ALGORITHMS FOR THE INTEGER MULTICOMMODITY NETWORK DESIGN PROBLEM

A THESIS

SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Mustafa Rasim Kılınç

July 2004

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Oya Ekin Karaşan (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Ezhan Karaşan

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Bahar Yetiş Kara

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

ALGORITHMS FOR THE INTEGER MULTICOMMODITY NETWORK DESIGN PROBLEM

Mustafa Rasim Kılınç
M.S. in Industrial Engineering
Supervisor: Assist. Prof. Dr. Oya Ekin Karaşan
July 2004

In this thesis, we study the problem of logical network design in telecommunication networks. Given a set of nodes and a set of commodities, we aim to locate lightpaths(links) between nodes and route the commodities over these lightpaths. The cost to be minimized is the number of lightpaths used. The problem has capacity, degree and delay constraints. An important characteristic of our problem is that the commodities can not be split, therefore they must be routed on a single path.

We present two integer programming formulations of the problem and consider four sets of valid inequalities. Additionally, a relaxation of the problem is presented to obtain a lower bound to the problem. Finally, we propose two algorithms of generating good feasible solutions to the problem. Our results prove to be close to the lower bounds.

Keywords: Network Topology Design, Integer Multicommodity Flow Problem, Tabu Search, Capacitated Network Design, Branch-and-Price Algorithm.

ÖZET

TAMSAYI ÇOKLU AĞ TASARIMI PROBLEMLERİ İÇİN ALGORİTMALAR

Mustafa Rasim Kılınç
Endüstri Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. Oya Ekin Karaşan
Temmuz, 2004

Bu tezde İletişim Ağlarında Mantıksal Ağ Tasarımı Problemi üzerinde calışıldı. Düğümler kümesi ve bu düğümler arasındaki trafik verildigi halde, yerleştirme maliyetini en azlamayı amaçladık. Problemimizin kapasite, derece ve gecikme kısıtları vardır. Düğümler arasındaki trafiğin bölünerek dağıtılamaması da problemimizin bir başka önemli özelliğidir.

Problemin iki farklı tamsayı programlama modelini verdikten sonra dört farklı geçerli eşitsizlik sunduk. Ayrıca probleme alt sınır bulmak için bir gevşetme programlama modeli sunduk. Problemimiz için iki farklı sezgisel yöntem geliştirdik. Sonuçlarımız ürettiğimiz alt sınırlara yakındır.

Anahtar sözcükler: Ağ Yerleşke Tasarımı,Tamsayı Çok Ürünlü Akım Problemi, Kapasiteli Ağ Tasarımı, Dallandırma-Fiyatlandırma Algoritması.

To my family...

Acknowledgement

I would like to express my most sincere gratitude to my advisor, Assist. Prof. Dr. Oya Ekin Karaşan for her encouragement and trust in the supervision of the thesis. She has been supervising me with patience and everlasting interest.

I would like to express my special thanks and gratitude to Assist. Prof. Ezhan Karaşan for showing keen interest to the subject matter and for his invaluable guidance, remarks and recommendations.

I am also indebted to Assist. Prof. Dr. Bahar Yetiş Kara for accepting to read and review this thesis and for her suggestions.

I would like to express my deepest thanks to Mehmet Oğuz Atan, Salih Öztop, Aysegul Altın, Zümbül Bulut, Güneş Erdoğan, Selin Damla Erdoğan, Kamer Kaya, Yunus Emre Karamanoğlu, Kürşad Derinkuyu, Emrah Zarifoğlu, Esra Büyüktahtakın and Gökhan Metan, for their keen friendship, morale support and helps.

Finally, I would like to express my gratitude to my father Resul Kılınç, my mother Rukkiye Kılınç, my brothers Mehmet Rauf Kılınç and Ahmet Kılınç and my sister Rabia Kılınç for their patience, love and understanding not only for the time of the study but for my lifetime.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With fast and easy access to information, Internet has become one of our lives' indispensable tools. In a short time, it has become widely and abundantly used. With this tremendous usage of Internet, speed and capacity requirements have increased considerably. Existing computer networks did not satisfy this enormous requirement. Now "Service Providers" are building optic trunk lines in which multiple fiber optic cables are combined. Fiber optic cables provide higher speed, better reliability and more capacity. Building optic trunk lines is quite expensive so it should be done wisely to satisfy the requirements with minimal cost.

