algorithm for lexicographically optimal flows in multi-terminal networks," Bulletin of the American Mathematical Society 83 (1977) 407-409.

[14] Megiddo, N.,"Combinatorial optimization with rational objective functions," Mathematics of Operations Research 4 (1979) 414-424.

[15] Sleator, D.,"An O(nm log n) algorithm for maximum network flow," Ph. D. Dissertation, Stanford University, 1980.

[16] Stanat, D.F. and G.A. Magó,"Minimizing maximum flows in linear graphs," Networks 9 (1979) 333-361.

[17] Swarc, W.,"Some remarks on the time transportation problem," Naval Research Logistics Quarterly 18 (1971) 475-485.

CHAPTER 4

SOME APPLICATIONS OF SHORTEST PATHS

4.1 Introduction

In this chapter, we consider some applications of shortest path problems already solved. The first is concerned with the following routing problem. We wish to route a vehicle through a traffic network where there are only certain vertices at which the vehicle can refuel completely. Suppose that the vehicle can move a specified distance without refueling. How can we determine the shortest path between a specified origin and a specified destination along which the vehicle can move without running out of fuel on the way?

The straightforward method for this problem is to compute repeatedly the shortest paths from the origin to the destination with known algorithms until the desired path is obtained. In view of the computational complexity, however, this method is not recommendable.

We develop here a polynomially bounded algorithm for finding the optimal route. The time complexity thereof is $O(pn^2)$ where n is the number of vertices in a network and p is that of refueling vertices.

The second is also concerned with the routing problem with fuel limitation. In a traffic network, there are some refueling vertices, including a vehicle depot. Let us assume a vehicle starts from the depot and can move distance L at most after filling up with fuel. On receiving a service call from any vertex, it travels from the depot to the vertex and returns to the depot. During this trip, the vehicle may visit some refueling vertices for the caution of fuel exhaustion even though the route may not be shortest.

The problem considered here is to determine the minimum value of L subject to the constraint that the vehicle can travel to any vertex of G and return to the depot without exhausting fuel on the way. An efficient algorithm is developed for this problem and the computational complexity is estimated.

The third is a problem of finding the minimum number of workers required to satisfy the weekly staffing requirements b_j $(j = 1,2,\ldots, 7)$ which may vary, depending upon the day of the week, assuming that each worker is allowed to be off his work for two consecutive days each week. A simple solution procedure is proposed for this problem, by which the optimal solution is obtained in closed form in terms of b_j $(j = 1,2,\ldots, 7)$. By this solution procedure, the computational effort is free from the values of b_j $(j = 1,2,\ldots, 7)$.

The last is concerned with the multifacility minimax location problem with rectilinear distances. Rectilinear distance of two points in the plane is the sum of the differences of x and y coordinates of these two points. The travel cost between two facilities is defined to be the rectilinear distance, times a unit distance cost, plus a nonnegative fixed cost. When m facilities are already located in the plane, n new facilities are to be located so that the maximum travel cost beween any two facilities i.e. old and new, or new and new, may be minimized.

This problem can be transformed into a parametric shortest path problem. This transformation plays a crucial role to construct an improved algorithm with computational complexity $O(n \max(m \log m, n^3))$. For a special case, the computational complexity is further reduced to $O(n \max(m, n^2))$.

4.2 Vehicle Routing Problem (I) with Constant Fuel Limitation

Consider a traffic network $G = (V, A)$ where V is the vertex set and A is the directed arc set. Length $a_{ij} > 0$ is associated with each arc from vertex i to j.

When a path is referred to, we intend a directed path which is permitted to traverse vertices and arcs more than once. The length of path is defined to be the sum of the lengths of the arcs along that path.

A vehicle starts at a specified origin s and arrives at a specified destination t, visiting some refueling vertices on the way. Suppose that the vehicle can move distance L at most, after filling up with fuel. Suppose in addition that there are p refueling vertices, among which are the origin and destination vertices. Then the routing problem is the problem of determining the shortest path between the origin and the destination along which the vehicle can move without running out of fuel. In the sequel we assume for simplicity that such shortest path always exists.

Before describing our algorithm, we will present the following two difficulties which we may encounter in our routing problem, but not in the conventional shortest path problems.

The first difficulty is illustrated by the example shown in Fig. 4.1, where L = 10, s, r and t are refueling vertices, and the number on each arc is its length. A vehicle with L = 10 can go along the path s–v–r–v–t with the cycle v–r–v, but can not go along the simple path s–v–t because of the fuel limitation.

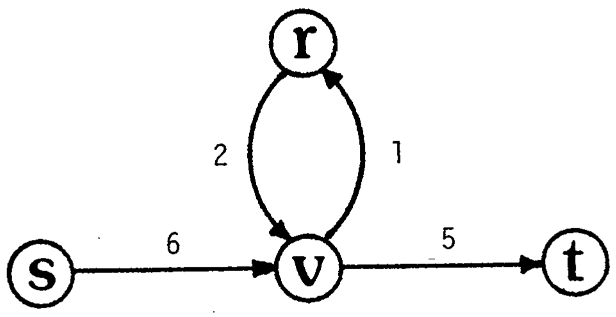


Fig. 4.1

The above illustration implies that an optimal route may contain cycles in our routing problem, different from the conventional shortest path problems.

The second difficulty is illustrated by the example shown in Fig. 4.2, where L = 10, s, r and t are refueling vertices, and where the numbers of the arcs are their lengths. The optimal route between s and t is the path s-x-r-z-t. However the optimal route between s and z is not s-x-r-z, but s-x-y-z. The second illustration implies that the socalled "Principle of Optimality" is not satisfied in this context. Here we note that these two difficulties are related each other in some sense.

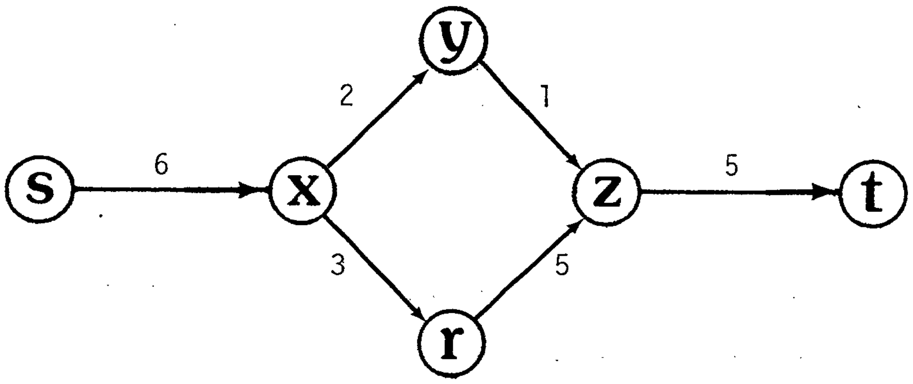


Fig. 4.2

These two illustrations imply that it is impossible to solve our routing problem by applying one of the conventional shortest path algorithms directly, i.e., by applying it only once.

Now we discuss a solution method for this routing problem. First consider the problem of finding shortest paths between all pairs of p refueling vertices in the traffic network $G = (V, A)$. This problem can be easily solved by applying Dijkstra's shortest path algorithm [5] p times. Since the time complexity of Dijkstra's algorithm is $O(n^2)$, the complexity of the problem posed above is $O(pn^2)$ where $n = |V|$. Let u_{ij} denote the length of shortest path P_{ij} from refueling vertex i to refueling vertex j.

Suppose $u_{ij} > L$ for some pair i and j, for the moment. If P_{ij} does not contain any refueling vertices except vertices i and j, then the vehicle can not move from i to j, namely it must stop somewhere on the way from i to j. Otherwise, P_{ij} must contain at least one refueling vertex, say k. Now we have

$$u_{ij} = u_{ik} + u_{kj},$$

and the path P_{ij} consists of two shortest paths P_{ik} and P_{kj}, by the "Principle of Optimality".

Thus, all the values u_{ij} such that $u_{ij} > L$ can be discarded. After discarding u_{ij} larger than L, all the remaining u_{ij} are $\leq$

L, which means that the vehicle can traverse any path without stopping.

Let us construct the reduced network $G' = (V', A')$ which contains an arc (i,j) in A' and vertices i, j in V' for each $u_{ij} \leq L$. Here we note that $s, t \in V'$ by assumption. In the network G' formed above, assign to each arc (i,j) length $u_{ij} \leq L$. Then it is clear that the 1st shrtest path from s to t in G' has its corresponding shortest path from s to t in the original network G and that the lengths of both shortest paths are equal. Since the 1st shortest path in G' contains only the arcs of lengths $\leq L$, its corresponding shortest path in G contains only the sub-paths such that the lengths between successive refueling vertices thereon are less than or equal to L. Accordingly the vehicle can go through the network G along this path, which corresponds to the 1st shortest path in G'. The computational time for the 1st shortest path in G' is $O(p^2)$, noticing $|V'| = p$.

Algorithm 4.1 and Time Complexity

Step 1. Compute u_{ij} for all pairs of p refueling vertices. --- $O(pn^2)$.

Step 2. Discard u_{ij} if $u_{ij} > L$ and construct G'. ---$O(p^2)$.

Step 3. Compute the 1st shortest path in G'. ---$O(p^2)$. (This path corresponds to the desired one in G.)

Remark 1. The theoretical time bound required for the entire algorithm is therefore proportional to pn^2.

Remark 2. Although the actual construction of the optimal route in G is not explicitly described here, such an issue is relatively straightforward and hence omitted.

4.3 Vehicle Routing Problem (II) with Variable Fuel Limitation

Consider a traffic network $G = (V, A)$ where V is the set of vertices, A is the set of undirected arcs. Assume G is connected. Length $a_{ij} > 0$ is associated with each arc (i,j) between vertices i and j. We assume that there are p refueling vertices, including a depot. The others are referred to as non-refueling vertices. A vehicle proceeds from the depot to any vertex and therefrom back to the depot, visiting some refueling vertices on the way. Suppose that the vehicle can move distance L at most, after filling up with fuel. Then the routing problem here is to determine the minimum value of L such that the vehicle can travel to any vertex and return to the depot without running out of fuel on the way. Or equivalently,

(P1) $\min L$

subject to the following constraints:

(i) each non-refueling vertex is connected to at least one refueling vertex through a path of length $\leq L/2$;

(ii) each refueling vertex is reachable from the depot along a path on which the lengths between successive refueling vertices are $\leq L$.

Now we present a solution procedure for this problem. Let our refueling vertices be $x_1, x_2, \ldots, x_p$, let x_1 be the depot, and let the other non-refueling vertices be $y_1, y_2, \ldots, y_q$, where $q = |V| - p$.

First, consider the problem of finding shortest paths between all pairs of p refueling vertices in $G = (V, A)$. This problem can be easily solved by applying Dijkstra's shortest path algorithm, p times repeatedly. Hence the problem posed here is solvable in time $O(pn^2)$, where $n = |V|$.

Let u_{ij} denote the length of the shortest path formed as above between refueling vertices x_i and x_j. Denote the shortest path itself as P_{ij}. Construct the reduced network $G' = (V', A')$ which contains as undirected arc (i,j) in A' and vertices i, j in V' for each u_{ij}. In the network G' thus formed, we assign length u_{ij} to each arc (i,j).

Secondly, consider the problem of finding a minimum (or shortest) spanning tree of the reduced network G'. This problem can be solved in time $O(p^2)$ by implementing Prim-Dijkstra's minimum spanning tree algorithm [4, p. 138].

Let T be the resulting minimum spanning tree. And let $\max_{(i,j)\in T} u_{ij} = u^*$ and L^* denotes the minimum value of L in (P1).

Theorem 4.1 L^* is greater than or equal to u^*.

Proof. First we show that the shortest path of G, corresponding to each arc contained in the minimum spanning tree T of G', does not contain any refueling vertex except for both end-vertices of the path. Assume on the contrary that the shortest path P_{ij} of G contains some refueling vertex, say x_k, strictly between refueling vertices x_i and x_j. Then we have the relation $u_{ij} = u_{ik} + u_{kj}$. Consider the cycle $\{(i,j), (j,k), (k,i)\}$ of G'. Since $u_{ij} > u_{ik}$ and $u_{ij} > u_{kj}$ follow from the above relation (recall that every arc lenth is positive), the arc (i,j) cannot be contained in T, the minimum spanning tree of G'. Hence each arc (i,j) contained in T corresponds to the shortest path P_{ij} between refueling vertices x_i and x_j in the original network G which does not contain any refueling vertex strictly between x_i and x_j.

