# NETWORK DESIGN PROBLEMS FOR IMPROVING FACILITY LOCATIONS

by

DIVINAGRACIA I. INGCO

B. S. Mathematics
University of the Philippines, 1986

Submitted in Partial Fulfillment of
the Requirements of the Degree of
Master of Science in Operations Research

at the

Massachusetts Institute of Technology

March 1989



Signature of
Author_____
Interdepartmental Program in Operations Research
March 1989

Certified by_____
Amedeo R. Odoni
Professor, Aeronautics and Astronautics, and Civil Engineering
Thesis Supervisor

Accepted by_____
Amedeo R. Odoni
Codirector, Operations Research Center

# NETWORK DESIGN PROBLEMS FOR IMPROVING FACILITY LOCATIONS

by

DIVINAGRACIA I. INGCO

Submitted in Partial Fulfillment of
the Requirements of the Degree of
Master of Science in Operations Research

## ABSTRACT

This thesis addresses two problems. The first problem, which serves as a background to the second problem, is a simple variation of the network design problem where given a number of facilities and a set of nodes with associated demands, the optimal set of links that would minimize the median problem (or the center problem) is identified. The second problem is concerned with improving an existing network through either reducing the length of existing links or adding new links to the network. The total amount of link reduction or link addition is subject to a budget constraint, where costs of reduction or addition may vary across arcs.

All problems are initially solved for a spanning tree, then solved again for a network. All algorithms are written for the one-facility case. It will be shown, at the end, that a k-facility problem may be reduced to a one-facility case through a simple transformation.

Using the median objective function value as the measure of improvement, we identify the links and the amounts to reduce in a given spanning tree or network. In the spanning tree case, an iterative algorithm identifies the optimal strategy. This procedure, however, when extended to the case of networks may not generate the optimal solution. For some special situations, however, a nonlinear mixed integer program can be formulated to determine the optimal strategy.

Using the center objective function value as the measure of improvement, we identify the links and amounts to reduce in a given spanning tree or network. In the spanning tree case, the solution is generated optimally. For the case of networks, a calibrating algorithm that requires solving an integer program repeatedly can determine the optimal solution.

A spanning tree or network may also be improved by adding new arcs. If we are required to add only one arc to the spanning tree or network, the solution may be determined optimally. If several arcs may be added, an iterative heuristic that approximates the optimal strategy is presented for each objective function.

Thesis Supervisor: Prof. Amedeo Odoni
Title: Professor of Aeronautics and Astronautics, and of Civil Engineering

Thesis Supervisor: Prof. Oded Berman
Title: Professor of Management Science

## ACKNOWLEDGEMENTS

I would like to thank my adviser, Professor Amedeo Odoni, for introducing me to the problems addressed in this thesis, and, through his invaluable counsel, for helping me develop the material and clearly articulate my ideas. His continued guidance, assistance, and support have motivated me through my academic endeavors and have never been more evident than in the course of writing this thesis. All these are most deeply appreciated.

I would like to express my sincerest thanks to Professor Oded Berman whose expert advice and patient guidance have been crucial to the writing of this thesis. His suggestions have given greater depth to the material and clarity in the presentation, and his encouragement has sustained my interest in this work. More importantly, through his commitment to this undertaking, I have come to fully appreciate the challenges and rewards inherent to the writing of a thesis.

To all my friends, who have been family to me in Boston, I extend my genuine thanks. To old friends who have always been with me in spirit, I express my gratitude.

Finally, it is with deepest affection that I thank my father and mother. In moments of doubt and difficulty, it is from their unfailing support that I derive renewed purpose and strength. Accomplishments, such as this, gain more meaning when shared with them.

## TABLE OF CONTENTS

# Chapter 1

# INTRODUCTION

In the field of networks, there is a special class of problems known as facility location problems. In their discrete form, facility location problems can be stated as follows: given an undirected graph with nodes that have associated demands, a set of points (or facility locations) of prespecified cardinality is chosen so as to minimize either the sum of the weighted shortest paths from the vertices to any of the facilities, or the maximum of the weighted distance from any of the vertices to the closest facility to it. Depending on whether the first criterion was used or the second, the problem is referred to as the median problem or the center problem respectively.

A second class of problems is known as network design problems. In its simplest form, a network design problem can be described as follows: given an undirected graph with nodes that have associated demands (to be referred henceforth as a weighted network), one must construct a network, connecting all the nodes of the graph, such that the sum of the shortest paths between all pairs of vertices is minimized and the total length of the links does not exceed a specified budget.

## 1.1. The Problem

This thesis addresses two problems. The first problem, which serves as a background to the second problem, is a simple variation of the network design problem where given a number of facilities and a set of nodes with associated demands, the optimal set of links that would minimize the median problem (or the center problem) is identified.

The second problem, which comprise the bulk of the thesis, is concerned with improving an existing network through either reducing the length of existing links or adding new links to the network. By improving a network we mean reducing the

median objective function value or the center objective function value. The appropriate criterion to use will depend on the real-life application that motivated the problem. The total amount of link reduction or link addition is subject to a budget constraint, where costs of reduction or addition may vary across arcs.

**1.2. Scope and Preview**

To guide and facilitate the reading of the succeeding chapters, we outline here the organization of the thesis. All algorithms are written for the one-facility case. It will be shown, at the end, that a k-facility problem may be reduced to a one-facility case through a simple transformation. Hence, algorithms developed for the one-facility problem will apply.

All problems are initially solved for a spanning tree, then solved again for a network, extending related concepts and procedures whenever possible. Median problems and center problems are discussed separately.

We begin our analysis in Chapter 2 by addressing the first problem described in Section 1.1. Given a network with one-facility, the optimal spanning tree that minimizes the median or center objectives is found to be the shortest path tree rooted at the facility.

Chapters 3, 4, 5, and 6 address the second problem described in Section 1.1. Chapters 3 and 4 use the median objective function as the measure of improvement and Chapters 5 and 6 use the center objective function.

In Chapter 3, we identify the links and the amounts to reduce in a given spanning tree or network, to best improve the median objective function. The cost of reducing an arc may or may not vary across arcs. In the spanning tree case, an iterative algorithm identifies the optimal strategy . This procedure, however, when extended to the case where we are given a network may not generate the optimal solution. For some special situations, however, a nonlinear mixed integer program can be formulated to determine the optimal strategy.

In Chapter 4, we try to improve the median objective function value by adding arcs to the given spanning tree or network. The cost of adding an arc may or may not vary across arcs. If we are required to add only one arc to the spanning tree or network, the solution may be determined optimally. If several arcs may be added, an iterative heuristic that approximates the optimal strategy is presented. It is difficult to solve this problem optimally because of the dependency of arcs on one another. This property will be illustrated in Section 4.1.

In Chapter 5, we identify the links and amounts to reduce in a given spanning tree or network, to best improve the center objective function. In the spanning tree case, where the cost of reduction is equal for all arcs, the solution is generated optimally by an iterative algorithm. However, the iterative algorithm cannot be extended to the spanning tree case, where the costs of reduction vary across arcs. Instead, a mixed integer program may be solved to obtain the optimal strategy. For the case of networks, a calibrating algorithm that requires solving an integer program repeatedly can determine the optimal solution.

In Chapter 6, we try to improve the center objective function value by adding arcs to the given spanning tree or network. The cost of adding an arc may or may not vary across arcs. As in Chapter 4, if we are required to add only one arc, the solution obtained from the suggested algorithm is optimal. However, if more than one arc may be added, the algorithm we designed only approximates the optimal solution. Again, this is due to the dependency of arcs on one another.

In Chapter 7, we will show how a k-facility problem may be transformed into a one-facility problem.

### 1.3. A Brief Review of Related Literature

Magnanti and Wong (1984) compiled and reviewed the then-existing literature on the network design problem and its various extensions. In its most general form, the network design problem has been defined for a directed graph, with multiple

commodities that have specific origins and destinations. Arcs have flow capacities and routing costs that vary for each commodity. Inclusion of arcs in a network may incur certain fixed costs (like construction expenses) and the selection of arcs may hence be constrained by a budget. The objective function may be linear or nonlinear. If it is linear, it is the sum of the cost incurred to move the commodities and the fixed costs incurred when establishing the various arcs. When other factors, like congestion are incorporated into the model, the objective function may become nonlinear. The network design problem then solves for two sets of unknowns: the set of arcs to be included in the network, as indicated by binary variables and the amount of flow of each commodity on each chosen arc, as indicated by continuous variables.

From this general model, Magnanti and Wong have shown that other network problems such as the minimal spanning tree problem, shortest path problem, traveling salesman problem, vehicle routing problem, and facility location problem, among others, may be derived by imposing certain conditions on the network specifications.

Johnson, Lenstra, and Rinnooy Kan (1978) established that the network design problem defined on a weighted undirected graph is NP-complete. This justified research into enumerative algorithms to find the optimal solution and into heuristic procedures to determine reasonable approximations.

The problem was investigated by Scott (1969), Dionne and Florian (1979), and Leblanc (1975) as a combinatorics problem of examining all the possible combinations of zero-one arc incidences. In each case, the set of solutions is continually partitioned into smaller subsets with the aim of eventually eliminating a subset from further consideration. The criteria for branching and bounding differ among these three papers. A large part of the literature according to Magnanti and Wong is devoted to finding better lower bounds; they also noted that Bender's decomposition has also been used to solve the network design problem optimally.

The heuristics that have been devised employ variations of three basic steps: add arcs one at a time to a feasible solution (for example, the minimal spanning tree as suggested by Scott); delete arcs one at a time from a completely connected network; and interchange arcs iteratively.

Since we are concerned with networks with known facility locations that provide some service to the other nodes of the network, some concepts from facility location theory will also be utilized. Given a network, denote a node in the network as $i$, and denote the node set as I. We want to choose $k$ points in the network: $x_1, x_2, \ldots, x_k$ such that we minimize either:

$$Z_M = \sum_i h_i \min\{d(x_1,i), d(x_2,i),\ldots,d(x_k,i)\}$$

or,

$$Z_C = \max_i \left[h_i \min\{d(x_1,i), d(x_2,i),\ldots,d(x_k,i)\}\right]$$

where, $h_i$ = weight of node $i$

and $d(x_\alpha,i)$ = shortest distance from point $x_\alpha$ to node $i$, $\alpha = 1,2,\ldots,k$

If we minimize $Z_M$, we refer to the problem as the $k$-median problem. If we minimize $Z_C$, we refer to the problem as the $k$-center problem.

A theorem established by Hakimi (1964) states that there is at least one $k$-node subset of I that minimizes $Z_M$. Hence, the search for a set of optimal facility locations for the $k$-median problem may be confined to the set of nodes I.

There is no analogous theorem for the center problem. The optimal facility locations may be anywhere in the network. Median and center facility location problems are reviewed extensively in Handler and Mirchandani, Location on Networks: Theorems and Algorithms (1979).

Essentially the problem addressed by this thesis is determining the proper set of arcs to satisfy predetermined criteria under specified conditions. Inasmuch as

network design problems solve for an optimal set of arcs and facility location problems search for an optimal set of points, it is expected that the methods that will be applied in this thesis are more closely related to the procedures used in solving the network design problem.

Also, the Dijkstra algorithm, that solves for the shortest path tree rooted at a specific node, and ordering algorithms which arrange variables in some ascending or descending order are repeatedly used.

# Chapter 2

## THE OPTIMAL SPANNING TREE

### 2.1 Optimal Subgraph for the One-Facility Median Problem

Given a set of weighted nodes N of an undirected graph, a fixed facility X, $X \notin N$, we want to find a subgraph that minimizes the average distance (or the total weighted distance) of the nodes to the facility, i.e.

$$\text{minimize } Z_M = \sum_{j \in N} h_j \, d(X,j)$$

where $h_j$ = weight of the node $j$, $h_j \geq 0, j = 1,2,...,N$

and $d(X,j)$ = shortest distance from node $j$ to node X, $d(X,j) \geq 0$

It is obvious that the desired subgraph is a spanning tree.

Proof: Consider the shortest path tree T rooted at X, and let $d_T(X,j)$ be the distance between node $j$ and node X according to T. Suppose this is not the same as the desired spanning tree that minimizes the median problem. Then there exists an optimal spanning tree $T_M$, with shortest distances $d_M(X,j)$ between node $j$ and node X, where

$$\sum_{j \in N} h_j d_M(X,j) < \sum_{j \in N} h_j d_T(X,j) \qquad (2.1)$$

By definition,

$$d_T(X,j) \leq d_M(X,j) \qquad \text{for all } j \in N$$

$$\Rightarrow \; h_j d_T(X,j) \leq h_j d_M(X,j) \qquad \text{since } h_j \geq 0$$

$$\Rightarrow \sum_{j \in N} h_j d_T(X,j) \leq \sum_{j \in N} h_j d_M(X,j)$$

which contradicts (2.1). Hence, the shortest path tree T rooted at X must be the spanning tree that minimizes the median objective function.

## 2.2 Optimal Subgraph for the One-Facility Center Problem

In exactly the same manner as in Section 2.1, if the objective is to :

minimize $Z_C = \max h_j d(X,j)$

we can show that T is the optimal solution. Otherwise there exists another spanning tree $T_C$ with shortest distances $d_C(X,j)$ between node $j$ and node X, where

$$\max_{j\in N} h_j d_C(X,j) < \max_{j\in N} h_j d_T(X,j) \qquad (2.2)$$

By definition,

$$d_T(X,j) \le d_C(X,j) \qquad \text{for all } j\in N$$

$$\Rightarrow\ h_j\, d_T(X,j) \le h_j\, d_C(X,j) \qquad \text{since } h_j \ge 0$$

$$\Rightarrow\ \max_{j\in N} h_j\, d_T(X,j) \le \max_{j\in N} h_j\, d_C(X,j)$$

which contradicts (2.2). So the shortest path tree rooted at X must also be the spanning tree that minimizes the center objective function.

## 2.3 Determining the Objective Function Value

Given $G(N \cup X,A)$ where X is the fixed facility, we can use the Dijkstra Algorithm to find the shortest path tree, with the inclusion of counters that will eventually provide the value of the median and center objective functions. Let these counters be $Z_M$ and $Z_C$, respectively. Every time a node $j$ is closed, $Z_M$ is increased by $h_j d(X, j)$ and $Z_C$ takes the value of the maximum between the current $Z_C$ and $h_j d(X, j)$. When all N nodes have been closed, the final values of $Z_M$ and $Z_C$ are equal to the objective function values. The algorithm is more precisely described in Appendix A.

## Chapter 3

## LINK REDUCTIONS IN THE ONE-MEDIAN PROBLEM

### 3.1. Marginal Contibution of Arcs in a Spanning Tree to the Median Function

*Suppose that given a spanning tree with a fixed facility, we want to find the arc with the largest marginal contribution to the median objective function.*

If we could reduce any link, the link we would reduce to obtain maximum improvement in the objective function would be the link with the highest marginal contribution. This motivates the problem under consideration.

Consider the following illustration:

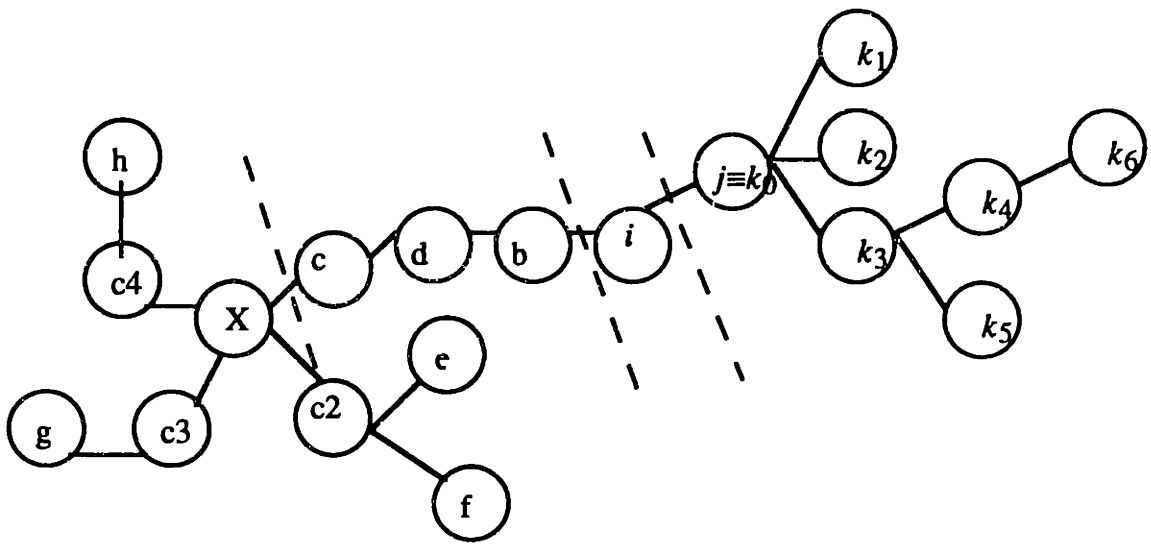



*Figure 3.1*

Let $a_{ij}$, define the arc incident to nodes $i$ and $j$, where $d(X,i) < d(X,j)$. A reduction of $a_{ij}$, by a marginal amount $\Delta$ will improve by $\Delta$ the following: $d(X,j)$, and $d(X,k)$ where k is a node in the cutset performed on $a_{ij}$ (nodes $j$, $k_1$, $k_2$, ..., $k_6$ in figure 3.1). All other nodes are unaffected. The total improvement to $Z_M$ is $I_{ij} = \Delta(\sum_k h_k.)$ If we perform the

cut on $a_{bi}$, the total improvement to $Z_M$ is $I_{bi} = I_{ij} + \Delta h_i$, which is larger than $I_{ij}$. If we follow this line of thought, in general, as we move closer to the facility, I increases. To obtain, maximum improvement in $Z_M$, we want to perform the reduction on some arc $a_{Xc}$ incident to the facility X (using this notation, we henceforth call "c-nodes" those nodes that are adjacent to the facility), and the corresponding improvement is $I_{Xc} = \Delta(\sum_k h_k.)$, where $k$ is a node in the cutset resulting from a cut on $a_{Xc}$.

We confine the search for the desired arc to the various $a_{Xc}$. For each cutset formed from a cut on each of the $a_{Xc}$, a corresponding weight $\sum_k h_k$ is associated; call this weight $H_c$. A proposed algorithm (Algorithm C1) that determines this weight is discussed in Appendix C1. Finding the arc that has the largest marginal contribution to $Z_M$ reduces to identifying the arc $a_{Xc}$ whose corresponding cutset has the heaviest weight.

To illustrate the preceding discussion, consider figure 3.1 again. The arcs incident to the facility and the weights of their corresponding cutsets are tabulated below:

| | |
|---|---|
| $a_{Xc}$ | $H_c = h_c + h_d + h_b + h_i + \sum_k h_k$ |
| $a_{Xc_2}$ | $H_{c_2} = h_{c_2} + h_e + h_f$ |
| $a_{Xc_3}$ | $H_{c_3} = h_{c_3} + h_g$ |
| $a_{Xc_4}$ | $H_{c_4} = h_{c_4} + h_h$ |

The arc with the largest corresponding H-value has the largest marginal contribution to $Z_M$.

### 3.2. Link Reduction of Y Units in a Spanning Tree

*Given a spanning tree and a fixed facility, we want to know which arcs should be reduced and by how much, if we could reduce a total of Y arc-units throughout the spanning tree, to obtain maximum improvement of $Z_M$.*

This could translate physically to the improvement of certain existing systems such that time or distance to the facility is reduced.

From Section 3.1, the arc with largest marginal contribution to $Z_M$ can be determined. Suppose we have $\gamma$ nodes $c_1, c_2, ..., c_\gamma$ which are linked to the facility by $a_{Xc_1}, a_{Xc_2}, ..., a_{Xc_\gamma}$, respectively. The nodes have associated weights $H_{c_1}, H_{c_2}, ..., H_{c_\gamma}$ as discussed previously, and suppose that the nodes have been named in order so that $H_{c_1} > H_{c_2} > ... > H_{c_\gamma}$

Step 1. It would be better to "invest" as much of our Y arc-units as possible in $a_{Xc_1}$. If we require that an arc may not be reduced to zero length and that there is a limit L* to how short an arc may be (where L* as specified by the decision maker is less than or equal to the minimum of all the arc lengths in the spanning tree) we can reduce $a_{Xc_1}$ by:

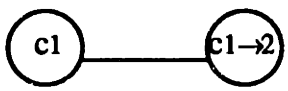
$l(X,c_1) - L^*$ units  if $Y > l(X,c_1) - L^*$

or  Y units  if $Y \leq l(X,c_1) - L^*$

Step 2. If $Y \leq l(X,c_1) - L^*$, we are done, with improvement on $Z_M$, $I = YH_{c_1}$. Otherwise , let $I = [l(X,c_1) - L^*] H_{c_1}$. Let $Y = Y - [l(X,c_1) - L^*]$ = remaining budget. We next examine what happens when an arc has been reduced to the limit.

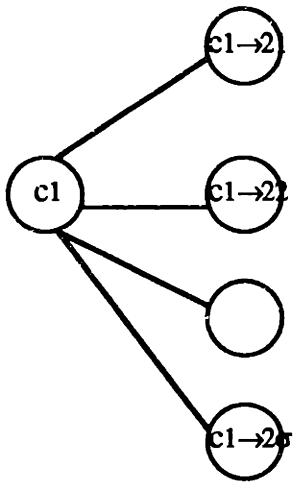
Suppose $a_{Xc_1}$ has been reduced to the length L*. One of three things may occur.

Case 1.




Besides X, there is only one node adjacent to $c_1$, called $c_{1\to2}$. Since $a_{Xc_1}$ has been reduced to the limit L*, $c_{1\to2}$ will act like a c-node. The node $c_{1\to2}$ will carry a weight $H_{c_{1\to2}} = H_{c_1} - h_{c_1}$. Locate the place of $H_{c_{1\to2}}$ in the order of remaining $H_c$'s.

Case 2.