In this thesis, the problem of logical network design with minimum installation cost is explored. We are given a set of nodes and an associated traffic(commodity) matrix for all pairs of these nodes. Our job is to determine how many optical connections called *lightpaths*, between which node pairs are to be installed so as to route the traffic. We assume that the cost of each lightpath(link) is identical regardless of which node pairs are connected. Besides flow conservation constraints, the structure of the communication network we analyze requires some additional constraints that have to be satisfied.

Before going further, the structure of the communication network will be presented. We are given a set of nodes, and it is possible to install lightpaths between any pair of nodes. Every lightpath has a traffic carrying capacity. We

assume that all lightpaths in the logical topology have the same capacity. If one lightpath is not sufficient for carrying all the traffic routed between two nodes, multiple lightpaths can be established between this pair of nodes. Fiber optic cables are unidirectional, and installing a fiber between a node pair provides the same capacity separately in both directions. In other words, routing a commodity in one direction does not use capacity in the other direction.

Transmission of signals over a fiber optic cable goes through a propagation delay which is determined by the length of the cable. Because of the time requirements, every commodity must be routed within the time no longer than a maximum delay value. Given a delay matrix representing delay values between node pairs, the end-to-end delay for traffic routed between two nodes corresponds to the sum of delay values of every lightpath on the path of the commodity. We ensure that this delay value is less than or equal to some given maximum delay value.

Due to hardware and software constraints, it may not be possible to set up lightpaths between each pair of nodes, because only a limited number of lightpaths can be located on a node. Therefore, the number of lightpaths originating at a node can not exceed a predetermined number.

Another important characteristic of the communication network is that every commodity must be routed on a single path. It is not possible to split the flow of a commodity and route it on several paths. This is merely because splitting a commodity complicates routing and results in out-of-order packet delivery at the destination.

Under these constraints, our aim is to find a feasible logical network design and route the commodities on this network while minimizing the cost value. Since we assume that the costs of the links are identical, our objective is to minimize the number of lightpaths used. A feasible solution should satisfy all of the following conditions:

- Every commodity must be routed through a single path which is a sequence of lightpaths.

- Every commodity must be routed within a delay no longer than the maximum delay value.
- Summation of the flow amounts routed over a lightpath must be less than or equal to the bandwidth capacity of the installed lightpath.
- Number of lightpaths installed on a node must be less than or equal to the maximum nodal degree which is the maximum number of interfaces that can be built at a node.

Our preliminary attempts showed that finding an optimum solution to this problem within reasonable time is outside the capabilities of the off-the-shelf software for even small problem instances. Even, finding a feasible solution has not been possible for moderate instances with 20 nodes. Therefore, our approach to the problem will be first finding a "good" feasible solution heuristically. Then, we will find a lower bound to our problem to be able to judge the quality of our heuristic solution, i.e., how close is the gap between our upper and lower bounds.

The remainder of the thesis is organized as follows:

In Chapter 2, we will provide a review of the literature in network design and multicommodity flow problems.

In Chapter 3, two Integer Programming(IP) formulations of the problem at hand are presented. In the first formulation, routing of the commodities is represented by binary variables and location of lightpaths are represented by integer variables. Second formulation is a path-based formulation where our model contains one binary variable for each path, for every commodity. This formulation has fewer constraints but an exponential number of variables. It is also suitable for column generation.

In Chapter 4, some valid cuts are introduced for the models presented. Since our problem is a minimization problem, LP relaxation value of the problem is a lower bound to the optimal value. By adding valid cuts to the problem, the lower bound can be improved. We also present an aggregated model of our problem where commodities are consolidated according to their source nodes. Aggregated

formulation is a relaxation of our original problem and greatly reduces the number of variables. At the end of the chapter, the effects of these cuts and results of aggregated formulation are presented on test problems.

In Chapter 5, two algorithms are proposed. The first one is the Tabu Search algorithm, which is an improved version of [12]. The heuristic is based on a local search and approaches to the problem as a two step problem: Location Problem and Routing Problem. It starts with a feasible solution. In each iteration, it finds neighborhood solutions to the Location Problem. Then for every neighborhood solution, it routes the commodities in descending order of their flow amounts via shortest paths. After each routing, link capacities of the edges on the routing path are lowered by the flow amount of the commodity. When finding the shortest path of the next commodity, only the links that have enough capacity for the commodity are used. Before going to the next iteration, it selects the feasible neighborhood with the smallest cost as a move. For more on Tabu Search and its application areas we refer the reader to Glover and Laguna [11].