Next it should be noted that the vehicle must traverse the shortest path of G without refueling which corresponds to the longest arc with length u^* in T. Since a minimax i.e. bottleneck spanning tree is exactly a minimum spanning tree (e.g. see [4, p. 139]), the conclusion follows. Q.E.D.

Now, consider the problem of finding a path between each non-refueling vertex y_k and its nearest refueling vertex x'_k in the original network G, letting v_k denote the path length. This problem is also solved by implementing Dijkstra's shortest path algorithm q times, hence in time $O(qn^2)$.

Let $\max_{k=1,2,\ldots,q} v_k = v^*$. Then apparently L^* must be greater than or equal to $2v^*$. Hence we have the following important fact.

Theorem 4.2 $L^* = \max(u^*, 2v^*)$.

From the fact we have mentioned above, we can present the following algorithm together with each computational time bound.

Algorithm 4.2 and Computational Complexity

Step 1. Compute u_{ij} for all pairs of p refueling vertices. --- $O(pn^2)$.

Step 2. Construct the reduced network G'. ---$O(p^2)$.

Step 3. Compute the minimum spanning tree T. ---$O(p^2)$.

Step 4. Set $u^* = \max_{(i,j) \in T} u_{ij}$.

Step 5. Compute v_k for q non-refueling vertices. ---$O(qn^2)$.

Step 6. Set $v^* = \max_{k=1,2,\ldots,q} v_k$.

Step 7. Set $L^* = \max(u^*, 2v^*)$.

Remark. The theoretical time bound for the entire algorithm is $O(pn^2) + O(qn^2) = O(n^3)$. Although the actual construction of routes for L* in G is not explicitly described here, such an issue is relatively straightforward and hence omitted. (See Corollary 4.4 below.)

We conclude this section with the following theorem which may save the computation of the values of some v_k's (although this is not embodied in Algorithm 4.2) and its corollary.

Theorem 4.3 Let arc (i,j) be contained in the minimum spanning tree T of G'. Let the corresponding shortest path P_{ij} of G contain a non-refueling vertex y. Then the nearest refueling vertex of y is x_i or x_j in G.

Proof. Suppose on the contrary that the nearest vertex is x_k which is distinct from both x_i and x_j. Let d(i,j) denote the shortest path length between vertices i and j in G. (Note d(i,j) = d(j,i)). Then we have $d(y, x_k) < d(y, x_i)$ and $d(y, x_k) < d(y, x_j)$ by assumption. Hence we obtain

$$d(y, x_k) + d(y, x_i) < u_{ij}$$

$$d(y, x_k) + d(y, x_j) < u_{ij},$$

noticing $d(x_i, y) + d(y, x_j) = u_{ij}$. Since

$$u_{ik} \le d(x_i, y) + d(y, x_k)$$

$$u_{jk} \le d(x_j, y) + d(y, x_k),$$

we have $u_{ik} < u_{ij}$, $u_{jk} < u_{ij}$.

Consider the cycle $\{(i,j), (j,k), (k,i)\}$ in the reduced network G'. Then the above relations $u_{ij} > u_{ik}$, $u_{ij} > u_{jk}$ contradict with the fact that the arc (i,j) is contained in T, the minimum spanning tree of G'. Hence the result follows. Q.E.D.

Corollary 4.4 There exists a spanning tree of G such that the routes in the tree give L*.

Remark. The routes given by a spanning tree of G are not shortest paths in general (but feasible for L*) from the depot to the vertices. Such shortest paths may constitute cycles, as shown before.

Consider the numerical example of (P1) shown in Fig. 4.3, where the number on each arc is the length. The reduced network G' is as shown in Fig. 4.4. Since the minimum spanning tree T of G' is $\{(1,2), (1,3), (3,4)\}$, $u^* = 5$. Since the nearest refueling vertices of y_1, y_2, y_3, y_4, y_5 and y_6 are x_2, x_1, x_1, x_1, x_2 and x_4, respectively, we have $v^* = 2$. Hence $L^* = 5$. The routes for $L^* = 5$ are shown in Fig. 4.5. Here note that a spanning tree results.

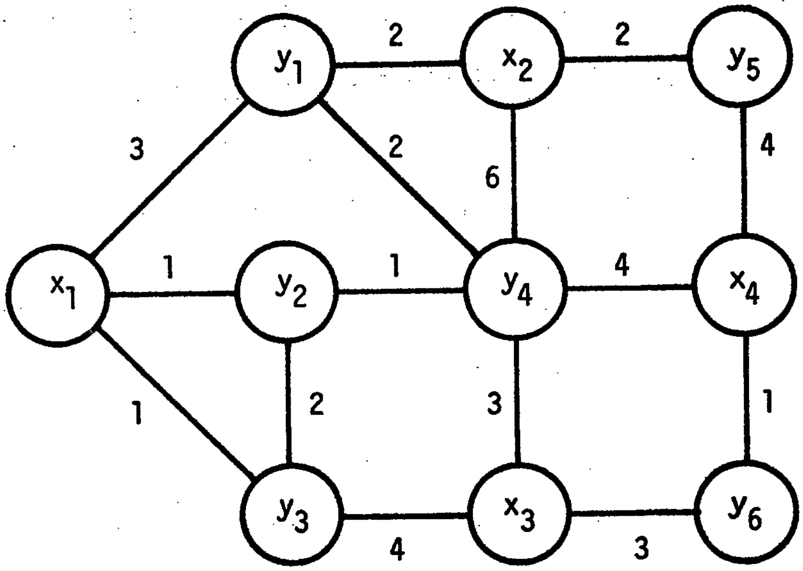


Fig. 4.3

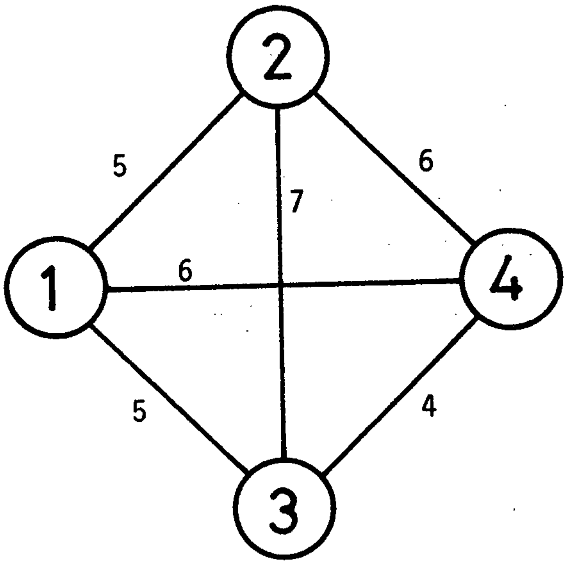


Fig. 4.4

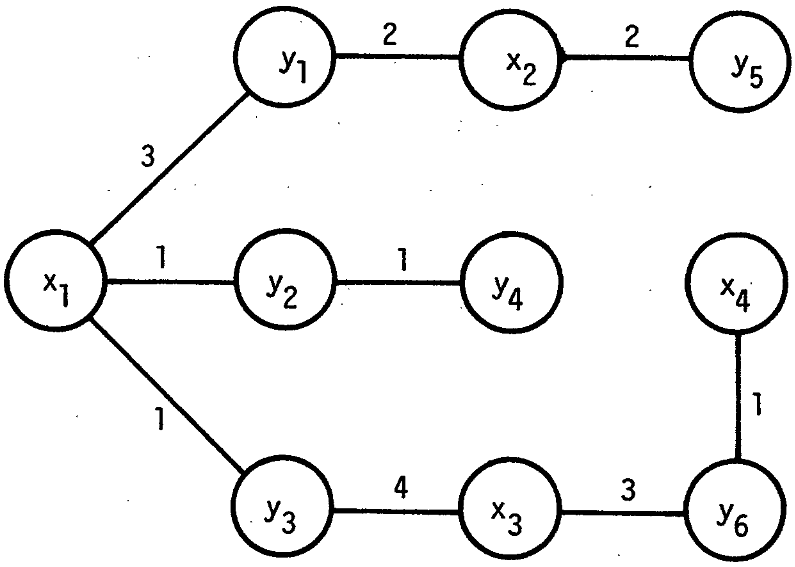


Fig. 4.5

4.4 Cyclic Staffing Problem with Two Consecutive Days Off Each Week

Usually staffing requirements are cyclic in week, i.e., the requirements may vary within the week but repeat weekly. Employees want to have two consecutive days off each (assume that the first and the last days of the week are consecutive). Thus it is worth while considering the following cyclic staffing problem:

$$\text{minimize} \quad x_1 + x_2 + \dots + x_7 \tag{4.1}$$

$$\text{subject to} \quad \begin{bmatrix} 1&0&0&1&1&1&1 \\ 1&1&0&0&1&1&1 \\ 1&1&1&0&0&1&1 \\ 1&1&1&1&0&0&1 \\ 1&1&1&1&1&0&0 \\ 0&1&1&1&1&1&0 \\ 0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \geq \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix},$$

where each column of the above matrix represents a work pattern, x_i is the number of workers on work pattern i, and b_j denotes the number of workers required on day j.

The cyclic staffing problem in which two consecutive days off are permitted to workers each week, has been studied by Tibrewala et al. [22], Baker [1], Bartholdi and Ratliff [3], Bartholdi et al. [2], Karp and Orlin [14] and others. The cyclic staffing problem considered in [2] is rather generalized and the objective is to minimize the linear cost of assigning workers to an n-day cyclic schedule where each worker is to be idle for consecutuive k days and on duty for the other (n - k) days. In [14], the problem with the unit costs and with n-day cyclic requirements is considered. (Note that if the costs are all unity, then the objective is equal to minimizing the number of workers required to satisfy the requirements. In fact, this is true of problem (4.1).) The algorithm in [14] can give the optimal solution efficiently in the sense that the computational effort is independent of the values of b_j for $j = 1, 2, \ldots, 7$, while the others cannot do efficiently in this sense. Since the algorithm is developed to slove the n-day problem with consecutive $k < n$ days off rather than the seven-day version, it may not be appropriate to the seven-day problem or to hand calculations.

In addition, considering that the seven-day problem is of wide applicability and has been studied for a long time, it would be important to show a specialized solution method where the optimal solution is expressed in terms of b_j only. This would be most helpful to the staff of the personnel department. The calculation can be carried out by hand easily and the time required for it is indifferent to the values of b_j.

By using a transformation given in [2], Karp and Orlin [14] have reduced the cyclic n-day staffing problem with the unit costs to a problem of determining whether or not there exists a negative cycle in the network formed by the given constraints, which is shown (implicitly) in [2] as well. Following their line of solution method and restricting $n = 7$, we form the network which represents the problem constraints. Then we investigate cycles and paths in the network, and express the optimal solution explicitly.

By the transformation of variables:

$$y_0 = 0,\ y_i = x_1 + \dots + x_i \quad \text{for } i = 1, 2, \dots, 6,$$

$$y = x_1 + x_2 + \dots + x_7,$$

problem (4.1) may be rewritten as:

$$\text{minimize} \quad y$$

subject to

$$y_i + (y - y_{i+2}) \geq b_i \quad \text{for } i = 1,2,3,4$$

$$y_5 \geq b_5$$

$$y_6 - y_1 \geq b_6$$

$$y - y_2 \geq b_7$$

$$y_{i+1} - y_i \geq 0 \quad \text{for } i = 1,2,3,4,5$$

$$y - y_6 \geq 0$$

$$y,\ y_i \text{ integer for } i = 1,2,3,4,5,6.$$

Or equivalently,

$$\text{minimize} \quad y$$

subject to

$$\begin{aligned}
&y_{i+2} \le y_i + (y - b_i) \quad \text{for } i = 1, 2, 3, 4\\
&y_0 \le y_5 + (-b_5)\\
&y_1 \le y_6 + (-b_6)\\
&y_2 \le y_0 + (y - b_0) \qquad\qquad\qquad\qquad (4.2)\\
&y_i \le y_{i+1} + (0) \quad \text{for } i = 1, 2, 3, 4, 5\\
&y_6 \le y_0 + (y)\\
&y, y_i \text{ integer for } i = 1, 2, 3, 4, 5, 6
\end{aligned}$$

where $b_0 = b_7$.

