2000

Analysis of a network design problem

Shalu Wadhwa
Lehigh University

Wadhwa, Shalu

Analysis of a Network Design Problem

January 2001

Analysis of a Network Design Problem

by

Shalu Wadhwa

A Thesis
Presented to the Graduate and Research Committee
of Lehigh University
in Candidacy for the Degree of
Master of Science

in
Industrial and Manufacturing Systems Engineering

Lehigh University

December, 2000

This thesis is accepted and approved in partial fulfillment of the requirements of the Master of Science.

_____

Date

Thesis Advisor

Chairperson of the Industrial and
Manufacturing Systems Engineering
Department

Acknowledgments

I would like to thank Professor Joseph Hartman for his guidance throughout the development of this study. His expertise and encouragement was invaluable. I would also like to thank Professor Ted Ralphs, who helped me with my programming difficulties in this study, and Robert Barbieri, one of my co-students in the department, who introduced me to this problem indirectly, through a class presentation.

In addition, I would like to thank my depatment for providing me funding for the first year of my Master of Science studies.

This research was supported by National Science Foundation under grant numbers DMI-9713690, and DMI-9984891.

Table of Contents

List of Tables v

List of Figures vi

1 Introduction **2**

1.1 Network Design Problem (Cable and Trench Problem) 2

1.1.1 Minimum Spanning Tree Problem 2

1.1.2 Shortest Path Tree Problem 2

1.1.3 Capital Cost vs. Utilization Cost 3

1.2 Background 3

1.2.1 Savings Algorithm 4

1.3 Contributions 5

2 Problem Formulation **6**

2.1 Mixed Integer Program 6

2.2 Computational Complexity of the Network Design Problem 7

2.2.1 Fixed Charge Network Flow Problem (FCNFP) 8

2.3 Strengthening of Lower Bound 9

3 Savings Algorithm for Network Design Problem **11**

3.1 Example 12

3.2 Complexity of the Heuristic 15

3.2.1 Find the Minimum Spanning Tree for the given network using Prim's Algorithm 15

3.2.2 Calculation of Savings 16

3.3 A modification to the Savings Algorithm 17

3.3.1 Complexity of the Modification. 18

4 Computational Testing **19**

4.1 Performance Analysis 19

4.1.1 Solutions 19

4.1.2 Running Times 24

4.1.3 Summary 29

4.2 Case Study 30

5 Conclusions **32**

6 Appendix **33**

Bibliography **37**

Vita **40**

List of Tables

Table 4-1: Solutions and Running times for different methods 31

List of Figures

Figure 3-1: A given network 12
Figure 3-2: Minimum Spanning Tree 12
Figure 3-3: Try arc (1,2) 13
Figure 3-4: Try arc (2,3) 13
Figure 3-5: New Spanning Tree 14
Figure 3-6: Try arc (2,3) 14
Figure 3-7: Try arc (2,4) 15
Figure 4-1: Solutions for problem numbers 1-35 20
Figure 4-2: Solutions for problem numbers 36-70 21
Figure 4-3: Solutions for problem numbers 71-105 22
Figure 4-4: Solutions for problem numbers 106-127 23
Figure 4-5: Running Times for problem numbers 1-35 25
Figure 4-6: Running Times for problem numbers 36-70 26
Figure 4-7: Running Times for problem numbers 71-105 27
Figure 4-8: Running Times for problem numbers 106-127 28
Figure 4-9: Average Running Times (seconds) 29
Figure 4-10: Solution Gaps 30
Figure 6-1: Network obtained from Savings Algorithm 34
Figure 6-2: Network obtained by solving Integer Program 34
Figure 6-3: Network obtained from Savings Algorithm 35
Figure 6-4: Network obtained by solving Integer Program 36

Abstract

In this study, we examine different methods of solving a Network Design Problem. The Network Design Problem is a combination of the minimum spanning tree and the shortest path problem, which represents a trade-off between utilization costs and capital costs for network construction. A larger network, (the shortest path tree) may cost more to build but may reduce utilization costs by including more attractive origin-destination paths. Conversely, a smaller network, (minimum spanning tree) may increase the utilization costs. This problem has been shown to be a NP complete problem.We develop and test a heuristic, which is a modification of the Savings algorithm, given by Clarke and Wright in 1964, for solving a vehicle routing problem. Using the proposed heuristic, we can get good solutions to the problem, fairly quickly. At the same time, we formulate the problem as an integer program and test valid inequalities to improve its solution time.

Keywords: Savings Algorithm, Network Design Problem, Cable and Trench Problem

Chapter 1

Introduction

1.1 Network Design Problem (Cable and Trench Problem)

The Network Design Problem considered here is a combination of the shortest path tree problem and the minimum spanning tree problem.

1.1.1 Minimum Spanning Tree Problem

A spanning tree is a tree (that is, a connected acyclic graph) that spans (touches) all the nodes of an undirected graph. The cost of a spanning tree is the sum of the costs (or lengths) of its arcs. The minimum spanning tree is a spanning tree of minimum cost. Minimum spanning tree problem is a problem solvable in polynomial time. Given an undirected graph $G = (N, A)$ with $n = |N|$nodes and $m = |A|$ arcs and with a *length* or *cost* c_{ij} associated with each arc $(i, j) \in A$, we find a *minimum spanning tree*, that has the smallest total cost (or length) of its constituent arcs, measured as the sum of costs of the arcs in the spanning tree. Some of the greedy algorithms used for solving it are Kruskal's Algorithm,[10], Prim's Algorithm,[11] and Sollin's Algorithm(1961) [12].

1.1.2 Shortest Path Tree Problem

In the shortest path problem, we wish to determine a shortest path from the source node to all other $(n - 1)$ nodes. We can always find a directed out-tree rooted from the source with the property that the unique path from the source to any node is a shortest path to that node. Such a tree is called a shortest path tree. Since, it touches upon all the nodes in the graph, it is a spanning tree. For any given graph, $G = (N, A)$ with a specified vertex $n_0 \in N$, the shortest path tree problem is a problem solvable in polynomial time. There are two algorithmic approaches to solving shortest path problem:

label setting and *label correcting*.Label setting algorithms designate one label as permanent (optimal) at each iteration. Some of the label setting algorithms are Dijkstra's algorithm, [1] and Dial's algorithm, [2] which have various implementations, [3], [4], [5]. Label correcting algorithms consider all labels as temporary until the final step, when they all become permanent. First label correcting algorithm was given by Ford, [6]. Subsequently, many modifications and implementations of the generic label correcting algorithm were given by various researchers, [7], [8], [9].

1.1.3 Capital Cost vs. Utilization Cost

In the Network Design Problem (Cable and Trench Problem), a trade-off between utilization costs and capital costs for network construction is examined. A larger network, (the shortest path tree) may cost more to build but may reduce utilization costs by including more attractive origin-destination paths. Conversely, a smaller network, (minimum spanning tree) may increase the utilization costs. It is called a cable and trench problem due to its physical application in the following problem: A university needs to connect several buildings on campus to one building which holds the mainframe for the computer system. Each building needs to be connected directly to the main building through an underground cable. Before, any cable is laid, a trench must be dug; however, once a trench is dug, any number of cables may be laid in the trench. The cable cost is the product of the total length of cable required and per unit cable cost. The solution to the cable problem alone, is the shortest path network with the main building as the vertex. The trench cost is the product of the total length of trench needed to be dug and per unit trench cost. The solution to the trench problem alone is the minimum spanning tree. The solution to our problem is a trade-off between these two problems.

This problem can also be thought of as a trade-off between capital and operating costs. For example, in urban planning, when we need to build roads from a hospital's location or a firestation's location, to various other locations in town, we are tempted to build them along the minimum spanning tree to save on building cost, which is the capital cost. But, once the hospital or the firestation are operating, we require a shortest path tree, so that we are able to reach any location in minimum possible time, to minimize the life or property damages, caused by any delays. This would be the utilization cost. Therefore, before actually building the roads, we need to have a trade-off between these two costs to establish where the roads would actually be built.

1.2 Background

The problem of finding trade-offs between shortest path trees and minimum spanning trees has been looked into by several researchers. Khuller et al., [14], gives an algorithm which finds a spanning tree in

which the distance between any vertex and the root of the shortest path tree is atmost $1+\sqrt{2}d$ times the shortest path distance, and yet the total weight of the tree is atmost $1+\frac{\sqrt{2}}{d}$ times the weight of a minimum spanning tree where $d>0$ is given. Booth and Westbrook, [13], developed an algorithm that can be used to perform edge cost sensitivity analysis, find replacement edges, and verify their minimality for both the minimum spanning tree and shortest path tree in a planar graph. The algorithm uses the properties of a planar embedding, combined with a heap-ordered queue data structure. Saltzman, [17], provides a counterexample to the question "Is a Minimum Spanning Tree also a Shortest Path Tree for a determined vertex in a positive weighted connected graph?" Eppstein, [18], proved that finding the minimum spanning tree that minimizes the path length between a particular set of vertices is NP-complete. Vasko et al,[16], developed a heuristic for the cable and trench problem, in which they start with the shortest paths spanning tree, and perform a one-opt neighborhood search with backtracking until the minimum spanning tree is created. This method produces a collection of spanning trees such that each tree is optimal, or near optimal, for a range of per unit trench and cable costs. The collection of spanning trees covers all positive values of per unit costs. The backtracking allows two consecutive spanning trees in the solution to differ by more than one arc. Calvete and Mateo, [15], deal with network flow problems with multiple objectives. They rank the objective functions according to their importance and then perform a lexicographical optimization of the objectives by assigning preemptive priorities to them. The proposed approach enables them to maintain the network structure of the problem and hence to develop network-based algorithms which usually are proved to be more efficient than general ones.