In the second algorithm, we present a column-generation model and a branch-and-price algorithm. At the root node, our model contains only one variable(one path) for each commodity. At the nodes of the branch-and-bound tree, the linear programming relaxation of the problem is solved and by implicit pricing of nonbasic variables, new variables are generated or LP optimality is proved. A branching rule is used to preserve the structure of the subproblem and it allows columns to be generated efficiently at each node of the branch-and-bound tree. At the end of Chapter 5, computational experiments with test problems are given.

The last chapter is the summary of thesis. The results of the thesis are discussed and possible areas of future research are highlighted.

Chapter 2

Literature Review

Network design models have been known as useful planning tools in areas of transportation, telecommunications and logistics. Network design problems concern the selection of arcs in a graph in order to satisfy flow requirements. In other words, it contains location and routing problems simultaneously. Comprehensive survey on the models and algorithms for network design problems can be found in Magnanti and Wong [15] and Minoux [16]. In the literature, network design problems can be characterized in two main classes as uncapacitated(UNDP) and capacitated(CNDP) problems. The problems can also be characterized according to the charges of the arcs. There are three main classes known as fixed charge(single-facility), two level fixed charge(two-facility) and step increasing cost function. In fixed charge networks, there is only one type of technology, which can be installed multiple times on a link. In two level fixed charge, the technology for each link should be chosen from given two technologies. In the third class, the cost of purchasing capacity for a link is given by a step-increasing cost-capacity function.

In the literature, network flow(routing) problems also drew the attention of many researchers. In network flow problems, arc capacities are given apriori and flow requirements are the main concern while optimizing some objective function value. Network flow problems can also be characterized in two main classes: Linear(splittable) and Integer(unsplittable) flow problems. The problems can

further be characterized according to the origin and destination of the flows. There are three main classes known as single(one-to-one) flow, single-source(one-to-all) flow and multicommodity(all-to-all) flow. Unlike linear multicommodity flow problems, there are not many studies that focus on integer multicommodity flow problems. A comprehensive survey of linear multicommodity flow (LMF) models and solution procedures was presented in Ahuja et al [1] .

Capacitated fixed charge network design problem with integer multicommodity is not thoroughly investigated. The main target of this thesis is to fill this gap. Our problem of focus is Integer Multicommodity Capacitated Fixed Charge Network Design Problem with additional degree and delay constraints.

In this chapter, we present studies from the literature to form a basis on the theoretical background of the related topics such as network design problems and integer multicommodity flow problems.

2.1 Network Design Problems

Gendron and Crainic [10] study the Linear Multicommodity Fixed Charge Capacitated Network Design Problem. Three formulations of the problem are presented. Various relaxations are defined and analyzed theoretically. Relaxations are compared empirically by performing heuristic based computational experiments on a large set of test problems. The linking constraints they define can also be applied to Integer Multicommodity Network Design Problems. We used similar inequalities on our problem called as Link Inequalities to improve optimum value of LP relaxation of Integer Programming Formulation.

Magnanti et al. [14] study the Linear Multicommodity Two-Facility Capacitated Network Design Problem. They introduce three facet defining valid inequalities: Cutset Inequality, Arc Residual Capacity Inequality and 3-Partition Inequality. They propose a Lagrangian relaxation strategy and a cutting plane approach that uses these inequalities and showed that these inequalities provide a bound at least as efficient as the Lagrangian relaxation lower bound. Cutset

inequalities are also valid for our problem and they are used to improve optimum value of LP relaxation.

Bienstock and Günlük [8] study the Linear Multicommodity Network Design Problem. They have a degree constraint stating that the number of links assigned to any node in the network must be equal to a constant number. The goal is to minimize the maximum aggregate flow on any edge. To obtain a lower bound, they deal with the aggregated formulation of the problem where commodities are consolidated and identify according to their source nodes. They introduce flux inequalities and use basic network equalities that are first introduced by Van Roy and Wolsey [18] for the aggregate formulation and present a cutting plane algorithm. Note that flux inequalities will be explained further in Chapter 4. To make our problem easier, we also used aggregated formulation that reduces the number of variables in the original formulation.