Consider the relaxation of problem (4.2) resulting from dropping the integrality constraints of y and y_i for $i = 1, 2, 3, 4, 5, 6$. The relaxation is to minimize y subject to the linear inequalities of (4.2). By the shortest path theory (see [15, p. 65]), it follows that linear inequalities imply the network G with vertex set {0, 1, 2, 3, 4, 5, 6} and an arc from vertex i to j of weight (.) for each inequality of (4.2): $y_j \le y_i + (.)$. The network is shown in Fig. 4.6, where weights are written on arcs.

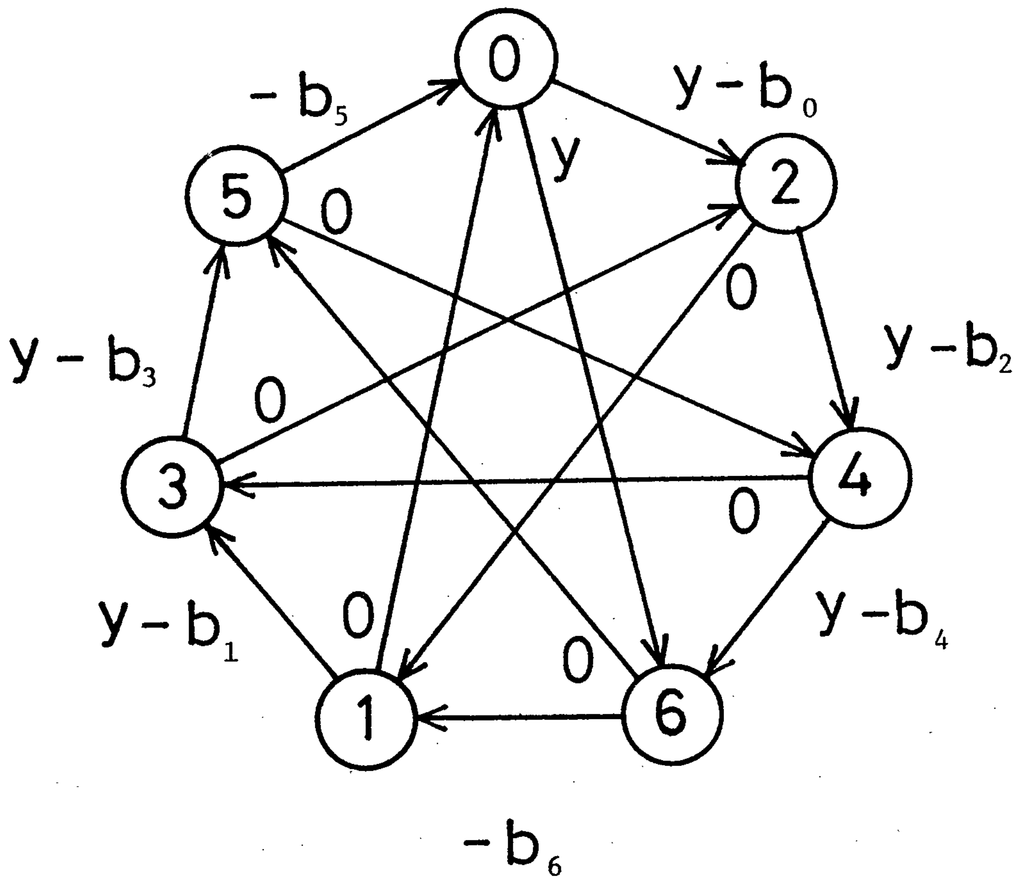


Fig. 4.6 The network corresponding to the linear inequalities of equation (4.2). The number on each arc is its weight.

That the linear inequalities can be satisfied for some fixed y means that there is no negative cycle in G, and vice versa. Hence the minimum value $\bar{y}$ of y subject to the linear inequalities only can be obtained by investigating all the cycles in G. Obviously, G has the following sixteen cycles:

$$C_i = (i,\ i+2,\ i+8) \quad \text{for } i = 1, 2, \ldots, 7$$

$$C_{7+i} = (i,\ i+2,\ i+4,\ i+6,\ i+8) \quad \text{for } i = 1, 2, \ldots, 7$$

$$C_{15} = (0, 2, 4, 6, 1, 3, 5)$$

$$C_{16} = (0, 6, 5, 4, 3, 2, 1),$$

where node number is to be taken modulo 7.

If each cycle is a non-negative one, the following is obtained:

$$y \geq b_i \quad \text{for } i = 1, 2, \ldots, 7$$

$$3y \geq b_i + b_{i+2} + b_{i+4} + b_{i+6} \quad \text{for } i = 1, 2, \ldots, 7$$

$$5y \geq b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7$$

$$y \geq 0.$$

Here and in the sequel, subscript of b_i is to be taken modulo 7. Hence we have

$$\bar{y} = \max\ (y_\alpha, y_\beta, y_\gamma)$$

where

$$y_\alpha = \max_{i=1,\dots,7} b_i \tag{4.3}$$

$$3y_\beta = \max_{i=1,\dots,7} (b_i + b_{i+2} + b_{i+4} + b_{i+6})$$

$$5y_\gamma = b_1 + \dots + b_7.$$

From the above, the optimal solution to problem (4.1) is $y^* = \lceil \bar{y} \rceil$, where $\lceil x \rceil$ denotes the least integer greater than or equal to x.

Now we consider the problem of expressing each value of x_i in terms of b_j only. For that purpose, we enumerate all the paths from vertex 0 to the others, which are shown in Fig. 4.7. From this diagram it follows that y_i^* for i = 1, ..., 6 given below, satisfy the inequalities of (4.2):

$$y_1^* = \min(y^* - b_6,\ y^* - b_7,\ 3y^* - (b_2 + b_4 + b_6 + b_7))$$

$$y_2^* = \min(y^* - b_7,\ 2y^* - (b_1 + b_6))$$

$$\begin{aligned} y_3^* = \min(&y^*,\ 2y^* - (b_1 + b_6),\ 2y^* - (b_1 + b_7), \\ &2y^* - (b_2 + b_7),\ 4y^* - (b_1 + b_2 + b_4 + b_6 + b_7)) \end{aligned}$$

(4.4)

$$\begin{aligned} y_4^* = \min(&y^*,\ 2y^* - (b_2 + b_7),\ 3y^* - (b_1 + b_2 + b_6), \\ &3y^* - (b_1 + b_3 + b_7),\ 3y^* - (b_1 + b_3 + b_7)) \end{aligned}$$

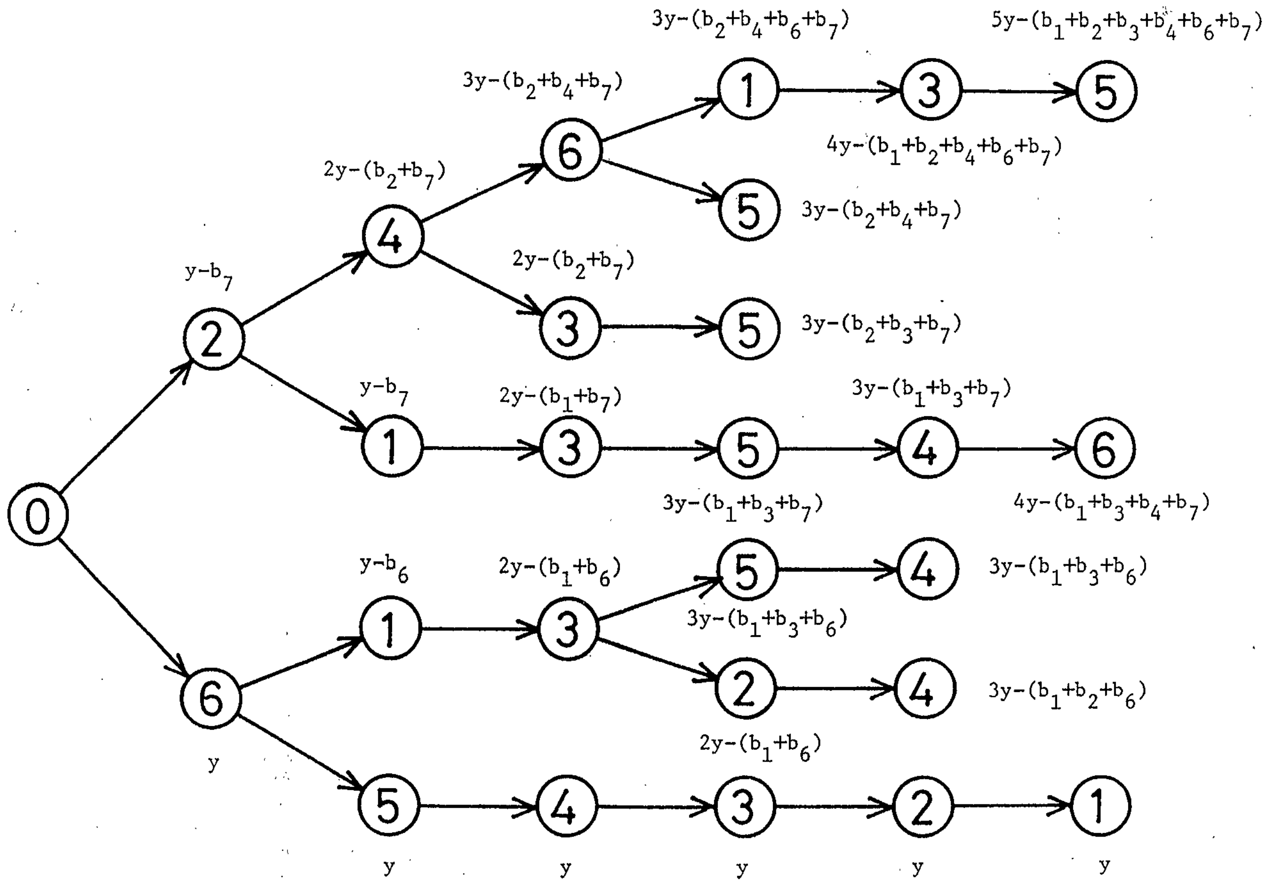


Fig. 4.7 All the paths from vertex 0 to the others. The number next to each vertex denotes the sum of weights on the arcs in the path from vertex to each vertex.

$$y_5^* = \min(y^*, 3y^* - (b_1 + b_3 + b_6), 3y^* - (b_1 + b_3 + b_7),$$
$$3y^* - (b_2 + b_3 + b_7), 3y^* - (b_2 + b_4 + b_7),$$
$$5y^* - (b_1 + b_2 + b_3 + b_4 + b_6 + b_7))$$

$$y_6^* = \min(y^*, 3y^* - (b_2 + b_4 + b_7), 4y^* - (b_1 + b_3 + b_4 + b_7)).$$

Since each y_i^* is integer, these y_i^* , $i = 1,.., 6$ are optimal for problem (4.2). Hence

$$x_i^* = y_i^* - y_{i-1}^* \quad \text{for } i = 1,..., 7 \qquad (4.5)$$

are optimal for problem (4.1) where $y_0^* = 0$, $y_7^* = y^*$. From (4.3)-(4.5) the algorithm follows.

Algorithm 4.3

Step 1. Compute $\bar{y}$ by (4.3). Set $y^* = \lceil \bar{y} \rceil$.

Step 2. Compute y_i^* for $i = 1,.., 6$ by (4.4).

Step 3. Compute x_i^* for $i = 1,.., 7$ by (4.5).

Consider the example given in [22]:

$b_1 = 16$, $b_2 = 11$, $b_3 = 17$, $b_4 = 13$, $b_5 = 15$, $b_6 = 19$, $b_7 = 14$.

By (4.3) it follows that

$$y_\alpha = 19,$$

$$3y_\beta = \max (62, 59, 57, 65, 53, 67, 57) = 67,$$

$$5y_\gamma = 105,$$

and $$\bar{y} = \max (19, 67/3, 105/5) = 67/3.$$

Hence $y^* = \lceil 67/3 \rceil = 23$.

From (4.4) we obtain that

$$y_1^* = \min (4, 9, 12) = 4,$$

$$y_2^* = \min (9, 11) = 9,$$

$$y_3^* = \min (23, 11, 16, 21, 19) = 11,$$

$$y_4^* = \min (23, 21, 23, 17, 22) = 17,$$

$$y_5^* = \min (23, 17, 22, 27, 31, 25) = 17,$$

$$y_6^* = \min (23, 31, 32) = 23.$$

Hence the optimal solution is as follows:

$x_1^* = 4$, $x_2^* = 5$, $x_3^* = 2$, $x_4^* = 6$, $x_5^* = 0$, $x_6^* = 6$ and $x_7^* = 0$.