1.2.1 Savings Algorithm

The Savings Algorithm is a greedy algorithm developed by Clarke and Wright, [19], to get an optimum or near optimum routing of a fleet of trucks of varying capacities from a central depot to a number of delivery points. It is described as follows:

Suppose, that there is a single depot from which all vehicles depart and return. Customers' locations and needs are known. Identify the depot as location 0 and the customers as location $1,2,...,n$. We assume that there are known costs of traveling from each location to another.

C_{0j} =Cost of making one trip from the depot to customer j.

C_{ij} = Cost of making a trip from customer location i to customer location j.

$C_{ij}=C_{ji}$ for all $1\leq i,j\leq n$. (not necessary)

The method proceeds as follows: Suppose initially that there is a separate vehicle assigned to each customer location. Then the initial solution consists of n separate routes from the depot to each customer location and back. It follows that the total cost of all round trips for the initial solution is $2\sum_{j}C_{0j}$. Now, suppose that we link customers i and j. That is, we go from the depot to i to j and back to the depot

again. In doing so, we would save one trip between the depot and location i and one trip between the depot and location j. However, there would be an added cost of C_{ij} for the trip from i to j (or vice versa). Hence, the savings realized by linking i and j is

$S_{ij} = C_{0i} + C_{0j} - C_{ij}$.

The method is to compute S_{ij} for all possible pairs of customer locations i and j, and then rank the S_{ij} in decreasing order. Then consider each of the links in descending order of savings and include (i, j) in a route if it does not violate feasibility constraints. If including the current link violates feasibility, go to the next link on the list and consider including that on a single route. Continue in this manner until the list is exhausted. Whenever link (i, j) is included on a route, the cost savings is S_{ij}. The total number of calculations of S_{ij} required is $\frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$. (When C_{ij} and C_{ji} are not equal, twice as many savings terms must be computed.)

As long as the constraints are not too complex, the method can be easily implemented on a computer. However, it is only a heuristic, and does not necessarily produce an optimal routing. The problem is that forcing the choice of a highly ranked link may preclude other links that might have slightly lower savings but might be better choices in a global sense by allowing other links to be chosen downstream.

1.3 Contributions

The main contribution of this work is to find a good heuristic for a network design problem. Previous studies in this area have focused on more general situations, and hence make more compromises on the optimality of the solutions. This heuristic utilizes the given information about the network completely, and comes up with an optimal or near optimal solution. A slight modification of this heuristic has also been presented which makes it faster, at the cost of the goodness of the solutions.

The savings algorithm given by Clarke and Wright,[19], finds the savings, by introducing various arcs, ranks them, and then greedily selects the feasible arcs. All this is performed in a single iteration. Our approach is to do it in several iterations. We find the savings on the arcs, use the arc with maximum savings, and then start all over again to find the savings on all the arcs again, so as to get an optimal or near-optimal solution. In the modification, some of the arcs which cost less than a certain percentage of maximum edge cost in the graph are marked permanent at the very start of the problem, thus, decreasing the running time of the heuristic.

Chapter 2

Problem Formulation

Given positive parameters T and C and a connected graph, $G = (N, A)$, with root, $n_0 \in N$, $n = |N|$, and positive arc lengths, minimize $\alpha T + \beta C$, where α is the total length of the spanning tree and β is the total path length from the root n_0, to all other nodes in N.

The parameters T and C can be thought of , in regards to the cable and trench example, as the per unit distance trench cost and per unit distance cable cost, respectively. Notice that with a very large T as compared to C, the solution will result in a minimum spanning tree. Analogously, with a very large C as compared to T, the solution will result in shortest paths spanning tree. It is interesting that for other values of T and C, the solution tree may be neither the minimum spanning tree, nor the shortest paths spanning tree.

2.1 Mixed Integer Program

The following are the variables used in a zero-one mixed integer programming formulation of the Network Design Problem.

Variables:

x_{ij} The flow value from node i to node j.

y_{ij} =1, if there is an edge from node i to node j in the solution, else 0.

Parameters:

n total number of nodes in the graph.

d_{ij} The length/cost of arc (i, j) in the graph.

C per unit cable cost.

T per unit trench cost.

Formulation(P1):

$$\textit{Minimize}$$

$$C\sum_i\sum_j d_{ij}x_{ij} + T\sum_i\sum_j d_{ij}y_{ij}$$

subject to

$$\sum_j x_{0j} - \sum_j x_{j0} = n-1 \tag{1}$$

$$\sum_j x_{ij} - \sum_j x_{ji} = -1 \qquad i = 1,...,n-1 \tag{2}$$

$$x_{ij} + x_{ji} - (n-1)y_{ij} \leq 0 \qquad i = 0,...,n-2 \quad j = i+1,...,n-1 \tag{3}$$

$$x_{ij} \in \{0,1,2,...,n-1\} \qquad i = 0,...,n-1 \quad j = 0,...,n-1 \tag{4}$$

$$y_{ij} \in \{0,1\} \qquad i = 0,...,n-2 \quad j = i+1,...,n-1 \tag{5}$$

The objective function minimizes overall cost, which is a linear combination of minimum spanning tree cost and shortest path tree cost. The constraints model a graph which would be a spanning tree. There is a flow of value $(n-1)$ from node n_0, given by constraint (1). One unit of flow is absorbed by each node in the graph, as given by constraint (2). Therefore, there would be $(n-1)$ edges in the graph, and every node is touched upon, so we would get a spanning tree. y_{ij} is 1, only if an edge connects i and j. If no edge connects i and j, there cannot be any flow from i to j. This restriction is imposed by constraint (3). Constraint (4), imposes that the flow should always be positive.

2.2 Computational Complexity of the Network Design Problem

This problem can be shown to be a NP-complete problem.

Let $s_{ij} = Td_{ij}$, and $c_{ij} = Cd_{ij}$. x is a matrix of all x_{ij} values. We express constraints (1), and (2) together in matrix form, as constraint (1), as they all are linear constraints on x_{ij}.

Then the programming formulation is:

Formulation (P2):

$$\textit{Minimize}$$

$$\sum_i\sum_j c_{ij}x_{ij} + \sum_i\sum_j s_{ij}y_{ij}$$

subject to

$$Ax = b \tag{6}$$

$$x_{ij} + x_{ji} - (n-1)y_{ij} \leq 0 \quad i = 0,...,n-2 \quad j = i+1,...,n-1 \tag{7}$$

$$x_{ij} \in \{0,1,2,...,n-1\} \quad i = 0,...,n-1 \quad j = 0,...,n-1 \tag{8}$$

$$y_{ij} \in \{0,1\} \quad i = 0,...,n-2 \quad j = i+1,...,n-1 \tag{9}$$

This is a fixed charge problem. A fixed charge problem is a NP-complete problem,[40], therefore since we are able to represent a Network Design Problem as a fixed charge network flow problem, Network Design Problem is also a NP-complete problem. Constraint (7) is a *forcing* constraint, which prohibits flow through inactive arcs. This is also the constraint, which actually makes this problem a MIP problem, so we cannot solve it as a linear program, thus, making the problem a hard problem. This is because, if we try to solve this problem by linear programming, for $x_{ij} + x_{ji} < (n-1)$, which is the case for all edges, (with the exception of the edge connected to the source node, if it is the only edge touching upon the source node), we will get a y_{ij}, that is fractional, which is an infeasible solution for our problem. If we somehow knew the exact value of $x_{ij} + x_{ji}$ for each (i,j), we could put that as the coefficient of y_{ij}, instead of $(n-1)$, and solve the problem as a linear programming problem. But, this information is not known a priori.

2.2.1 Fixed Charge Network Flow Problem (FCNFP)

A wide variety of applied problems can be effectively modeled as uncapacitated FCNFP. These problems include lot-sizing (planning production setups) problems, designing single commodity utility and logistics networks, and planning for warehouse and distribution systems. It is a special subclass of the Minimum Concave-cost Network Flow Problem, therefore, it has the same characteristics as Minimum Concave-cost Network Flow Problem. The objective function in FCNFP is discontinuous at the origin. Hence, most solution approaches have utilized branch and bound techniques to find an exact solution by transforming the fixed charge problem into an equivalent 0-1 mixed integer programming problem, [21]. Gray, [22], has attempted to provide an exact solution to this problem by decomposing it into a master integer program and a series of transportation sub-programs. Another classical exact solution approach is the vertex ranking procedure proposed by Murthy, [23], which exploits the property that a global solution occurs at the vertex of the feasible domain , [24]. Palekar et al., [25] and Steinberg, [26], on the other hand, attempt to provide exact algorithms based on branch-and-bound methods. Sandrock, [27], presents a simple algorithm for the solution of small, fixed-charge problems. Due to the requirement of massive computational efforts implementing these approaches, it is still not practical to solve general large-scale problems.