Besides x, there are $\sigma$ nodes adjacent to $c_1$. Since $a_{Xc_1}$ has been reduced to the limit L*, each of nodes $c_{1\to21}, c_{1\to22}, \ldots, c_{1\to2\sigma}$ acts like a c-node. We can find the corresponding weights associated with the nodes $c_{1\to21}, c_{1\to22}, \ldots, c_{1\to2\sigma}$ through Algorithm C2 described in Appendix C2. We can find $H_{c_{1\to21}}, H_{c_{1\to22}}, \ldots, H_{c_{1\to2\sigma}}$ and we order these again with the remaining $H_c$'s.

Case 3. $c_1$ is an end node. Do nothing.

Step 3. Having a new ordering of weights $H_c$, pick again the arc with the largest $H_c$, and reduce it by:

$l(X,c_1) - L^*$ units if $Y > l(X,c_1) - L^*$

or $Y$ units if $Y \le l(X,c_1) - L^*$

To determine the total improvement, add the improvement due to this reduction, which is $YH_{c_1}$ or $[l(X,c_1) - L^*]H_{c_1}$, to any improvements made through previous iterations. Repeat Steps 2 and 3 until Y has been exhausted or until all arcs in the spanning tree have been reduced to the limit L*.

### 3.3 Link Reduction on Spanning Tree Arcs with Varying Costs

*Suppose that it costs $c_{ij}$ to reduce $a_{ij}$ by one unit and we have a budget of C units. We want to know how to effect the reductions of the arcs of a spanning tree with a fixed facility to best improve $Z_M$.*

Section 3.2 becomes a special case of this problem, where the $c_{ij}$ are equal for all links in the spanning tree. In the more general case examined in this section, since arcs have different costs, reduction on $a_{Xc}$ is not necessarily preferred.

For every node in the network $j$, we can calculate, $H_j$, the weight of all nodes that follow $j$ away from X through Algorithm C2. Knowing these weights $H_j$, the reduction strategy that would best improve $Z_M$ in this spanning tree subject to our budget constraint would be to reduce $x_{ij}$ units in arc $a_{ij}$ where $x_{ij}$ is given by the following linear program:

$$\text{maximize} \quad I = \sum x_{ij}H_j$$

$$\text{subject to} \quad \sum x_{ij}c_{ij} \le C$$

$$l(i,j) - x_{ij} \ge L^* \qquad \text{for all arcs}$$

$$x_{ij} \ge 0$$

For each arc we can define:

$$r_{ij} = \frac{H_j}{c_{ij}} = \text{amount of improvement per unit cost spent on } a_{ij}$$

We then order the different $r_{ij}$ from largest to smallest, and reduce as much as possible the arc $a_{ij}$ that corresponds to the largest $r_{ij}$. If the budget has not yet been exhausted, move to the arc with the second highest $r_{ij}$ and reduce it as much as possible. We go through the ordered list of $r_{ij}$ values to determine which $a_{ij}$ to reduce next. Once we have exhausted the budget or reduced all arcs in the spanning tree to the limit L*, we are done. More precisely, we can refer to the following algorithm.

Step 1. Order the different $r_{ij}$ from largest to smallest.

Step 2. Pick the arc with largest $r_{ij}$ and let the reduction on this arc be

$$x_{ij} = \begin{cases} l(i,j) - L^* & \text{if } c_{ij}[l(i,j) - L^*] \le C \\ \dfrac{C}{c_{ij}} & \text{otherwise} \end{cases}$$

Since $\frac{C}{c_{ij}}$ is real and may be impractical physically we can require it to be $\left\lfloor \frac{C}{c_{ij}} \right\rfloor$.

Step 3. Adjust $C = C - (c_{ij})(x_{ij})$. C is our remaining budget. If this value is zero, we are done. Otherwise, we choose the arc with the next largest $r_{ij}$ and determine the appropriate $x_{ij}$, as defined in Step 2.

Step 4. Once the budget has been exhausted or all the arcs in the spanning tree have been reduced to the limit L*, we are done. However if we are using the $\left\lfloor \frac{C}{c_{ij}} \right\rfloor$ criterion, it is possible that we have some budget surplus and that we "skipped" arcs.

Example:

We have four arcs: A, B, C, D where,

| Arc ($\alpha$) | Arc Length $l(\alpha)$ | Cost $c_\alpha$ | Ratio $r_\alpha$ |
|---|---|---|---|
| A | 15 | 20 | 10 |
| B | 20 | 20 | 9 |
| C | 40 | 4 | 3 |
| D | 8 | 2 | 2 |

Let L* = 5, C = 105. If we impose the $\left\lfloor \frac{C}{c_{ij}} \right\rfloor$ criterion, $c_A[l(A) - L^*] = 20\ [15 - 5] = 200 \not\leq 105$. Hence, $x_A = \left\lfloor \frac{C}{c_A} \right\rfloor = \left\lfloor \frac{105}{20} \right\rfloor = 5$. $C = 105 - (20)(5) = 5$. We move to arc B since $r_B = 9$. $c_B[l(B) - L^*] = 20\ [20 - 5] = 300 \not\leq 5$, hence $x_B = \left\lfloor \frac{C}{c_B} \right\rfloor = \left\lfloor \frac{5}{20} \right\rfloor = 0$. We do not reduce arc B. $c_C[l(C) - L] = 4\ [40 - 5] = 140 \not\leq 5$. $x_C = \left\lfloor \frac{5}{4} \right\rfloor = 1$ and $C = 5 - (4)(1) = 1$. We move to arc D. $c_D[l(D) - L^*] = 2\ [8 - 5] = 6 \not\leq 1$. Hence, $x_D = \left\lfloor \frac{C}{c_D} \right\rfloor = \left\lfloor \frac{1}{2} \right\rfloor = 0$ and C = 1. There is some budget leftover.

This algorithm, by construction allocates the budget to the "cheapest" arc. As such the resulting strategy is optimal. We could use this technique in lieu of the linear program mentioned initially.

### 3.4. Marginal Contribution of Arcs in a Network to the Median Function

Instead of a spanning tree, suppose we started with a graph with one facility and we want to answer the questions about link reductions that were asked in the

past sections. In the most general case, the problem was to define the arcs to reduce and by how much, if each arc costs different to reduce and a budget constraint exists.

*Given a network with a fixed facillity, determine which arc has the largest marginal contribution to $Z_M$. (This is an extension of Section 3.1.)*

To do this, create the shortest path tree (SPT) rooted at X. Arcs that are not in the shortest path tree are irrelevant to $Z_M$ so their marginal contributions are zero. The search is confined to arcs on the SPT and this reduces our problem to the case of having a spanning tree as described in Section 3.1.

### 3.5. Link Reduction of Y Units in a Network

*Given a network with a fixed facility, and a capacity to reduce Y arc-units, we want to determine the links to reduce and by how much to obtain maximum improvement on $Z_M$. (This is an extension of Section 3.2.)*

#### 3.5.1 Extending the Spanning Tree Solution

We start with the SPT rooted at X. From Section 3.4, we know that this contains all the arcs that have marginal contributions to $Z_M$. Also, the arcs adjacent to both X and a c-node gives a larger improvement to $Z_M$ when reduced compared with other arcs of the spanning tree. But we simply cannot follow the technique in Section 3.2 (which briefly is reducing arcs in the SPT starting with the one with the heaviest $H_j$, then the next heaviest, and so on by as much as allowed by Y and L*) because the situation differs in one major aspect - the shortest path tree which is the current solution may change as we reduce the length of some arcs.

Illustration.

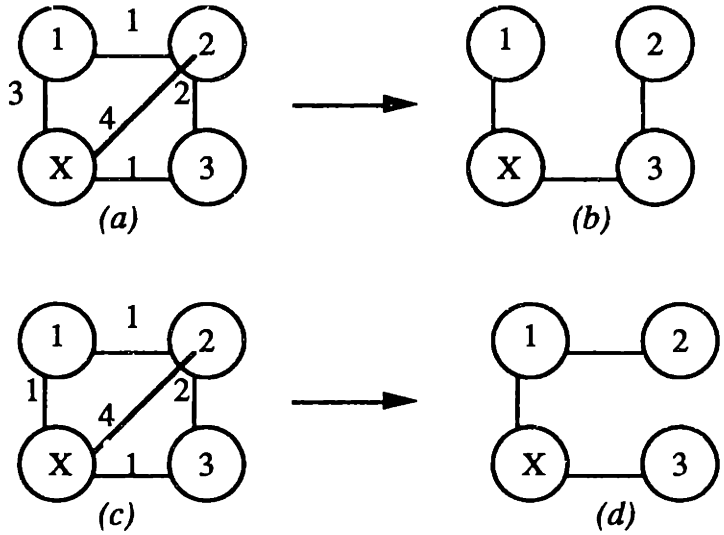



*Figure 3.2*

Given the network in figure 3.2(a), the current SPT rooted at X is shown in figure 3.2(b). If $a_{X1}$ is reduced by 2 units then the new network is shown in figure 3.2(c) and the new SPT rooted at X is in figure 3.2(d). Arc $a_{12}$ which was previously not in the SPT becomes part of it, and $a_{32}$ is not a part of the SPT after the reduction. Therefore with each reduction we may alter $H_c$, the weight of the c-nodes in the process because $H_c$ depends on the underlying SPT. In the example, in the first SPT, $H_1 = h_1$ and $H_3 = h_3 + h_2$ and in the second SPT $H_1 = h_1 + h_2$ and $H_3 = h_3$.

Consider the following:

| | |
|---|---|
| nodes | $c_1, c_2, c_3$ |
| arcs | $a_{Xc_1}, a_{Xc_2}, a_{Xc_3}$ |
| weights | $H_{c_1} > H_{c_2} > H_{c_3}$ |

according to our spanning tree T. We are going to make reductions in increments of one to facilitate our algorithm. In this case, we are requiring that the reductions be integers. We could impose however that reductions be made in any smaller increments, for example halves or tenths of a unit, until we sufficiently meet the

standards of detail set by the decision maker. If we reduce $a_{Xc_1}$ by one, we improve $Z_M$. In the process, we may or may not alter T. If we do not, we can go on and reduce $a_{Xc_1}$ again by one unit (because it still has the largest weight). But if we happened to change T, we have shortened the paths along the subtree rooted at $c_1$, which might have made it more attractive for nodes rooted at $c_2$ or $c_3$, to take paths through $c_1$ leading to X. In other words, nodes originally weighed with $H_{c_2}$ or $H_{c_3}$ have been absorbed by $H_{c_1}$. So the new shortest path tree T' will yield new weights $H'_{c_1} \neq H_{c_1}$, $H'_{c_2} \neq H_{c_2}$, but the "heaviest" remains the same because movement is from $H_{c_2}$ or $H_{c_3}$ to $H_{c_1}$. We are sure that $H'_{c_1} > H'_{c_2}$ and $H'_{c_1} > H'_{c_3}$, so we can reduce $a_{Xc_1}$ by one more unit. In general, we can reduce $a_{Xc_1}$ by:

$$Y_c = \begin{cases} \lfloor l(X,c_1) - L^* \rfloor \text{ units} & \text{if } Y > \lfloor l(X,c_1) - L^* \rfloor \\ Y \text{ units} & \text{if } Y \leq \lfloor l(X,c_1) - L^* \rfloor \end{cases}$$

If $Y \leq \lfloor l(X,c_1) - L^* \rfloor$, we are done. Otherwise, we are left with $H'_{c_2}$ and $H'_{c_3}$ the larger of which we do not know. Also, we have to consider the $c_{1\rightarrow 2\sigma}$-nodes (described in Section 3.1) in the new T' in the evaluation of the new $H'_c$ weights.

A possible way of finding the new SPT is to perform the "modified" shortest path tree algorithm described in Appendix C4. In this new spanning tree we have c-nodes. But we have reduced $a_{Xc_1}$ to the limit. There are three cases that occur when an arc has been reduced to the limit as discussed in Section 3.1. Whatever the case, we can determine the arcs $a_{ij}$ (i may be X) closest to X that have not yet been reduced to the limit, and the corresponding $H_j$'s. Adjust Y to reflect the remaining amount of the budget.

Example.

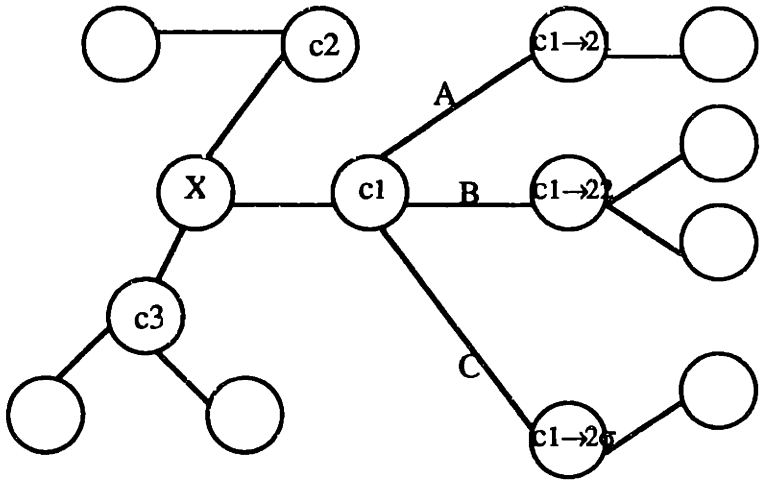



*Figure 3.3*

When $a_{Xc_1}$ has been reduced to the limit L*, the relevant arcs become $a_{Xc_2}$, $a_{Xc_3}$, A, B, and C and the corresponding weights are $H_{c_2}$, $H_{c_3}$, $H_{c_{1\to 21}}$, $H_{c_{1\to 22}}$, and $H_{c_{1\to 2\sigma}}$.

We order the $H_c$'s and pick the largest and reduce its corresponding arc as much as possible. At this point we have completed an iteration. We begin the next iteration by determining the new SPT. Repeat this procedure until Y is exhausted or until all arcs in the current SPT have been reduced to the limit L*.

This procedure does not guarantee the optimal solution, however. Consider the following network:

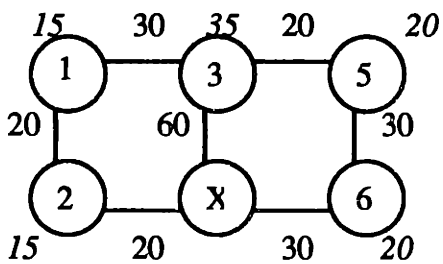



*Figure 3.4 (a)*

The demand on the nodes are italicized (these are the $h_j$'s).Suppose we have the capacity to reduce 15 units.

The SPT to the network in figure 3.4 (a) is

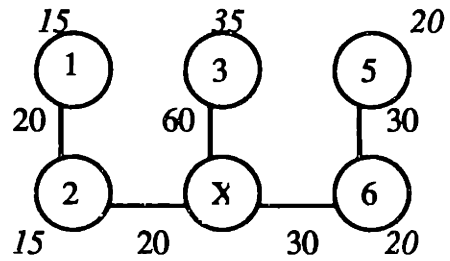



*Figure 3.4 (b)*

According to the solution suggested above, cut on the arc adjacent to the "heaviest" subtree. This is $a_{X6}$. The new network is

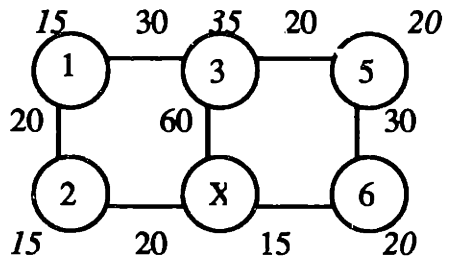



*Figure 3.4 (c)*

and the new corresponding SPT is

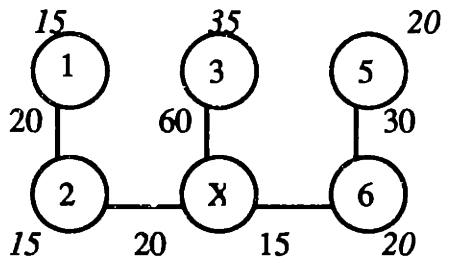



*Figure 3.4(d)*

Improvement = $15(h_5 + h_6) = 15(20 + 20) = 600$ and $Z_{M(New)} = Z_M - 600$. Suppose we reduced $a_{X2}$ instead, by 15 units. The new network is

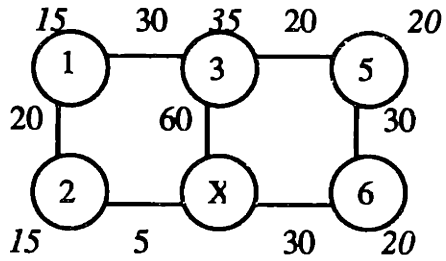



*Figure 3.4(e)*

and the corresponding SPT is

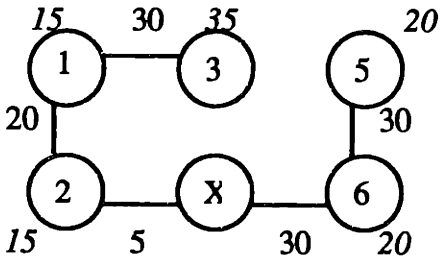



*Figure 3.4(f)*

Improvement = $15(h_2 + h_1) + (60 - 55)h_3 = 15(15 + 15) + 5(35) = 625$. $Z_{M(New)} = Z_M - 625$. The second strategy is better than the first strategy.

### 3.5.2.The Nonlinear Programming Approach

Reexamining the problem, it appears that the approach should not be iterative because succeeding decisions are largely influenced by the entire history of reductions. In the example, if we reduce $a_{X2}$ by larger than 10, it becomes more "profitable" to reduce $a_{X2}$ thereafter than $a_{X6}$ , meaning for every one unit beyond 10 that $a_{X2}$ is reduced we get a larger improvement than from a one unit reduction on $a_{X6}$. But before 10 units, it is more "profitable" to reduce $a_{X6}$. With a reduction capacity of 15 units, the initial advantage of $a_{X6}$ over $a_{X2}$ has been overcome by the fact that the 11th, 12th,..., 15th unit of reduction on $a_{X2}$ improves $Z_M$ more than the 11th, 12th,..., 15th unit of reduction on $a_{X6}$. Because of this dependent nature, which was not considered in the previous approach, a second approach is herein suggested. Whereas the former is a mere heuristic, the following method is more precise, and in special situations yield the optimal solution.

Let us try the following approach: for each node n, (other than X), compute

l(n) = largest number of units the path X $\rightarrow$ n in the SPT is reduced that does not cause a change (i. e. there is an indifference of paths) in the shortest path tree at node n; anything larger than l(n) will cause a change of the SPT at node n

= min $[v_{nj} = d(X,n) + l(n,j) - d(X,j)]$ taken over all j such that $a_{nj}$ exists and $a_{nj}$ is not part of the shortest path tree.

and label node n with l(n) if l(n) is less than the possible number of units of reduction in the path from X$\rightarrow$n. (The possible number of units of reduction in the path from X$\rightarrow$n is determined by adding $l(i,j)$ - L* over all arcs in the said path.) Otherwise, leave the node unlabelled. Order the labels l(n) from smallest to largest and name them $l_1(n_1)$, $l_2(n_2)$, and so on.

For the network in figure 3.4(a),

at n = 1: $l(1) = \min [v_{13} = d(X,1) + l(1,3) - d(X,3)]$

$= \min [v_{13} = 40 + 30 - 60]$

$= 10$

at n = 3 $l(3) = \min [v_{31} = d(X,3) + l(3,1) - d(X,1), v_{35} = d(X,3) + l(3,5) - d(X,5)]$

$= \min [v_{31} = 60+30 - 40, v_{35} = 60 + 20 -60]$

$= 20$

at n =5 $l(5) = 20$

and l(2) and l(6) do not exist.

For $Y \in [0, l_1(n_1)]$

We can reduce anywhere and not disturb any of the shortest paths. The SPT remains unchanged. So it is optimal to cut Y units where we have the heaviest subtree of nodes. Our problem reduces to the case where we are given a spanning tree, and we follow the procedure described in Section 3.2. In the above example, if the budget was

8 instead of 15, we would treat it as a spanning tree case and the optimal solution is to reduce $a_{X6}$ by 8 units.

For $Y \in [l_1(n_1), l_2(n_2)]$

Case 1. $n_1$ is in the current heaviest subtree, and the first arc going out of X in this subtree has length greater than Y + L*. Performing the reduction on the heaviest subtree and even changing the SPT can only increase our benefits , so we should reduce the arc closest to X in the heaviest subtree. In figure 3.4(a), if node 1 had a weight of 30 instead of 15, and Y = 15 units, it is optimal to reduce $a_{X2}$ by 15.

Case 2. $n_1$ is not in the current heaviest subtree, and both the arcs going out of X to the heaviest subtree and $n_1$'s subtree have length greater than Y + L*. If we cut Y units from the heaviest subtree, we have an improvement of Y times the weight of the heaviest subtree. If we cut Y units from $n_1$'s subtree, we have an improvement of Y times the weight of $n_1$'s subtree+ $\sum_{p} [Y - v_{n_1 p}]H_p$ where p is a node that will be pulled in by $n_1$ when $Y > l_1(n_1)$. Pick the strategy which yields the larger improvement. The discussion about figures 3.4(a) through 3.4(f) is precisely this case. If we cut 15 units from the heaviest subtree, we have an improvement of 600 units. If we cut 15 units from $n_1$'s (or node 2's) subtree, we have an improvement of 625 units. The second strategy is better.

Case 3. If Y + L* is greater than the length of the initial arc going out of X in the heaviest subtree or Y + L* is greater than the initial arc going out of X in the subtree to which $n_1$ belongs to, the strategy may not be determined iteratively. Note that changes in the SPT will occur if a reduction of more than $l_1(n_1)$ occurs on the shortest path from $X \rightarrow n_1$. Denote the arcs that comprise this path $a_{ij}$* and the reductions on each arc as $x_{ij}$*. Note that if $\sum x_{ij}^* = l_1(n_1)$, no nodes from other subtrees are pulled into node $n_1$ and hence $n_1$'s subtree. Rather distances become equal via the existing shortest path tree and via node $n_1$. Actually changes in the

shortest path tree may also occur if a reduction of $\sum x_{ij}^*$ is less than $l_1(n_1)$ as long as an arc out of $n_1$ which is not in the current SPT is reduced appropriately. However, we know that it is profitable to reduce wherever it is closest to X and we also know that the number of allowable units of reduction on the path from $X \rightarrow n_1$ is greater than $l_1(n_1)$, so we cannot reduce on an arc going out of $n_1$ unless the shortest path from $X \rightarrow n_1$ has been fully reduced, and these are the $\sum x_{ij}^*$. In figure 3.4(a) the $a_{ij}^*$ arcs are $a_{X2}$ and $a_{21}$. The SPT will change if $x_{X2} + x_{21} > 10 = l(1)$. However, even if $x_{X2} + x_{21} \leq 10$ as is the case when $x_{X2} = x_{21} = 3$, the SPT will change if $a_{13}$ is reduced appropriately, say $x_{13} = 5$. (Observe that $x_{X2} + x_{21} + x_{13} = 11$ and the path from X to node 3 becomes shorter via nodes 1 and 2 than straight to X.) However, it is more profitable to reduce nodes closer to X , so we would not reduce $a_{21}$ without first reducing $a_{X2}$ to the limit nor reduce $a_{13}$ without first reducing $a_{21}$ to the limit.