The relation:

$$\begin{bmatrix}4\\4\\4\\4\\4\\0\\0\end{bmatrix} + \begin{bmatrix}0\\5\\5\\5\\5\\5\\0\end{bmatrix} + \begin{bmatrix}0\\0\\2\\2\\2\\2\\2\end{bmatrix} + \begin{bmatrix}6\\0\\0\\6\\6\\6\\6\end{bmatrix} + \begin{bmatrix}6\\6\\6\\0\\0\\6\\6\end{bmatrix} = \begin{bmatrix}16\\15\\17\\17\\17\\19\\14\end{bmatrix} \geq \begin{bmatrix}16\\11\\17\\13\\15\\19\\14\end{bmatrix}$$

shows that x_i^* for $i = 1, \ldots, 7$ satisfy the requirements.

4.5 Multifacility Minimax Location Problem with Rectilinear Distances

There are m facilities already located at points (a_i, b_i) for $i = 1, 2, \ldots, m$ in the plane, and n new facilities are to be located at points (x_j, y_j) for $j = 1, 2, \ldots, n$ in the plane. The travel distance between two points (X_1, Y_1) and (X_2, Y_2) in the plane is measured by the rectilinear distance, i.e., $|X_1 - X_2| + |Y_1 - Y_2|$. In urban situations, rectilinear distances are typically used. Define the travel cost between an old facility at (a_i, b_i) and a new facility at (x_j, y_j) as

$$w_{ij}(|a_i - x_j| + |b_i - y_j|) + g_{ij}$$

and the travel cost between a new facility at (x_j, y_j) and another new facility at (x_k, y_k) as

$$v_{jk}(|x_j - x_k| + |y_j - y_k|) + h_{jk}.$$

The constraints w_{ij}, v_{jk} are considered as nonnegative costs per unit of distance, while the constants g_{ij}, h_{jk} are considered as nonnegative fixed costs.

The problem of interest, denoted by (P2), is to minimize

$$\max \{ w_{ij}(|a_i - x_j| + |b_j - y_j|) + g_{ij} \text{ for } i = 1,\dots, m,\ j = 1,\dots, n,$$
$$v_{jk}(|x_j - x_k| + |y_j - y_k|) + h_{jk} \text{ for } j = 1,\dots, n-1$$
$$k = j + 1,\dots, n\} \tag{4.6}$$

subject to

$$\begin{aligned} |a_i - x_j| + |b_i - y_j| &\le d_{ij} \quad \text{for all } i \text{ and } j \\ |x_j - x_k| + |y_j - y_k| &\le c_{jk} \quad \text{for all } j \text{ and } k > j. \end{aligned} \tag{4.7}$$

The constraints (4.7) give the upper bounds, $d_{ij} \ge 0$ and $c_{jk} \ge 0$, on how far apart facilities may be. The distance constrains such as (4.7) may be important to facilities of some kind, as mentioned by Schaefer and Hurter [20] and Francis et al. [12]. As an example, a fire station may be required to be within a specified driving distance of any point that it serves. For simplicity, assume that (P2) has a feasible solution that satisfies the constraints (4.7). Of course, this assuption may be checked in the algorithm proposed here.

The minimax location problem such as (P2) may be important to the poor to whom the travel costs are the most significant factors, or may be important to emergency service facilities such as fire, police and hospital stations, as pointed out by Hakimi [13]. For another application see [21].

Special case of problem (P2) have been studied by some authors. The single new facility case, i.e. $n = 1$, without constraints (4.7) has been considered by Francis [9]. And then, Elzinga and Hearn [7] and Francis [10] have independently given a closed-form solution to problem (P2) where $n = 1$, $w_{1j} = 1$, and $g_{1j} \geq 0$ for all j, and constraints (4.7) are deleted.

Problem (P2) with constraints (4.7) deleted and with $g_{ij} = 0$, $h_{jk} = 0$ for all i, j and k has been studied by Wesolowsky [23], Elzinga and Hearn [8] and Morris [18].

Morris [17] has considered problem (P2) with constraints (4.7) and $g_{ij} = 0$, $h_{jk} = 0$ for all i, j and k in the context of linear programming.

Dearing and Francis [6] have solved (P2) with $h_{jk} = 0$ for all j and k as a parametric shortest path problem. However, their solution procedure needs a shortest path algorithm which must permit negative length arcs, and it cannot solve (P2) if $h_{jk} \neq 0$. Moreover the time bound of it depends on the input data such as costs and upper bounds on travel distances. The input data must be integral, otherwise their algorithm is no longer polynomially bounded (see [15, p. 97]). Hence the statement in [6] that their algorithm can run in time $O(n^3 \log n)$ is incorrect, although the statement holds if the input data are all integers, they increase as polynomial functions of n, and m is constant. (Note: The time bound of their algorithm must depend on m. Consider the case $m = \infty$.)

In order to improve the drawbacks presented above, we transform problem (P2) into a parametric shortest path problem (different from that of Dearing and Francis) which has been studied and solved efficiently. This leads to an efficient solution procedure with time complexity $O(n \max(m \log m, n^3))$. It has the merits that do not require shortest path algorithms which must permit negative length arcs or integral input data. Of course, the case $h_{jk} \neq 0$ is allowed. Furthermore, since the parametric shortest path problem is solved more efficiently for a special case, it is shown that the above time bound is reduced to $O(n \max(m, n^2))$ for the special case.

A selected bibliogaphy of location problems appears in [11]. For location problems involving generalized distances, see, e.g., [19].

Transformation of (P2) and Solution Procedure

We will tranform problem (P2) into a parametric shortest path problem so that we may exploit Megiddo's algorithm [16] of solving combinatorial problems with fractional objective functions and

Dijkstra's shortest path algorithm [5] which treats nonnegative arc lengths only. Although the transformation is related to the work in [6], it is essentially new in that both the above two efficient algorithms are applicable to the transformed parametric shortest path problem, which yields an efficient algorithm.

We will discuss the transformation in detail since it is the main subject. However the transformed parametric problem will be considered briefly since the problem has been already studied by Megiddo.

Lemma. 4.5 For two real numbers X and Y, $|X| + |Y| = \max(|X + Y|, |X - Y|)$.

The changes of variables:

$$\alpha_i = a_i + b_i \qquad \text{for all } i$$

$$\beta_i = a_i - b_i \qquad \text{for all } i$$

$$s_j = x_j - y_j \qquad \text{for all } j$$

$$t_j = x_j - y_j \qquad \text{for all } j$$

together with Lemma 4.5 yield a problem, to be denoted by (P3), which is equivalent to (P2). Problem (P3) is to minimize

$$\max(A_{ij} \text{ for all } i \text{ and } j,\ B_{jk} \text{ for all } j \text{ and } k > j)$$

where

$$A_{ij} = \max(w_{ij}|\alpha_i - s_j| + g_{ij},\ w_{ij}|\beta_i - t_j| + g_{ij}),$$

$$B_{jk} = \max(v_{jk}|s_j - s_k| + h_{jk},\ v_{jk}|t_j - t_k| + h_{jk})$$

subject to

$$\max(|\alpha_i - s_j|,\ |\beta_i - t_j|) \le d_{ij} \quad \text{for all } i \text{ and } j,$$

$$\max(|s_j - s_k|,\ |t_j - t_k|) \le c_{jk} \quad \text{for all } j \text{ and } k > j.$$

Since (P3) is separable, it suffices to consider the following two problems (P4) and (P5):

(P4) $$\text{min-max}(w_{ij}|\alpha_i - s_j| + g_{ij} \text{ for all } i \text{ and } j,$$
$$v_{jk}|s_j - s_k| + h_{jk} \text{ for all } j \text{ and } k > j)$$

subject to

$$|\alpha_i - s_j| \le d_{ij} \quad \text{for all } i \text{ and } j,$$

$$|s_j - s_k| \le c_{jk} \quad \text{for all } j \text{ and } k > j.$$

For simplicity assume $\alpha_i \ge 0$ for all i. If the assumption fails, replace α_i by $\alpha_i + M$ for all i and s_j by $s_j + M$ for all j, where M is a suitably large positive constant. Note that this replacement makes (P4) remain as it is.

(P5) $\text{min-max}(w_{ij}|\beta_i - t_j| + g_{ij}$ for all i and j,

$v_{jk}|t_j - t_k| + h_{jk}$ for all j and $k > j)$

subject to

$|\beta_i - t_j| \le d_{ij}$ for all i and j,

$|t_j - t_k| \le c_{jk}$ for all j and $k > j$.

Similarly, assume $\beta_i \ge 0$ for all i. Note that (P4) and (P5) can be considered on a line. Since they are of the same type, we concentrate on (P4) only. Problem (P4) can be restated as follows:

(P6) minimize λ

s.t. $|\sigma_i - s_j| \le \min((\lambda - g_{ij})/w_{ij}, d_{ij})$ for all i and j,

(4.8)

$|s_j - s_k| \le \min((\lambda - h_{jk})/v_{jk}, c_{jk})$ for all j and $k > j$,

where we define $(\lambda - g_{ij})/w_{ij} = \infty$ and $(\lambda - h_{jk})/v_{jk} = \infty$ if $w_{ij} = 0$ and $v_{jk} = 0$, respectively.

As a notational convenience, the minimum value of the objective function of (P6) will be denoted by λ^*. Let $\underline{\lambda} = \max(g_{ij}$ for all i and j, h_{jk} for all j and $k > j)$. Then it is obvious that $\lambda^* \ge \underline{\lambda}$. So in the sequel restrict $\lambda \ge \underline{\lambda}$. Note that all the constraints (4.8) are not satisfied for $\lambda < \lambda^*$. Let

$$\delta_{ij} = \min((\lambda - g_{ij})/w_{ij}, d_{ij}) \quad \text{for all } i \text{ and } j \tag{4.9}$$

$$F_{0j} = \min_{i=1,2,\ldots,m} (\alpha_i + \delta_{ij}) \quad \text{for all } j \tag{4.10}$$

$$F_j = \max_{i=1,2,\ldots,m} (\alpha_i - \delta_{ij}) \quad \text{for all } j \tag{4.11}$$

$$F_{jk} = \min((\lambda - h_{jk})/v_{jk}, c_{jk}) \quad \text{for all } j \text{ and } k > j. \tag{4.12}$$

Since F_{0j}, F_j and F_{jk} are functions of λ, we may sometimes denote them by $F_{0j}(\lambda)$, $F_j(\lambda)$ and $F_{jk}(\lambda)$, respectively. Obviously, $F_{0j}(\lambda)$ is concave and piecewise linear with at most $(m + 1)$ linear pieces, $F_j(\lambda)$ is convex and piecewise linear with at most $(m + 1)$ linear pieces, and $F_{jk}(\lambda)$ is concave and piecewise linear with at most two linear pieces.

The constraints (4.8) are expressed as (4.13)-(4.15):

$$s_j \le s_0 + F_{0j} \qquad \text{for all } j \tag{4.13}$$

$$s_j \ge F_j \qquad \text{for all } j \tag{4.14}$$

$$|s_j - s_k| \le F_{jk} \qquad \text{for all } j \text{ and } k > j \tag{4.15}$$

where $s_0 = 0$. By the shortest path theory, it follows that linear inequalities (4.13) and (4.15) imply the network G with vertex set {0, 1, 2,..., n} and a dircted arc from vertex 0 to j with length F_{0j} for each inequality of (4.13) and an underected arc between vertices j and k with length F_{jk} for each inequality of (4.15).

The network G for n = 3 is illustrated in Fig. 4.8.

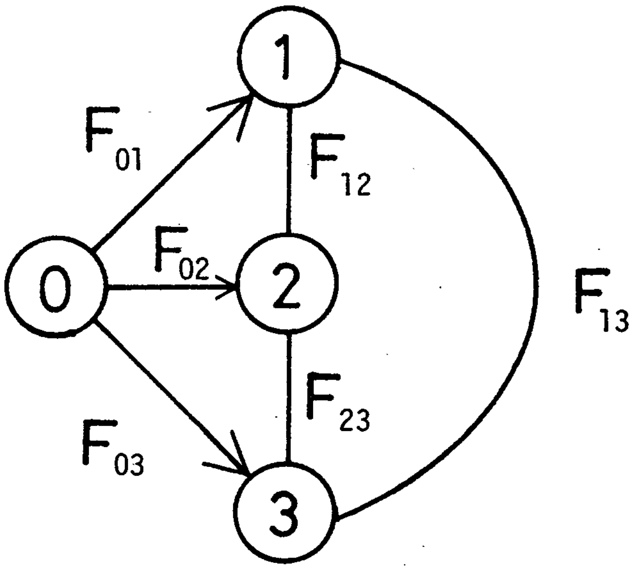


Fig. 4.8 G for n = 3.