Because of the complexity involved in examining many local minima, early attempts to solve this problem consisted of finding an approximate solution. The earliest one was proposed by Balinski,[28], who observed that there exists an optimal solution to the relaxed version of fixed charge transportation problem formed by ignoring the integer restriction on y_{ij} variables. Other well-known heuristic approaches are the ones by Cooper and Drebes [29], Denzler [30], Diaby [31], and Kuhn and Baumol [32]. The generic model of FCNFP has applications for problems of distribution, transportation, communication, and routing, [33].

Adlakha and Kowalski, [34], developed a quick sufficient condition to identify candidate markets and supply points to ship more for less in fixed-charge transportation problem. The more-for-less paradox occurs when it is possible to ship more total goods for less (or equal) total cost, while shipping the same amount or more from each origin and to each destination and keeping all the shipping costs nonnegative. Stallaet ,[35] discussed a simple procedure to derive network inequalities for capacitated fixed charge network problems. Properties of the fractional extreme points of the LP relaxation are used to construct a class of inequalities and to construct a computational heuristic procedure for generating violated cutting planes. A new concept of the dynamic slope scaling procedure was proposed by Kim and Pardalos [36], to solve the general capacitated (or uncapacitated) FCNFP and some computational results on a wide range of test problems were reported. Sun, et al. [37], developed a tabu search heuristic procedure for the fixed charge transportation problem. FCNFP has also been extended to other situations, such as multi-commodity flow FCNFP [38], and teacher assignment problem [39].

2.3 Strengthening of Lower Bound

We can strengthen the mathematical formulation (P1), by adding the following constraint,[16]:

$$\sum_i \sum_j y_{ij} = n-1 \qquad i = 0, ..., n-2 \quad j = i+1, ..., n-1. \qquad (10)$$

It should be clear that (10) is valid, as there are $n-1$ arcs in any feasible solution, (as it is a spanning tree). This cut, strengthens the formulation by reducing the solution space that needs to be searched.

Another cut that strengthens the formulation is:

$$\sum_{j,i<j} y_{ij} + \sum_{j,j<i} y_{ji} \geq 1 \quad i = 0, ..., n-1. \quad (11)$$

Again it should be clear that this cut is feasible as each node in the network must be supported by atleast one arc. We have tested both of these cuts in our example problems, and they seem to give good reductions in the CPLEX running times, from the original formulation.

Another cut that can be used is:

$$x_{ij} + x_{ji} - y_{ij} \geqslant 0 \qquad i = 0, ..., n-2 \quad j = i+1, ..., n-1. \qquad (12)$$

This cut is feasible because there can be flow along an arc, only if that arc exists in the tree.

If we define S as a subset of the graph G, containing s nodes, then for every subset S,

$$\sum_i \sum_j y_{ij} \leq s - 1 \quad i, j \in S, \text{ and } i < j. \quad (13)$$

This would amount to an exponential number of constraints in the formulation, so we need to select a subset, or generate them in a reformulation procedure. For this, we solve the problem as a linear program, with no integer constraints. Now, y_{ij} variables, can be fractional, which will lead to more number of non-zero y_{ij} variables than $n-1$. This gives us cycles in the graph, as a non-zero y_{ij} variable means an arc exists from i to j. For every cycle C in the graph, containing c nodes, we introduce the following:

$$\sum_i \sum_j y_{ij} \leq c - 1 \quad i, j \in C, \text{ and } i < j. \quad (14)$$

Future research will examine the computational benefits of including (12), (13), and (14) in P1.

Chapter 3

Savings Algorithm for Network Design Problem

The Savings Algorithm can be used to find "good" solutions to the Network Design Problem (cable and trench problem). The heuristic starts with finding the minimum spanning tree for a given network. This can be accomplished efficiently, using any of the greedy algorithms. Here, we use Prim's algorithm. The spanning tree obtained has $(n-1)$ edges, and since there can be at maximum $\frac{n(n-1)}{2}$ edges, in a graph of n nodes, we have $\frac{(n-1)(n-2)}{2}$ edges to explore.

Introducing a new edge in a spanning tree creates a cycle, and to make the graph acyclic again, we need to remove an edge from that cycle. So, we start by considering an edge (i,j) which is not there in the spanning tree. If we put (i,j) into the graph we get a set of edges, that form a cycle, and we can remove any one member of this set, to make the graph acyclic again. For every pair of edges that we can put into the graph, and remove from the graph, there is a difference in cost, and possibly savings on the original graph. For every edge being put in, find the edge to be removed which gives the maximum savings. Once we have computed the maximum savings for each edge that can be put in, rank them, and pick up the edge, which gives us overall, maximum positive savings. Put this edge into the graph, removing the edge that was giving us maximum savings, on being replaced. Now, repeat the whole process on this graph, till we reach a point, when we get no savings from replacing any arc. That is the solution to our problem.

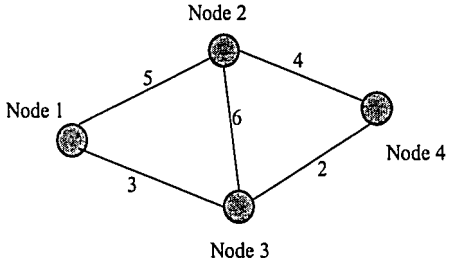

Figure 3-1: A given network

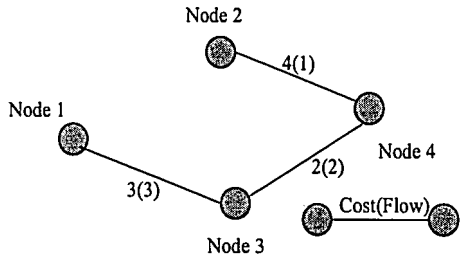

Figure 3-2: Minimum Spanning Tree

3.1 Example

We have a graph $G = (N, A)$, (see Figure 3-1), where $N = \{1, 2, 3, 4\}$, $A = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4)\}$, per unit trench cost $T = 11$, per unit cable cost $C = 4$. The costs of the arcs are 5,3,6,4, and 2 respectively.

STEP 1: Find the minimum spanning tree.

The minimum spanning tree for this graph is shown in Figure 3-2.

The cost of the minimum spanning tree is $(3*3+2*2+4*1)C+(3+2+4)T = 9T+17C = 167$.

STEP 2: Compute Savings.

(I) Now, if we introduce arc $(1, 2)$ in the graph, (see Figure 3-3) we get a cycle, in which other arcs are $(1, 3)$, $(2, 4)$ and $(3, 4)$.

(a) If we delete arc $(1, 3)$, the cost of the spanning tree obtained is $(5*3+4*2+2*1)C+ (5+4+2)T = 25C+11T = 221$.

(b) If we delete arc $(2, 4)$, the cost of the spanning tree obtained is $(5*1+3*2+2*1)C+ (5+3+2)T = 13C+10T = 162$.

(c) If we delete arc $(3, 4)$, the cost of the spanning tree obtained is $(5*2+4*1+3*1)C + (5+4+3)T =$

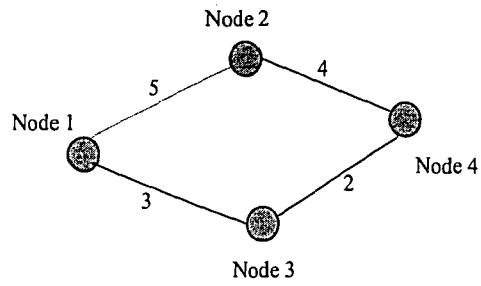

Figure 3-3: Try arc (1,2)

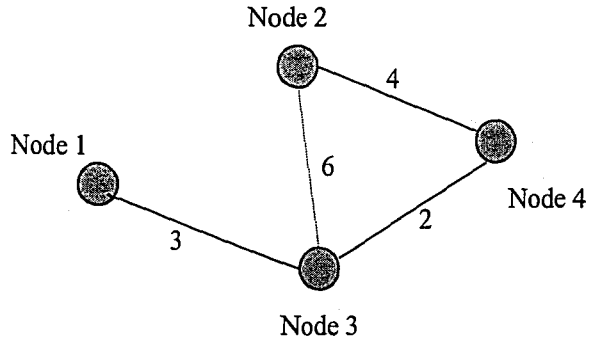

Figure 3-4: Try arc (2,3)

$17C+12T = 200$.

Since we get least cost, in above three choices for replacing arc $(2,4)$, that is the arc we would remove if we wanted to put in arc $(1,2)$. The saving if we put arc $(1,2)$ in the graph, and take away arc $(2,4)$ is 167-162=5.

(II) If we introduce arc $(2,3)$ in the graph, (see Figure 3-4) we get a cycle in which other arcs are $(2,4)$, and $(3,4)$.

(a) If we delete arc $(2,4)$, the cost of the spanning tree obtained is (3*3+6*1+2*1)C+(3+6+2)T= $17C+11T = 189$.

(b) If we delete arc $(3,4)$, the cost of the spanning tree obtained is (3*3+6*2+4*1)C+(3+6+4)T= $25C+13T = 243$.

Since we don't get an improvement over the minimum spanning tree solution, by putting in arc $(2,3)$, we have no savings from it, and it will not be considered.