The following figure is a portion of a larger network.

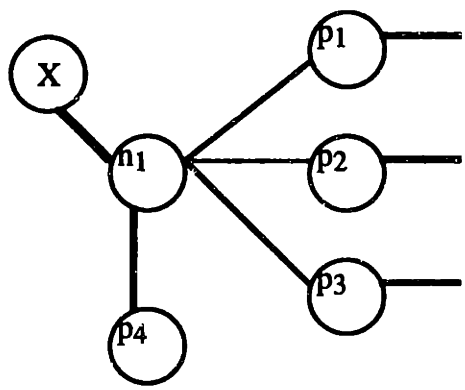



*Figure 3.5*

$p_1$, $p_2$, and $p_3$ are somehow connected to X via other nodes currently ignored. The bold lines are arcs of the SPT and the narrow lines are other arcs in the network. The nodes $p_1$, $p_2$, and $p_3$ are candidates to be pulled in. Call the arcs that join them to $n_1$ $a_{n_1p_k}$ and the amount to cut from them $x_{n_1p_k}$. $p_4$ is already attached to $n_1$ in the SPT. $v_{n_1p_1}$, $v_{n_1p_2}$, and $v_{n_1p_3}$ are the lengths needed to reduce the path $X \rightarrow n_1 \rightarrow p_k$ for $p_1$, $p_2$, and $p_3$ to have two indifferent paths to X. Any reduction larger than $v_{n_1p_1}$ will

pull $p_k$ towards $n_1$ in the new SPT. Sometimes, however, there are not enough allowable units for reduction in the path $X \rightarrow n_1 \rightarrow p_k$ to reach lengths of $v_{n_1p_1}$, $v_{n_1p_2}$, or $v_{n_1p_3}$ because each arc of the network should be L* units long.

If $\sum x_{ij}^* + x_{n_1p_k} \leq v_{n_1p_k}$, then $p_k$ has not been pulled in by $n_1$.

If $\sum x_{ij}^* + x_{n_1p_k} > v_{n_1p_k}$, then $p_k$ has been pulled in by $n_1$

Define $W_k = 0$ if $p_k$ has been pulled in

1 if $p_k$ has not been pulled in

So, $\sum x_{ij}^* + x_{n_1p_k} \leq v_{n_1p_k} \Rightarrow W_k = 1$ and $\sum x_{ij}^* + x_{n_1p_k} > v_{n_1p_k} \Rightarrow W_k = 0$. This can be translated to

$$\sum x_{ij}^* + x_{n_1p_k} - Bu_{k_1} > v_{n_1p_k} \qquad (3.5.1)$$

$$W_k - Bu_{k_2} \leq 1 \qquad (3.5.2)$$

$$W_k + Bu_{k_2} \geq 1 \qquad (3.5.3)$$

$$u_{k_1} + u_{k_2} = 1 \qquad (3.5.4)$$

$$\sum x_{ij}^* + x_{n_1p_k} - Bt_{k_1} \leq v_{n_1p_k} \qquad (3.5.5)$$

$$W_k - Bt_{k_2} \leq 0 \qquad (3.5.6)$$

$$W_k + Bt_{k_2} \geq 0 \qquad (3.5.7)$$

$$t_{k_1} + t_{k_2} = 1 \qquad (3.5.8)$$

where B is a large positive number and u and t are binary variables.

Recall $H_j$. It is conditional on the SPT, hence, on what nodes have been pulled already. So, the actual $H_j$ after considering the pulling is called $\overline{H_j}$ where

$$\overline{H_j} = H_j - \sum_a (1 - W_a)H_{p_a}$$

where the nodes $p_a$ were originally counted in $H_j$, implying that the a-nodes differ across j-nodes.

Example. Let us redefine the network in figure 3.4(a).

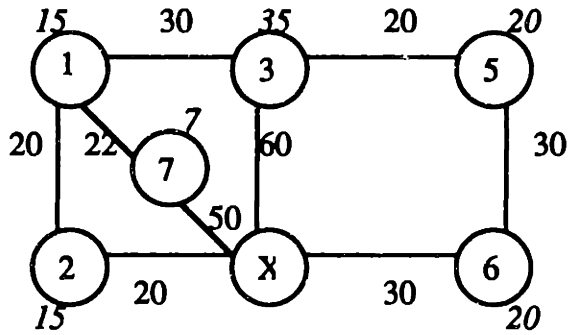



*Figure 3.6(a)*

The corresponding SPT is

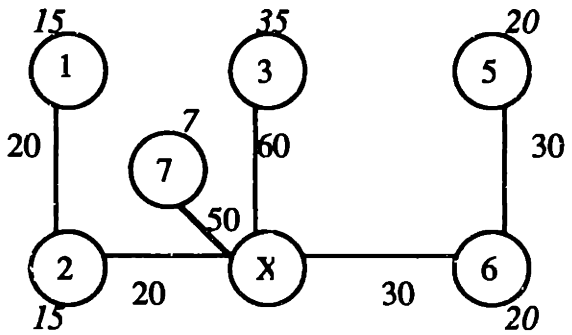



*Figure 3.6(b)*

Let Y = 15 and L* = 4.

l(1) = min[40 + 30 - 60, 40 +22 - 50] = 10

l(7) = 50 + 22 - 40 = 32

l(3) = 20

l(5) = 20

l(2) and l(6) do not exist.

$H_1 = 15$, $H_2 = 15+15 = 30$, $H_3 = 35$, $H_5 = 20$, $H_6 = 20 +20 = 40$, $H_7 = 7$.

Y is between 10 and 20, and pulling may only occur at node 1. The p-nodes are nodes 3 and 7. For each H, the corresponding $p_a$- nodes and $\overline{H}$ are listed as follows:

| | | |
|---|---|---|
| $H_1 = 15$ | none | $\overline{H_1} = 15$ |
| $H_2 = 30$ | none | $\overline{H_2} = 30$ |
| $H_3 = 35$ | 3 | $\overline{H_3} = 35 - (1 - W_3)35$ |
| $H_5 = 20$ | none | $\overline{H_5} = 20$ |
| $H_6 = 40$ | none | $\overline{H_6} = 40$ |
| $H_7 = 7$ | 7 | $\overline{H_7} = 7 - (1 - W_7)7$ |

$a_{ij}^*$ are $a_{X2}$ and $a_{21}$.

We can now define a nonlinear program that could give us the optimal strategy.

$$\text{(P3.5)} \quad \text{maximize} \quad \sum_{\substack{k \\ \text{(counts the nodes that could be pulled by } n_1)}} \left\{ \left[ \left( \sum x_{ij}^* + x_{n_1 p_k} \right) - v_{n_1 p_k} \right] H_{p_k} [1 - W_{p_k}] \right\}$$

$$+ \sum_{\substack{\text{arcs in SPT but not in the path from X to } n_1}} \left\{ x_{ij} \left[ H_j - \sum_{\substack{a \\ \text{(a-nodes are special p-nodes that originally were counted with } H_j)}} (1 - W_a) H_a \right] \right\}$$

$$+ \sum_{\substack{\text{arcs in SPT on the path from X to } n_1}} \left\{ x_{ij}^* H_j \right\}$$

subject to

$$x_{ij} \le l(i, j) - L^* \qquad (3.5.9)$$

$$x_{ij}^* \le l^*(i, j) - L^* \qquad (3.5.10)$$

$$x_{n_1 p_k} \le l(n_1, p_k) - L^* \qquad (3.5.11)$$

$$\sum x_{ij}^* + \sum x_{ij} + \sum x_{n_1 p_k} = Y \qquad (3.5.12)$$

$$\sum x_{ij}^* + x_{n_1 p_k} - B u_{k_1} > v_{n_1 p_k} \qquad \text{(3.5.1 - 3.5.8 for each}$$

$W_k - Bu_{k_2} \leq 1$ of the k nodes which

$W_k + Bu_{k_2} \geq 1$ may be pulled.)

$u_{k_1} + u_{k_2} = 1$

$\sum x_{ij}^* + x_{n_1 p_k} - Bt_{k_1} \leq v_{n_1 p_k}$

$W_k - Bt_{k_2} \leq 0$

$W_k + Bt_{k_2} \geq 0$

$t_{k_1} + t_{k_2} = 1$

$W_k = 0,1$

$x_{ij}^*, x_{ij}, x_{n_1 p_k} \geq 0$

$u_{k_1}, u_{k_2}, t_{k_1}, t_{k_2}$ are binary variables

B is a large positive number

Equations (3.5.9), (3.5.10), and (3.5.11) are feasibility constraints, (3.5.12) is the budget constraint, and (3.5.1)-(3.5.8) for each candidate for pulling records the SPT changes that occur.

In our example, the program may be formulated as

$$\text{maximize} \quad \sum_{k=3,7} \left\{ [(x_{X2} + x_{21} + x_{1k}) - v_{1k}] H_k [1 - W_k] \right\}$$

$$+ x_{X2}(30) + x_{21}(15) + x_{X7}[7 - (1 - W_7)7]$$

$$+ x_{X3}[35 - (1 - W_3)35] + x_{X6}(40) + x_{65}(20)$$

or maximize $(x_{X2} + x_{21} + x_{13} - 10)(35)(1 - W_3)$

$$+ (x_{X2} + x_{21} + x_{17} - 12)(7)(1 - W_7)$$

$$+ x_{X2}(30) + x_{21}(15) + x_{X7}[7 - (1 - W_7)7]$$

$$+ x_{X3}[35 - (1 - W_3)35] + x_{X6}(40) + x_{65}(20)$$

subject to (feasibility constraints:)

$x_{X2} \leq 20 - 4 = 16$ $x_{X3} \leq 60 - 4 = 56$

$x_{21} \leq 20 - 4 = 16$ $x_{X6} \leq 30 - 4 = 26$

$x_{X7} \leq 50 - 4 = 46$ $x_{65} \leq 30 - 4 = 26$

$x_{17} \leq 22 - 4 = 18$ $x_{13} \leq 30 - 4 = 26$

(budget constraint:)

$x_{X2} + x_{21} \div x_{X7} + x_{X3} + x_{X6} + x_{65} + x_{17} + x_{13} = 15$

(constraints for node 3)

$x_{X2} + x_{21} - Bu_{31} > 10$

$W_3 - Bu_{32} \leq 1$

$W_3 + Bu_{32} \geq 1$

$u_{31} + u_{32} = 1$

$x_{X2} + x_{21} + Bt_{31} \leq 10$

$W_3 - Bt_{32} \leq 0$

$W_3 + Bt_{32} \geq 0$

$t_{31} + t_{32} = 1$

(constraints for node 7:)

$x_{X2} + x_{21} - Bu_{71} > 12$

$W_7 - Bu_{72} \leq 1$

$W_7 + Bu_{72} \geq 1$

$u_{71} + u_{72} = 1$

$x_{X2} + x_{21} + Bt_{71} \leq 12$

$W_7 - Bt_{72} \leq 0$

$W_7 + Bt_{72} \geq 0$

$t_{71} + t_{72} = 1$

$W_3$, $W_7$, u and t are binary variables

$x_{X2}, x_{21}, x_{X7}, x_{X3}, x_{X6}, x_{65}, x_{17}, x_{13} \geq 0$

B is a large positive number.

Suppose L* = 18 (instead of 4), it becomes impossible to pull node 7 because the maximum amount of reduction from X → 2 → 1 → 7 is (20 -18) + (20 - 18 ) + (22 - 18) = 8 which is less than $v_{17}$ = 12. So node 7 is not a $p_k$-node in this case, and the corresponding program is smaller.

Another way of making the program smaller is by including only the arcs within the radius of Y possible reductions centered at X. This means that starting from X in the SPT, we will measure in every possible outward direction Y reducible units. Arcs beyond this radius will never be reduced because we have a budget of Y units only, so these arcs will not be included in the formulation. In the preceding example, Y = 15. Since $a_{X2}$ yields 20 - 4 = 16 reducible units, we have no need need to examine $a_{21}$ even though it is in the path from X to node 1. Similarly, $a_{13}$, $a_{17}$, and $a_{65}$ will not be reduced. We can delete these arcs from the preceding formulation and the smaller mixed integer program that results is described in Appendix C5.

If two nodes, $n_{1A}$ and $n_{1B}$, had the same label $l_1(n_1)$, we can formulate a similar program as long as $n_{1A}$ and $n_{1B}$ do not share a common candidate for pulling (i.e. $p_k$-node). With double-pulling, computations become harder, because we have to note which node, $n_{1A}$ or $n_{1B}$, actually pulls the common $p_k$-node.

### 3.5.3. Double-Pulling

Consider the following network:

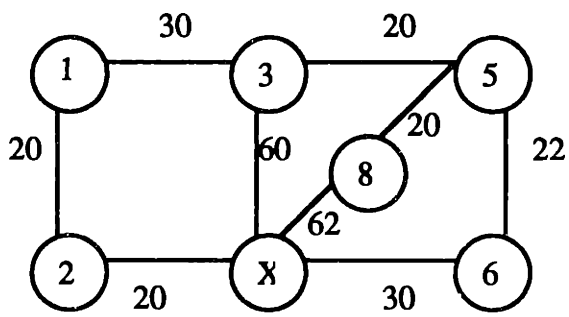



*Figure 3.7(a)*

and its corresponding SPT:

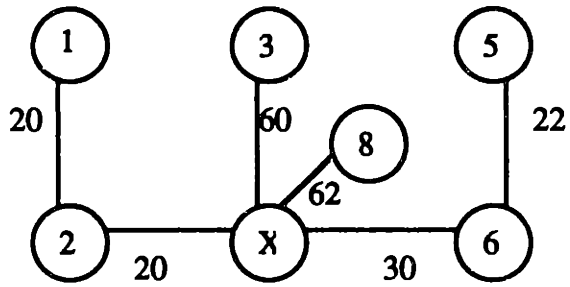



*Figure 3.7(b)*

l(1) = 10

l(3) = min [60 + 30 - 40, 60 + 20 - 52] = 18

l(5) = min [52 + 20 - 60, 52 + 20 - 62] = 10

l(8) = 62 + 20 - 52 = 30

l(2) and l(6) do not exist

l(1) = l(5). Node 3 may be pulled by node 1 and node 5.

After the reductions, if the length of the path X → 6 → 5 → 3 is denoted by P(5) and the length of the path X → 2 → 1 → 3 is denoted by P(1), then node 3 is pulled by the larger of P(5) - 12 and P(1) -10. Only when we know where node 3 was pulled, if it was pulled at all, could we solve for the $\overline{H_j}$'s. If we know that min [P(5) - 12, P(1) - 10] > 0, node 3 was pulled. It is not sufficient to have a variable $W_3$; we have to redefine it as $W_{m3}$ where m is the node that pulled it. In our example, the new binary variables are $W_{13}$ and $W_{53}$ defined as :

if P(5) - 12 > P(1) -10, then $W_{53} = 0$ and $W_{13} = 1$

if P(5) - 12 < P(1) - 10 then $W_{53} = 1$ and $W_{13} = 0$

otherwise, $W_{53} + W_{13} = 1$

In general, (P3.5) may be modified by changing the objective function to

$$\text{maximize} \sum_{\substack{m \\ \text{(nodes with tied } l(n))}} \sum_{\substack{k \\ \text{(counts the nodes that could be pulled by m)}}} \left\{\left[\left(\sum x_{ij}^* + x_{mp_k}\right) - v_{mp_k}\right] H_{p_k}[1 - W_{mp_k}]\right\}$$

$$+ \sum_{\substack{\text{arcs in SPT but not in the path from X to any m}}} \left\{ x_{ij}[H_j - \sum_{\substack{a \\ \text{(a-nodes are special p-nodes that originally were counted with } H_j\text{)}}} (1 - \sum_{m} W_{ma})H_a] \right\}$$

$$+ \sum_{\substack{\text{arcs in SPT on the path from X to any m}}} \{x_{ij}^* H_j\}$$

The constraints have to incorporate the condition that for every path that has a chance of pulling node p: $P(m_1)$, $P(m_2)$, $P(m_3)$, and so on then

if $P(m_1) - v_{m_1p} = \max_m [P(m) - v_{mp}]$, then $W_{m_1p} = 0$ and all other $W_{mp} = 1$

if $P(m_2) - v_{m_2p} = \max_m [P(m) - v_{mp}]$, then $W_{m_2p} = 0$ and all other $W_{mp} = 1$

and so on.

This would require an additional number of constraints and variables. Besides, the formulation becomes more complicated when we have more common $p_k$-nodes, and more nodes having the same value of l(n).

### 3.5.4. Second-Order Pulling

Go back to figure 3.4(a) and the accompanying problem:

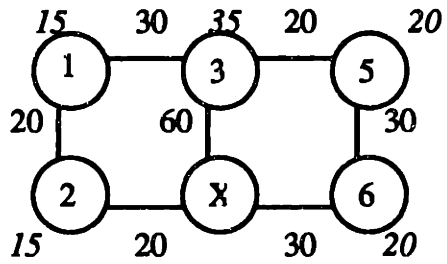



*Figure 3.4 (a)*

The demand on the nodes are italicized (these are the $h_j$'s). Suppose we have the capacity to reduce 15 units.

The SPT to the network in figure 3.4 (a) is

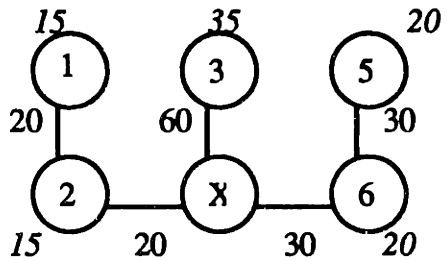



*Figure 3.4 (b)*

Make the slight change that Y = 35 instead of Y = 15 and L = 1. Second-order pulling happens when we reduce $a_{X2}$ by 18 and $a_{21}$ by 17. The resulting network is

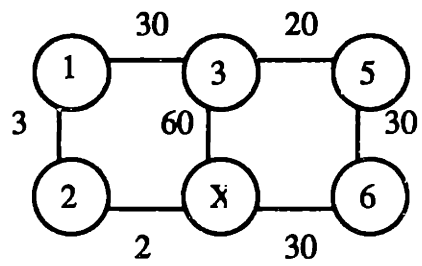



*Figure 3.8(a)*

and the corresponding SPT is

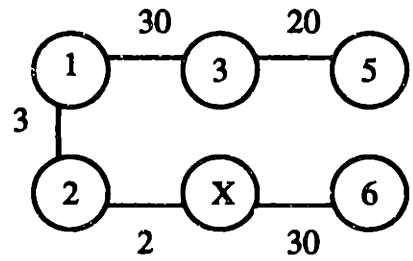



*Figure 3.8(b)*

Node 3 has been pulled by node 1 by a sufficient amount for node 3 to still pull node 5.

Call candidates for second order pulling f-nodes., and the reduction on the arc from $p_k$ to $f_r$, $x_{p_k f_{p_k r}}$. The figure below shows the relationship between $n_1$, p, and f nodes.

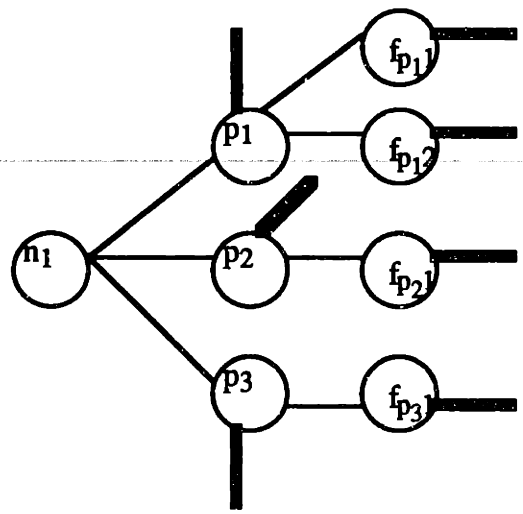



*Figure 3.9*

The bold lines are lines in the SPT and the narrow lines are other arcs in the network. Like figure 3.5, this is just a portion of a network.

Actual second-order pulling will occur if

$$[\sum x_{ij}^* + x_{n_1 p_k} + x_{p_k f_{p_k r}} - v_{n_1 p_k}] > v_{p_k f_{p_k r}}$$

and the improvement this brings in due to $f_{p_k r}$ is

$$[\sum x_{ij}^* + x_{n_1 p_k} + x_{p_k f_{p_k r}} - v_{n_1 p_k} - v_{p_k f_{p_k r}}](H_{f_{p_k r}})$$

We can introduce a dummy variable $g_{f_{p_k r}}$ that indicates if the second order node $f_{p_k r}$ has been pulled:

if $[\sum x_{ij}^* + x_{n_1 p_k} + x_{p_k f_{p_k r}} - v_{n_1 p_k}] > v_{p_k f_{p_k r}}$, then $g_{f_{p_k r}} = 0$

otherwise, $g_{f_{p_k r}} = 1$

Modify integer program (P3.5) to allow for second-order pulling. To the objective function, add another term:

$$\sum_{f_{p_k r}} [\sum_{k} x_{ij}^* + x_{n_1 p_k} + x_{p_k f_{p_k r}} - v_{n_1 p_k} - v_{p_k f_{p_k r}}](H_{f_{p_k r}})(1 - g_{f_{p_k r}})$$

To the constraints add:

(1) a set of constraints similar to (3.5.1) - (3.5.8) that tranforms:

if $[\sum x_{ij}^* + x_{n_1 p_k} + x_{p_k f_{p_k r}} - v_{n_1 p_k}] > v_{p_k f_{p_k r}}$, then $g_{f_{p_k r}} = 0$

otherwise, $g_{f_{p_k r}} = 1$

For each f-node there are 8 constraints.