It is well known that linear inequalities (4.13) and (4.15) can be satisfied by the shortest path lengths from vertex 0 to the other vertices. For any $\lambda \geq \underline{\lambda}$, the shortest path lengths can be computed by Dijkstra's shortest path algorithm [5] since each arc length F_{0j}, F_{jk} is nonnegative, recalling $\alpha_i \geq 0$, $\lambda \geq g_{ij}$ and $\lambda \geq h_{jk}$. Let $P_j(\lambda)$ be the shortest path length from verex 0 to j in G for λ. Then, from the above, we have the equivalent problem (P7).

(P7) minimize λ

s.t. $P_j(\lambda) \geq F_j$ for $j = 1, 2, \ldots, n$.

We will solve (P7) by computing the shortest path lengths from vertex 0 to the other vertices parametrically in the parameter λ, noting that λ^* is the smallest value of λ such that

$$P_j(\lambda) \geq F_j \quad \text{for all } j.$$

From the property of each function of λ, F_{0j} or F_j, it can be expressed as the form

$$\begin{array}{ll} A_1\lambda + B_1 & \underline{\lambda} \leq \lambda \leq \omega_1 \\ A_2\lambda + B_2 & \omega_1 \leq \lambda \leq \omega_2 \\ \cdots\cdots\cdots\cdots & \\ A_{r+1}\lambda + B_{r+1} & \omega_r \leq \lambda < \infty \end{array} \qquad (4.16)$$

where A_j and B_j for $j = 1, 2, \ldots, r + 1$ are real numbers, $r \leq m$, and $\underline{\lambda} < \omega_1 < \omega_2 < \ldots < \omega_r < \infty$. It is shown in [16] that the expression entails $O(m \log m)$ time for each F_{0j}, F_j and hence $O(nm \log m)$ time for all F_{0j} and F_j for $j = 1, 2, \ldots, n$. Each F_{jk} can be also expressed as the form (4.16) with $r \leq 1$. This takes $O(n^2)$ time for all j and $k > j$. For fixed value λ, the feasibility test for problem (P7), i.e., whether $P_j(\lambda) \geq F_j$ for all j are satisfied or not, is denoted by (FT). For fixed value λ, the computation time of determining the value of each $F_{0j}(\lambda)$, $F_j(\lambda)$ is $O(\log m)$ by finding an interval including this λ with binary search, while that of each $F_{jk}(\lambda)$ is $O(1)$. After the computation of F_{0j}, F_j and F_{jk} for all j and $k > j$, which takes $O(n \log m + n^2)$ time in all, $P_j(\lambda)$ for all j are determined in $O(n^2)$ time by Dijkstra's algorithm. And then, the comparisons of $P_j(\lambda)$ and F_j for $j = 1, 2, \ldots, n$ need $O(n)$ time. Hence the time complexity of (FT) for fixed value λ is $O(n \log m + n^2)$.

Since each function F_{0j} or F_j has at most m breaking points and each F_{jk} has at most one breaing point, there are at most $2mn + n(n - 1)/2$ breaking points in all. Let the distinct λ-coodinates of the breaking points greater than $\underline{\lambda}$ be $\Omega_1 < \Omega_2 < \Omega_3 < \ldots$. By perfoming binary search over intervals $[\underline{\lambda}, \Omega_1]$, $[\Omega_1, \Omega_2]$, $[\Omega_2, \Omega_3]$, ... and feasibility test (FT), we can obtain the quantities of the form:

$$F_{ij}(\lambda) = e_{ij}\lambda + f_{ij} \qquad \Omega_r < \lambda \le \Omega_{r+1} \tag{4.17}$$

for each directed or undirected arc (i,j) in G, and of the form:

$$F_j(\lambda) = e_j\lambda + f_j \qquad \Omega_r < \lambda \le \Omega_{r+1} \tag{4.18}$$

for all vertices j in G, where

$$\Omega_0 = \underline{\lambda},$$

$$P_j(\Omega_r) < F_j(\Omega_r) \quad \text{for some } j$$

and

$$P_j(\Omega_{r+1}) \ge F_j(\Omega_{r+1}) \quad \text{for all } j,$$

i.e.,

$$\Omega_r < \lambda^* \le \Omega_{r+1}.$$

To derive the quantities of the form (4.17) or (4.18), it takes time

$$O(\log(2mn + n(n-1)/2))O(n \log m + n^2)$$
$$= O(n^2 \log(n+m)) + O(n \log m \log(n+m))$$
$$= O(n \log(n+m) \max(n, \log m)).$$

At this point, all data (i.e. arc lengths) for computing shorteat path lengths are of the form (4.17) or (4.18). Hence Megiddo's parametric approach is applicable to our purpose. A good illustration of Megiddo's approach is given in Chapter 3, Section 5, although in the context of maximum flow. As mentioned previously, we do not give a full detail of Megiddo's algorithm here.

The parametric computation for the shortest path lengths in G needs $O(n^2)$ comparisons since Dijkstra's algorithm does so. Each comparison involves a feasibility test (FT), which needs $O(n^2)$ time since each F_{0j} or F_j is now of the form (4.17) or (4.18) and hence the value of $F_{0j}(\lambda)$ or $F_j(\lambda)$ can be determined in $O(1)$ time for fixed value of λ. Therefore the shortest path lengths, $P_j(\lambda)$ for all j can be obtained in time

$$O(n^2)O(n^2) = O(n^4).$$

This is exactly the result given in [16] for this problem. Let the parametric computation yield

$$P_j(\lambda) = e_j^*\lambda + f_j^* \qquad L < \lambda \le U, \tag{4.19}$$

$$F_j(\lambda) = e_j\lambda + f_j \qquad L < \lambda \le U,$$

for all j, where $P_j(L) < F_j(L)$ for some j and $P_j(U) \ge F_j(U)$ for all j. Denote the set of all j such that $P_j(L) < F_j(L)$ by J.

Theorem 4.6 Define $\lambda_j = (f_j - f_j^*)/(e_j^* - e_j)$ for $j \in J$. Then

$$\lambda^* = \max_{j \in J} \lambda_j.$$

Proof. Let $\lambda^* = \lambda_{j^*}$. Then, for $\lambda < \lambda^*$ we have $P_j(\lambda) < F_j(\lambda)$ for $j = j^*$ (see Fig. 4.9). On the other hand, for $\lambda \geq \lambda^*$ we have $P_j(\lambda) \geq F_j(\lambda)$ for all $j = 1, 2, \ldots, n$. Q.E.D.

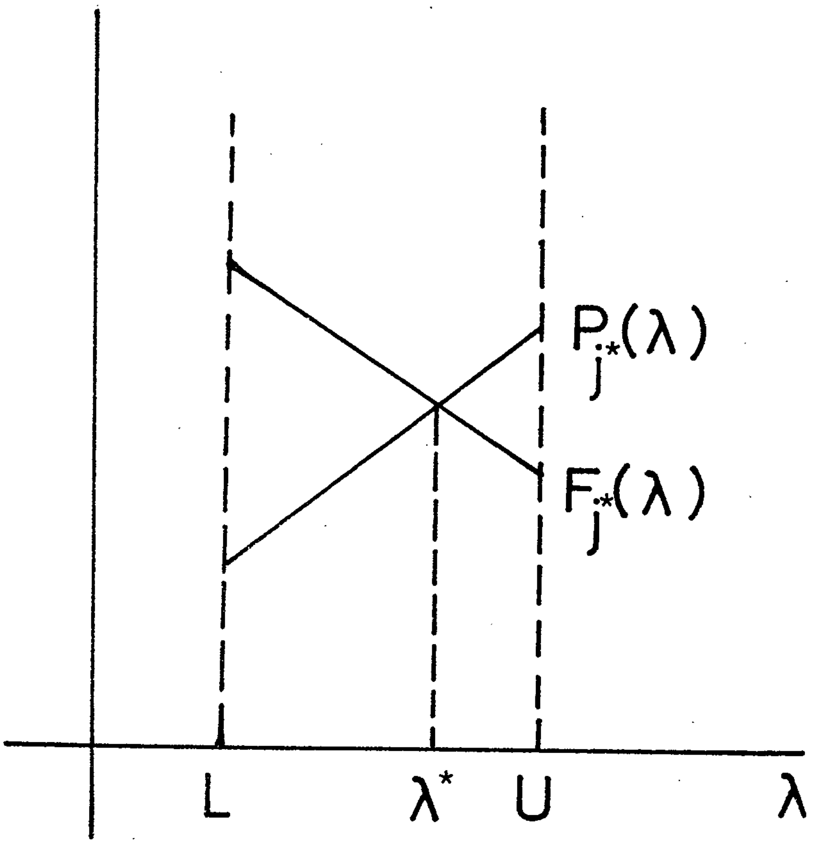


Fig. 4.9

Summarizing what has been discussed above, the algorithm for (P7) follows.

Algorithm 4.4 and Time Complexity

Step 1. Express F_{0j}, F_j and F_{jk} for all j and k > j as the form (4.16). ---$O(nm \log m + n^2)$.

Step 2. Express F_{ij} for all arcs and F_j for all vertices in G as the form (4.17) and (4.18), respectively. ---$O(n \log(n + m) \max(n, \log m))$.

Step 3. Compute $P_j(\lambda)$ of the form (4.19) for all j. ---$O(n^4)$.

Step 4. Compute λ^* by Theorem 4.6. ---$O(n)$.

Remark. The optimal solution to (P4) is given by $s_j^* \leftarrow P_j(\lambda^*)$ for $j = 1, 2, .., n$.

Theorem 4.7 The overall time complexity of the preceding algorithm is $O(n \max(m \log m, n^3))$.

Proof. It is obvious that the time bounds of Steps 1, 3 and 4 are not greater than $O(n \max(m \log m, n^3))$. Consider the case of Step 2.

(i) Let $n \geq \log m$, or $2^n \geq m$. Then

$$\begin{aligned} O(n \log(n + m) \max(n, \log m)) &\leq O(n^2 \log(n + 2^n)) \\ &\leq O(n^2 \log 2^{2n}) \end{aligned}$$

$$= O(n^3)$$

$$< O(n^4).$$

(ii) Let $n < \log m$, which yields $n < m$. Then

$$O(n \log(n + m) \max(n, \log m)) = O(n \log(n + m) \log m)$$

$$\leq O(n \log(2m) \log m)$$

$$< O(nm \log m).$$

Hence the theorem follows. Q.E.D.

Theorem 4.7 states that the preceding algorithm can solve (P7) in time $O(n \max(m \log m, n^3))$. Hence it means that Algorithm 4.4 can solve problem (P2) in the same time as well, noting that the time for transformations from (P2) to (P7) is negligible compared with $O(n \max(m \log m, n^3))$.

Now consider the case $w_{ij} = 0$ or 1 for all i and j, $v_{jk} = 0$ or 1 for all j and $k > j$, briefly. Of course, the problem of this case can be solved by the algorithm presented above. However its time complexity can be reduced in this case. Step 1 needs $O(nm + n^2)$ time since the slope of each F_{0j} or F_{jk} is 0 or 1 and that of each F_j is 0 or -1. With respect to Step 2, each (FT) entails $O(n + n^2) = O(n^2)$ time and there are at most $2n + n(n - 1)/2$ breaking points in all. Hence Step 2 needs time

$$O(\log(2n + n(n-1)/2))O(n^2) = O(n^2 \log n).$$

Step 3 can be performed by Karp and Orlin's algorithm in time $O(n^3)$ (see [14]), noting that each arc length contained in G is of the form:

$$A\lambda + B$$

where $A = 0$ or 1, B is a real number. Step 4 is $O(n)$. Hence the entire algorithm for this case requires only $O(n \max(m, n^2))$ time. We note that the reduction of the time bound for this case is possible since the slopes of functions of λ, F_{0j} and F_{jk} for all j and $k > j$, are restricted to 0 and 1 (hence the slope of $P_j(\lambda)$ is 0, 1, ..., or n for each j), while this is not true for the general case.