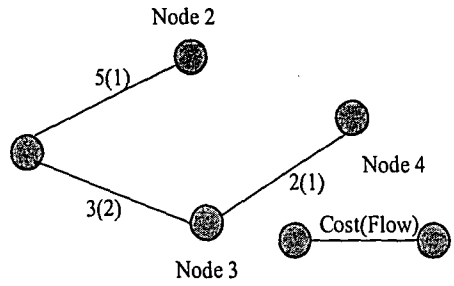

Figure 3-5: New Spanning Tree

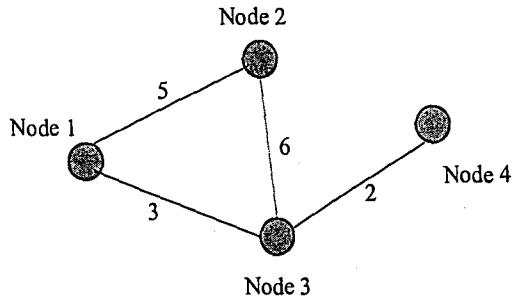

Figure 3-6: Try arc (2,3)

STEP 3: Order the savings and select the largest.

The maximum saving value that we got in the last step is 5 for replacing arc $(2,4)$ with $(1,2)$, the cost being 162. Therefore, we make this change and goto STEP 2. (see Figure 3-5).

(I) If we introduce arc $(2,3)$ in the graph, (see Figure 3-6) we get a cycle in which other arcs are, $(1,2)$ and $(1,3)$.

(a) If we delete arc $(1,2)$, the cost of the spanning tree obtained is (3*3+6*1+2*1)C+(3+6+2)T= $17C+11T = 189$.

(b) If we delete arc $(1,3)$, the cost of the spanning tree obtained is (5*3+6*2+2*1)C+(5+6+2)T= $29C+13T = 259$.

Since we don't get an improvement over the minimum spanning tree solution, by putting in arc $(2,3)$, we have no savings from it, and it will not be considered.

(II) If we introduce arc $(2,4)$ in the graph, (see Figure 3-7) we get a cycle in which other arcs are, $(1,2)$, $(1,3)$ and $(3,4)$.

(a) If we delete arc $(1,2)$, the cost of the spanning tree obtained is (3*3+2*2+4*1)C+(3+2+4)T= $17C+9T= 167$.

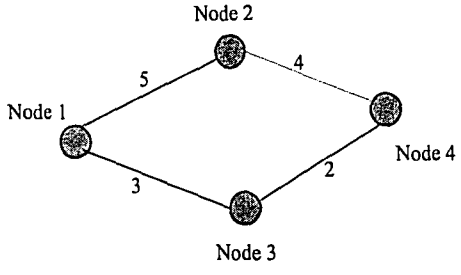

Figure 3-7: Try arc (2,4)

(b) If we delete arc $(1,3)$, the cost of the spanning tree obtained is (5*3+4*2+2*1)C+(5+4+2)T= 25C+11T= 221.

(c) If we delete arc $(3,4)$, the cost of the spanning tree obtained is ((5*2+4*1+3*1)C+(5+4+3)T= 17C+12T= 200.

Since we don't get an improvement over the minimum spanning tree solution, by putting in arc $(2,3)$, we have no savings from it, and it will not be considered.

STEP 4: If we did not get any savings, STOP.

We don't get any savings from any exchange in the last step, so the heuristic ends here, and the solution is the graph with edges, $(1,2)$, $(1,3)$, and $(3,4)$, and cost=162.

3.2 Complexity of the Heuristic

The heuristic involves several steps:

3.2.1 Find the Minimum Spanning Tree for the given network using Prim's Algorithm, [11]:

This algorithm is a greedy algorithm that builds a spanning tree from scratch by fanning out from a single node and adding arcs one at a time. It maintains a tree spanning on a subset S of nodes and adds a nearest neighbor to S. The algorithm does so by identifying an arc (i,j) of minimum cost in the cut $[S,\overline{S}]$. It adds arc (i,j) to the tree, node j to S, and repeats this basic step, until S includes all the nodes. We use arrays to implement this algorithm which gives us a running time of $O(n^2)$.

3.2.2 Calculation of Savings

While no improvement is possible:

(I) Find the path from every node to every other node, for the present spanning tree. $O(n^4)$.

(1) Initially, our search node is node 0.

(2) Find all the nodes connected to the search node by a single arc (nodes which have not been touched upon before), and record the paths from the search node to these nodes. This takes $O(n)$, running time.

(3) If the search node is not 0, record the paths from node 0, to the nodes adjacent to the search node, as the sum of the path from node 0 to the search node, and from the search node to each of these adjacent nodes. This takes $O(n^2)$, running time as proved later.

(4) Recursively, perform steps (2) and (3), with new search nodes as the nodes found adjacent to our present search node. This way we would end up covering all the nodes in the network, and will have the paths from node 0 to all the other nodes in the network. Since there are n nodes in the graph, the total running time for step (2) is $O(n^2)$, and for step (3) is $O(n^3)$, giving an overall running time of $O(n^3)$.

(5) The path from node 0 to node i, is the reverse of path from node i to node 0. Add path from node i to node 0, to that from node 0 to node j, to get the path from node i to node j. Get the paths for all values of i and j. There are $\frac{n(n-1)}{2}$, possible (i, j) pairs in a graph, and each addition of paths has a running time of $O(n^2)$, therefore, this step requires a running time of $O(n^4)$. Since this step dominates all the other steps in terms of running times, it takes $O(n^4)$, running time to find the paths from every node to every other node.

An important function used in these steps is addition of paths. If we know the paths from node i to node j and from node j to node k, we can get the path from node i to node k. The steps involved in this kind of addition are:

(1) Move along the path from node k to node j. For every node m along the path:

(a) Check if m lies along the path from node i to node j. If it does, the path from node i to node k is all the arcs from node i to node m, along the path i to j, and all the arcs from node m to node k, along the path j to k. If it doesn't, continue moving along the path from node k to node j.

(2) If we have found no m which satisfies condition (a), the path from node i node k, includes all the arcs along the path from node i to node j, and all the arcs along the path from node j to node k.

Since, the maximum number of arcs along any path can be $(n-1)$, the worst case running time for the path addition function is $O(n^2)$.

(II) Number of arcs to be tested= $\frac{n(n-1)}{2} - (n-1) = \frac{(n-1)(n-2)}{2}$**, which is** $O(n^2)$.

(1) Track the cycle formed by an additional arc, (i,j)**.** The cycle formed on adding arc (i,j) consists of all the arcs along Path(i,j).

(2) Try deleting every other arc (m,n) **in the cycle.** In this step we need to estimate the cost of the network, when arc (i,j) has been deleted and arc (m,n) has been added.

(a) First, find the paths from every node to every other node for the new network. $O(n^4)$.

(b) Estimate the cost of the new network. This takes $O(n^2)$.

Since, the running time for (a) dominates that of (b), testing each (m,n) in the cycle takes a running time of $O(n^4)$. There can be a maximum of $(n-1)$ arcs in a cycle. Therefore, checking for each additional arc (i,j) is $O(n^5)$, and since we have $O(n^2)$, arcs to be tested, step (II) is $O(n^7)$.

(III) Find the best of all replacements.

In the previous step, the best cost estimate for each arc that can be added, is recorded. Also, the arc which needs to be deleted to get this cost is saved in the memory. In this step, a sorting operation is done on all these cost estimates, and possible savings over the previous network are obtained. Corresponding to the highest positive savings obtained, the switch of the arcs is performed. This takes a running time of $O(n^2)$. The algorithm is repeated again, with the present tree as the starting network. The dominant step is (II) which took $O(n^7)$, therefore the algorithm runs in $O(n^7)$ time.

3.3 A modification to the Savings Algorithm

A slight modification has been tested to make the heuristic run faster. At the very start, some of the edges which cost less than x% of the maximum edge cost in the graph, are labelled permanent. If some of these edges form any cycles, all the edges in the cycle are unlabeled. This telescoping reduces the processing time, as we need to check lesser number of edges now. We would expect that higher the x value, lesser the processing time, at the expense of better solutions. Also, if x value goes beyond a certain limit, we end up getting more number of cycles, as a result of which all the edges in those cycles are disqualified for permanency, and our additional work of making them permanent in the first place, goes in vain, and only adds to the processing time. Several values of x have been tested, and it turns out that the optimum value of x is somewhere around 2% for our test set.

3.3.1 Complexity of the Modification

The modification part runs as a labeling algorithm. All the arcs in the graph, which is $O(n^2)$ in number, are scanned one by one. If the cost of any arc is below $x\%$, it is included in the graph permanently, and the nodes connected by it, are given a common label. Four possibilities may occur when such an arc is located:

(a) Both the nodes have no labels.

In this case, both the nodes are given a new label.

(b) One of the nodes already has a label, while the other one does not.

In this case, the unlabeled node is given the same label as the already labeled one.

(c) Both the nodes are labeled and the labels are different.

In this case, one of the nodes is taken, and all the nodes in the graph having the same label as that node, have their labels changed to that of the other node. This is done in $O(n)$ running time.

(d) Both the nodes are labeled and the labels are same.