Ramaswami and Sivarajan [17] study the Linear Multicommodity Network Design Problem. Their aim is to design a logical topology over a wavelength-routed all-optical network physical topology. All-optical networks are networks where information is converted to light, transmitted as light, and reaches its final destination directly without being converted to electronic form in between. Their aim is to minimize the maximum load on a link, i.e., the congestion, subject to the restriction that the delay of a commodity can not be more than a predetermined value. They also have degree constraints and wavelength number constraints, stating that multiple data streams to be transferred concurrently along the same fiber-optic cable must be assigned to separate wavelengths. They approach the problem in two stages: logical topology design stage and routing stage. We used a similar approach for our problem in the Tabu Search algorithm. They propose five different heuristics for logical topology design and compare the performances of these heuristics.

An important study on the Integer Multicommodity Fixed Charge Capacitated Network Design Problem is that of Barahona [3]. The paper first classifies the nodes into local and backbone. Local nodes must send their traffic through backbone nodes, thus this reduces the number of nodes that have to be dealt

with. Then, a relaxation of the problem involving only the link variables and backbone nodes is presented and a cutting plane algorithm based on cut and partition(multicut) inequalities is introduced. To find a feasible solution to the original problem, they incorporate a branch-and-bound procedure along with the cutting plane algorithm. Partition Inequalities will be explained further in Chapter 4, and will be incorporated to improve the LP relaxation value of our original formulation.

Dahl et al. [9] study a different version of the Integer Multicommodity Fixed Charge Capacitated Network Design Problem. Given a physical network, a demand matrix and a pipe network(virtual network), the problem is to select the pipes(ligthpaths) that are to be used and to determine on which path of the selected pipe set each of the demands should be routed. In addition to the capacity constraints, the number of the pipes on a link must not exceed a predetermined number. They introduce several facet defining valid inequalities of the problem. Then, they propose a cutting plane algorithm that uses separation and primal heuristics. This problem is very similar to our problem. Different from this study, we have a degree constraint on nodes and not on links and furthermore we have delay constraints.

Another important study on Network Design Problem is of Karaşan et al. [13]. They study Mesh Topology Design in Overlay Virtual Private Networks. The problem can be characterized as Integer Multicommodity Uncapacitated Network Design Problem. They also have a degree constraint stating that the number of links assigned to any node must be equal to a predetermined number. The goal of the problem is to determine where to locate the links and route the commodities while minimizing the routing cost value. Two types of valid inequalities are generated: flux inequalities and distance inequalities. They apply a Tabu Search based heuristic. The gap between upper and lower bound is found to be around %2 for reasonable networks of 20 nodes. Link Interchange method that we used in our Tabu Search algorithm is borrowed from them.

2.2 Integer Multicommodity Flow Problems

For small to moderate size LMF problems, there exist both heuristics and exact solution procedures in the literature [1]. However, as the problem size increases, these procedures require large amounts of memory and are therefore computationally impractical. Barnhart and Sheffi [7] develop a network-based primal-dual heuristic for large-scale LMF problems. They relax the problem and use a pure network-based solution strategy. This network-based strategy is embedded within an iterative primal-dual framework.

Barnhart et al. [4] present a column generation and partitioning solution procedure for these huge problems. They consider two different formulations for LMF: arc-chain formulation and cycle formulation. Using a cycle-based formulation, a solution procedure is proposed adopting column generation techniques and constraint relaxation. They are able to solve LMF problems with 500 nodes, 1300 arcs and 5850 commodities in a reasonable time.

Barnhart et al. [6] introduce a branch-and-price algorithm for solving huge integer programs using column generation techniques. Barnhart et al. [5] successfully implement this algorithm to the Integer Multicommodity Flow Problem. They state path(column generation) formulation of the problem which has fewer constraints and exponential number of variables in comparison to the conventional formulation. In column generation, sets of columns are left out of the linear programming(LP) because most of them will have their associated variable equal to zero in an optimal solution. Then to check optimality of an LP solution, a subproblem called the pricing problem, which is a separation problem for the dual LP, is solved to try to identify columns to enter the basis. They also propose a new branching strategy at the branch-and-bound procedure which is compatible with the structure of the pricing problem. They describe lifted cover inequalities that can be generated at each node of the branch-and-bound tree. Alvelos and Carvalho [2] study the same problem and present knapsack decomposition. They propose a new branching rule at the branch-and-bound procedure. They also compare path and knapsack decomposition and branching rules with computational results. We modified this branch-and-price algorithm and applied

to our problem. We use same branching strategy as they state for path variables [6]. In their formulation they do not have link variables. We propose a branching procedure for link variables that preserves the structure of the pricing problem.