Consider the following numerical example of problem (P4) with $m = n = 3$:

$$(w_{ij}) = \begin{pmatrix} 1.2 & 1 & 0.5 \\ 1 & 1.5 & 2 \\ 1 & 1.25 & 4 \end{pmatrix} (g_{ij}) = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 1 & 2 \\ 2 & 0 & 1 \end{pmatrix} (d_{ij}) = \begin{pmatrix} 7 & 15 & 12 \\ 13 & 11 & 10 \\ 20 & 15 & 14 \end{pmatrix}$$

$$(v_{jk}) = \begin{pmatrix} - & 10 & 1 \\ - & - & 1.5 \\ - & - & - \end{pmatrix} \quad (h_{jk}) = \begin{pmatrix} - & 2 & 1 \\ - & - & 1 \\ - & - & - \end{pmatrix} (c_{jk}) = \begin{pmatrix} - & 1 & 3 \\ - & - & 10 \\ - & - & - \end{pmatrix}$$

$(\alpha_i) = (2, 3, 6)$.

From the preceding data we have:

$\underline{\lambda} = 2,$

$$F_{01} = \begin{cases} (5/6)\lambda + 7/6 & 2 \le \lambda \le 47/5 \\ 9 & \lambda \ge 47/5 \end{cases}$$

$$F_{02} = \begin{cases} \lambda + 1 & 2 \le \lambda \le 4 \\ (2/3)\lambda + 7/3 & 4 \le \lambda \le 16 \\ (1/2)\lambda + 5 & 16 \le \lambda \le 18 \\ 14 & \lambda \ge 18 \end{cases}$$

$$F_{03} = \begin{cases} \lambda & 2 \le \lambda \le 25/3 \\ (1/4)\lambda + 25/4 & 25 \le \lambda \le 47 \\ 18 & \lambda \ge 47 \end{cases}$$

$$F_{1} = \begin{cases} 10 - 2\lambda & 2 \le \lambda \le 6 \\ 4 - \lambda & 6 \le \lambda \le 7 \\ 17/6 - (5/6)\lambda & 7 \le \lambda \le 47/5 \\ -5 & \lambda \ge 47/5 \end{cases}$$

$$F_{2} = \begin{cases} 7 - (1/2)\lambda & 2 \le \lambda \le 22 \\ -4 & \lambda \ge 22 \end{cases}$$

$$F_{3} = \begin{cases} 25/4 - (1/4)\lambda & 2 \le \lambda \le 57 \\ -8 & \lambda \ge 57 \end{cases}$$

$$F_{12} = \begin{cases} (1/10)\lambda - 1/5 & 2 \le \lambda \le 12 \\ 1 & \lambda \ge 12 \end{cases}$$

$$F_{13} = \begin{cases} \lambda - 1 & 2 \le \lambda \le 4 \\ 3 & \lambda \ge 4 \end{cases}$$

$$F_{23} = \begin{cases} (2/3)\lambda - 2/3 & 2 \le \lambda \le 16 \\ 10 & \lambda \ge 16. \end{cases}$$

The above functions have the distinct λ-coordinates of the breaking points:

2, 4, 6, 7, 25/3, 47/5, 12, 16, 18, 22, 47, 57.

Since the median[†] is 47/5, we perform (FT) for $\lambda = 47/5$ on G, which is shown in Fig. 4.10. Here $F_1 = -5$, $F_2 = 2.3$, $F_3 = 3.9$, $P_1 = 9$, $P_2 = 8.6$, $P_3 = 8.6$. Since $P_j \ge F_j$ for j = 1, 2, 3, we obtain $2 \le \lambda^* \le 47/5$.

Since the median of 2, 4, 6, 7 and 25/3 is 6, (FT) is done for $\lambda = 6$ on G, which is shown in Fig. 4.11. Here $F_1 = -2$, $F_2 = 4$, $F_3 = 19/4$, $P_1 = 37/6$, $P_2 = 19/3$, $P_3 = 6$. Since $P_j \ge F_j$ for j = 1, 2, 3, we have $2 \le \lambda^* \le 6$. For $\lambda = 4$, we have $F_1 = 2$, $F_2 = 5$, $F_3 = 5.25$, $P_1 = 4.5$, $P_2 = 4.7$, $P_3 = 4$ (see Fig. 4.12). Since $P_2 < F_2$, we have $4 < \lambda^* \le 6$. Since there is no breaking point strictly between 4 and 6, the functions above are expressed as follows:

† The median of the set $\{a_1, a_2, \ldots, a_n\}$ refers to the $\lceil n/2 \rceil$th smallest element.

$$F_{01} = (5/6)\lambda + 7/6$$

$$F_{02} = (2/3)\lambda + 7/3$$

$$F_{03} = \lambda$$

$$F_1 = 10 - 2\lambda$$

$$F_2 = 7 - (1/2)\lambda$$

$$F_3 = 25/4 - (1/4)\lambda$$

$$F_{12} = (1/10)\lambda - 1/5$$

$$F_{13} = 3$$

$$F_{23} = (2/3)\lambda - 2/3$$

$$4 < \lambda \le 6.$$

Here we have the parametric shortest path problem with additional constraints $P_j(\lambda) \ge F_j(\lambda)$ for $j = 1, 2, 3$, which is shown in Fig. 4.13. The result is as follows:

$$P_1 = (5/6)\lambda + 7/6$$

$$P_2 = (14/15)\lambda + 29/30$$

$$P_3 = \lambda$$

$$4 < \lambda \le 41/8.$$

Since $P_2(4) < F_2(4)$, $P_3(4) < F_3(4)$, we have $J = \{2, 3\}$,

$$\lambda_2 = (7 - 29/30)/(14/15 - (-1/4)) = 181/43,$$

$$\lambda_3 = (25/4 - 0)/(1 - (-1/4)) = 5,$$

hence $\lambda^* = \max(\lambda_2, \lambda_3) = 5$. Thus, the optimal solution to (P4) is $s_1^* = 16/3$, $s_2^* = 169/30$, $s_3^* = 5$.

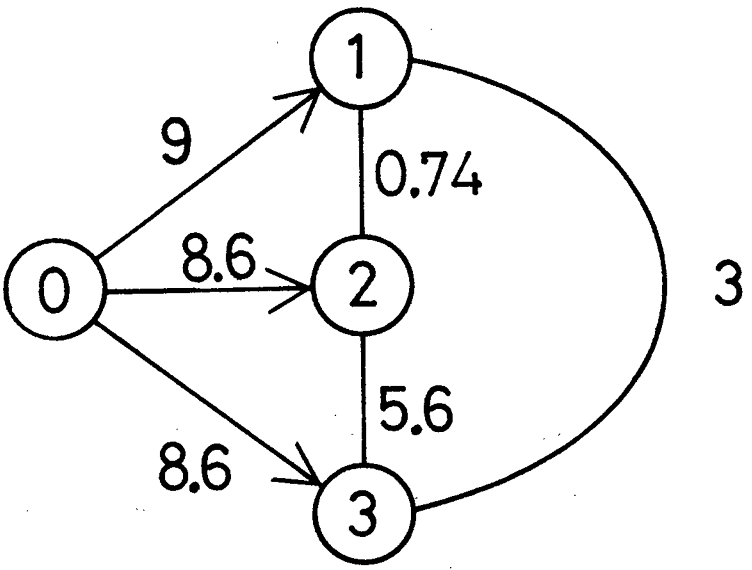


Fig. 4.10 $\lambda = 47/5$.

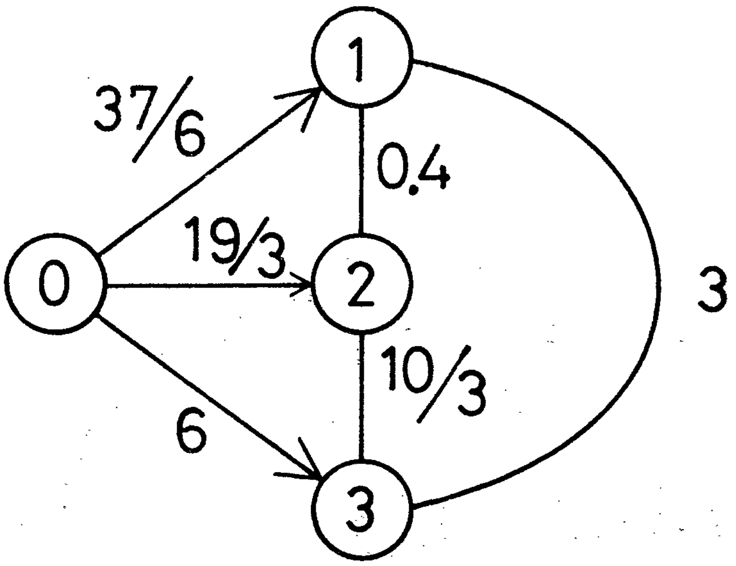


Fig. 4.11 λ = 6.

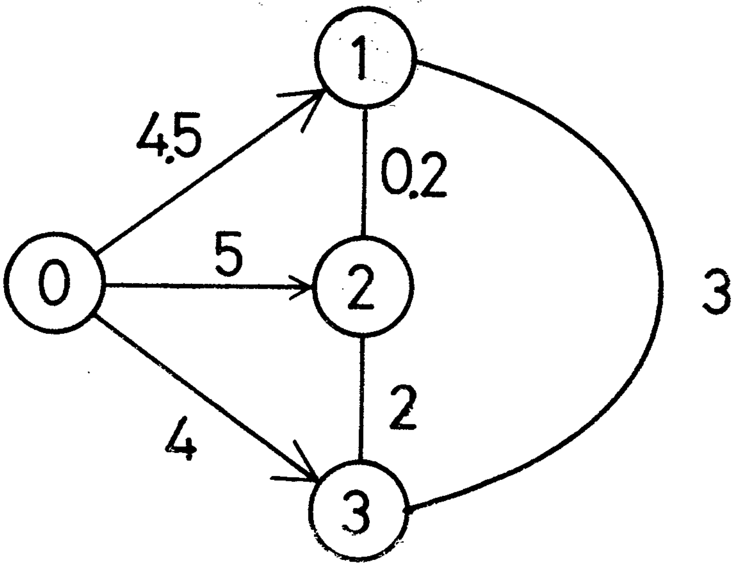
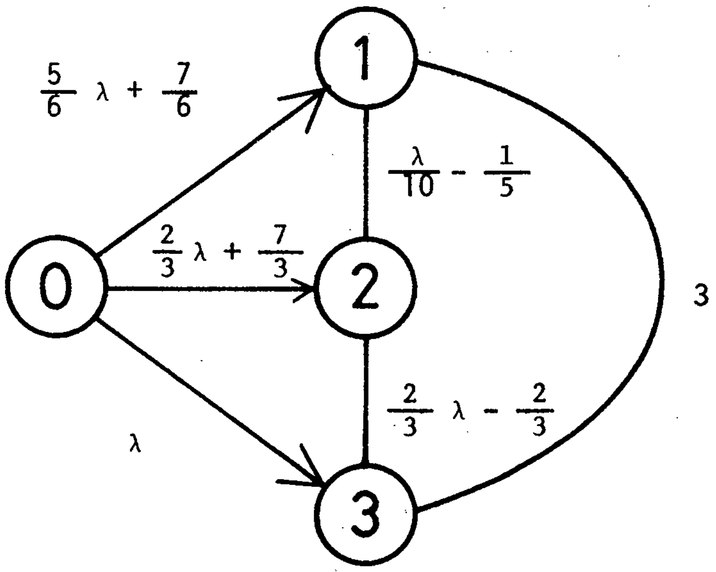


Fig. 4.12 λ = 4.



$\frac{5}{6}\lambda + \frac{7}{6}$
$\frac{\lambda}{10} - \frac{1}{5}$
$\frac{2}{3}\lambda + \frac{7}{3}$
3
$\frac{2}{3}\lambda - \frac{2}{3}$
λ
0
1
2
3

Fig. 4.13

[1] Baker, K.R.,"Scheduling a full-time workforce to meet cyclic staffing requirements," Management Science 20 (1974) 1571-1568.

[2] Bartholdi III, J.J., J.B. Orlin and H.D. Ratliff,"Cyclic scheduling via integer programs with circular ones," Operations Research 28 (1980) 1074-1085.

[3] Bartholdi III, J.J. and H.D. Ratliff,"Unnetworks, with applications to idle time scheduling," Management Science 24 (1978) 850-858.

[4] Christofides, N., Graph Theory: an algorithmic approach, Academic Press, New York, 1975.

[5] Dijkstra, E.W.,"A note on two problems in connection with graphs," Numeriche Mathematik 1 (1959) 269-271.

[6] Dearing, P.M. and R.L. Francis,"A network flow solution to a multifacility minimax location problem involving rectilinear distances," Transportation Science 8 (1974) 126-141.