(2) $\overline{H_j}$ has to be redefined to

$$\overline{H_j} = H_j - \sum_{\substack{a \\ \text{(a-nodes are special p-nodes that originally were counted with } H_j)}} (1 - W_a)H_a - \sum_{\substack{b \\ \text{(b-nodes are special f-nodes that originally were counted with } H_j)}} (1 - W_b)H_b$$

and the second term in the objective function becomes

$$\sum_{\substack{\text{arcs in SPT but not in the path from X to } n_1}} \{ x_{ij}[H_j - \sum_{\substack{a \\ \text{(a-nodes are special p-nodes that originally were counted with } H_j)}} (1 - W_a)H_a - \sum_{\substack{b \\ \text{(b-nodes are special f-nodes that originally were counted with } H_j)}} (1 - W_b)H_b] \}$$

There are new real variables, $x_{p_k f_r}$, and new binary variables, $g_{f_{p_k r}}$, plus the binary variables that will be used in the transformation described in (1).

### 3.5.5. The General Case

For Y exceeding two or more labels, (i.e. in the general case) we have to consider the following:

(1) Double-pulling,

(2) Second order-pulling

(3) Multiple-pulling (when three or more nodes can pull a common node)

(4) Higher-order pulling (when a node can pull a second node which in turn pulls a third node which in turn pulls a fourth node and so on)

These make the formulation of a program more complicated. The heuristic extension of the spanning tree case discussed in Section 3.5.1 may be applied. But as long as there is no multiple pulling or higher-order pulling, the programming approach may be used by applying the modifications discussed in Sections 3.5.3 and 3.5.4.

## 3.6. Link Reduction on Network Arcs with Varying Costs

*Suppose that it costs $c_{ij}$ to reduce one unit of arc $a_{ij}$. Determine the optimal strategy of reductions, within a specified budget C that would best improve the median objective function.*

The heuristic in Section 3.5.1 can easily be extended by considering ratios $r_{ij} = \frac{H_j}{c_{ij}}$. Pick to reduce first the arc with the largest $r_{ij}$. Reduce $a_{ij}$ as much as our budget allows. If C is not big enough to reduce $a_{ij}$ by the smallest increment allowed, pick to reduce the arc with the second largest $r_{ij}$. Go through the ordered list of $r_{ij}$ until an $a_{ij}$ that could be reduced is found. If an arc has been reduced, determine the new SPT, the new ratios, and the remaining budget. Order the new ratios and begin the next iteration by choosing to reduce the arc with the largest $r_{ij}$. Do this until C is exhausted or until all arcs of the SPT has been reduced to the limit L*.

The nonlinear program could be extended by changing the budget constraint $\sum x_{ij}^* + \sum x_{ij} + \sum x_{n_1 p_k} = Y$ to $\sum x_{ij}^* c_{ij}^* + \sum x_{ij} c_{ij} + \sum x_{n_1 p_k} c_{n_1 p_k} = C$. The

analysis on double-pulling and second-order pulling is the same. However this formulation is not guaranteed to provide the optimal reductions because the candidate links for reductions are only the links in the SPT. When all links have equal costs of reduction as is the case in Section 3.5, it is best to reduce in the SPT. But now this may not be the case; although the path not in the SPT is longer, its cheaper cost may actually compensate for its added length. Hence all arcs in the network become candidates for reduction.

To make the programming approach accurate, compute for each node n and each path P from X to n:

l(n,P) = largest number of units the Pth path from X to n is reduced that does not change (i.e. there is an indifference of paths at most) in the SPT at node n; anything larger than l(n,P) will cause a change in the SPT at node n

= min $[v_{njP} = l(P) + l(n,j) - d(X,j)]$ taken over all j such that $a_{nj}$ exists and $a_{nj}$ is not part of the SPT and $l(P)$ is the length of path P and l(n,j) is the length of $a_{nj}$

For figure 3.6(a), there are four paths from X to node 1: [X, 2, 1], [X, 7, 1], [X, 3, 1], [X, 6, 5, 3, 2]. The corresponding l(n,P) values are:

$l(1, [X, 2, 1]) = \min [v_{13[x,2,1]}, v_{17[X,2,1]}] = [40 + 30 - 60, 40 + 22 - 50] = 10$

$l(1, [X, 7, 1]) = \min [v_{13[X,7,1]}] = [72 + 30 - 60] = 42$

$l(1, [X, 3, 1]) = \min [v_{17[X,3,1]}] = [90 + 22 - 50] = 62$

$l(1, [X, 6, 5, 3, 1]) = \min [v_{17[X,6,5,3,1]}] = 110 + 22 - 50 = 82$

Observe that l(1, [X, 2, 1]) where [X, 2, 1] is the path from X to node 1 defined by the SPT, is the smallest of all l(1,P). In general, for a particular n and among all paths from X to n, l(n,P), where P is the path according to the SPT, is the smallest.

If l(n,P) is larger than or equal to the possible number of reductions along path P, the SPT will not change even if the path P is reduced to the limit, hence, ignore this

particular l(n,P). Otherwise, order the remaining l(n,P) from smallest to largest and name them $l_1(n_1P_1)$, $l_2(n_2P_2)$, and so on. (Observe that $P_1$ will always be a path in the SPT.) As in Section 3.5, nodes with labels less than Y have the potential to pull nodes towards themselves and hence change the SPT. The nodes that could be pulled in by node n will still be called p-nodes in the ensuing discussion, but the $v_{njP}$ is now path dependent.

If Y does not exceed the first label, the SPT will not change wherever we perform the reductions. As mentioned before, we may reduce on the SPT or not. If we reduce $x_{ij}$ from an arc in the SPT, there will be an immediate improvement of $x_{ij}$ times $H_j$ to $Z_M$. If we reduce $x_{ij}$ from an arc not in the SPT, it is impossible to have an improvement of $Z_M$ because an improvement will only occur if the current shortest path from X to any node is reduced. But cutting on an arc not on the current SPT cannot reduce the path of X to any node unless the SPT changes. But since Y does not exceed the first label, the SPT cannot change. Hence, only arcs in the SPT can improve $Z_M$. The problem reduces to the spanning tree case whose solution is described in Section 3.3.

If Y exceeds one label, $l_1(n_1P_1)$, let $a_{ij}$* be the arcs that comprise $P_1$. Because $P_1$ is the path of the smallest label, we know that $a_{ij}$* belong to the SPT. It is not profitable to reduce on arcs not in the SPT because changes in the SPT are only possible at $n_1$ through $P_1$. Reductions on arcs not in the SPT cannot change the SPT because Y exceeds only one label. We can identify the p-nodes, the corresponding $v_{njP}$, and apply (P3.5) exactly.

If Y exceeds two or more labels, $l_1(n_1P_1)$ and $l_2(n_2P_2)$, pulling may occur at $n_1$ through $P_1$ and $n_2$ through $P_2$. Although we are definite that $P_1$ is in the SPT, we are not sure if $P_2$ is part of the SPT. If $P_2$ is entirely in the SPT, $n_2$ must be different from $n_1$. In this case, our problem can be solved by an integer program which confines the set of reductions to arcs in the SPT like the formulation in the previous section. This

integer program is precisely defined in Appendix C6. On the other hand, if $P_2$ includes arcs in the SPT as well as arcs not in the SPT, $n_2$ must be the same as $n_1$. Then the arcs that are candidates for reduction are the arcs that comprise $P_1$ and $P_2$ and all the other arcs in the SPT. The p-nodes are obviously common. This problem can be formulated into a nonlinear mixed integer program with double-pulling which was discussed in the previous section.

If Y exceeds more than two labels, the nonlinear mixed integer program may still be used to obtain optimal solutions as long as there is a manageable degree of double-pulling or second-order pulling. In the more general case, the heuristic solutions that were suggested in the earlier part of this section may be used instead.

# Chapter 4

## ARC ADDITIONS IN THE ONE-MEDIAN PROBLEM

### 4.1. Defining the Test Arcs

Throughout this chapter, the problem of improving $Z_M$ through arc additions will be discussed. The set of arcs which may be added to a given spanning tree or network will be initially specified by the decision maker. In the most general case, all arcs may be considered in this initial specification stage. Then the eligibility of these arcs are further evaluated by two tests.

#### 4.1.1. The First Test

For every arc $a_{ij}$ that may be added to the existing graph, identify the two nodes i and j. By the way we defined $a_{ij}$, it follows that $d(X,i) < d(X,j)$. Arc $a_{ij}$ gives an improvement to the median objective function only if $d(X,i) + l(i,j) < d(X,j)$, i.e. if $l(i,j) < d(X,j) - d(X,i)$. An arc satisfying this condition is said to have passed the first test.

#### 4.1.2. The Second Test

There are problems in this chapter that impose contraints on the length or number of arc-units we may add to an existing spanning tree or network. The second test is designed as a test of feasibility. If L arc-units could be added to a spanning tree or network, all arcs with lengths greater than L fail the second test. If it costs $c_{ij}$ to add arc $a_{ij}$, and we have a budget of C, all arcs with corresponding costs greater than C fail the second test.

#### 4.1.3. The Test Arcs

Define a test arc as an arc initially specified by the decision maker which we should actually consider adding to our existing graph because of its compliance with either the first test or the second test. Each problem in this chapter has a different

way of determining the test arcs from the initial specification. If we are concerned with adding only one arc to a spanning tree or network with no budget constraints (as in Sections 4.2 and 4.5), the test arcs are arcs that have passed the first test.

If we could add more than one arc to a spanning tree or network in the case where we have budget constraints (as in Sections 4.3, 4.4, 4.6, and 4.7), the test arcs are arcs that have passed the second test. Whether they satisfy the first test or not is irrelevent. Consider any arc that has passed the second test and failed the first test. Having failed the first test means that the current distances of the nodes will be unaffected by the addition of that arc. It is of no immediate use. However, since we could add more arcs, it is possible that in conjunction with another arc in the network arc $a_{ij}$ becomes useful. To illustrate this consider the following example.

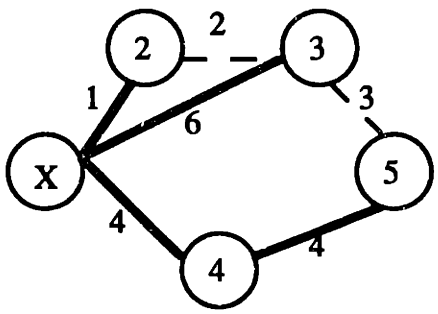



*Figure 4.1*

The solid lines are arcs in the SPT and the dashed lines are arcs specified by the decision maker. Adding $a_{35}$ fails the first test and does not change the SPT; hence, it cannot improve $Z_M$. But adding $a_{35}$ and $a_{23}$ at the same time causes a change in the SPT, and in the process improves $Z_M$. Node 3 is closer to X by 3 units and node 5 is closer to X by 2 units.

## 4.2. Addition of One Arc to a Spanning Tree

*Suppose we want to add a new arc to a spanning tree with a facility in order to best improve $Z_M$.*

### 4.2.1. The Cutset Method

Step 1. Pick a test arc $a_{ij}$.

Step 2. Determine the path from X to j. Denote the nodes along this path $[W_{n+1}, W_n, ..., W_1, W_0]$ where $W_{n+1} = X$ and $W_0 = j$. For the precise algorithm, the reader is referred to Appendix D1.

Step 3. Find $S_{W_k}$ where $S_{W_k}$ is the difference of the distance from $W_k$ to X before and after the addition of link $a_{ij}$. For each node in the path, beginning from $W_n$, $W_{n-1}, ..., W_0$, determine the difference $S_{W_k} = d(X, W_k) - [d(X,i) + l(i,j) + d(j, W_k)]$, where $d(j, W_k)$ is the distance from node j to $W_k$.

Step 4. We would like to partition the spanning tree into two subtrees where one subtree, called $T_1$, contains nodes whose distances to X were altered by the reduction, and another subtree, called $T_2$, contains nodes whose distances to X remain unchanged after the reduction.

Starting from $k = n + 1$ until $k = 0$, check $S_{W_k}$ and in the first instance that $S_{W_k}$ is greater than zero, make a cut on arc $a_{W_k W_{k+1}}$ and arc $a_{ij}$. This partitions the spanning tree into two subtrees. $T_2$ should contain X; $T_1$ is the other subtree. Algorithm D2 (described in Appendix D2) determines the nodes in $T_1$ and the nodes in $T_2$. The improvement of $Z_M$ is entirely due to the nodes in $T_1$.

Step 5. Determine the savings or improvement of $Z_M$ due to the addition of arc $a_{ij}$ using Algorithm D3 (described in Appendix D3).

Step 6. Pick another arc and repeat Steps 2 - 5.

Step 7. After choosing all the test arcs, the arc to add is the one which yields the largest associated improvement of $Z_M$.

### 4.2.2. The Cycle Method

Instead of using cutsets, we may use cycles. For any test arc $a_{ij}$, we can compute for the corresponding improvement it brings to the objective function.

Step 1. Pick a test arc $a_{ij}$.

Step 2. Determine the path from X to i.

Step 3. Determine the path from X to j

Step 4. Find the node common to both paths that is farthest from X.Graphically,

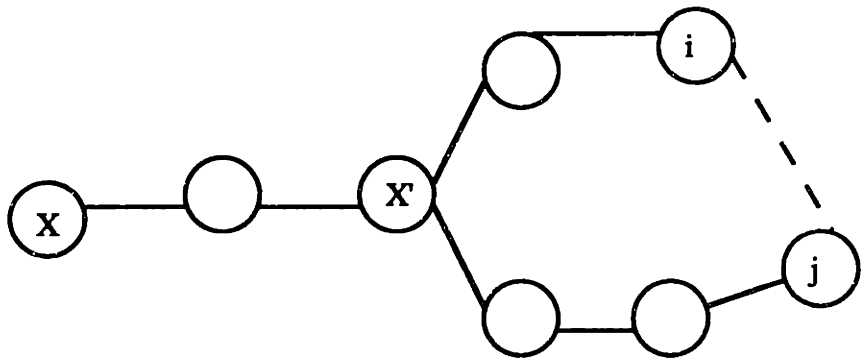



*Figure 4.2*

We want to find X'. The procedure is precisely described by Algorithm D4 in Appendix D4.

Step 5. Referring to figure 4.2, for every node in the path from X to j beginning from node X' until node j (refer to these nodes as $W_{k^*}$; $W_{k^*}$ = X', ..., j), determine the difference,

$$D_{W_{k^*}} = d(X,W_{k^*}) - [d(X,i) + l(i,j) + d(j,W_{k^*})]$$

and define the variable $S_{W_{k^*}} = D_{W_{k^*}}$ if $D_{W_{k^*}} > 0$

$= 0$ otherwise

Step 6. Determine the savings associated with $a_{ij}$. All nodes n in the spanning tree which have $W_{k^*}$ in the path from X to n (provided n is not in the cycle) and have $W_{k^*}$ as the nearest node to it compared to others in {X', ..., j}, will have a savings of $h_n S_{W_{k^*}}$. We like to find how many such nodes exist. By including the weight of $W_{k^*}$ we can determine the weight of the subtree rooted at $W_{k^*}$. Algorithm C2 discusses the computation of $H_{W_{k^*}}$

Step 7. The total improvement of $Z_M$ or the savings associated with test arc $a_{ij}$ is:

$$\sum_{W_{k^*}} (S_{W_{k^*}})(H_{W_{k^*}})$$

Step 8. Pick another test arc and repeat Steps 2 - 7.

Step 9. After choosing all test arcs, add the arc which yields the largest improvement of $Z_M$.

### 4.2.3. An Example of the Cutset Method and the Cycle Method

Consider the following spanning tree.

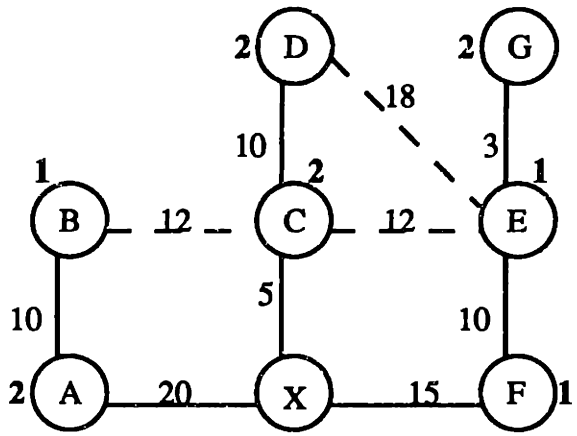



*Figure 4.3*

The nodes and the solid lines define a spanning tree. The weights of the nodes are indicated by the bold numbers. Suppose that the arcs specified by the decision maker for addition are the broken lines in the graph.

$a_{ED}$ fails the first test, hence it is not a test arc.

The Cutset Method

If we add $a_{CE}$, the nodes along the path from X to E are [X,F,E]. $S_X$ =-42, $S_F$ = -12, $S_E$ = 8. Perform the cut on arcs $a_{FE}$ and $a_{CE}$. $T_1$ contains two nodes E and G. The improvement associated with $a_{CE}$ is (8)(1+2) = 24.

If we add $a_{CB}$, the nodes along the path from X to B are [X,A,B]. $S_X$ =-47, $S_A$ = -7, $S_B$ = 13. Perform the cut on arcs $a_{AB}$ and $a_{CB}$. $T_1$ contains one node, B. The improvement associated with $a_{CB}$ is (13)(1) = 13.

Since 24 > 13, add $a_{CE}$.

The Cycle Method

Consider arc $a_{CE}$; the nodes along the path from X to C are [X,C] and those from X to E are [X,F,E]. X' = X. $W_{k*}$ = X,F,E. $S_X$ =0, $S_F$ = 0, $S_E$ = 8. The improvement associated with $a_{CE}$ is 8(1+2) = 24.

Consider arc $a_{CB}$; the nodes along the path from X to B are [X,A,B]. X' = X. $W_{k*}$ = X,A,B. $S_X$ =0, $S_A$ = 0, $S_B$ = 13. The improvement associated with $a_{CB}$ is (13)(1) = 13.

Since 24 > 13, add $a_{CE}$.

**4.3. Addition of L Arc Units to a Spanning Tree**

*Suppose we had the capacity to add L arc-units instead of one arc to a spanning tree. Find the arcs that would best improve $Z_M$.*

For each test arc, we can compute the associated improvement of the objective function by using either the cutset method or the cycle method. The first arc that we add to the spanning tree is the arc with the highest potential improvement. At this point, the associated improvements due to the other test arcs may change. An arc may increase or decrease its potential improvement after the addition of our first link. Because of this, the approach to this problem cannot be iterative. We do not even know for sure if the arc with the largest initial potential improvement is part of the optimal solution.

The following heuristic based on this iterative approach is proposed to solve the problem.

Step 1. Determine each arc's potential savings. For each test arc, compute the associated potential savings using either the cutset method or the cycle method. Order the results from largest to smallest. Let the one with the largest improvement be $a_{ij}$*.

Step 2. Check feasibility. If $l^*(i,j) \leq L$, add arc $a_{ij}$*. Otherwise go down the list of ordered potential savings until the corresponding arc length is less than or equal to

L, or until the end of the list. If the former happens, call the added arc $a_{ij}$* and let L = L - l*(i,j), $Z_M = Z_M$ - savings of $a_{ij}$*. Otherwise, stop. We cannot further improve $Z_M$ because L is insufficient.

Step 3. Determine the new shortest path tree. Then relative to this spanning tree do Steps 1 and 2.

Step 4. Repeat Steps 1 - 3 until L is exhausted.

**Remark**. Suppose instead of having a capacity to add L arc-units, we had the capacity to add L arcs, our analysis of the problem is the same except that the test arcs consist of all arcs specified by the decision maker. We terminate the heuristic procedure when we have gone through Steps 1, 2, and 3 L times. Also we do not need to order the potential savings of each arc; it is sufficient that the largest is known.

### 4.4. Arc Addition on a Spanning Tree with Varying Costs

*Suppose there are costs $c_{ij}$ associated with every arc $a_{ij}$ that we could add, which arcs should be added to best improve $Z_M$ and yet remain within a budget C.*

The property of dependency of an arc's potential savings on the other arcs present in the resulting network as seen in Section 4.1 extends to the case where there are varying costs across arcs. The heuristic presented in the previous section is extended to this case, where the concern is cost rather than length.

Step 1. Instead of computing merely the potential savings of each test arc, we compute for $\frac{\text{savings}}{c_{ij}}$ per arc. (It is important to differentiate between the definition of $c_{ij}$ in link reduction problems and in link addition problems. In link reduction problems, $c_{ij}$ is the cost of reducing one unit of arc $a_{ij}$, while in link addition problems, $c_{ij}$ is the cost of adding the entire arc $a_{ij}$.) Order the ratios from largest to smallest. Let the one with the largest ratio be $a_{ij}$*.

Step 2. Check feasibility. If c*(i,j) ≤ C, add $a_{ij}$* to the graph. Let C = C - c*(i,j), $Z_M = Z_M$ - savings of $a_{ij}$*. Otherwise go down the list of ordered ratios until an arc

$a_{ij}$* with c*(i,j) ≤ C is found. If the list is exhausted, stop. We cannot further improve $Z_M$ because C is insufficient.

Step 3. Determine the new shortest path tree. Then relative to this spanning tree do Steps 1 and 2.

Step 4. Repeat Steps 1 - 3 until C is exhausted.

### 4.5. Addition of One Arc to a Network

*Given a network, find the best arc to add to best improve $Z_M$.*

This problem can be thought of as an extension of the spanning tree case in Section 4.2. Observe, however, that while the techniques for the spanning tree problem (both the cutset method and the cycle method) relied on paths from the facility to each node, this would be difficult to extend in a network because paths are numerous and interconnected. One way of approaching this problem is by adding every test arc $a_{ij}$, finding the new shortest path tree rooted at X (through the method described in Appendix A or the method in Appendix C4) and comparing the new $Z_M$ with the old $Z_M$ to determine the improvement or savings associated with $a_{ij}$. Pick the test arc with the largest associated improvement.