If such a case occurs, it means we will form a cycle in the network by including this arc, and all the arcs in this cycle have to be disqualified for permanency. This is done in $O(n)$ running time.

Therefore, the total running time of the modification part is $O(n^3)$.

Chapter 4

Computational Testing

Testing was done for various solution methods. For this, problem instances of 15 to 60 nodes were generated.

4.1 Performance Analysis

Testing has been done for the following solution methods:

(1) *Savings*: Savings algorithm without the modification.

(2) *SavingsII*: Savings algorithm with the modification, x value being 5% and 10%.

(3) *MIP*: Mixed Integer Program solution by CPLEX.

(4) *MIPwithcut*1: Mixed Integer Program solution by CPLEX with constraint (10).

(5) *MIPwithcuts*1&2: Mixed Integer Program solution by CPLEX with constraints (10) and (11).

Also, a problem has been solved to show how the results from savings algorithm, can be used to provide a good upper bound for the MIP, so that we get the optimal answer in time lesser than that taken by CPLEX, without the upper bound.

n: The problem number *nodes* : Number of nodes in the problem

4.1.1 Solutions

This testing showed that the Savings Algorithm performs very well, giving optimal and near optimal answers most of the time. For the modification, as the value of x increases, the frequency of getting non-optimal answers increases. But still, we are near optimal.

Solutions					
n	nodes	Savings	Savingsll-5%	Savingsll-10%	MIP
1	15	5748	5748	5748	5748
2	15	7572	7572	7572	7572
3	15	5559	5559	5559	5559
4	15	4688	4688	**5039**	4688
5	15	5842	5842	**6016**	5842
6	15	7596	7596	7596	7596
7	15	3317	3317	3317	3317
8	15	**6241**	**6241**	**6413**	6178
9	15	6080	6080	6080	6080
10	15	**5952**	**5952**	**5952**	5943
11	15	4804	4804	4804	4804
12	15	3990	3990	3990	3990
13	15	7084	7084	7084	7084
14	16	3945	3945	**4097**	3945
15	16	2009	2009	**2040**	2009
16	16	3402	3402	**3443**	3402
17	16	2940	**2960**	2940	2940
18	16	5538	5538	5538	5538
19	16	5605	5605	**5689**	5605
20	16	3598	3598	**3740**	3598
21	16	4582	4582	**4753**	4582
22	16	6279	6279	6279	6279
23	16	6719	6719	6719	6719
24	16	3239	3239	3239	3239
25	16	6169	6169	6169	6169
26	16	4190	4190	4190	4190
27	16	7152	7152	7152	7152
28	16	5656	5656	5656	5656
29	16	4929	4929	4929	4929
30	16	7145	7145	7145	7145
31	16	6105	6105	**6622**	6105
32	16	4126	4126	4126	4126
33	16	5993	5993	5993	5993
34	16	4716	4716	4716	4716
35	16	6726	6726	6726	6726

Figure 4-1: Solutions for problem numbers 1-35

n	nodes	Savings	SavingsII-5%	SavingsII-10%	MIP
36	16	3982	3982	3982	3982
37	16	2621	2621	2621	2621
38	16	6976	6976	6976	6976
39	16	5957	5957	5957	5957
40	16	2555	2555	**2916**	2555
41	17	5300	5300	5300	5300
42	17	4565	4565	4565	4565
43	17	3015	3015	3015	3015
44	17	5127	5127	5127	5127
45	17	6179	6179	6179	6179
46	17	5528	5528	**5533**	5528
47	17	3558	3558	**3581**	3558
48	17	1969	1969	**2049**	1969
49	17	5646	5646	5646	5646
50	17	4111	4111	4111	4111
51	17	3253	3253	3253	3253
52	17	7540	7540	**7681**	7540
53	17	5215	5215	5215	5215
54	18	4422	4422	**4646**	4422
55	18	5045	5045	5045	5045
56	18	5675	5675	5675	5675
57	18	5215	5215	5215	5215
58	18	6137	6137	**6185**	6137
59	18	4219	**4315**	**4302**	4219
60	18	5565	5565	5565	5565
61	18	7009	7009	**7408**	7009
62	18	4426	4426	4426	4426
63	18	6483	6483	6483	6483
64	18	4521	4521	4521	4521
65	18	5753	5753	5753	5753
66	18	6640	6640	**6717**	6640
67	18	**3614**	**3614**	**3614**	3612
68	18	5717	5717	5717	5717
69	19	4543	4543	**4588**	4543
70	19	3979	3979	**4147**	3979

Figure 4-2: Solutions for problem numbers 36-70

n	nodes	Savings	SavingsII-5%	SavingsII-10%	MIP
71	19	7484	7484	7484	7484
72	19	3453	**3590**	3453	3453
73	19	4106	**4154**	4106	4106
74	19	5794	5794	**5892**	5794
75	19	5821	5821	5821	5821
76	19	5275	5275	5275	5275
77	19	5208	**5280**	**5313**	5208
78	19	**6271**	**6271**	**6332**	6225
79	20	4051	4051	**4141**	4051
80	20	5485	5485	5485	5485
81	20	6311	6311	6311	6311
82	20	5927	**5944**	**6232**	5927
83	20	3763	3763	**4290**	3763
84	20	4916	4916	4916	4916
85	20	4921	4921	4921	4921
86	20	5053	5053	**6329**	5053
87	20	5248	**5373**	5248	5248
88	20	4066	**4979**	4153	4066
89	20	6343	6343	**6358**	6343
90	21	5904	5904	5904	5904
91	21	5034	5034	5034	5034
92	21	4974	4974	**6445**	4974
93	21	3856	3856	**5063**	3856
94	21	1628	1628	**1797**	1628
95	21	3859	3859	**4205**	3859
96	21	2880	2880	2880	2880
97	21	4710	4710	**5556**	4710
98	21	5662	5662	**5756**	5662
99	21	6876	6876	6876	6876
100	22	**6057**	**6057**	**6604**	6046
101	23	4060	4060	**4348**	4060
102	23	9852	9852	**9913**	9852
103	23	5068	5068	**5114**	5068
104	23	5488	5488	**5715**	5488
105	24	6440	6440	**6445**	6440

Figure 4-3: Solutions for problem numbers 71-105

n	nodes	Savings	SavingsII-5%	SavingsII-10%	MIP
106	24	5706	5706	**5712**	5706
107	24	4974	4974	**5273**	4974
108	24	5682	5682	**6185**	5682
109	24	5104	5104	5104	5104
110	24	4173	**4316**	**4316**	4173
111	24	5069	**5071**	5069	5069
112	25	7004	**7036**	7004	7004
113	26	6867	6867	**8919**	6867
114	26	6632	6632	6632	6632
115	26	5264	5264	5264	5264
116	29	3823	3823	3823	3823
117	30	3268	3268	**3333**	3268
118	30	5420	5420	5420	5420
119	31	4488	**4522**	**4763**	4488
120	32	4875	**5094**	**5189**	4875
121	33	3242	**3285**	**3619**	3242
122	34	**5158**	**5158**	**5705**	5147
123	34	5530	**5900**	**6046**	5530
124	37	7679	7679	7679	7679
125	48	6181	**6993**	**6930**	6181
126	51	6054	**6059**	**6167**	6054
127	59	5803	5803	**6260**	5803

Figure 4-4: Solutions for problem numbers 106-127

4.1.2 Running times

The MIP takes much more time than the Savings algorithm. The MIP with cuts performs better than the Savings Algorithm in terms of running time. But, by implementing the modification in the Savings algorithm with a suitable x, we are able to beat the MIP with cuts. The cost given by the Savings algorithm with modification, can be used as an upper bound for the MIP with cuts, which enables us to get the optimal answer in lesser time than that used by the MIP with cuts alone. This has been demonstrated using a problem, in the next section.

Running Times							
n	nodes	Savings	SavingsII-5%	SavingsII-10%	MIP	MIP with cut 1	MIP with cuts 1&2
1	15	1.48	0.25	0.368	2.31	2.09	0.55
2	15	2.14	0.351	0.401	2.53	0.77	0.55
3	15	2.14	0.431	0.33	2.14	0.77	0.82
4	15	2.75	0.474	0.046	1.21	0.88	0.49
5	15	2.25	0.447	0.218	2.53	0.77	0.6
6	15	2.19	0.426	0.412	1.16	0.66	0.66
7	15	1.48	0.192	0.189	1.32	0.6	0.44
8	15	2.85	0.447	0.381	1.97	0.83	0.54
9	15	4.23	0.83	0.809	4.89	1.48	0.71
10	15	1.43	0.233	0.216	5.44	1.04	0.93
11	15	2.14	0.323	0.333	1.82	0.71	0.82
12	15	0.71	0.121	0.085	1.15	0.55	0.44
13	15	3.52	0.587	0.597	4.77	1.76	0.88
14	16	1.76	0.288	0.266	1.42	0.83	0.55
15	16	2.58	0.484	0.126	1.15	0.88	0.55
16	16	2.63	0.596	0.734	4.45	1.27	1.1
17	16	1.75	0.297	0.36	1.65	0.72	0.55
18	16	3.41	0.666	0.53	1.1	0.71	0.55
19	16	1.81	0.366	0.378	2.75	0.82	1.05
20	16	1.76	0.32	0.452	1.37	0.82	0.66
21	16	3.52	0.723	0.508	2.14	1.87	21.47
22	16	1.71	0.241	0.346	1.2	0.77	0.55
23	16	1.76	0.342	0.336	1.7	0.83	0.66
24	16	0.83	0.136	0.136	0.98	0.77	0.5
25	16	2.53	0.477	0.475	8.35	1.04	0.71
26	16	2.74	0.628	0.52	5.06	0.82	0.77
27	16	1.7	0.352	0.314	3.35	1.21	0.66
28	16	4.23	0.666	0.705	6.43	1.37	1.05
29	16	1.7	0.253	0.351	0.88	0.66	0.43
30	16	1.7	0.258	0.326	1.37	1.65	0.5
31	16	4.5	0.932	0.455	9.67	1.26	0.99
32	16	0.88	0.122	0.17	1.7	1.87	1.37
33	16	2.58	0.429	0.516	2.8	0.93	0.65
34	16	1.76	0.369	0.328	1.54	0.66	0.5
35	16	1.71	0.354	0.319	4.73	0.87	0.61