[7] Elzinga, L. and D.W. Hearn,"Geometrical solution for some minimax location problems," Transportation Science 4 (1972) 379-394.

[8] Elzinga, L. and D.W. Hearn,"A note on a minimax location problem," Transportation Science 7 (1973) 100-103.

[9] Francis, R.L.,"Some aspects of a minimax location problem," Operations Research 15 (1967) 1163-1169.

[10] Francis, R.L.,"A geometrical solution procedure for a recti-

linear distance minimax location problem," AIIE Transactions 4 (1972) 328-332.

[11] Francis, R.L. and J.M. Goldstein,"Location theory: a selective bibliograph," Operations Research 22 (1974) 400-410.

[12] Francis, R.L., T.J. Lowe and H.D. Ratliff,"Distance constraints for tree network multifacility location problem," Operations Research 26 (1978) 570-596.

[13] Hakimi, S.L.,"Optimum location of switching centers and the absolute centers and medians of a graph," Operations Research 12 (1964) 450-459.

[14] Karp, R.M. and J.B. Orlin,"Parametric shortest path algorithm with an application to cyclic staffing," Discrete Applied Mathematics 3 (1981) 37-45.

[15] Lawler, E.L., Combinatorial Optimization: networks and matroids, Holt, Rinehart and Winston, New York, 1976.

[16] Megiddo, N.,"Combinatorial optimization with rational objective functions," Mathematics of Operations Research 4 (1979) 414-424.

[17] Morris, J.G.,"A linear programming approach to the solution of constrained multifacility minimax location problem where distances are rectangular," Operational Research Quarterly 24 (1973) 419-435.

[18] Morris, J.G.,"A further note on a minimax location problem,"

Transportation Science 10 (1976) 405-408.

[19] Morris, J.G.,"Convergence of the Weiszfeld algorithm for Weber problems using a generalized "distance" function," Operations Reseach 29 (1981) 37-48.

[20] Schaefer, M.K. and A.P. Hurter,"An algorithm for the solution of a location problem with metric constraints," Naval Research Logistics Quarterly 4 (1974) 625-636.

[21] Smallwood, R.D.,"Minimax detection station placement," Operations Research 13 (1965) 632-646.

[22] Tibrewala, R., D. Philippe and J. Browne,"Optimal scheduling of two idle periods," Management Science 19 (1972) 71-75.

[23] Wesolowsky, G.O.,"Rectangular distance location under the minimax optimality criterion," Transportation Science 6 (1972) 103-113.

CHAPTER 5

TWO APPLICATIONS OF THE MAXIMUM FLOW AND MINIMUM SPANNING TREE PROBLEMS

5.1 Introduction

In this chapter, we consider the following two problems. One is concerned with the maximum flow problem. The other is concerned with the minimum spanning tree problem.

The first problem is as follows. We often come across the problem in which each unit of a network flow must go through a certain fixed vertex called a check vertex. For example, in the transportation of milk from a dairy farm, milk is processed at a milk company, which corresponds to the check vertex, on the way from the farm to consumers. Then the volume of milk may increase or decrease at the milk company which means that the check vertex has a positive gain denoted by γ. Our objective is to find a flow whose value is the maximum under the above constraint. The problem is equivalent to the special case of the problem of Hu [3]. However, solving our problem efficiently needs some developments and improvements of Hu's two-commodity flow algorithm. In the following statement, the case in which the gain of the check vertex is equal to one is discussed. Moreover, it is shown that the case in which $\gamma \neq 1$ can be easily extended from the case $\gamma = 1$.

The second problem is as follows. When weights of arcs are independent random variables according to the normal distributions, we seek a spanning tree maximizing the probability that the sum of arc-weights in the tree (the weight of the tree) is not greater than a given constant value. In this context we refer to this spanning tree as a minimum spanning tree. We assume in addition that the given constant value is greater than the maximum value of the mean of sum of arc-weights among all spanning trees. If the assumption fails, since no spanning tree has a probability greater than 0.5 (this is shown below), we cannot accept any spanning tree in an ordinary sence. We discuss this problem and then develop an algorithm which is bounded polynomially.

5.2 Flows in a Network with a Check Vertex

Let $G = (V, A)$ be a directed network, where V is the vertex set and A is the arc set. Let $n = |V|$ and $m = |A|$. We associate with each arc (i,j) in A positive capacity $c(i,j)$. Let s be the source, t is the sink and u be the check vertex. A function $f(i,j)$ defined on A is called a flow of value v in G if

$$|f(i,j)| \leq c(i,j) \qquad (i,j) \in A \qquad (5.1)$$

$$\sum_{i\in A(j)} f(i,j) - \sum_{i\in B(j)} f(j,i) = \begin{cases} v & j = s \\ 0 & j \neq s, t \\ -v & j = t \end{cases} \qquad (5.2)$$

where A(j) (resp. B(j)) is the parent (resp. child) set of vertex j.

Let $f_b(i,j)$ and $f_a(i,j)$, respectively, denote the flows before and after visiting the check vertex u. Then the following relations must hold:

$$|f_b(i,j)| + |f_a(i,j)| \leq c(i,j) \qquad (5.3)$$

$$\sum_{i\in A(j)} f_b(i,j) - \sum_{i\in B(j)} f_b(j,i) = \begin{cases} v & j = s \\ 0 & j \neq s,t \\ -v & j = t \end{cases} \qquad (5.4)$$

$$\sum_{i\in A(j)} f_a(i,j) - \sum_{i\in B(j)} f_a(j,i) = \begin{cases} v & j = u \\ 0 & j \neq u,t \\ -v & j = t. \end{cases} \qquad (5.5)$$

We consider the following problem (5.6):

$$\text{maximize} \quad v \qquad (5.6)$$

subject to the constraints (5.3) - (5.5). Let v* denote the maximum value of v in problem (5.6). With the change of variables

$$g_1(i,j) = f_b(i,j) + f_a(i,j)$$

$$g_2(i,j) = f_b(i,j) - f_a(i,j),$$

we have the following constraits (5.7)-(5.10) in place of (5.3)-(5.5).

$$|g_1(i,j)| \le c(i,j) \tag{5.7}$$

$$|g_2(i,j)| \le c(i,j) \tag{5.8}$$

$$\sum_{i \in A(j)} g_1(i,j) - \sum_{i \in B(j)} g_1(j,i) = \begin{cases} v & j=s \\ 0 & j \ne s,t \\ -v & j=t \end{cases} \tag{5.9}$$

$$\sum_{i \in A(j)} g_2(i,j) - \sum_{i \in B(j)} g_2(j,i) = \begin{cases} v & j=s \\ -2v & j=u \\ v & j=t \\ 0 & j \ne s,u,t. \end{cases} \tag{5.10}$$

From the max-biflow min-cut theorem of Hu [3], the relation between the values of f_b and f_a becomes as curve ABCD shown in Fig. 5.1. If $\gamma \ne 1$, the value of f_b multiplied by γ should be equal to that of f_a, which is indicated by a line like OE shown in Fig. 5.1. Here, point A corresponds to the maximum value of flow f_b when $f_a(i,j) = 0$ for every arc (i,j), point D to the maximum value of f_a when $f_b(i,j) = 0$ for every arc (i,j), and segment BC to the maximum sum of the values of two flows f_b and f_a. The coordinates of points A, B, C and D are $(v_{su1}, 0)$, (v_{su1}, v_{su1}), (v_{su2}, v_{tu2}) and $(0, v_{tu2})$, respectively, and these $v_{\dots}$ are given by the next algorithm. Here we have to distinguish among the following three cases: (1) point E is on segment DC; (2) E is on segment CB; (3)

E is on segment BA, which is determined by the value of γ and the relation between f_b and f_a. When $\gamma = 1$, we note that in case (2)

$$v^* = (v_{su1}v_{su2} - v_{su2}v_{tu1})/(v_{su1} + v_{tu2} - v_{su2} - v_{tu1})$$

and that in case (1)

$$v^* = v_{tu2},$$

in case (3)

$$v^* = v_{tu1},$$

which shows that v* determined in the following algorithm is optimum.

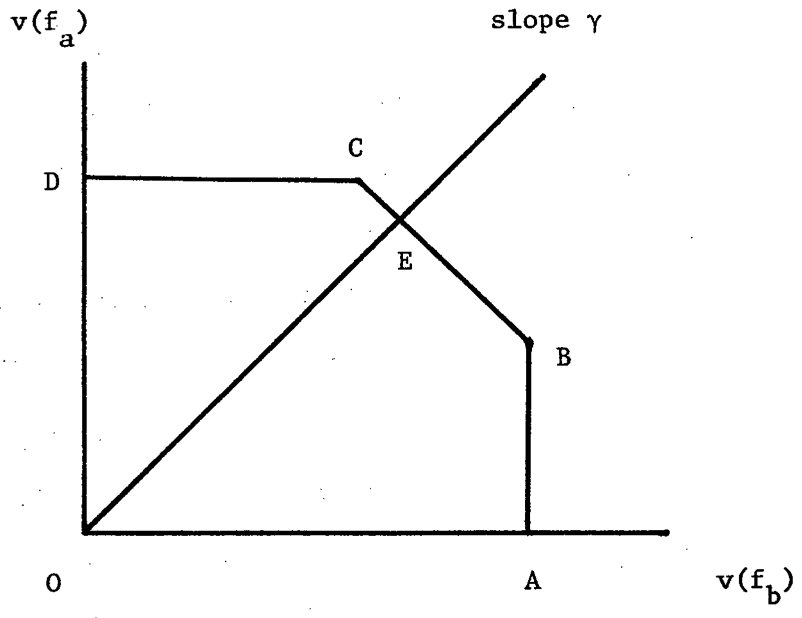


Fig. 5.1 v(f.) is the value of f..

Algorithm 5.1

```
(Determination of $v^*$)
Step 1.  Set $j = 1$, $S = \{s\}$, $T = \{u\}$ and $f_i = 0$.  Go to Step 6.
Step 2.  Set $v_{su1} = v$, $S = \{s, u\}$, $T = \{t\}$ and $f_i = f$.  Go to Step 6.
Step 3.  Set $v'_{tu1} = v$, $S = \{s, t\}$, $T = \{u\}$.  Go to Step 6.
Step 4.  Set $v''_{tu1} = v - v_{su1}$, $v_{tu1} = \min(v'_{tu1}, v''_{tu1})$, $S = \{u\}$, $T =$
         $\{t\}$ and $f_i = 0$.  Go to Step 6.
Step 5.  Set $v_{tu2} = v$, $v_{su2} = v_{su1} + v_{tu1} - v_{tu2}$.  Go to Step 7.
Step 6.  Find a maximum flow $f$ and its value $v$ from $S$ to $T$ in $G$
         with initial flow $f_i$.  Set $j = j + 1$.  Go to Step $j$.
Step 7.  Set $w = (v_{su1} - v_{tu1})(v_{su2} - v_{tu2})$.  If $w \geq 0$, then set
         $v^* = \min(v_{su2}, v_{tu2})$.  Otherwise, set
         $v^* = (v_{su1}v_{tu2} - v_{su2}v_{tu1})/(v_{su1} + v_{tu2} - v_{su2} - v_{tu1})$.
(Determination of $f_b$ and $f_a$)
Step 8.  Set $j = 8$, $S = \{s\}$, $T = \{u\}$, $f_i = 0$ and $v = v^*$.  Go to
         Step 12.
Step 9.  Set $f_i = f$, $S = \{s, u\}$, $T = \{t\}$, $v = v^*$.  Go to Step 12.
Step 10. Set $g_1 = f$, $S = \{s, t\}$, $T = \{u\}$, $v = 2v^*$.  Go to Step 12.
Step 11. Set $g_2 = f$.  Go to Step 13.
Step 12. Find a flow of value $v$ from $S$ to $T$ in $G$ under the
```

constraint $\sum_{i \in B(s)} f(s,i) = v^*$, taking f_i as the initial flow. Set $j = j + 1$. Go to Step j.

Step 13. Set $f_b = (g_1 + g_2)/2$ and $f_a = (g_1 - g_2)/2$. Stop.

Number of Operations of Algorithm 5.1

We note that a maximum flow can be found in time $O(n \max(n^2, m \log n))$, see [4] and [6]. So, Steps 6 and 12 need so much time. The time for the other steps is negligible as compared with that required in Steps 6 and 12. Therefore the network flow problem with a check vertex can be solved in time $O(n \max(n^2, m \log n))$.