### 4.6. Addition of L Arc Units to a Network

*Given a network and the capacity to add L arc units. Determine the set of arcs to be added to the network to best improve $Z_M$.*

This is an extension of the spanning tree case in Section 4.3. Like the spanning tree problem, it cannot be solved optimally using the iterative approach. However, the heuristic used for the spanning tree case can be extended directly to the case of networks by just finding the SPT of the given network to start the algorithm.

**Remark.** To the variation of the problem wherein we can add L arcs instead of L arc-units, the same comments apply.

## 4.7. Arc Addition on a Network with Varying Costs

*Suppose there are costs associated with every arc that we could add to a network, how should L arc units be allocated to best improve $Z_M$.*

If costs are associated with the arcs we may add to a given network, the heuristic procedure for the spanning tree case discussed in Section 4.4 may be extended directly by just finding the SPT of the given network to start the algorithm.

Example:

Consider the following network:

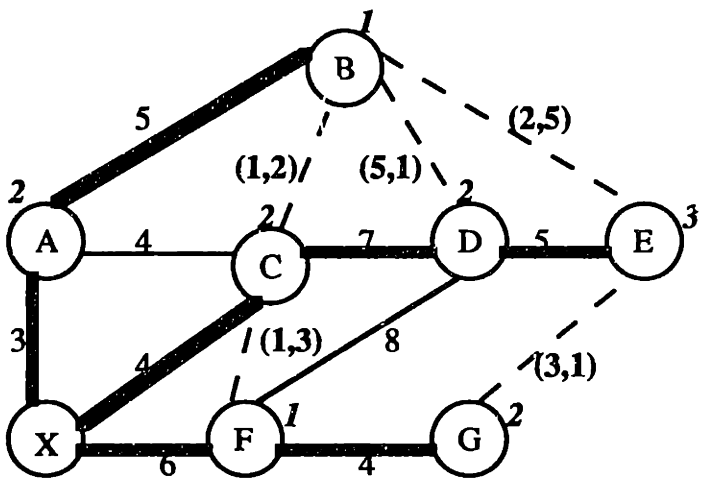



*Figure 4.5*

The bold lines are arcs in the SPT, the other solid lines are the other arcs in the network and the broken lines are the arcs specified by the decision maker with corresponding arc length and unit cost, respectively, in parenthesis. The bold italicized numbers are the demands of the nodes. Suppose we had a budget of C = 4. The current $Z_M$ = 118 = (2)(3)+(1)(8)+(2)(4)+(2)(11)+(3)(16)+(1)(6)+(2)(10). Recall that the savings due to a particular arc is the difference of the objective function values before and after the addition of the arc.

The cost of $a_{BE}$ is greater than C; hence it is not a test arc.

| Test Arcs | New $Z_M$ | Savings | Ratio(Savings/ $c_{ij}$) |
|---|---|---|---|
| $a_{CB}$ | 115 | 3 | 1.5 |
| $a_{BD}$ | 118 | 0 | 0 |
| $a_{CF}$ | 115 | 3 | 1 |
| $a_{GE}$ | 109 | 9 | 9 |

Add arc $a_{GE}$, C = 4 -1 = 3, $Z_M$ = 109. According to the new SPT,

| Test Arcs | New $Z_M$ | Savings | Ratio(Savings/ $c_{ij}$) |
|---|---|---|---|
| $a_{CB}$ | 106 | 3 | 1.5 |
| $a_{BD}$ | 0 | 0 | 0 |
| $a_{CF}$ | 106 | 3 | 1 |

Add arc $a_{CB}$, C = 3 - 2 = 1, $Z_M$ = 106. According to the new SPT,

| Test Arcs | New $Z_M$ | Savings | Ratio(Savings/ $c_{ij}$) |
|---|---|---|---|
| $a_{BD}$ | 104 | 2 | 2 |
| $a_{CF}$ | 103 | 3 | 1 |

Add arc $a_{BD}$, C = 1 - 1 = 0, $Z_M$ = 104. We have exhausted C. From this heuristic the set of arc additions is $\{a_{GE}, a_{CB}, a_{BD}\}$ and the final spanning tree is:

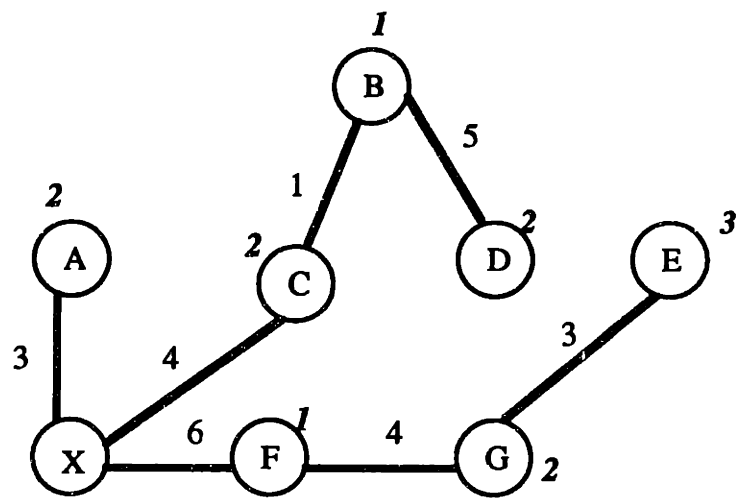



*Figure 4.6*

# Chapter 5

## LINK REDUCTIONS IN THE ONE-CENTER PROBLEM

Until now our criterion has been to minimize average distance from a node to the facility. There are practical situations where it is more appropriate to minimize the maximum weighted distance from any node to the facility. Emergency services fall in this category. In the next two chapters, this criterion, better referred to as the center problem will be applied to the various problems we sought to solve in the preceding two chapters. Notations, concepts, and techniques will be extended whenever possible.

### 5.1. Marginal Contribution of Arcs in a Spanning Tree to the Center Function

*Given a spanning tree with a fixed facility, we want to find the arc with the largest marginal contribution to the center objective function.*

#### 5.1.1. Defining M

Define $u_j = h_j d(X,j)$, for each node j and let j* be the node with the largest $u_j$. If the nodes have equal weights, $u_j$ is just the distance of node j from X, and j* is necessarily an end node of the spanning tree. If different weights are associated with the nodes, j* may be any node in the spanning tree.

We will now define an array M which will be relevant to all the sections of this chapter. M will contain only the end nodes if the spanning tree or network is unweighted, and will contain all nodes if the spanning tree or network is weighted. M orders the nodes in a descending manner according to their $u_j$ values. $m_1$, the first entry of M, has the largest $u_j$, $m_2$, the second entry of M, has the second largest $u_j$, and so on.

Illustration.

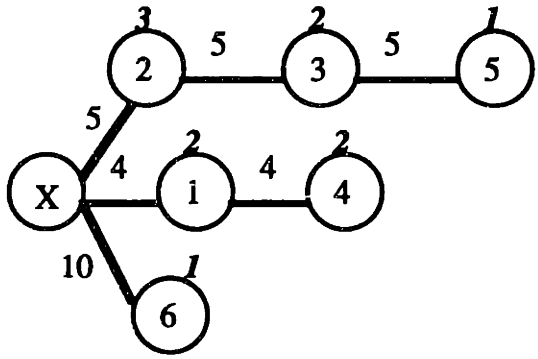



*Figure 5.1*

The bold italicized numbers are the weights of the nodes. The other numbers represent arc lengths.

The node set is {1 2 3 4 5 6}.

M = (5 6 4) if we consider the unweighted spanning tree.

M = (3,4,2,5,6,1) if we consider the weights of the nodes.

### 5.1.2. Paths Relevant to $Z_C$

Looking at M, $m_1$ has the largest $u_j$. If $u_{m_2} < u_{m_1}$, the solution to the problem is trivial. The only path that affects $Z_C$ is the path from X to $m_1$. Any arc on this path when reduced improves $Z_C$ and no arc is better than another. All other arcs when reduced do nothing to $Z_C$.

It could also happen that $u_{m_2} = u_{m_1}$. In general, there may be many such nodes tied for the largest value of $u_j$. If this is the case, it is not sufficient to reduce one arc because $Z_C$ will remain the same. If there are z nodes tied for the largest $u_j$, $Z_C$ is dependent on z paths. Inasmuch as these paths may not be mutually exclusive, at most z-arcs and at least one arc has to be reduced to lower $Z_C$. The marginal contribution of an arc is then conditional on what other arcs are reduced.

## 5.2. Link Reduction of Y Units in a Spanning Tree

*Given a spanning tree and a fixed facility, we want to know which arcs should be reduced and by how much, if we could reduce a total of Y arc-units throughout the spanning tree, to obtain maximum improvement of $Z_C$.*

Step 1. Determine M.

Step 2. Identify the path from X to $m_1$.

Step 3(a) $u_{m_2} < u_{m_1}$

We can reduce the path from X to $m_1$ by y until $h_{m_1}[d(X, m_1) - y] = h_{m_2}d(X, m_2)$ if $y \leq Y$. If $y > Y$, we can reduce all y units from this path. Suppose, that reductions have to be integer. This means that

$$\text{if} \qquad \Omega = \frac{h_{m_1}d(X, m_1) - h_{m_2}d(X, m_2)}{h_{m_1}}$$

$$\text{then} \qquad y = \min \{\lfloor \Omega \rfloor , Y\}.$$

Suppose we name the arcs along the path from X to $m_1$ as $p_1$, $p_2$, and so on. Although it does not matter to $m_1$ which arc on the path we reduce, to affect more nodes, one should reduce the arc closest to the facility first, the second arc closest to the facility second, and so on along the path X to $m_1$ until we consume y or until every arc on the path from X to $m_1$ has been reduced to the limit L*. If the latter happens, stop. $Z_C$ cannot be improved further by link reductions. Otherwise, adjust Y and go to Step 4.

Step 3(b) Nodes are tied for the largest value of $u_j$.

This means that reducing only one path will not improve our center objective function. Suppose there are z nodes tied for the largest value of $u_j$. The configuration of the spanning tree is important to the best strategy of reductions. In particular, how the z nodes are linked to one another helps us minimize the number of arc reductions necessary to lower $Z_C$.

Define the z x N matrix $\pi$ with components

$\pi_{ij} = 1$ if node j is in the path from X to i

0 otherwise

Every row corresponds to a tied node z. Algorithm E1 (described in Appendix E1) constructs this matrix.

Once $\pi$ has been constructed, we will group the tied nodes according to their connections to X. Two nodes that share a common intermediate node from X will require a reduction of one unit before the node of intersection to produce the same effect as a reduction of two units after the node of intersection.

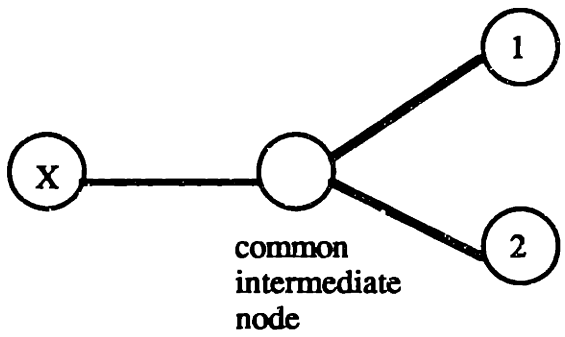



*Figure 5.2*

Knowledge of these groupings of tied nodes will help us in deciding where to reduce. The groupings can be determined as follows:

(i) Define $T_j = \pi_{1j} + \pi_{2j} + ... + \pi_{zj}$ .

(ii) Grouping the tied nodes

Find out if j exists such that $T_j = z$. If such a j exists, this means that all the tied nodes emanate from one node called j. Many such j 's may exist as illustrated in the following figure,

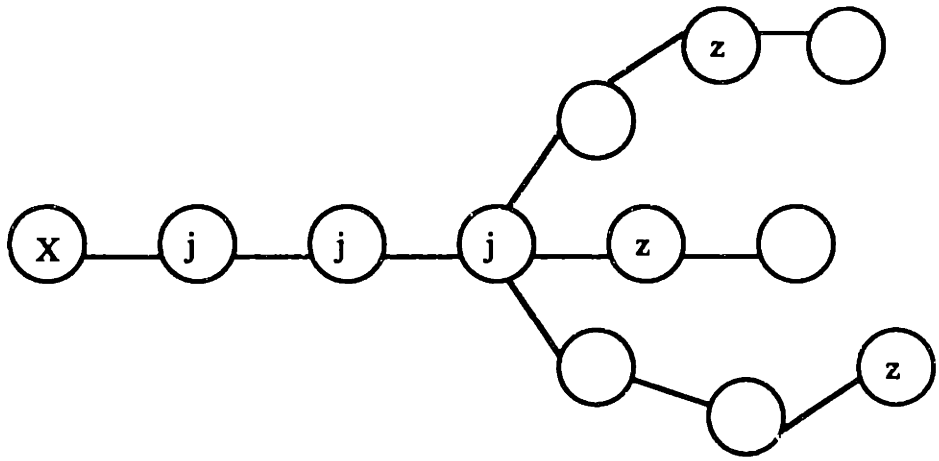



*Figure 5.3*

and it is equally profitable to reduce arcs in the path from X to any j.

Suppose no such j exists. Find out if there is a j such that $T_j = z-1$. Again many such j's may exist. For any such j chosen arbitrarily, determine $\pi_{ij} = 1$ and group those i's together. Let $z_j = \{i | \pi_{ij} = 1\}$. This is one group of tied nodes. Because we have a spanning tree and paths from X to n are unique, there are at most $\lfloor (z - g)/T_j \rfloor$ groups of size $T_j$ that are still unknown to us where g is the sum of the cardinalities of all the groups we have discovered so far. If $\lfloor (z - g)/T_j \rfloor$ is zero, we have discovered all groups of size $T_j$ or larger. Next, find the groups of smaller size, so let $T_j = T_j - 1$. If we have found all the groups of size 2 and larger, all the remaining ungrouped z-nodes are isolated from one another.

Example.

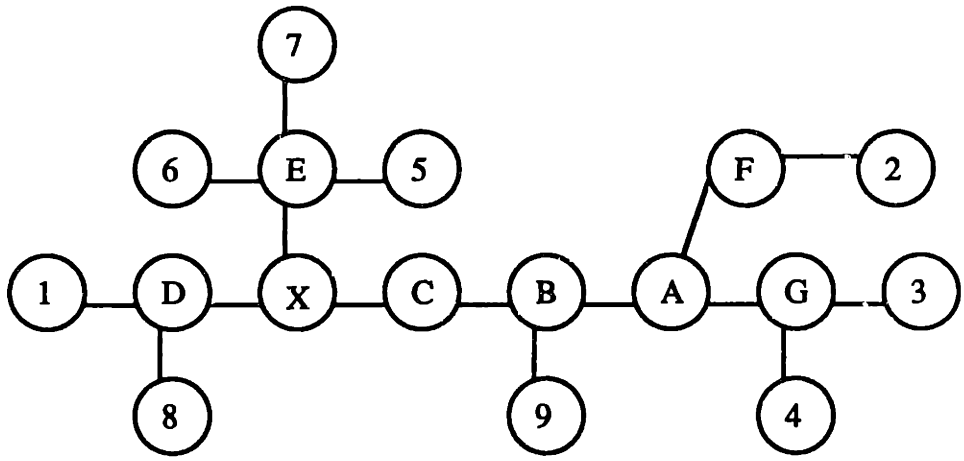



*Figure 5.4*

Let figure 5.4 represent an unweighted spanning tree with 9 ends. Suppose end nodes 1, 2, ...,7 are all tied for the largest distance from X. These are the z-nodes. Nodes 8 and 9 have lesser distances from X. The matrix $\pi$ is as follows:

$$[A\ B\ C\ E\ G]$$

$$\begin{bmatrix}1\\2\\3\\4\\5\\6\\7\end{bmatrix}\begin{bmatrix}0&0&0&0&0\\1&1&1&0&0\\1&1&1&0&1\\1&1&1&0&1\\0&0&0&1&0\\0&0&0&1&0\\0&0&0&1&0\end{bmatrix}$$

From the matrix $\pi$, we can see that A, B, C, and E have 3 z-nodes attached to them. We do not know however if these nodes are unique. Pick one arbitrarily, say B. The corresponding group of end nodes is {2, 3, 4}. There is at most $\lfloor(z - g)/T_j\rfloor = \lfloor(7 - 3)/3\rfloor = 1$ distinct group yet undiscovered. If we choose node C or node A next, we define the same group of end nodes. However, if we pick E, we find a new group of size 3, {5, 6, 7}. Checking the number of ungrouped nodes left, we have 7 - 3 - 3 = 1. This is a singleton, {1}.

(iii) After grouping the z-nodes, we know that for each group there may be more than one node from which the group emanates. It is our objective to find that

which is farthest from X, and denote it as n*. In the example, node A is the farthest among nodes A, B, and C from where nodes 2, 3, and 4 emanate; n* = A. For singletons, n* is the singleton node itself. It is also possible that a z-node is also an n*-node. Suppose there are N* such n*-nodes.

(iv) For unweighted spanning trees if Y is as large or exceeds N*, the strategy of reduction is to allocate the budget Y on the arcs before n* of each group. One unit at a time (or for any other increment used), reduce the path from X to every n* at the closest allowable arc to X. After reducing all N* paths by one unit, adjust the remaining Y and check for four things. First, has a path from X to any z-node been reduced to the limit, meaning have all the arcs on this path been reduced to the length L*? If it has, terminate the reduction process. Second, has the path from X to any n* been reduced to the limit? If it has, regroup the nodes in that group and determine the new n*s. In the above example, if the arc from X to E has been reduced to the limit, 3 new singletons are formed. Third, is the adjusted Y smaller than the new N*? If so, terminate the reduction process. Fourth, knowing that the objective function has been reduced by one unit, is there a new addition to the set of tied nodes? If so, determine its grouping, and the new n* if one is necessary. Having checked these four states, perform a next set of reductions of a unit each on the paths from X to the n*s. (Observe that once we enter a state with tied nodes we can not get out of it because of the construction of the algorithm.)

For weighted spanning trees reducing all paths from X to n* by the smallest increment possible would most likely break the ties, and move us out to the untied node state. Perform the smallest possible reduction on the paths form X to n* starting from that closest to X. Adjust Y after the reductions. Go to Step 4.

Step 4. Updating the Distances and Reordering M

If arcs $a_{ij_1}$, $a_{ij_2}$, and so on were reduced, all nodes which have $j_1, j_2$, etc. in their paths to X will now have shorter distances to X. All other nodes retain their distance from X prior to the reductions in length.

To reorder M, the reader is referred to Appendix E2.

Step 5. Having adjusted Y, updated the distances and reordered M, go back to Step 3. Terminate when we have exhausted Y or when the path from X to one of the nodes having the largest $u_j$ has been reduced to the limit.

### 5.3. Link Reductions on Spanning Tree Arcs with Varying Costs

*Suppose that it costs $c_{ij}$ to reduce $a_{ij}$ by one unit and we have a budget of C units, we want to know how to reduce the arcs of a spanning tree with a fixed facility to best improve $Z_M$.*

With different costs of reduction on different arcs, the choice of where to reduce becomes critical. We know that we have to cut along the path from X to $m_1$ or along each of the paths of the tied nodes, but, whereas before, we reduced the arc closest to X, it may be too expensive to do this now; along the path there may be a cheaper arc to reduce.

For every node j in the network, determine the path from X to j. Let $p_j$ be the path from X to j and $P_j$ be the set of all arcs $a_{ij}$ that comprise $p_j$. If $x_{ij}$ is the amount we will reduce from $a_{ij}$, then the new length of $p_j$ after link reductions will be $\sum_{P_j} l(i,j) - x_{ij}$ and the new $u_j$ is $h_j\left\{\sum_{P_j} l(i,j) - x_{ij}\right\}$. The optimal strategy of link reductions is generated by the following linear program:

$$\text{minimize} \quad v$$

$$\text{subject to} \quad h_j\left\{\sum_{P_j} l(i,j) - x_{ij}\right\} \le v \qquad \text{for all } j \in M$$

$$l(i,j) - x_{ij} \ge L^* \qquad \text{for all arcs}$$

$$\sum_{\text{all arcs}} x_{ij}c_{ij} \leq C$$

$$x_{ij} \geq 0$$

The first set of constraints guarantee that we are minimizing the maximum $u_j$. The relevant nodes are only those in M. The second set of constraints are feasibility constraints. And the last constraint is the budget specification.

Example. To illustrate the preceding discussion, consider the following spanning tree.

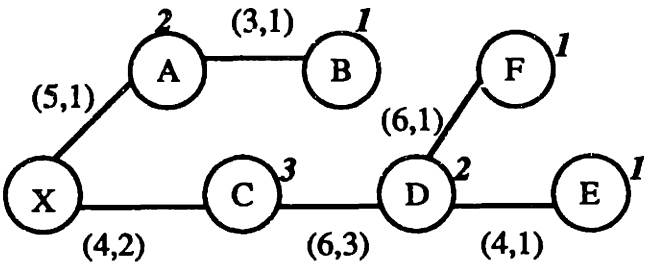



*Figure 5.5*

The numbers in parenthesis represent the arc length l(i,j) and the cost of reduction $c_{ij}$. The bold italicized numbers are the demands associated with the nodes. Suppose L* = 1, C = 6.

M = [D, F, E, C, A, B] with corresponding distances from X of 20, 16, 14, 12, 10, and 8 respectively. The linear program that gives the optimal strategy of link reductions is:

minimize v

subject to $(2)(5 - x_{XA}) \leq v$

$(1)[(5 - x_{XA})+(3-x_{AB})] \leq v$

$(3)(4 - x_{XC}) \leq v$

$(2)[(4 - x_{XC})+(6-x_{CD})] \leq v$

$(1)[(4 - x_{XC})+(6-x_{CD})+(4-x_{DE})] \leq v$

$(1)[(4 - x_{XC})+(6-x_{CD})+(6-x_{DF})] \leq v$

$$x_{XA} \leq 4$$

$$x_{AB} \leq 2$$

$$x_{XC} \leq 3$$

$$x_{CD} \leq 5$$

$$x_{DE} \leq 3$$

$$x_{DF} \leq 5$$

$$x_{XA} + x_{AB} + 2x_{XC} + 3x_{CD} + x_{DE} + x_{DF} \leq 6$$

$$x_{XA}, x_{AB}, x_{XC}, x_{CD}, x_{DE}, x_{DF} \geq 0$$

If we require $x_{ij}$ to be integers, we have to solve the previous formulation as an integer program.