Figure 4-5: Running Times for problem numbers 1-35

n	nodes	Savings	Savingsll-5%	Savingsll-10%	MIP	MIP with cut 1	MIP with cuts 1&2
36	16	1.82	0.407	0.369	1.7	0.71	0.55
37	16	0.88	0.147	0.137	1.32	0.71	0.55
38	16	1.87	0.393	0.369	4.67	0.77	0.55
39	16	2.47	0.389	0.426	1.49	0.82	0.6
40	16	0.82	0.135	0.262	1.38	0.82	0.61
41	17	4.4	0.961	0.843	3.13	0.82	0.61
42	17	4.39	1.003	0.837	2.41	0.88	0.72
43	17	2.14	0.282	0.366	2.25	0.77	0.66
44	17	2.03	0.386	0.346	10.38	0.99	0.76
45	17	3.29	0.797	0.557	3.52	1.43	0.82
46	17	3.24	0.663	0.348	4.83	1.93	1.43
47	17	3.96	0.652	0.279	3.08	1.48	1.21
48	17	3.02	0.468	0.35	1.04	1.48	0.93
49	17	2.2	0.501	0.463	2.53	0.93	0.71
50	17	1.04	0.197	0.21	2.26	1.15	1.1
51	17	3.46	0.624	0.711	1.82	0.82	0.6
52	17	4.01	0.859	0.687	71.68	1.43	1.6
53	17	2.31	0.599	0.422	2.25	0.77	0.61
54	18	4.94	0.933	0.819	3.68	1.54	0.99
55	18	3.73	0.71	0.947	4.39	1.43	0.82
56	18	6.31	1.08	1.58	6.31	2.37	1.43
57	18	1.21	0.245	0.235	2.91	1.54	0.99
58	18	4.07	0.9	0.622	7.25	1.37	1.09
59	18	3.9	0.552	0.706	3.51	1.76	0.93
60	18	1.32	0.308	0.278	1.31	0.88	0.94
61	18	3.9	1.007	0.871	2.8	0.99	0.77
62	18	2.58	0.384	0.693	2.2	0.88	0.93
63	18	6.43	1.498	1.204	27.41	1.7	2.14
64	18	4.01	1.05	1.001	3.95	1.81	1.26
65	18	5.16	1.157	1.3	7.25	2.91	4.01
66	18	2.74	0.642	0.232	2.14	1.32	0.71
67	18	3.9	0.826	0.446	2.74	1.32	0.77
68	18	3.9	0.823	0.853	7.03	0.88	0.72
69	19	6.64	1.47	1.667	3.57	2.09	1.15
70	19	3.13	0.703	1.118	4.39	1.71	1.32

Figure 4-6: Running Times for problem numbers 36-70

n	nodes	Savings	SavingsII-5%	SavingsII-10%	MIP	MIP with cut 1	MIP with cuts 1&2
71	19	6.04	1.505	1.538	8.02	1.48	1.1
72	19	4.62	0.571	1.274	3.3	3.07	1.7
73	19	5.93	1.271	1.427	3.3	1.15	0.88
74	19	4.61	1.141	0.906	3.3	0.99	0.71
75	19	1.49	0.387	0.39	17.36	1.1	0.82
76	19	4.78	1.218	1.432	4.01	1.32	0.88
77	19	7.19	0.676	2.3	6.1	1.87	1.38
78	19	9.5	2.731	1.975	37.57	3.62	2.25
79	20	3.85	0.789	0.896	2.64	1.04	0.77
80	20	6.97	1.488	2.158	15.99	2.63	3.02
81	20	5.49	1.457	1.501	18.89	2.09	4.01
82	20	9.61	2.251	1.542	7.58	1.59	2.41
83	20	1.76	0.262	0.39	2.19	2.69	2.47
84	20	3.74	1.191	1.189	3.85	3.4	1.65
85	20	7.63	1.796	1.785	6.26	1.43	2.14
86	20	6.92	1.552	2.044	10.76	2.04	1.37
87	20	9.12	2.321	2.799	4.12	4.23	1.05
88	20	5.33	1.255	1.348	6.37	2.19	2.8
89	20	7.64	2.536	2.293	4.88	1.87	1.65
90	21	4.67	1.218	1.326	5.71	4.12	2.25
91	21	4.45	1.526	1.43	4.72	1.54	1.48
92	21	6.81	1.232	1.799	7.36	2.14	1.05
93	21	4.66	1.537	1.165	7.42	2.53	1.76
94	21	4.06	1.063	1.194	1.82	1.15	0.83
95	21	6.98	1.203	2.52	6.65	1.15	1.16
96	21	6.81	2.012	2.406	5.33	1.64	1.59
97	21	14.06	4.4	1.309	12.97	2.3	1.27
98	21	6.59	2.086	1.939	7.41	2.14	5
99	21	9.07	2.626	2.74	8.95	3.13	1.76
100	22	4.61	1.273	1.068	8.84	4.89	3.95
101	23	5.38	2.114	3.278	5.82	4.18	3.35
102	23	14.99	5.021	5.888	23.24	3.24	8.4
103	23	2.25	0.543	1.038	10.54	4.56	4.67
104	23	10.05	3.147	4.66	6.71	3.46	3.13
105	24	15.21	4.184	5.18	20.33	3.46	3.57

Figure 4-7: Running Times for problem numbers 71-105

n	nodes	Savings	SavingsII-5%	SavingsII-10%	MIP	MIP with cut 1	MIP with cuts 1&2
106	24	9.66	3.181	3.84	32.35	1.49	1.37
107	24	6.37	2.516	3.161	8.62	1.87	3.57
108	24	5.71	1.802	1.623	3.79	3.3	4.39
109	24	5.82	2.075	2.316	8.57	2.58	1.27
110	24	5.66	1.988	2.369	14.61	4.17	3.85
111	24	8.51	1.331	4.253	11.04	1.05	1.26
112	25	23.18	7.002	10.893	120.73	30.32	21.31
113	26	11.75	3.833	6.609	80.41	4.78	5.05
114	26	8.51	2.729	4.354	12.09	1.59	1.97
115	26	27.96	8.396	16.518	556.23	3.79	2.75
116	29	18.35	8.724	10.629	30.81	6.76	5.43
117	30	24.72	12.831	15.322	14.28	5.71	5.44
118	30	37.24	17.129	22.411	116.11	5.05	5.66
119	31	33.83	10.403	21.38	42.46	2.42	2.42
120	32	41.08	13.396	47.289	131.27	6.15	3.3
121	33	53.5	32.502	35.634	98.32	3.79	8.02
122	34	48.22	24.396	30.578	24667	53.34	48.77
123	34	71.84	20.411	64.054	389.37	7.58	6.37
124	37	127.54	63.957	73.75	2125.3	12.3	7.19
125	48	765.39	657.75	952.004	3days+	1424.77	611.76
126	51	1094.01	782.827	957.335	>>10567	>10567.47	1540.38
127	59	1462.28	1024.28	1962.09	3days+	573.91	1038.21

Figure 4-8: Running Times for problem numbers 106-127

Average Running Times (seconds)						
nodes	Savings	SavingsII-5%	SavingsII-10%	MIP	MIP with cut 1	MIP with cuts 1&2
15	2.255	0.393	0.337	2.557	0.993	0.648
16	2.126	0.399	0.378	2.828	0.980	1.455
17	3.038	0.615	0.494	8.552	1.145	0.905
18	3.873	0.808	0.786	5.659	1.513	1.233
19	5.393	1.167	1.403	9.092	1.840	1.219
20	6.187	1.536	1.631	7.594	2.291	2.122
21	6.816	1.890	1.783	6.834	2.184	1.815
22	4.610	1.273	1.068	8.840	4.890	3.950
23	8.168	2.706	3.716	11.580	3.860	4.888
24	8.134	2.440	3.249	14.190	2.560	2.754
25	23.180	7.002	10.893	120.730	30.320	21.310
26	16.073	4.986	9.160	216.243	3.387	3.257
29	18.350	8.724	10.629	30.810	6.760	5.430
30	30.980	14.980	18.867	65.195	5.380	5.550
31	33.830	10.403	21.380	42.460	2.420	2.420
32	41.080	13.396	47.289	131.270	6.150	3.300
33	53.500	32.502	35.634	98.320	3.790	8.020
34	60.030	22.404	47.316	12528.085	30.460	27.570
37	127.540	63.957	73.750	2125.290	12.300	7.190
48	765.390	657.750	952.004	3days+	1424.770	611.760
51	1094.010	782.827	957.335	>>10567.47	>10567.47	1540.380
59	1462.280	1024.280	1962.090	3days+	573.910	1038.210

Figure 4-9:

4.1.3 Summary

In this subsection, we endeavour to summarize the tables in the previous subsections.