5.3 Minimum Spanning Tree with Normal Variates as Weights

Let $G = (V, A)$ be an undirected network where V is the vertex set and A is the arc set. Arcs (denoted by e(j) or only by j) have weights c(j) where weights c(j) are independent normal variates with means m(j) and variances v(j). Let T denote a spanning tree of G and let AST refer to the set of all spanning trees of G. For any tree T, we define the following:

$$C(T) = \sum_{j \in T} c(j)$$

$$M(T) = \sum_{j \in T} m(j)$$

$$S(T) = \left(\sum_{j \in T} v(j) \right)^{1/2}$$

$$V(T) = \sum_{j \in T} v(j).$$

We assume a given constant f is such that $f \geq \min_{T \in AST} M(T)$. Then the (stochastic) minimum spanning tree problem is written:

$$\max_{T \in AST} \Pr\{C(T) \leq f\}.$$

Noting that C(T) is also distributed according to the normal distribution with mean M(T) and variance V(T), the problem of solving the preceding problem is equivalent to solving the following one:

$$\max_{T \in AST} \Pr\{C(T) - M(T))/S(T) \leq (f - M(T))/S(T)\}.$$

Hence our task is to maximize $(f - M(T))/S(T)$. Note that $\max(f - M(T))/S(T) > 0$ is equivalent to $f > \min M(T)$. Hence if $f \leq \min M(T)$, then $\max \Pr\{C(T) \leq f\} \leq 0.5$.

Let T(t) denote a spanning tree minimizing $(M(T) + tS(T))$ for the parameter $t > 0$, and let T* and t* denote the spanning tree and value of t satisfying $\min(M(T) + tS(T)) = f$, i.e., $M(T^*) + t^*S(T^*) = f$. We note that from the results of ratio minimization problems (see [5]), we have $\max_{T \in AST} (f - M(T))/S(T) = t^*$. Therefore what we must do is to seek $T^* = T(t^*)$.

For two spanning trees T and T', if $M(T) = M(T')$ and $S(T) = S(T')$, then we write $T \cong T'$. In other words, we do not distinguish them even if the arcs in T and those in T' are different. Otherwise we write $T \neq T'$.

Lemma 5.1 For any $x > 0$, define $\overline{T}(x)$ to be a spanning tree minimizing $(M(T) + xV(T))$ over all T's in AST. Then $T^* \in \{\overline{T}(x) \mid x > 0\}$.

Proof. Let $T \cong T(t)$ and $T' \neq T(t)$. Then we have $M(T') + tS(T') \geq M(T) + tS(T)$ by the optimality of $T(t)$ or $M(T') - M(T) \geq t(S(T) - S(T'))$, and we have $M(T') \neq M(T)$ and/or $S(T') \neq S(T)$ by definition of different trees $T' \neq T$. If we set $x = t/(2S(T))$, then it follows that

$$\begin{aligned} &M(T') + xV(T') - (M(T) + xV(T)) \\ &\quad = M(T') + x(S(T'))^2 - (M(T) + x(S(T))^2) \\ &\quad = M(T') - M(T) - (t/(2S(T)))((S(T))^2-(S(T'))^2) \\ &\quad \geq t(S(T) - S(T')) - (t/(2S(T)))((S(T))^2 - (S(T'))^2) \\ &\quad = (t/(2S(T)))(S(T) - S(T'))(2(S(T)) - (S(T) + S(T'))) \\ &\quad = (t/(2S(T)))(S(T) - S(T'))^2 \\ &\quad \geq 0. \end{aligned}$$

If $M(T') + tS(T') = M(T) + tS(T)$ and $S(T') = S(T)$, then we have $M(T') = M(T)$, which is a contradiction. Hence $M(T') + tS(T') =$

$M(T) + tS(T)$ means $S(T') \neq S(T)$, and $S(T') = S(T)$ means $M(T') + tS(T') > M(T) + tS(T)$. Thus for $x = t/(2S(T))$ we have $M(T') + xV(T') > M(T) + xV(T)$, i.e., $T(t) \cong \overline{T}(x = t/(2S(T)))$. (Recall the definitions of $T(t)$ and $\overline{T}(x)$.) This shows $T(t) \in \{\overline{T}(x) | \ x > 0\}$. Since T^* is a spanning tree such that $M(T(t)) + tS(T(t)) = f$, we have $T^* \in \{\overline{T}(x) | \ x > 0\}$. Q.E.D.

Let $t(x) = (f - M(\overline{T}(x)))/S(\overline{T}(x))$ for $\overline{T}(x)$, then $t^* = \max\{t(x) | \ x > 0\}$ by definition of t^* and Lemma 5.1. Although it is very difficult to find $T(t)$, $\overline{T}(x)$ is easily found since

$$M(T) + xV(T) = \sum_{j \in T} (m(j) + xv(j)) = \sum_{j \in T} w_x(j)$$

where $w_x(j) = m(j) + xv(j)$ for any arc $e(j)$, namely since finding $\overline{T}(x)$ is equivalent to finding a usual minimum spanning tree. We refer to $w_x(j)$ as a (deterministic) weight of arc $e(j)$.

Define $R(i,j) = (m(i) - m(j))/(v(j) - v(i))$ for each pair of arcs such that $m(i) < m(j)$ and $v(i) > v(j)$. Rearrange different positive $R(i,j)$ as follows:

$$0 = R(0) < R(1) < \dots < R(h) < R(h + 1) = \infty$$

where h is the number of such $R(i,j)$.

Lemma 5.2 For any x and x' such that $R(k) < x,x' < R(k + 1)$ $k = 0, 1, \dots, h$, $\overline{T}(x) \cong \overline{T}(x')$ holds.

Proof. See [1]. Q.E.D.

Letting $x(k) = (R(k) + R(k + 1))/2$ for $k = 0, 1, \ldots, h - 1$ and letting $x(h) = R(h) + 1$, we have the following theorem.

Theorem 5.3 $T^* \in \{\overline{T}(x) \mid x = x(k),\ k = 0, 1, \ldots, h\}$.

Proof. From Lemma 5.2 it follows that $\overline{T}(x) \cong \overline{T}(x(k))$ holds for any x such that $R(k) < x < R(k + 1)$. This together with Lemma 5.1 shows this theorem. Q.E.D.

Let $R(k) = R(i,j)$, then we have $w_x(i) < w_x(j)$ for $x < R(k)$, $w_x(i) > w_x(j)$ for $x > R(k)$. That is, if $x = x(k - 1)$, then $w_x(i) < w_x(j)$, and if $x = x(k)$, then $w_x(i) > w_x(j)$.

Proposition 5.4 If $e(i) \in \overline{T}(x(k - 1))$, $e(j) \notin \overline{T}(x(k - 1))$ and $\overline{T}(x(k - 1)) \cup e(j)$ has a cycle in which both $e(i)$ and $e(j)$ are included, then $\overline{T}(x(k)) = \overline{T}(x(k - 1)) \cup e(j) - e(i)$. Otherwise, $\overline{T}(x(k)) = \overline{T}(x(k - 1))$.

Proof. With respect to $x(k - 1)$ and $x(k)$ the order among all weights is the same except $w_x(i)$ and $w_x(j)$. Then it is obvious that $\overline{T}(x(k))$ created as above is a minimum spanning tree since there exists no other spanning tree through an elementary tree transformation whose weight is less than that of $\overline{T}(x(k))$, see Deo [2, Theorem 3-16]. Q.E.D.

```
Algorithm 5.2
Step 1.  Compute $x(k)$'s.
Step 2.  (i)  Generate $\overline{T}(x(k))$ from $\overline{T}(x(k - 1))$.
         (ii) Set $t(k) = (f - M(\overline{T}(x(k)))/S(\overline{T}(x(k)))$.
Step 3.  Find $t^* = \max t(k)$.  Stop.
```

We will estimate the time complexity of the above algorithm. First we note that the number h is bounded by the number of intersection points of m functions ($m = |A|$) $w_x(j) = m(j) + xv(j)$, which means $h = O(m^2)$. Hence Step 1 is computed in time $O(m^2)$. Let us consider the time complexity of Step 2. Initially $\overline{T}(x(0))$ is obtained in time $O(m \log \log n)$, see [8], where $n = |V|$.

Let two arcs $e(i)$ and $e(j)$ for $R(k) = R(i,j)$ be written as $i(k)$ and $j(k)$ respectively. Label each arc as 1 if it belongs to a spanning tree and as 0 otherwise. Then the check whether $i(k)$ is contained by $\overline{T}(x(k - 1))$ or not is done in time $O(1)$ by looking up the label of $i(k)$. If $i(k) \in \overline{T}(x(k - 1))$ and $j(k) \notin \overline{T}(x(k - 1))$, then it is necessary to determine whether or not $i(k)$ and $j(k)$ are both included in a cycle of $\overline{T}(x(k - 1)) \cup j(k)$, i.e., whether or not $\overline{T}(x(k - 1)) \cup j(k) - i(k)$ is a spanning tree. If $\overline{T}(x(k - 1)) \cup j(k) - i(k)$ is not a spanning tree, then it must consist of two components, which is detected in time $O(n)$ by the depth first search due to Tarjan [7]. Hence Step 2 (i) needs $O(m^2n)$ time. If we have the value of $M(\overline{T}(x(k - 1)))$, then the value of $M(\overline{T}(x(k)))$ is given in time $O(1)$ since at most two arcs change. So is $S(\overline{T}(x(k)))$. Hence Step 2 (ii) is done in time $O(m^2)$. Lastly Step 3 takes $O(m^2)$ time. Therefore the overall time complexity is $O(m^2 n)$.

References

[1] Chandrasekaran, R.,"Minimal ratio spanning tree," Mathematics of Operations Research 4 (1979) 414-424.

[2] Deo, N., Graph Theory with Applications to Engineering and Computer Science, Prentice Hall, New Jersey, 1974.

[3] Hu, T.C.,"Multi-commodity network flows," Operations Research 11 (1963) 344-360.

[4] Karzanov, A.U.,"Determinig the maximum flow in a network by the method of preflows," Soviet Mathematical Doklady 15 (1974) 434-437.

[5] Lawler, E.L., Combinatorial Optimization: networks and matroids, Holt, Rinehart and Winston, New York, 1976.

[6] Sleator, D.,"An O(nm log m) algorithm for maximum network flow ," Ph. D. Dissertation, Stanford University, 1980.

[7] Tarjan, R.E.,"Depth first search and linear graph algorithms," SIAM journal on Computing 1 (1972) 146-160.

[8] Yao, A.C.,"An $O(|E| \log \log |V|)$ algorithm for finding minimum spanning trees," Information Processing Letters 4 (1975) 21-23.

LIST OF THE PAPERS DUE TO THE AUTHOR

1 A minimax flow problem, Mathematica Japonica 24 (1979) 65-71.

2 Flow in a network with a check node, Journal of the Operations Research Society of Japan 23 (1980) 111-117.

3 Minimax cost flow problem, Technology Reports of the Osaka University 30 (1980) 39-44.

4 Finding the weighted minimax flow in a polynomial time, Journal of the Operations Research Society of Japan 23 (1980) 268-272.

5 Weighted minimax real-valued flows, Journal of the Operations Research Society of Japan 24 (1981) 52-60.

6 Minimum spanning tree with normal variates as weights, Journal of the Operations Research Society of Japan 24 (1981) 61-66.

7 Algorithm for one machine job sequencing with precedence constraints, Journal of the Operations Research Society of Japan 24 (1981) 159-169.

8 Routing a vehicle with the limitation of fuel, Journal of the Operations Research Society of Japan 24 (1981) 277-281.

9 Shortest path algorithms, Technology Reports of the Osaka University 31 (1981) 197-204.

10 Optimal sharing, Mathematical Programming (to appear).

11 Algorithm for finding shortest paths from a fixed origin to other nodes (in preparation).

12 Note on a rectilinear distance round-trip location problem (submitted).

13 Two routing problem with the limitation of fuel (submitted).

14 Multifacility minimax location with rectilinear distances (submitted).

15 One- and two-commodity sharing problems on networks (in preparation).

16 On the cyclic staffing problem with two consecutive days off each week (submitted).

ACKNOWLEDGEMENTS

The author wishes to express his sincere appreciation to Professor T. Nishida for supervising this thesis. His continuous encouragement and valuable comments have helped to accomplish the thesis.

The author also wishes to express his sincere appreciation to Professor H. Sugiyama and Professor Y. Tezuka for their valuable suggestions and helpful comments for improving the thesis.

Furthermore, the author would like to express his gratitude to Dr. H. Ishii for his great encouragement and valuable comments to complete the thesis.