To sum up, our problem is a network design problem for the given node set and associated traffic between these nodes. Unlike most of the studies in the literature, the traffic flow of any commodity can not be split. This makes our problem harder than the linear multicommodity network design problems. Even for linear cases there are no proposed exact solution methods, therefore, one can not hope to find an exact method for integer cases. We have also additional degree and delay constraints which appear only in some of Linear Multicommodity Network Design Problems. There are only a few studies on the Integer Multicommodity Network Design Problems but none of these studies considers these two constraints jointly. We aim to fill this gap in this thesis.

Chapter 3

Problem Formulation

In this chapter, two IP formulations of the problem along with an analysis of their strong and weak points are presented. Although they do not prove to be useful in solving the problem itself, they will aid in better understanding of the structure of the problem and may offer many insights about ways of solving the problem. Before moving on to the formulations, it will be useful to state the notations that will be used through the thesis.

3.1 Notation

Let $G = (N, E)$ be the graph corresponding to the network topology where $N = \{1, ..., |N|\}$ is the set of nodes, $|N|$ is the cardinality of node set. Let $E = \{1, ..., |E|\}$ be the set of edges whose elements are all possible unordered pairs of distinct nodes (complete network), i.e. $\{i, j\} \in E$ for $i, j \in N, i \neq j$ and $|E|$ be the cardinality of edge set. Let $A = \{1, ..., |A|\}$ be the arc set associated with edge set E, whose elements are oriented versions of edges, i.e. $(i, j), (j, i) \in A$ for every edge $\{i, j\} \in E$ and $|A|$ be the cardinality of arc set.

Let K be the set of commodities, with cardinality $|K|$. Each element in the set K has a triplet (s_k, d_k, f_k) associated, where s_k denotes the source of

commodity k, d_k denotes the destination of commodity k and f_k denotes the flow amount(traffic) of commodity k. We can also represent the commodities by a traffic matrix F, where f_{ij} represents the traffic between node i and j.

We have two types of variables: location variables and routing variables.

Let Y_{ij} be the location variable denoting the number of links between nodes i and j. Since we can put an integer number of links between nodes, Y_{ij} takes integer values. It is clear that Y_{ij} is equal to Y_{ji}, because they represent the same parameter, the number of links between nodes i and j. In the model, we shall force them equal to each other by a constraint.

Let X_{ij}^k be the binary variable representing the flow of commodity k from node i to node j. In other words,

$$X_{ij}^k = \begin{cases} 1 & \text{if commodity } k \text{ is routed over arc } (i,j) \text{ from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

We are also given a symmetric delay matrix D that denotes the delay values between ever pair of nodes. An entry in D, say d_{ij}, represents the delay value between nodes i and j, clearly we have $d_{ij} = d_{ji}$.

Let r_i^k be the parameter which differentiates between supply, demand and transshipment nodes of each commodity. In other words,

$$r_i^k = \begin{cases} 1 & \text{if node } i \text{ is the source node of commodity } k \\ -1 & \text{if node } i \text{ is the destination node of commodity } k \\ 0 & \text{otherwise} \end{cases}$$

Let p be the maximum number of links that can be located on a node.

Let D be the maximum delay value.

Let c represent the bandwidth capacity of a link.