The average times taken by Savings, SavingsII-5%, SavingsII-10%, MIP, MIP with cut 1, and MIP with cuts 1&2, for different problem sizes are summarized in the table in Figure 4-9.

Solution Gaps									
nodes	Savings			SavingsII-5%			SavingsII-10%		
	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.
15	0.000%	0.010%	0.001%	0.000%	0.010%	0.001%	0.000%	0.075%	0.011%
16	0.000%	0.000%	0.000%	0.000%	0.007%	0.000%	0.000%	0.140%	0.020%
17	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.040%	0.005%
18	0.000%	0.001%	0.000%	0.000%	0.023%	0.002%	0.000%	0.057%	0.010%
19	0.000%	0.007%	0.001%	0.000%	0.040%	0.007%	0.000%	0.042%	0.010%
20	0.000%	0.000%	0.000%	0.000%	0.225%	0.023%	0.000%	0.253%	0.045%
21	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.313%	0.100%
22	0.002%	0.002%	0.002%	0.002%	0.002%	0.002%	0.092%	0.092%	0.092%
23	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.071%	0.032%
24	0.000%	0.000%	0.000%	0.000%	0.034%	0.005%	0.000%	0.089%	0.026%
25	0.000%	0.000%	0.000%	0.005%	0.005%	0.005%	0.000%	0.000%	0.000%
26	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.299%	0.100%
29	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
30	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.020%	0.010%
31	0.000%	0.000%	0.000%	0.008%	0.008%	0.008%	0.061%	0.061%	0.061%
32	0.000%	0.000%	0.000%	0.045%	0.045%	0.045%	0.064%	0.064%	0.064%
33	0.000%	0.000%	0.000%	0.013%	0.013%	0.013%	0.116%	0.116%	0.116%
34	0.000%	0.002%	0.001%	0.002%	0.067%	0.035%	0.093%	0.108%	0.101%
37	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
48	0.000%	0.000%	0.000%	0.131%	0.131%	0.131%	0.121%	0.121%	0.121%
51	0.000%	0.000%	0.000%	0.001%	0.001%	0.001%	0.019%	0.019%	0.019%
59	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.079%	0.079%	0.079%

Figure 4-10:

The gaps of the solutions given by Savings, SavingsII-5%, and SavingsII-10% from the optimal solution are summarized in the table in Figure 4-10.

4.2 Case Study

For one of the problems tested above, we have shown that using the solution given by the Savings algorithm with modification, as an upper bound for the MIP with cuts, we get the optimal answer in lesser time than that required by the MIP with cuts alone.

For problem number 126, with 51 nodes, the solutions and times taken by savings algorithm with modification have been obtained, for different values of x.

Algorithm	Solution	Time(seconds)
MIP	6054	>>10567.470
MIP with cut 1	6054	>10567.470
MIP with cuts1&2	6054	1540.380
SavingsII-0%	6054	1094.010
SavingsII-1%	6054	765.948
SavingsII-2%	6054	499.498
SavingsII-3%	**6067**	554.483
SavingsII-5%	**6059**	783.762
SavingsII-10%	**6167**	967.563
SavingsII-15%	**7273**	1364.920
SavingsII-25%	**7752**	1356.710
SavingsII-50%	**9605**	1333.900

Table 4-1: Solutions and Running times for different methods

Assuming that a suitable value of x is 2%, we use the solution given by SavingsII-2%, as an upper bound to the MIP with cuts, and solve it in 982.13 seconds. Since the time taken by SavingsII-2% is 499.498, we are able to obtain the optimal solution in a total time of $982.13 + 499.498 = 1481.628$ seconds, which beats the time taken by MIP with cuts, that is, 1540.38 seconds.

Chapter 5

Conclusions

In this work we have targeted a Network Design Problem (Cable and Trench Problem), which involves a trade-off between utilization costs and capital costs for network construction. A larger network,(the shortest path tree) may cost more to build but may reduce utilization costs by including more attractive origin-destination paths. Conversely, a smaller network, (minimum spanning tree) may increase the utilization costs. A heuristic has been provided which gives us optimal or near-optimal solutions. This heuristic is an adaptation of the Savings algorithm given by Clarke and Wright in 1964, for solving a vehicle routing problem. The heuristic provides us good solutions which can be used as upper bounds for branch and bound methods, giving us the optimal solutions in lesser times than that given by branch and bound without the upper bounds.

For future work on this problem, it will be interesting to look at the linear programming formulation of this problem, and come up with more cuts to improve the lower bound. We tried two cuts on our problems and got tremendous reductions in the running times of the branch and bound done by CPLEX. Also, the running time of the algorithm could be improved by using different data structures.

Chapter 6

Appendix

Example1(with near optimal answer)

Results Network obtained by Savings Algorithm (see Figure 6-1):

For Trench cost T=11, Cable cost C=4,

Total Network Cost= 262T+743C=5854.

Network obtained by CPLEX (see Figure 6-2):

Total Network Cost=279T+696C=5853.

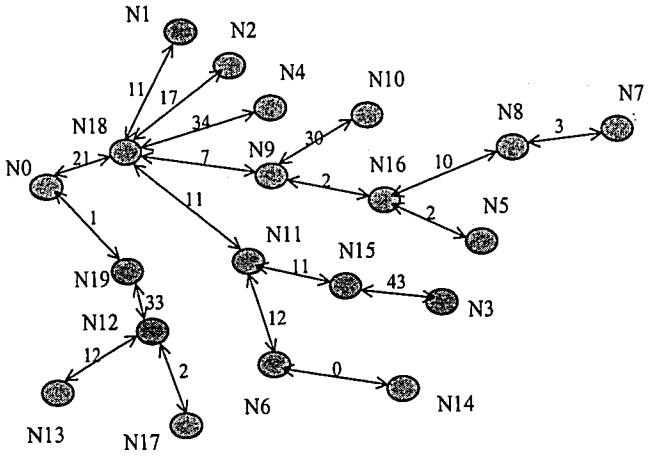

Figure 6-1: Network obtained from Savings Algorithm

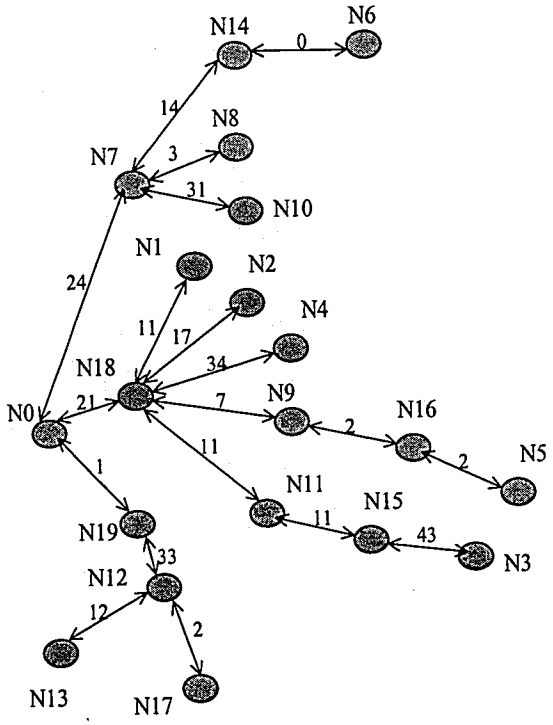

Figure 6-2: Network obtained by solving Integer Program

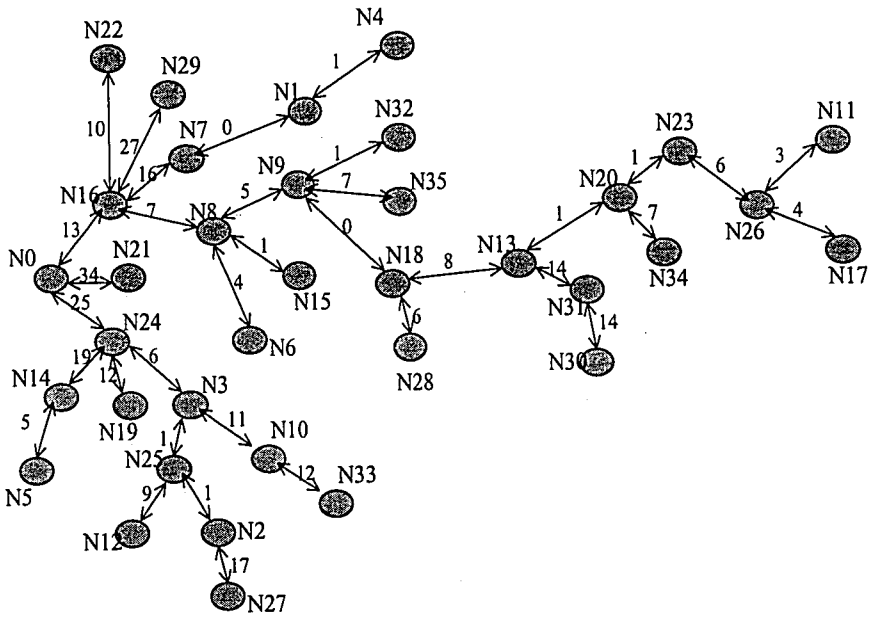

Figure 6-3: Network obtained from Savings Algorithm

Example 2(with near optimal answer)

Results Network obtained by Savings Algorithm (see Figure 6-3):

For Trench Cost T=11, and Cable cost C=4,

Total Network Cost = 308T+1221C = 8272.