**5.4. Marginal Contribution of Arcs in a Network to the Center Problem**

Instead of a spanning tree, suppose we started with a graph with one facility, and we want to answer the questions asked in the previous sections concerning link reductions, in the most general case - which arcs to reduce and by how much if arc reduction costs are different for each arc and we have a budget constraint.

*Determine the arc in the given graph which has the largest marginal contribution to $Z_C$.*

First determine the SPT rooted at X. Only arcs in this SPT matter to $Z_C$. Our problem then reduces to a spanning tree case and is determined by the procedure outlined in Section 5.1.

**5.5. Link Reduction of Y Units in a Network**

*Given a network and a budget of Y units, we want to find the optimal strategy of reduction to best improve $Z_C$.*

**5.5.1. Extending the Spanning Tree Solution**

Step 1. Create the SPT rooted at X and create matrix M. We will concentrate on reducing arcs in the SPT because they make marginal contributions to $Z_C$ as discussed in the preceding section. Treat this as a spanning tree problem and

determine the initial set of reductions according to methods in Section 5.2. After reducing the prescribed arcs, determine the new SPT, update distances, and reorder M.

Adjust the budget accordingly. Then, on the new SPT, use again the methods of Section 5.2 to find the appropriate arcs to reduce next. Repeat this procedure until our budget has been exhausted or until all arcs in the SPT have been reduced to the limit.

It is very possible that the path on the current SPT from X to $m_1$ has been reduced to the limit but we still have a remaining budget. This is a situation that results in the termination of our method in Section 5.2. However in a network, there may be other paths from X to $m_1$ which we may reduce. This characteristic that arcs not in the SPT are considered as candidates for reduction causes the extension of the spanning case procedure to the network case to yield results that are mere approximations to the optimal solution. This will be illustrated by the following example:

Given the network:

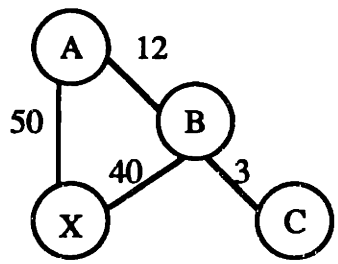



*Figure 5.6(a)*

Y = 13. How should we reduce the arcs?

Following the extension of the spanning tree solution, the SPT is

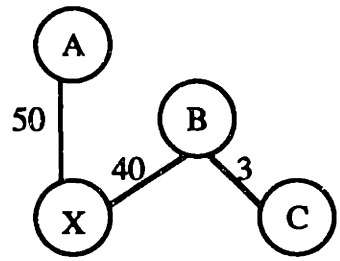



*Figure 5.6(b)*

M = [A, C, B] with distances from X of 50, 43, and 40 respectively. Reduce the path from X to A by 7 units. The new network is,

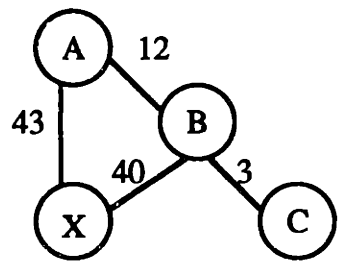



*Figure 5.6(c)*

and the corresponding SPT is

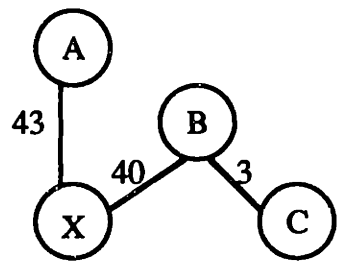



*Figure 5.6(d)*

M = [A C B] with distances from X of 43, 43, and 40 respectively. We have 6 units left to reduce. Reduce the path from X to A and the path from X to C by 3 units each (at arcs closest to X). The new network is

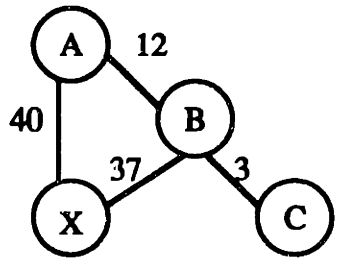



*Figure 5.6(e)*

and the corresponding SPT is

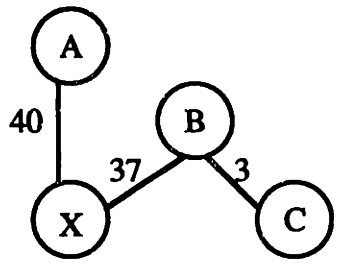



*Figure 5.6(f)*

M = [A C B] with distances from X of 40, 40, and 37 respectively. We have exhausted the budget. After reducing $a_{XA}$ by 10 units and $a_{XB}$ by 3 units, $Z_C$ becomes 40.

In the original network, suppose now we reduce $a_{XB}$ by 13 units. The new network is

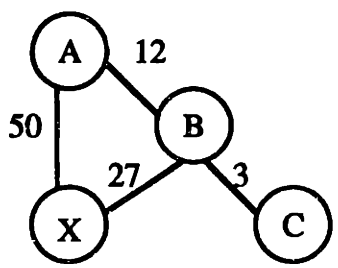



*Figure 5.6(g)*

and the corresponding SPT is

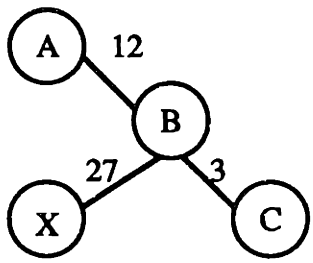



*Figure 5.6(h)*

M = [A C B] with distances from X of 39, 30, and 27 respectively. $Z_C$ = 39, which is smaller than what we obtain when using the solution prescribed by the extension of the spanning tree methods to networks.

Consider two paths from X to $m_1$. One is in the SPT , and the other is not. If we want to reduce $Z_C$ to $Z'_C$, the number of units that have to be reduced from the path not in the SPT is greater than the number of units that have to be reduced from the path in the SPT. Because of this, it may always seem more favorable to reduce paths in the SPT than paths not in the SPT. However, there should be another consideration, as illustrated by the previous example. Although the path may not be in the SPT, its reduction may also reduce the distance of other nodes from X which may eventually be critical to $Z_C$. In the example, there were two paths from X to A, X-A, which is in the SPT, and X-B-A, which is not in the SPT. Reducing X-B turned out to be better because it not only affected node A's distance from X but node C's distance as well. Because of this property, the iterative process that reduces only paths in the SPT, as discussed above, does not guarantee the optimal strategy.

### 5.5.2. Optimal Strategy for the Unweighted Network

We will try to obtain the optimal strategy for the case of the unweighted network. Since the network is unweighted, the farthest node will always be an end node.

**Reducing $Z_C$ to $Z'_C$**

Find the shortest path tree, and create matrix M. The paths of all the nodes which have distances from X via the SPT which are greater than $Z'_C$ need to be reduced if we are to attain a maximum distance of at most $Z'_C$. We will refer to these as our critical nodes: $m_1, m_2, ..., m_{\bar{c}}$.

For $m_1$, take the different paths from X to $m_1$ and determine by how much each path needs to be reduced to attain $Z'_C$.

Let $p_{m_1 l}$ = $l^{th}$ path from X to $m_1$, $l = 1, 2, ..., \overline{p_{m_1}}$

$r_{m_1 l}$ = the least amount path $p_{m_1 l}$ has to be reduced to attain $Z'_C$

= (current length of $p_{\bar{m}_1 l}$) - $Z'_C$

We can tabulate our results:

| | |
|---|---|
| $p_{m_1 1}$ | $r_{m_1 1}$ |
| $p_{m_1 2}$ | $r_{m_1 2}$ |
| ↓ | ↓ |
| $p_{m_1 \overline{p_{m_1}}}$ | $r_{m_1 \overline{p_{m_1}}}$ |

(For $m_2, m_3, ..., m_{\bar{c}}$ we can create similar tables.)

To reduce $m_1$'s distance from X to at most $Z'_C$,

$p_{m_1 1}$ must be reduced by $r_{m_1 1}$

or

$p_{m_1 2}$ must be reduced by $r_{m_1 2}$

or

↓

$p_{m_1 \overline{p_{m_1}}}$ must be reduced by $r_{m_1 \overline{p_{m_1}}}$ (5.1)

The statement "$p_{m_1 2}$ must be reduced by $r_{m_1 2}$" means that if we determine the arcs that comprise $p_{m_1 2}$ and call this set $P_{m_1 2}$ and define $x_{ij}$ as the amount to cut in arc $a_{ij}$, then

$$\sum_{P_{m_1 2}} x_{ij} \geq r_{m_1 2}$$

In general, the set of equations (5.1) can be translated to

$$\sum_{P_{m_1 1}} x_{ij} \geq r_{m_1 1}$$

or

$$\sum_{P_{m_1 2}} x_{ij} \geq r_{m_1 2}$$

or

↓

$$\sum_{P_{m_1 \overline{p_{m_1}}}} x_{ij} \geq r_{m_1 \overline{p_{m_1}}} \qquad (5.2)$$

At least one equation in the set (5.2) must be satisfied for each k, k=1, 2, ..., $\bar{c}$. For each $m_k$, if all $r_k$-values are greater than the allowable reductions on the corresponding paths, $Z'_C$ is not feasible.

The following mixed integer program will solve for the strategy that uses the fewest cuts to attain the predetermined value $Z'_C$:

(P5.5) minimize $R = \sum_{\substack{\text{all arcs in P,} \\ \text{the union of} \\ \text{all } P_{m_k 1}}} x_{ij}$

subject to

$$x_{ij} \leq l(i,j) - L^* \qquad \text{for all arcs in P}$$

For each k, one set of the following equations

$$\sum_{P_{m_k 1}} x_{ij} + By_{m_k 1} \geq r_{m_k 1} \qquad (5.3)$$

$$\sum_{P_{m_k 2}} x_{ij} + By_{m_k 2} \geq r_{m_k 2}$$

↓

$$\sum_{P_{m_k\overline{P_{m_k}}}} x_{ij} + By_{m_k\overline{P_{m_k}}} \geq r_{m_k\overline{P_{m_k}}}$$

$$\sum_{l=1}^{\overline{P_{m_k}}} y_{m_k l} \leq p_{m_k} - 1$$

$$x_{ij} \geq 0$$

$$y = 0 \text{ or } 1$$

B is a large positive number

The set of equations (5.3) guarantee that at least one of the equations in set (5.2) is satisfied.

Example. Refer to figure 5.6(a), and assume a limit of $L^* = 1$. If we choose $Z'_C = 37$, the critical nodes are A and C.

For node A, the different paths with corresponding r-values are

| | |
|---|---|
| X - A | 13 |
| X - B - A | 15 |

For node C,

| | |
|---|---|
| X - B - C | 6 |
| X - A - B - C | 28 |

$P = \{x_{XA}, x_{XB}, x_{BC}, x_{BA}\}$.

So ($x_{XA} \geq 13$ or $x_{XB} + x_{BA} \geq 15$) and ($x_{BA} + x_{BC} \geq 6$ or $x_{XA} + x_{BA} + x_{BC} \geq 28$).

The mixed integer program becomes

minimize $R = x_{XA} + x_{XB} + x_{BC} + x_{BA}$

subject to

(feasibility constraints)

$$x_{XA} \leq 49$$

$$x_{XB} \leq 39$$

$$x_{BC} \leq 2$$

$$x_{BA} \leq 11$$

(constraints for node A)

$$x_{XA} + By_{A1} \geq 13$$

$$x_{XB} + x_{XA} + By_{A2} \geq 15$$

$$y_{A1} + y_{A2} \leq 1$$

(constraints for node B)

$$x_{XB} + x_{BC} + By_{C1} \geq 6$$

$$x_{XA} + x_{BA} + x_{BC} + By_{C2} \geq 28$$

$$y_{C1} + y_{C2} \leq 1$$

$$x_{ij} \geq 0$$

$$y = 0 \text{ or } 1$$

B is a large positive number

**Calibrating $Z'_C$**

We claim that if we have chosen $Z'_C$ and determined that R is the fewest units of cuts needed to achieve $Z'_C$, if we choose $Z''_C < Z'_C$, the corresponding $R'' \geq R$.

Assume $R'' < R$. Consider the two formulations: F', which reduces $Z_C$ to $Z'_C$ and F", which reduces $Z_C$ to $Z''_C$. Let $x_{ij}$ be the units of reduction on arc $a_{ij}$ as specified by F' and $y_{ij}$ be the units of reduction on arc $a_{ij}$ as specified by F". The critical nodes $m_1, m_2, ..., m_{\bar{c}}$ of F' form a subset of the critical nodes of F", and the set of arcs P over which F" is minimized is a subset of the set of arcs P" over which F" is minimized. Since $Z''_C < Z'_C$, for every node m which is critical for both F' and F",

$$r_{mi} \text{ (for F')} \leq r_{mi} \text{ (for F")}$$

This means that for any arc $a_{ij}$ which is an element of P, and hence P", the corresponding $y_{ij}$ which is optimal in F", is feasible in F'. Also because P is a subset of P", $\sum_P y_{ij} \leq R''$. By assumption, $R'' < R$. So, $\sum_P y_{ij} < R$. If the set $y_{ij}$ is different from

the set $x_{ij}$ generated by F', there is a contradiction, because $x_{ij}$ is supposed to minimize R. If the set $y_{ij}$ is the same as the set $x_{ij}$ generated by F', there is also a contradiction, because $\sum_{P} x_{ij} = R$. So R" cannot be less than R' if we decrease $Z'_C$ to $Z''_C$.

Knowing that a decrease in $Z'_C$ will make R remain the same or increase R, we can set $Z'_C$ arbitrarily, determine the corresponding R and then calibrate until the ideal solution is found. The exact calibration algorithm is discussed in Appendix E3.

### 5.5.3. Optimal Strategy for the Weighted Network

The weighted network may be similarly solved. We choose the critical nodes for a predetermined $Z'_C$: $m_1, m_2, ..., m_{\bar{c}}$ where $\bar{c}$ has the property that

$$u_{m_{\bar{c}}} > Z'_C \text{ and } u_{m_{\bar{c}+1}} \leq Z'_C.$$

The $r_{m_k l}$ changes to

$$Z'_C \geq h_{m_k}[d_l(X, m_k) - r_{m_k l}]$$

$$r_{m_k l} \geq \frac{[h_{m_k} d_l(X, m_k)] - Z'_C]}{h_{m_k}} \quad \text{for all } k,\ k = 1, 2, ..., \bar{c} \text{ and } l = 1, 2, ..., \overline{p_{m_k}}$$

where $d_l(X, m_k)$ is the length of the $l^{th}$ path from X to $m_k$. As in the unweighted case, we must note that $x_{ij}$ may not be integers. A pure integer program may be done to generate integer solutions.

## 5.6. Link Reduction on Network Arcs with Varying Costs

*Suppose that it costs $c_{ij}$ to reduce one unit of arc $a_{ij}$. Determine the optimal strategy of reductions, within a specified budget C that would best improve the center objective function.*

The two approaches in the previous section may be extended to this case. First, the iterative heuristic which is an extension of the spanning tree method readily follows. Find the SPT of the network and create M. Treat this as a spanning tree problem and apply the methods in Section 5.3. Once the reductions have been made,

adjust the value of C. Detemine the new SPT and the corresponding M. Again apply the methods in Section 5.3 to this spanning tree. Do this until we cannot afford further reductions or until all arcs in a relevant path has been reduced to a minimum. The solution is not guaranteed to be optimal.

Second, the integer program approach needs the following change in the objective function of (P5.5):

$$\text{minimize} \quad \text{COST} = \sum_{\substack{\text{all arcs} \\ \text{in P}}} c_{ij}x_{ij}$$

The constraints remain the same. The calibrating phase is also modified. This is described in Appendix E4.

A third approach involving a larger mixed integer program is presented in Appendix E5.

# Chapter 6

## ARC ADDITIONS IN THE ONE-CENTER PROBLEM

The discussion in this chapter uses notations and concepts, that have been introduced in Chapter 4. Specifically, the reader is referred to Section 4.1 for a discussion of test arcs.

### 6.1. Addition of One Arc to a Spanning Tree

*We want to add a new arc to a spanning tree with a facility in order to best improve* $Z_C$.

#### 6.1.1. The Unweighted Spanning Tree

Suppose we have an unweighted spanning tree. Determine the corresponding M. Recall the analogous median problem in Section 4.2. In particular, the section on the cycle method is relevent to the material that follows.

To determine the test arc which will best improve $Z_C$, first, we will pick a test arc $a_{ij}$, and then perform Steps 2 -5 of the cycle method described in Section 4.2.2. Then the sixth step, described as follows, computes the improvement of $Z_C$ due to the addition of arc $a_{ij}$:

We start by choosing the farthest end node, and along with the other elements of M, try to use reasonable arguments and comparisons to find the effect of the added arc $a_{ij}$.

Initialize: $\alpha = 1$, MAX = $m_\alpha$, D(MAX) = d(X, $m_\alpha$)

MAX is the current farthest node from X, with distance D(MAX).

(i)Test the path of $m_\alpha$ to find out if it is connected with any of $W_{k^*} = X', \ldots, j$. If it is connected, its distance from X has been reduced, and its place in M has to be reevaluated in the light of this new distance. To do this

(a) Let $p = 0$, $Q_0 = m_\alpha$

(b) If $Q_p = X'$, $m_\alpha$ is not connected to the cycle, its distance from X has remained the same. Define $D(m_\alpha) = d(X, m_\alpha)$ and go to (ii)

If $Q_p$ is equal to any $W_{k^*}$ other than X', $m_\alpha$ is connected to the cycle, and define the new distance $D(m_\alpha) = \min \{d(X, m_\alpha), d(X, m_\alpha) - S_{W_{k^*}}\}$ and go to (ii)

Let $Q_p = A_{Q_p}$ and go to (b).

(ii) Did $m_{\alpha+1}$ distance from X change? (Determine by using part (i))

If it remained the same,

If $D(MAX) \geq d(X, m_{\alpha+1})$, stop.

Otherwise, $MAX = m_{\alpha+1}$, let $\alpha = \alpha + 1$, and go to (ii)

If it changed,

If $D(MAX) \geq D(m_{\alpha+1})$, let $\alpha = \alpha + 1$, and go to (ii)

Otherwise, let $MAX = m_{\alpha+1}$

Let $\alpha = \alpha + 1$, and go to (ii)

The savings associated with arc $a_{ij}$ is $I = d(X, m_1) - D(MAX)$.

Repeat this procedure for all the test arcs, and pick the arc with the largest associated savings. Since we are examining every possible solution to the problem, the solution is optimal.

**Example.**

Consider the following spanning tree:

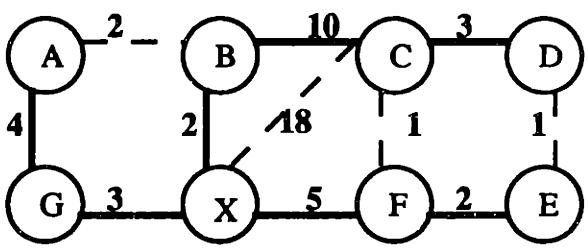



*Figure 6.1*

The bold lines are arcs in the spanning tree and the broken lines are the arcs specified by the decision maker. $a_{XC}$ is not a test arc. M = [D C A E F G B] with corresponding distances of 15, 12, 7, 7, 5, 3,and 2, respectively.

Testing $a_{BA}$:

Path from X to B: [X, B]

Path from X to A: [X, G, A]

X'=X, $W_k$*=X, G, A

$D_A = 7-(2+2) = 3$, $S_A = 3$

MAX = D, D(MAX) = 15

(i) Path from X to D: [X, B, C, D]

D is not equal to any $W_k$*

C is not equal to any $W_k$*

B is not equal to any $W_k$*

X = X'

∴ D is not connected to the cycle. $D(m_1) = D(D) = 15$

(ii) Did $m_2$=C change its distance from X?

Path from X to C: [X, B, C].

C is not equal to any $W_k$*

B is not equal to any $W_k$*

X = X'

∴ C is not connected to the cycle. D(C) = 12

D(MAX) = 15 ≥ d(X,C) = 12. Stop.

Associated savings is I = 15-15 = 0.

Testing $a_{FC}$:

Path from X to F: [X, F]

Path from X to C: [X, B, C]

X'=X, $W_k$* = X, B, C

$D_X = 0, S_X = 0$

$D_B = 2 - (5+1+10) = -14, S_B = 0$

$D_C = 12 - (5+1) = 6, S_C = 6$

MAX = D, D(MAX) = 15

(i) Path from X to D: [X, B, C, D]

D is not equal to any $W_{k^*}$

C is equal to one value of $W_{k^*}$ other than X'

$\therefore$ D is connected to the cycle. D(D) = min {15, 15-6} = 9

(ii) Did $m_2$=C change its distance form X?

Path from X to C: [X, B, C].

C is equal to one value of $W_{k^*}$ other than X'

$\therefore$ C is connected to the cycle. D(C) = min {12, 12-6} = 6

$D(MAX) = 9 \geq D(C) = 6$.

Let $\alpha = 2$

(ii) Did m3=A change its distance from X?

Path from X to A:[X, G, A]

A is not equal to any $W_{k^*}$

G is not equal to any $W_{k^*}$

X=X'

$\therefore$ A is not connected to the cycle. D(A) = 7

$D(MAX) = 9 \geq d(X,A) = 7$. Stop.

Associated savings is I = 15-9 =6.

Testing $a_{ED}$:

Path from X to E: [X, F, E]

Path from X to D: [X, B, C, D]

X'=X, $W_{k^*}$ = X, B, C, D

$D_X = 0, S_X = 0$

$D_B = 2 - (5+2+1+3+10) = -19$, $S_B = 0$

$D_C = 12 - (5+2+1+3) = 1$, $S_C = 1$

$D_D = 15 - (5+2+1) = 8$, $S_D = 7$

MAX = D, D(MAX) = 15

(i) Path from X to D: [X, B, C, D]

D is equal to one value of $W_{k^*}$

∴ D is connected to the cycle. D(D) = min {15, 15-7} = 8

(ii) Did $m_2$=C change its distance form X?

Path from X to C: [X, B, C].

C is equal to one value of $W_{k^*}$

∴ C is connected to the cycle. D(MAX) = 8 < D(C) = min {12, 12-1} = 11

MAX = C

Let $\alpha = 2$

(ii) Did $m_3$=A change its distance from X?

Path from X to A:[X, G, A]

A is not equal to any $W_{k^*}$

G is not equal to any $W_{k^*}$

X=X'

∴ A is not connected to the cycle. D(A) = 7

D(MAX) = 11 ≥ d(X,A) = 7. Stop.

Associated savings is I = 15-11 =4.

Therefore, the best arc to add is $a_{FC}$.

### 6.1.2. The Weighted Spanning Tree

The procedure is similar to that of the unweighted center problem. Determine M. For every test arc $a_{ij}$, perform Steps 2 -5 of the cycle method. Algorithm F1, described in detail in Appendix F1, is a modification of the algorithm described in

Section 6.1.1 and computes the improvement of $Z_C$ associated with $a_{ij}$. Repeat this procedure for all the test arcs, and pick the arc with the largest associated savings.

**6.2. Addition of L Arc Units to a Spanning Tree**

*Suppose we had the capacity to add L arc-units instead of one arc to a spanning tree. Find the arcs that would best improve $Z_C$.*

If we had L arc units to add to this spanning tree, to determine which arcs to add, first find M.

If $u_{m_1} > u_{m_2}$, determine the savings associated with the addition of each test arc. The procedures discussed in Section 4.2 may be used for this. Using the associated savings as the criterion for ordering our preferences for link addition, we prefer to add the link with the largest savings first. If this arc satisfies the feasibility constraint, such that $l(i,j) \leq L$, add it to the graph. Otherwise, try to add the link with the second largest savings. Again, check if this arc satisfies the feasibility constraint. If so, add it to the graph. Otherwise, go to the next link in our ordered list of test arcs. Repeat this prodedure until we have added one arc to our graph.

Then adjust the budget L, $L = L - l(i,j)$. Determine the new SPT and reorder M. We have completed one iteration. Begin the next iteration by determining whether or not $u_{m_1} > u_{m_2}$.

If $u_{m_1} = u_{m_2}$, there are nodes that are tied for the largest $u_j$. Determine the tied nodes $z_1, z_2, \ldots, z_T$. We have to reduce the distances to X of all the z-nodes so that $Z_C$ will decrease. In an approach similar to that in Section 5.5.2, we choose a value $Z'_C$, which is equal to the second largest $u_j$-value. We would like to reduce $Z_C$ to $Z'_C$. To do this, tabulate the following:

Test Arc  Length of Test Arc  Node $z_1$  Node $z_2$  ...  Node $z_T$

where under the first two columns we identify the test arcs $a_{ij}$ and their lengths. To complete the table, beginning with the first test arc, examine the new SPT that results when it is added to the existing spanning tree. According to this new SPT, determine

the value of $u_{z_1}$. If it is different from the former $u_{z_1}$, write the new value under column $z_1$ in the row of the first test arc. Otherwise leave the column entry blank. Repeat this procedure for all the other nodes $z_2, z_3, ..., z_T$. Then choose the second test arc and repeat the procedure to determine the entries in the second row of the table. Perform this until the table is complete.

Once the table has been constructed, define $b_{ijz}$ as an indicator variable that is equal to 1 when the entry under node z corresponding to arc $a_{ij}$ is less than or equal to $Z'_C$ and 0 otherwise.

Example:

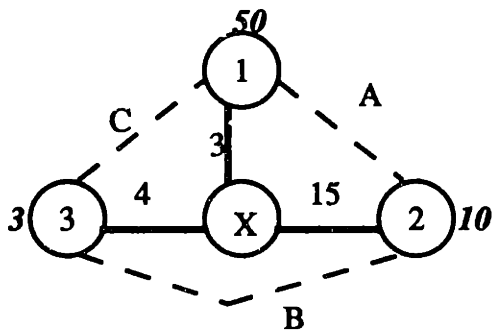



*Figure 6.2*

The nodes and the solid lines define our initial spanning tree. The broken lines are arcs specified by the decision maker. The italicized numbers are the weights of the nodes. $u_1 = 150$, $u_2 = 150$, $u_3 = 12$, hence M = [1,2,3], where nodes 1 and 2 are tied. Suppose that $L^* = 8$ and arcs A, B, and C have lengths of 8, 3, and 6 respectively. C is not a test arc.

| Test Arcs | Length of Arc | Node 1 | Node 2 |
|---|---|---|---|
| A | 8 | - | 110 |
| B | 3 | - | 70 |

where 110 = (10)(3+8) and 70 = (10)(4+3). Currently $Z'_C = 12$, the second largest $u_j$-value. $b_{A1} = b_{A2} = b_{B1} = 0$ and $b_{B2} = 0$. To reduce $Z_C$ to 12, we must add arcs that will ensure that the $u_j$-values of the z-nodes in the new graph will be less than or equal to

12. Examine the column under node 1. This shows us that the addition of any test arc does not affect node 1's $u_j$-value. Examine the column under node 2. This shows us that the addition of any test arc will reduce $u_2$ to at most 70 which is greater than 12. Hence $Z'_C = 12$ is infeasible. In fact because node 1 remains unchanged with any arc addition as shown by the empty column under it, $Z_C$ cannot be improved at all.

In general, solve the integer program:

$$\text{minimize} \sum_{\substack{\text{over all} \\ \text{test arcs}}} l(i,j)x_{ij}$$

$$\text{subject to} \sum_{\substack{\text{over all} \\ \text{test arcs}}} b_{ijz}x_{ij} \geq 1, \qquad \text{for each } z$$

$$x_{ij} = 0 \text{ or } 1 \text{ (not to add } a_{ij} \text{ or to add } a_{ij}, \text{ respectively)}$$

If we can afford the strategy that the program yields, add the arcs as specified by the indicator variables $x_{ij}$. Otherwise, among the $u_j$-values in the table which are larger than the current $Z'_C$ , choose the smallest value, and let the new $Z'_C$ be equal to this number. Solve another integer program using the new $Z'_C$. (In our example, the new $Z'_C$ is 70.) Repeat this procedure until the specified strategy is within our budget. Once we find an appropriate addition strategy, apply it to the existing graph then determine the new SPT and the corresponding M. Adjust the budget. We are ready for another iteration. Begin the next iteration by determining whether or not $u_{m_1} > u_{m_2}$.

We continue this procedure until we exhaust our budget or until the remaining budget is not sufficient to add even the shortest test arc.

The procedure discussed above is iterative in nature. As such it fails to capture the dependency of arcs we have earlier noted in other problems. The effect of an arc on a spanning tree depends on what arcs will be added to the spanning tree together with that arc.

**Remark.** For the variation that instead of adding L arc units we can add L arcs, only the termination criterion changes. The approach is still the same.

### 6.3. Arc Addition to a Spanning Tree with Varying Costs

*Suppose there are costs $c_{ij}$ associated with every arc we could add. Determine which arcs should be added to remain within a budget C and best improve $Z_C$.*

If it costs $c_{ij}$ to add arc $a_{ij}$ to the existing graph, the iterative heuristic in Section 6.2 can be slightly modified to address the problem. Instead of considering arc lengths to determine the feasibility of a suggested strategy, we use costs. Modify the procedure in Section 6.2 as follows:

(1) If $u_{m_1} > u_{m_2}$, the feasibility criterion becomes $c_{ij} \leq C$ instead of $l(i,j) \leq L$.

(2) If $u_{m_1} = u_{m_2}$, the integer program has a different objective function:

$$\text{minimize} \sum_{\substack{\text{over all} \\ \text{test arcs}}} c(i,j)x_{ij}.$$

### 6.4. Addition of One Arc to a Network

*Given a network, find the arc to add to best improve $Z_C$.*

Instead of spanning trees, suppose that X is located on a general network. We would like to investigate how the heuristic procedures of the past three sections will perform. In this section, the extension from the spanning tree case would be very similar to the analogous median problem which extends to a network the method for adding one arc to a spanning tree as discussed in Section 4.5.

Step 1. Find the SPT of the network.

Step 2. Choose a test arc.

Step 3. Add the test arc and determine the new SPT.

Step 4. Reorder M.

Step 5. The difference between the former $Z_C$ and the new $Z_C$ is the improvement associated with the test arc.

Step 6. Choose another test arc and repeat Steps 2 - 5.

Step 7. Terminate when all test arcs have been chosen. Add the test arc with the largest associated improvement.

**6.5 Addition of L Arc Units to a Network**

*Given a network and the capacity to add L units, determine the set of arcs to be added to the network to best improve $Z_C$.*

Given a budget of L arc units to add to a network, as in the spanning tree case, the optimal solution cannot be determined iteratively. But the heuristic in Section 6.2 may be extended directly to the case of networks by finding the SPT to start the algorithm.

**Remark.** To the variation of the problem wherein we can add L arcs instead of L arc units, the same comments apply.

Consider the following network:

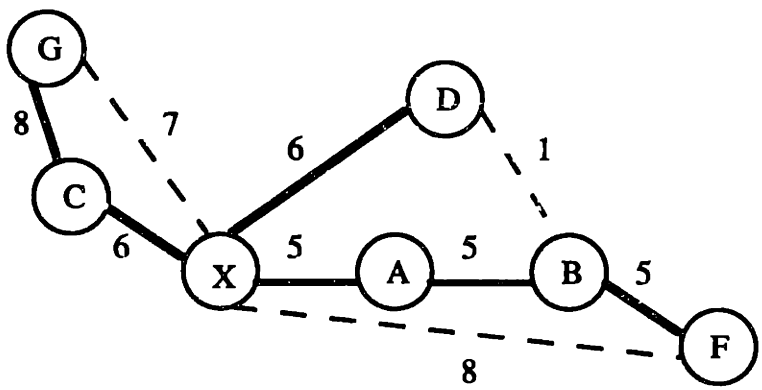



*Figure 6.3*

The bold lines are arcs in the SPT and the broken lines are test arcs. The current $Z_C = 15$ and $m_1 = F$. Suppose we have 8 units to add. Based on the previous algorithm we would do the following:

Add $a_{XF}$ (8 units), $Z_C = 14$, $m_1$ becomes G.

We have exhausted our budget, stop.

But suppose we allocate the 8 units in the following manner:

Add $a_{XG}$ (7 units) and $a_{DB}$ (1 unit), $Z_C = 12$, $m_1$ remains F. This is a better strategy than that specified by the iterative algorithm presented above. We notice from this example that our algorithm may cause "too much"" reduction to $m_1$ than is necessary. We therefore s ıggest another approach which may be used for both the spanning tree or network case.

Step 1. If we are given a network, determine the SPT.

Step 2. Determine M.

Step 3. Choose any value to which we want to reduce $Z_C$ and call it $Z'_C$. (We may initially let $Z'_C = Z_C - L$.) As before, we obtain a corresponding set of nodes $m_1, m_2, ..., m_{\bar{c}}$, where $u_{m_{\bar{c}}} > Z'_C$.

Step 4. Tabulate:

Test Arcs  Length  $m_1$  $m_2$  ...  $m_{\bar{c}}$

Under the first two column, identify the test arcs and their corresponding arc lengths. Entries to the other columns are determined using the procedure described in Section 6.2 where a similar table was constructed. Hence, under column $m_k$, place the value $u_{m_k}$ would become if arc $a_{ij}$ was added to the network by itself. Then, formulate the following program:

$$\text{minimize} \sum_{\text{test arcs}} l(i, j)x_{ij} \quad \text{(if we have L arc units to add)}$$

$$\text{or} \sum_{\text{test arcs}} x_{ij} \quad \text{(if we have L arcs to add)}$$

subject to

$$\sum_{\text{test arcs}} b_{ijk}x_{ij} \geq 1 \quad \text{where } b_{ijk} \text{ is as defined in Section 6.2,}$$

there are $\bar{c}$ constraints, one for each k

$x_{ij} = 0$ or 1 (not to add $a_{ij}$ or to add $a_{ij}$, respectively)

This integer program yields a proposed set of arc additions as specified by $x_{ij}$.

Step 5. Calibrate $Z'_C$ so that $\sum_{\text{test arcs}} l(i, j)x_{ij} > L$ and the cheapest strategy that achieves $Z'_C - 1$ is beyond our budget L.

In the example, M = [F, G, B, C, D, A] with $u_j$ values of 15, 14, 10, 9, 6, and 5, respectively. If we choose $Z'_C = 13$, the critical nodes are G and F.

| Test Arcs | Length | G | F |
|---|---|---|---|
| $a_{XG}$ | 7 | 7 | 15 |
| $a_{XF}$ | 8 | 14 | 8 |
| $a_{DB}$ | 1 | 14 | 12 |

The matrix of corresponding $b_{ijk}$ is

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

The integer program is:

minimize $7x_{XG} + 8x_{XF} + x_{DB}$

subject to

$x_{XG} \geq 1$

$x_{XF} + x_{DB} \geq 1$

$x_{XG}, x_{XF}, x_{DB} = 0 \text{ or } 1$

The solution generated by the integer program is an approximation of the optimal solution. The table we constructed considers only the effect of an arc by itself, when in fact we know that an arc is highly dependent to the entire set of arcs that are added along with it. We did not consider the effect of pairs of arcs, or groups of larger sizes. As we consider more groups of arcs, the more precise our solution will be. The program becomes very large however. Suppose we wanted to include the effect of pairs of arcs, then among the test arcs, we will determine every group of twos, add these successively to the network and record each pair's effect on each critical node. If there are P pairs, we will add P more rows to our table, and P more variables to our

linear program. Including the pairs does not mean we can ignore the single arcs. The optimal solution may add an odd number of arcs.

In our example, suppose there is another test arc $a_{AD}$ with length 0.5. If we perform the tabulation, we will get:

| Test Arcs | Length | G | F |
|---|---|---|---|
| $a_{XG}$ | 7 | 7 | 15 |
| $a_{XF}$ | 8 | 14 | 8 |
| $a_{DB}$ | 1 | 14 | 12 |
| $a_{AD}$ | 0.5 | 14 | 15 |

The set of constraints resulting from these values will imply that the best value of G's distance from X is 7 and the best value of F's distance from X is 8. It does not reflect that $a_{AD}$ and $a_{DB}$ together could improve F's distance from X to 11.5 while only exhausting 1.5 units of the budget. Adding the effect of pairs to the formulation will remedy this situation. But it still does not consider the effect of triples, or groups of larger sizes.

### 6.6. Arc Addition on a Network of Varying Costs

*Suppose there are costs associated with every arc we could add; how should arcs be added to a network to best improve $Z_C$ if we have a budget C.*

If costs are associated with the arcs, we can extend the heuristic for the spanning tree case, discussed in Section 6.3. by finding the SPT to start the algorithm.

The integer programming formulation in the previous section can also be applied to this problem. We merely change the objective function to

$$\text{minimize} \quad \sum_{\text{test arcs}} c_{ij}x_{ij}$$

and the termination criterion which has to consider costs rather than lengths.

## Chapter 7

# THE MULTI-FACILITY PROBLEM

In previous chapters, we have considered a spanning tree or a network with only one facility. The applicability of such configurations is limited. Graphs with two or more facilities are better representatives of actual service systems. It is imperative, therefore, to extend our discussion to the general case of k-facilities. Fortunately, this is easy to do.

Given a network and two facilities, a shortest path tree rooted at a conveniently placed dummy node minimizes both the median and center objective functions. The dummy node D has two arcs incident to it, one is $a_{DX_1}$ and the other is $a_{DX_2}$ where $X_1$ and $X_2$ are the two facilities. Let $a_{DX_1} = a_{DX_2}$. An illustration of this case is:

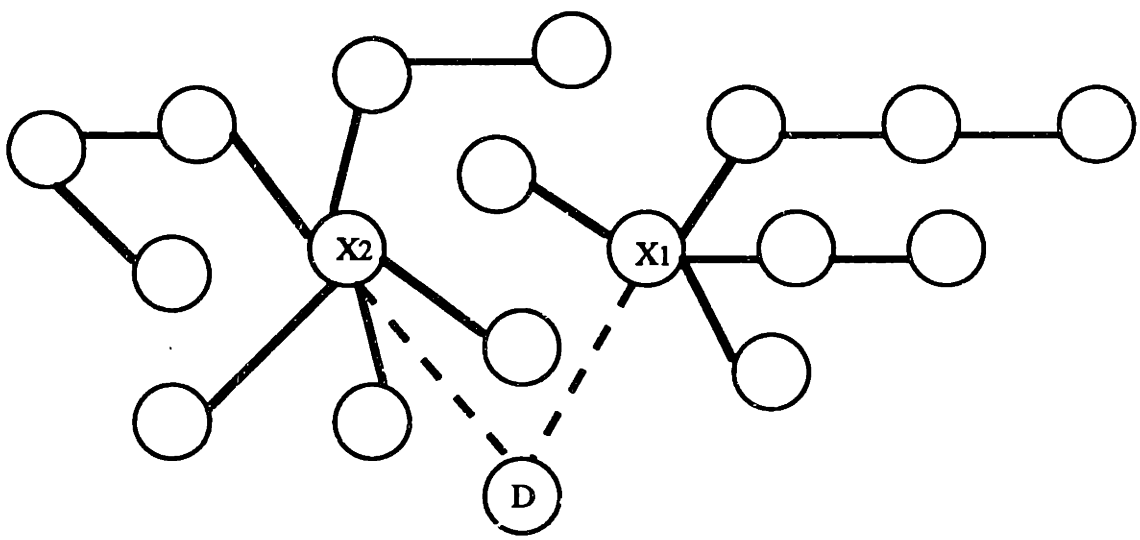



*Figure 7.1*

Because there are only two ways to reach D, via $X_1$ or via $X_2$, and $a_{DX_1} = a_{DX_2}$, the shortest path tree rooted at D will yield paths from nodes of the network to either $X_1$ or $X_2$ that minimizes the 2-median and 2-center objective functions.

If there are k-facilities, the dummy node D must have k arcs of equal length incident to it, one to each facility. Because these imaginary arcs provide the only way to reach D, and they are of equal length, the shortest path tree rooted at D will yield the necessary paths from nodes of the network to exactly one of the facilities that will minimize both the k-median and k-center objective functions.

In the one-facility case, the total savings associated with a set of link reductions or link additions is based on the new distances generated by the new SPT. The choice of which arcs to reduce or add is determined by the distances of the nodes relative to one another. But the addition of a dummy node that is equidistant to all the facilities maintains the order of the distances of the nodes to the facilities. Hence, all link reduction and link addition algorithms for the 1-facility case can be extended to the k-facility case by observing some restrictions:

(1) The actual distances that should be utilized to compute $Z_M$ and $Z_C$ are the distances on the pseudo-shortest path tree rooted at D minus the imaginary arc's length.

(2) Arcs adjacent to D may not be reduced in length.

(3) No arc additions adjacent to D may be made, nor arc additions between any two facilities.

These restrictions imply that the independence of each facility from another is guaranteed, and that the special property that the nodes can reach D through only one of the facilities is maintained. The arcs referred to in (2) and (3) are never candidates for reduction or addition, and may never enter the different computations.

The addition of a dummy node increases our node set from N to N + k. It is as if the k facilities are ordinary nodes (but do not carry any demand) in the 1-facility pseudo-network where D is the facility. This may increase substantially the amount of computations required inasmuch as our approaches have orders which are proportional to the number of nodes in the graph (excluding the facility). On the other hand, this

technique simplifies our extension to the multi-facility case and gives us flexibility in the implementation of all our earlier procedures.

## APPENDIX A

### The Dijkstra Algorithm with Counters that Measure the Median Objective Function and Center Objective Function

Given G(N $\cup$ X,A) where X is the fixed facility, we can use the Dijkstra Algorithm to find the shortest path tree, with the inclusion of counters that will eventually provide the value of the median and center objective functions. Let these counters be $Z_M$ and $Z_C$, respectively.

Let $a_{ij}$ = arc between node $i$ and node $j$

$l(i,j)$ = length of $a_{ij}$ if such an arc exists, $l(i,j) \geq 0$

C = set of all closed nodes (nodes for which the shortest paths to X have already been found)

O = set of all open nodes (nodes for which the shortest paths to X are still being determined)

At each iteration, every node $j$ will be labelled [$d(j)$, $p(j)$ ] where

$d(j)$ = length of the shortest path from node X to node $j$

$p(j)$ = the node before node $j$ in the shortest path from node X to node $j$

Initialize. $C = \{X\}$, $d(X) = 0$, $p(X) = 0$

$O = N$, $d(j) = \infty$, $p(j) =$ --- for all $j \neq X$

Set $k=X$, $Z_M = 0$, $Z_C = 0$

Step 1. For every open node $j$ such that $a_{kj}$ exists, determine

$$d(j) = \min [\, d(j), d(k) + l(k,j) \,]$$

Step 2. Find $d(j^*) = \min_{j \in O} d(j)$

Step 3. Find $i \in C$ such that $d(i) + l(i,j^*) = d(j^*)$

Step 4. Label node $j^*$ with $[d(j^*), i]$

Let $C = C \cup \{j^*\}$, $k = j^*$

$Z_M = Z_M + h_j d(j^*)$

$Z_C = \max\{Z_C, h_j d(j^*)\}$

If $C = N$, stop.

Otherwise, go to Step 1.

Upon termination,we can construct the shortest path tree by connecting nodes to their predecessors, $p(j)$, and the value of the median and center objective functions are $Z_M$ and $Z_C$ respectively.

# APPENDIX B

## Defining Initial Data

From the shortest path tree rooted at X, or any other arbitrary spanning tree, we can determine the following:

(1) $l(i,j)$ = lengths of the arcs between nodes $i$ and $j$ wherever they exist

(2) An array D where the $j$th entry, $d(X,j)$ is the distance from node $j$ to node X, for all $j \in N$.

(3) An array A where the $j$th entry $A_j$ is the node before node $j$ in the path from X to j

Reference to these data sets and notations will be made frequently in the subsequent appendices.