Network obtained by CPLEX (see Figure 6-4):

Total Network Cost = 294T+1256C=8258.

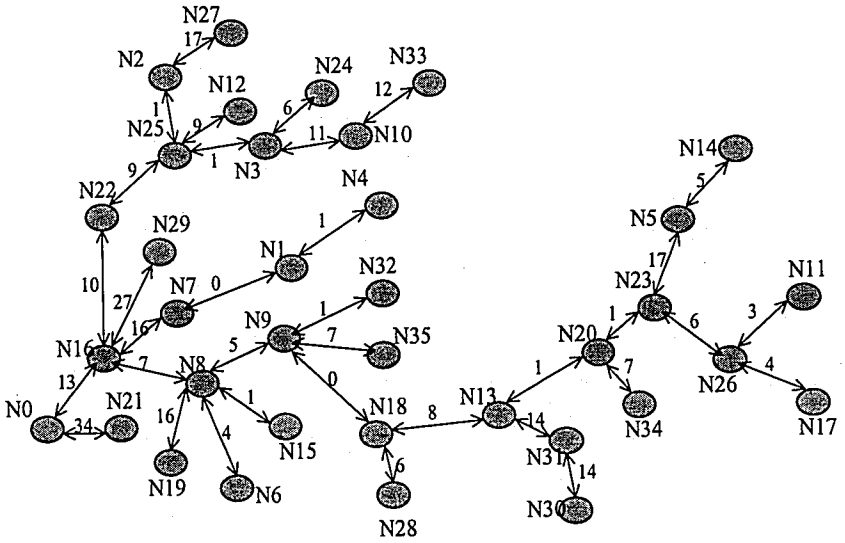

Figure 6-4: Network obtained by solving Integer Program

Bibliography

[1] E.W.Dijkstra, A Note on Two Problems in Connection with Graphs, *Numerische Mathematik* 1 (1959), 269-271.

[2] R. Dial, Algorithm 360: Shortest path forest with topological ordering, *Communications of ACM* 12 (1969), 632-633.

[3] R. Dial, F. Glover, D. Karney, and D. Klingman, A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees, *Networks* 9 (1979), 215-248.

[4] D. B. Johnson, Efficient shortest path Algorithms, *Journal of ACM.* 24 (1977), 1-13.

[5] M.L. Fredman, and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science* (1984), 338-346, Full paper in *Journal of ACM* 34 (1987), 596-615.

[6] L.R. Ford, Network Flow Theory, Report P-923 (1956), Rand Corp., Santa Monica, CA.

[7] E.F. Moore, The shortest path through a maze, In *Proceedings of the International Symposium on the Theory of Switching, Part II*; The Annals of the Computation Laboratory of Harvard University 30 (1957), Harvard University Press, 285-292.

[8] L.R. Ford, and D.R. Fulkerson, *Flows in Networks* (1962), Princeton University Press, Princeton, NJ.

[9] R. Bellman, On a routing problem, Quarterly of Applied Mathematics 16 (1958), 87-90.

[10] J.B. Kruskal, On the shortest spanning tree of graph and the travelling salesman problem, *Proceedings of the American Mathematical Society* 7 (1956), 48-50.

[11] R.C. Prim, Shortest connection networks and some generalizations, *Bell System Technical Journal* 36 (1957), 1389-1401.

[12] C. Berge, and A. Ghouila-Houri, *Programming, Games and Transportation Networks* (1962), Wiley, New York.

[13] H. Booth and J.Westbrook, A Linear Algorithm for Analysis of Minimum Spanning and Shortest-Path Trees of Planar Graphs, *Algorithmica* 11 (1994), 341-352.

[14] S.Khuller, B.Raghavachari, and N.Young, Balancing Minimum Spanning Trees and Shortest-path Trees, *Algorithmica* 14 (1995), 305-321.

[15] H.I. Calvete, and P.M. Mateo, An approach for the network flow problem with multiple objectives, *Computers and Operations Research* vol. 22, no. 9 (1995), 971-983.

[16] F.J. Vasko, R.S. Barbieri, and K.L. Reitmeyer, The Cable Trench Problem: Combining the shortest path and Minimum spanning tree problems, [Submitted to Journal, Nov. 1999] *Computers and Operations Research.*

[17] M.J.Saltzman, Minimum Spanning Tree = Shortest Path Tree?, *Sci Op-Research* (1995).

[18] D.Eppstein, Shortest path along an MST, *Computational theory*, (April 12, 1999).

[19] G. Clarke, and J.W. Wright, Scheduling of Vehicles from a Central Depot to a number of delivery points, *Operations Research* 12 (1964), 568-581.

[20] D. Kim, P.M. Pardalos, " A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure", *Operations Research Letters*, 24 (1999), 195-203.

[21] W.M. Hirsch, G.B. Dantzig, "The fixed charge problem", *Naval Research Logistics Quarterly*, 15 (1968), 413-424.

[22] P. Gray, "Exact solution of the fixed-charge transportation problem", *Operations Research*, 19 (1971), 1529-1538.

[23] K.G. Murty, "Solving the fixed charge problem by ranking the extreme points", *Operations Research*, 16 (1968) 268-279.

[24] D.I. Pardalos, J.B. Rosen, " Constrained global optimization: algorithms and applications", *Lecture Notes in Computer Science*, v268, Springer, Berlin, 1987.

[25] U.S. Palekar, M.H. Karwan, S.A. Zionts, "A branch-and-bound method for the fixed charge transportation problem", *Management Science*, 36 (1990), 1092-1105.

[26] D.I. Steinberg, "The fixed charge problem", *Naval Research Logistics Quarterly*, 17 (1970), 205-225.

[27] K. Sandrock, "A simple algorithm for solving small fixed-charge transportation problems", *Journal of Operations Research Society*, 39 (1988), 467-475.

[28] M.L. Balinski, "Fixed cost transportation problems", *Naval Research Logistics Quarterly*, 8 (1961), 41-54.

[29] L. Cooper, C. Drebes, "An approximation algorithm for the fixed charge problem", *Naval Research Logistics Quarterly*, 14 (1967), 101-113.

[30] D.R. Drenzler, "An approximation method for the fixed charge problem", *Naval Research Logistics Quarterly*, 16 (1969), 411-416.

[31] M. Diaby, "Successive linear approximation procedure for generalized fixed-charge transportation problems", *Journal of Operations Research Society*, 42 (1991), 991-1001.

[32] H.W. Kuhn, W.J. Baumol, "An approximation algorithm for the fixed charge transportation problem", *Naval Research Logistics Quarterly*, 15 (1968), 413-424.

[33] F.R.B. Cruz, J. M. Smith, G.R. Mateus, "Solving to optimality the uncapacitated fixed-charge network flow problem", *Computers Operations Research*, 25(1) (1998), 67-81.

[34] V. Adlakha, K. Kowalski, "On the fixed-charge transportation problem", *Omega, International Journal of Management Science*, 27 (1999) 381-388.

[35] Stallaert, J. "Valid inequalities and separation for capacitated fixed charge flow problems", *Discrete Applied Mathematics*, 98 (2000) 265-274.

[36] D. Kim, P.M. Pardalos, "A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure", *Operations Research Letters*, 24 (1999) 195-203.

[37] M. Sun, J.E. Aronson, P.G. McKeown, D. Drinka, "A tabu search heuristic procedure for the fixed charge transportation problem", *European Journal of Operational Research*, 106 (1998) 441-456.

[38] P.H. Ng, R.L. Rardin, "Commodity family extended formulations of uncapacitated fixed charge network flow problems", *Networks*, 30(1) (1997) 57-71.

[39] T.H. Hultberg, D.M. Cardoso, "The teacher assignment problem: A special case of the fixed charge transportation problem", *European Journal of Operational Research*, 101 (1997) 463-473.

[40] G.M. Guisewite and P.M. Pardalos, Minimum concave-cost network flow problems. *Ann. Oper. Res.* 25 (1990), pp. 75-100.

Vita

The author, Shalu Wadhwa, was born in Dehradun, India in 1977. She obtained her Bachelor of Technology degree in Mechanical Engineering from Indian Institute of Technology, in Delhi, India. She worked in India with Defense Research and Development Organization for an year, before joining Lehigh University. During her stay at Lehigh University, she worked as a Research Assistant for the Industrial and Manufacturing Systems Engineering Department. In addition, she did her summer internship in the Risk Management Department at American Express, NY.

END OF TITLE