## APPENDIX C1

### ALGORITHM C1. Finding $H_c$

Define c as any node adjacent to the facility, and $H_c$ as the total weight of the nodes that have node c in their paths to X

Step 1. A matrix called D is first constructed. It is a matrix with N columns. Under the nth column are the nodes which have node n immediately before them in the path from X to them. Entries to this matrix will be denoted $f_{row,column}$.

Illustration.

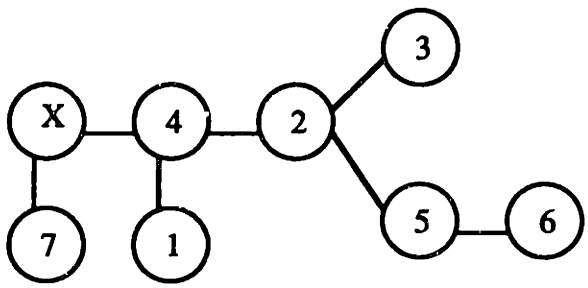



*Figure A1*

Nodes (1 2 3 4 5 6 7)

A = (4 4 2 X 2 5 X)

$$D = \begin{bmatrix} 0 & 3 & 0 & 2 & 6 & 0 & 0 \\ 0 & 5 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

More exactly, the procedure for finding D is as follows:

Initialize: Let k = 1, $R_1 = 1$, $R_2 = 1$, ..., $R_n = 1$, where $R_i$ is the row of the first zero entry in column i, $f_{row,\ column} = 0$ everywhere

(i) Let $f_{R_k, A_k} = k$ for all times k appears in A

Let $R_{A_k} = R_k + 1$

(ii) If k=N, stop.

Otherwise, let k = k+1 and go to (i).

The final matrix size is $(\max_n R_n) \times N$. Denote $\max_n R_n = r$

Step 2. To determine the H-values for each of the c-nodes. The following algorithm may be used:

Starting from c and moving away from X, a set of nodes immediately following c is determined and we call this T. We add to $H_c$ (which is currently equal to $h_c$) the weights of all nodes in T. Then, one at a time we take an element in T and determine what node immediately follows it. We add the weight of every "successor" node to $H_c$ and put the "successor" node in set T, meaning we will later check if it is an end node or not. (If not, the weights of the nodes following it must be added to $H_c$ as well.) If an element in T has been checked for "successor" nodes, remove it from T and check the next element in T. Do this until T is empty. Note that at any point in the algorithm, set T is the set of nodes whose weights are already added to $H_c$ but we still have to check if there are nodes attached to them whose weights have to be further added to $H_c$. With the use of previous data and notations, it is more precisely defined by the following:

Initialize: Let $H_c = h_c$, $s = c$, $T = \varnothing$

(i) Let $z = 1$

(ii) If $f_{z,s} = 0$, go to (iii)

Otherwise, let $H_c = h_c + h_{f_{z,s}}$ , $T = T \cup \{f_{z,s}\}$

If $z = r$, go to (iii)

Otherwise, let $z = z + 1$, and go to (ii)

(iii) If $T = \varnothing$, stop.

Otherwise, let $s = t^*$, where $t^*$ is the first element of T (Let T be a first in first out system.)

Go to (i)

# APPENDIX C2

**ALGORITHM C2. Finding $H_j$ for a given j**

Define j as any node in the graph, and $H_j$ as the total weight of the nodes that have node j in their paths to X. This is exactly Algorithm C1 except that we replace c by j.

## APPENDIX C3

**ALGORITHM C3. Finding $H_j$, for all $j \in N$**

To find $H_j$ for all $j \in N$ would require going Algorithm C2 N times. The alternative algorithm presented here determines the $H_j$-values for all $j \in N$ when the algorithm terminates.

Illustration.

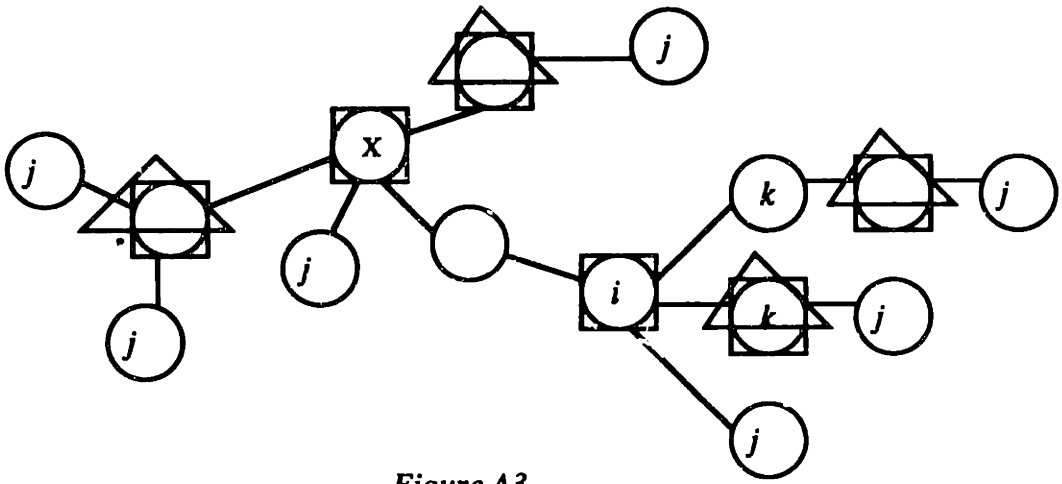



*Figure A3*

With the above figure, illustrating the notation used in the algorithm, we determine $H_j$ for all $j \in N$ as follows:

Label all ends $l_j = h_j$. Check all nodes $i$ such that $a_{ij}$ exists and $j$ is labelled node. For each $i$, there are arcs $a_{ik}$ (where by definition $d(X,k) > d(X,i)$). If all nodes $k$ are labelled, then label node $i$: $l_i = h_i + \sum_k l_k$ and for all these $k$'s , let $H_k = l_k$ and truncate $k$'s from the tree.

In the above figure, the ends are called $j$'s. The "squared" nodes are all i-node-candidates for labelling. However only the "triangled" nodes are labelled on the present step of the algorithm because the two "untriangled" nodes have adjacent nodes (in the direction away from X) that are not yet labelled. So nodes have to wait until all nodes adjacent to them (in the direction away from X) are labelled before they themselves are labelled.

After labelling all possible nodes among the candidate i-nodes, the newly labelled nodes become the ends of the "new truncated" tree. We can repeat the procedure described above. Terminate the procedure when $H_X$ has been determined.

## APPENDIX C4

### The Modified Shortest Path Tree Algorithm

If one arc has been added to an existing SPT or one arc in an SPT has been reduced, this algorithm may be used to find the new SPT due to the addition or reduction.

We were originally given a network with a facility and were asked to find the SPT. The final labels in that algorithm is what we are going to use to start the modified SPT.

This is a modification of Dijkstra's labelling algorithm used to find the SPT rooted at X in Appendix A. If $a_{ij}$ was added or reduced, start the "modified" algorithm from node *j*. From node *j*, label out as in Dijkstra's algorithm reinitializing C, the set of closed nodes to include only node *j*. All other nodes are open. The closed nodes are nodes whose distance from X cannot be improved further by the reduction $Y_c$.

Step 1. Label node *j* as [ m(*j*), q] where

m(*j*) = new shortest distance from node j to X due to the addition or reduction

q = predecessor node of *j*

Step 2. Check all arcs out of *j*, called $a_{jk}$ for k unclosed, and compute for each such k: $m(k) = \min \{m(j) + l(j,k), m(k)\}$.

If $m(j) + l(j,k) = m(k)$ for all open k, stop.

Otherwise, among m(k)'s not equal to d(X,k), choose

m(k*) = smallest of the chosen subset of m(k)'s

Find $q^* \in C$ such that $m(q^*) + l(q^*,k^*) = m(k^*)$

Let $C = C \cup \{k^*\}$

Step 3. Search matrix D (described in Appendix C1) for k*. Move k* from where it is to column q*. This is not part of Dijskstra's algorithm; the data set is just being updated for further use.

Step 4. Label node k* as [m(k*), q]. Let $j$ = k*.

If C = N, stop.

Otherwise, go to Step 2.

From the labelled predecessor nodes in the previous algorithm, the new SPT resulting from the reduction or addition can be determined.

## APPENDIX C5

**The Nonlinear Mixed Integer Program for Link Reduction in a Network Described in Section 3.5.2**

maximize $(x_{X2} - 10)(35)(1 - W_3)$

$+ (x_{X2} - 12)(7)(1 - W_7)$

$+ x_{X2}(30) + x_{X7}[7 - (1 - W_7)7]$

$+ x_{X3}[35 - (1 - W_3)35] + x_{X6}(40)$

subject to (feasibility constraints)

$x_{X2} \leq 20 - 4 = 16$ $\quad$ $x_{X3} \leq 60 - 4 = 56$

$x_{X7} \leq 50 - 4 = 46$ $\quad$ $x_{X6} \leq 30 - 4 = 26$

(budget constraint:)

$x_{X2} + x_{X7} + x_{X3} + x_{X6} = 15$

(constraints for node 3)

$x_{X2} - Bu_{31} > 10$

$W_3 - Bu_{32} \leq 1$

$W_3 + Bu_{32} \geq 1$

$u_{31} + u_{32} = 1$

$x_{X2} + Bt_{31} \leq 10$

$W_3 - Bt_{32} \leq 0$

$W_3 + Bt_{32} \geq 0$

$t_{31} + t_{32} = 1$

(constraints for node 7)

$x_{X2} - Bu_{71} > 12$

$W_7 - Bu_{72} \leq 1$

$W_7 + Bu_{72} \geq 1$

$u_{71} + u_{72} = 1$

$x_{X2} + Bt_{71} \leq 12$

$W_7 - Bt_{72} \leq 0$

$W_7 + Bt_{72} \geq 0$

$t_{71} + t_{72} = 1$

$W_3$, $W_7$, u and t are binary variables

$x_{X2}, x_{X7}, x_{X3}, x_{X6} \geq 0$

B is a large positive number.

## APPENDIX C6

**The Nonlinear Mixed Integer Program for Link Reduction in a Network Described in Section 3.6**

Let $a^*_{ijP_1}$ be arcs in $P_1$, and let $a^*_{ijP_2}$ be arcs in $P_2$. The objective function will be:

$$\text{maximize} \sum_{\substack{k \\ \text{(counts the nodes that could be pulled by } n_1\text{)}}} \left\{\left[\left(\sum x^*_{ijP_1} + x_{n_1p_{k_1}}\right) - v_{n_1p_{k_1}P_1}\right]H_{p_{k_1}}[1 - W_{p_{k_1}}]\right\}$$

$$+ \sum_{\substack{k \\ \text{(counts the nodes that could be pulled by } n_2\text{)}}} \left\{\left[\left(\sum x^*_{ijP_2} + x_{n_2p_{k_2}}\right) - v_{n_2p_{k_2}P_2}\right]H_{p_{k_2}}[1 - W_{p_{k_2}}]\right\}$$

$$+ \sum_{\substack{\text{arcs in the SPT but not in } P_1 \cup P_2}} \left\{ x^*_{ij}[H_j - \sum_{\substack{a \\ \text{(a-nodes are special p-nodes that originally were counted with } H_j\text{)}}} (1 - W_a)H_a] \right\}$$

$$+ \sum_{\substack{\text{arcs in} \\ P_1 \cup P_2}} \left\{x^*_{ij}H_j\right\}$$

There should be:

(1) feasibility constraints for all arcs in the SPT, all arcs in $P_1 \cup P_2$, and all arcs from $n_1$ and $n_2$ to their corresponding p-nodes;

(2) a budget constraint:

$$\sum x_{ij}c_{ij} + \sum x^*_{ijP_1}c^*_{ijP_1} + \sum x^*_{ijP_2}c^*_{ijP_2} + \sum x_{n_1p_{k_1}}c_{n_1p_{k_1}} + \sum x_{n_2p_{k_2}}c_{n_2p_{k_2}} \leq C$$

(3) for each node that may be pulled, a set of eight equations similar to Equations (3.5.1) - (3.5.8) which records whether the node has been pulled or not.

## APPENDIX D1

**ALGORITHM D1. Finding the Path from X to j**

Define j as any node in the graph and X is the facility.

(i) Let $n = 0$, $W_0 = j$

(ii) Let $W_{n+1} = AW_n$

(iii) If $W_{n+1} = X$, stop. The path is $[W_{n+1}, W_n, ..., W_1, W_0]$

Otherwise, go to (ii)

## APPENDIX D2

**ALGORITHM D2. Determining the Nodes in $T_1$**

This algorithm uses notations and concepts introduced in Section 4.2.1. Let n be any node in the spanning tree.

(i) Let p =0, $Q_0 = n$

If $n = W_k$, then $n \in T_1$, stop.

(ii) Let $Q_{p+1} = AQ_p$

If $Q_{p+1}$ is any of $W_k$, $W_{k-1}$, ..., $W_0$, then $n \in T_1$, stop.

If $Q_{p+1} = X$, $n \in T_2$, stop.

Otherwise, let p = p+1 and go to (ii)

## APPENDIX D3

**ALGORITHM D3. Determining the Improvement of $Z_M$ Due to the Addition of Arc $a_{ij}$**

This algorithm uses notations and concepts introduced in Section 4.2.1 and Appendix D2.

If $n \in T_1$ the associated savings is $s_n = h_n\{d(X,n) - [d(X,i) + l(i,j) + d(j,n)]\}$, where $d(j,n) = \sum_{\beta=0}^{p} l(Q_\beta, Q_{\beta+1}) + \sum_{\gamma=0}^{k^*} l(W_\gamma, W_{\gamma+1})$, where $k^*$ is such that $Q_{p+1} = W_{k^*}$.

Do this for all $n \in T_1$. If $n \in T_2$, there is no improvement to the objective function.

The total savings or improvement to $Z_M$ is

$$\sum_{n \in T_1} s_n$$

## APPENDIX D4

**Determining the Node of Intersection of Two Paths that is Farthest from X**

This algorithm uses notations and concepts discussed in Section 4.2.2. Given the paths $[Y_{m+1}, Y_m, ..., Y_0]$, where $Y_{m+1} = X$ and $Y_0 = i$, and $[W_{n+1}, W_n, ..., W_0]$, where $W_{n+1} = X$ and $W_0 = j$.

(i) Let $i = m+1$, $j = n+1$

(ii) If $Y_i = W_j$, let $i = i+1$, $j = j+1$ and go to (ii)

Otherwise, $X' = Y_{i+1} = W_{j+1}$ = the node of intersection farthest from X.

# APPENDIX E1

## ALGORITHM E1. Constructing the $\pi$ Matrix

Define the z x N matrix $\pi$ with components

$\pi_{ij} = 1$ if node j is in the path from X to i

0 otherwise

Every row corresponds to a tied node z. The matrix $\pi$ may be constructed as follows:

Initialize: $t = 1$, $w = m_1$, $\pi_{ij} = 0$

(i) Let $\pi_{t,w} = 1$

If $A_w = X$, go to (ii)

Otherwise, let $w = A_w$, go to (i)

(ii) If $t = z$, go to (iii).

Otherwise let $t = t + 1$ and $w = m_t$

go to (i)

(iii) Delete all columns that have only one entry of 1. All columns will have at least one entry of 1 because every node is in the path from X to itself.

## APPENDIX E2

### ALGORITHM E2. Reordering M

Initialize: Let w = $M$, where $M$ is the number of elements of M

(i) If $u_{m_{w-1}} \geq u_{m_w}$, then let w = w - 1 and go to (i).

Let t = $m_{w-1}$ (the node whose correct rank we want to detemine)

Let $m_{w-1} = m_w$ ($m_w$ was moved one slot forward)

Let v = w (v is the slot which is currently vacant)

(ii) If v = $M$, go to (iii)

If v < $M$, let v = v+1

If $u_t \geq u_{m_{v+1}}$ go to (iii)

Otherwise let $m_v = m_t$, v = v+ 1 and go to (ii)

(iii) Let $m_v = m_t$

If w > 2 let w = w - 1 and go to (i)

Otherwise, stop.(M has been reordered.)

## APPENDIX E3

**ALGORITHM E3. Calibrating $Z'_C$ When No Costs Are Associated with Arc Reductions**

Define Y as the maximum number of arc reductions that could be performed

Step 1. Choose $Z'_C$ arbitrarily. (We could let $Z'_C = Z_C - Y$ to find out if we could make all our Y units count. This is the lowest possible $Z'_C$ we can have.)

Step 2. Do (P7.5) and find R.

Step 3. If $Y < R$, choose a new $Z'_C$ which is larger than the old $Z'_C$ and go to Step 2. Otherwise, choose a new $Z'_C$ which is less than the old $Z'_C$ and go to Step 2.

Step 4. Terminate the procedure when in the current $Z'_C$, $R \leq Y$ but at $Z'_C - 1$, $R > Y$. Given Y units, we can improve the center objective function to $Z'_C$ at best.

## APPENDIX E4

**ALGORITHM E4. Calibrating $Z'_C$ When Costs Are Associated with Arc Reductions**

Define C as the maximum cost of arc reductions that could be performed.

Step 1. Choose $Z'_C$. (We could choose $Z'_C = Z_C - \lfloor C/C_{ave} \rfloor$, where $C_{ave} = \frac{\sum c_{ij} l(i,j)}{\sum l(i,j)}$

Step 2. Do (P5.5) with the objective function changed as mentioned above, and find COST.

Step 3. If Y < COST, choose a new $Z'_C$ which is larger than the old $Z'_C$ and go to Step 2. Otherwise, choose a new $Z'_C$ which is smaller than the old $Z'_C$ and go to Step 2.

Step 4. Terminate the procedure when in the current $Z'_C$, COST $\leq$ C but at $Z'_C - 1$, COST > C. Given a budget of C, we can improve the center objective function to $Z'_C$ at best.

## APPENDIX E5

### A Third Approach to the Problem of Link Reduction on Network Arcs with Varying Costs

This section uses notations and concepts dicussed in Section 5.6.

Consider the following program:

minimize $Z_C$

subject to

$$x_{ij} \leq l(i,j) - L^* \qquad \text{for all arcs in the network}$$

$$\sum_{\substack{\text{all arcs} \\ \text{in network}}} c_{ij}x_{ij} = C$$

and for each node in the network:

$$h_{m_k}[d_1(X, m_k) - \sum_{\substack{x_{ij} \text{ in} \\ p_{m_k 1} \text{ path}}} x_{ij}] \leq Z_C$$

or

$$h_{m_k}[d_2(X, m_k) - \sum_{\substack{x_{ij} \text{ in} \\ p_{m_k 2} \text{ path}}} x_{ij}] \leq Z_C$$

or

$$\downarrow$$

$$h_{m_k}[d_{\overline{p_{m_k}}}(X, m_k) - \sum_{\substack{x_{ij} \text{ in} \\ p_{m_k p_{m_k}} \text{ path}}} x_{ij}] \leq Z_C$$

(This condition imposed on every node could be transformed to linear constraints by adding indicator variables, as done in the previous formulations.)

This mixed integer program could give us the strategy of where and how much to reduce in a network, with weighted nodes and different costs at each arc. We do not have to do the calibrating procedure. However, not predetermining $Z'_C$ requires us to list and consider all the nodes in the network, resulting in a much larger formulation.

## APPENDIX F1

**ALGORITHM F1. Determining the Improvement of $Z_C$ Due to the Addition of Arc $a_{ij}$ to a Weighted Spanning Tree**

This section uses notations and concepts discussed in Sections 6.1.1

Initialize: $\alpha = 1$, MAX = $m_\alpha$, U(MAX) = $h_{m_\alpha} d(X, m_\alpha)$

MAX is current farthest node from X, with weighted distance U(MAX).

(i)Test the path of $m_\alpha$ to find out if it is connected with any of $W_{z'}, W_{z'-1}, \ldots, W_0$. If it is connected, its distance from X has improved, and its place in M has to be reevaluated in the light of this new distance. To do this

(a) Let $p = 0$, $Q_0 = m_\alpha$

(b) Let $Q_{p+1} = A_{Q_p}$

If $Q_{p+1} = X$, $m_\alpha$ is not connected to the cycle, its distance from X has remained the same. Define $U(m_\alpha) = u_{m_\alpha}$

If $Q_{p+1} = W_k$, $k = 0, 1, \ldots, z'$, $m_\alpha$ is connected to the cycle, and

define the new distance $U(m_\alpha) = \min \{u_{m_\alpha}, h_{m_\alpha}[d(X, m_\alpha) - S_{W_k}]\}$

Let $p = p + 1$ and go to (b).

(ii) Did $m_{\alpha+1}$ distance form X change? (Determine by using part (i))

If it remained the same,

If $U(MAX) \geq u_{m_{\alpha+1}}$, stop.

Otherwise, MAX = $m_{\alpha+1}$, let $\alpha = \alpha + 1$, and go to (ii)

If it changed,

If $U(MAX) \geq U(m_{\alpha+1})$, stop.

$U(m_{\alpha+1}) > U(MAX)$, let MAX = $m_{\alpha+1}$

Let $\alpha = \alpha + 1$, and go to (ii)

The associated savings of arc $a_{ij}$ is $I = h_{m_1} d(X, m_1) - U(MAX)$.

## REFERENCES


Dionne, R. and M. Florian. 1979. Exact and Approximate Algorithms for Optimal Network Design. *Networks*. **9**. 37-59.

Handler, G. Y. and P. B. Mirchandani. 1979. *Location on Networks: Theory and Algorithms*. MIT Press. Cambridge, Mass.

Johnson, D. S., J. K. Lenstra, and A. H. G. Rinnooy Kan. 1978. The Complexity of the Network Design Problem. *Networks*. **8**. 279-285.

Larson, R. C. and A. R. Odoni. 1981. *Urban Operations Research*. Prentiss Hall. Englewood Cliffs, N. J.

Leblanc, L.J. 1975. An Algorithm for the Discrete Network Desigh Problem. *Transportation Science*. **9**. 183-199.

Magnanti, T. L. and R. T. Wong. 1984. Network Design and Transportation Planning: Models and Algorithms. *Transportation Science*. **18**. 1-55.

Scott, A.J. 1969. The Optimal Network Problem: Some Computational Procedures. *Transportation Research*. **3**. 201-210.