3.2 Integer Programming Formulation

The IP formulation of the problem based on the above definitions is as follows:

$$\begin{aligned}
&\text{Minimize} && \textstyle\sum_{\{(i,j)\,:\,i<j\}} Y_{ij} && && \\
&\text{subject to} && && && \\
&\sum_{j\in N,\, j\neq i} X_{ij}^k - \sum_{j\in N,\, j\neq i} X_{ji}^k && = && r_i^k \quad \forall i \in N, \quad \forall k \in K && (3.1)\\
&\sum_{k\in K} f_k X_{ij}^k && \leq && c * Y_{ij} \quad \forall (i,j) \in A && (3.2)\\
&\sum_{j} Y_{ij} && \leq && p \quad \forall i \in N && (3.3)\\
&\sum_{(i,j)\in A} d_{ij} X_{ij}^k && \leq && D \quad \forall k \in K && (3.4)\\
&Y_{ij} && = && Y_{ji} \quad \forall \{i,j\} \in E && (3.5)\\
&X_{ij}^k && \in && \{0,1\} \quad \forall (i,j) \in A, \forall k \in K && (3.6)\\
&Y_{ij} && \in && Z^+ \cup \{0\} \quad \forall (i,j) \in A && (3.7)
\end{aligned}$$

In this thesis, we assumed that the cost of a link is identical regardless of which node pairs are connected. So in the objective function, we only add up Y_{ij} values. Since both Y_{ij} and Y_{ji} represent the same parameter, the total number of links is the summation of Y_{ij} values over all (i, j) where i is less than j.

Constraints 3.1 are the usual flow conservation constraints. They state that, for each commodity, the difference between the flow that enters a node and the flow that leaves that node is equal to the supply/demand of that node. Nodes other than source and destination of the commodity(i.e., the transshipment nodes) must a have balance on entering and leaving flows.

Constraints 3.2 are the capacity constraints. They express that the total flow on each arc must be less than or equal to the bandwidth capacity installed on that arc. The bandwidth capacity is the number of links times the capacity of a single link. We use variables Y_{ij} and Y_{ji} simultaneously, since installing a link between

a node pair provides same capacity in both directions and routing a commodity in one direction does not use capacity in the other direction. However, we set Y_{ij} equal to Y_{ji} for every i,j pair in Constraints 3.5. We need such a constraint because whenever a link is located between node i and node j, both Y_{ij} and Y_{ji} values increase by 1.

Constraints 3.3 are degree constraints. They express that there can be at most p links that can be located on a node. During our computational experiments, it is assumed that the maximum degree number for all nodes is the same and is equal to 8.

Constraints 3.4 are delay constraints and state that the delay value of a commodity must be less than or equal to the maximum delay value. The delay value of a commodity is found by summing the delay values of edges that are used in the routing of the commodity.

Constraints 3.6 express that routing variables are binary, and Constraints 3.7 express that location variables are integer.

We have $|A|$ location variables and $|A||K|$ routing variables, and the number of constraints is $|N||K| + 3|E| + |K| + |N|$.

The IP formulation of the problem has a very weak LP relaxation though it maybe be strengthened with valid cuts. Unfortunately, for a small sized problem of 10 nodes, 45 edges(90 arcs) and 90 commodities, the number of variables required is 8190 where 8100 of them are binary and 90 of them are integer. The number of constraints is 1145. For larger problems, the number of variables becomes too large for commercially available optimization softwares to solve to optimality.

3.3 Path Formulation

We can also formulate the problem using path flows instead of arc flows. For each commodity k, let $\mathcal{P}^k$ denote the collection of all directed paths from the source

node s_k to the destination node d_k in the underlying network $G = (N, A)$. Let X_p^k be binary flow variables indicating if commodity k is assigned to path $p \in \mathcal{P}^k$. In other words,

$$X_p^k = \begin{cases} 1 & \text{if commodity } k \text{ is routed through path } p \in \mathcal{P}^k \\ 0 & \text{otherwise} \end{cases}$$

Let δ_{ij}^p be an arc-path indicator variable, that is,

$$\delta_{ij}^p = \begin{cases} 1 & \text{arc } (i,j) \text{ is contained in the path } p \in \mathcal{P}^k \\ 0 & \text{otherwise} \end{cases}$$

The path formulation is then:

$$\begin{aligned}
&\text{Minimize} && \textstyle\sum_{\{(i,j)\,:\,i<j\}} Y_{ij} && && \\
&\text{subject to} && && && \\
&\sum_{p \in P^k} X_p^k && = && 1 \quad \forall k \in K && (3.8)\\
&\sum_{k \in K} \sum_{p \in \mathcal{P}^k} f^k \delta_{ij}^p X_p^k && \leq && c * Y_{ij} \quad \forall (i,j) \in A && (3.9)\\
&\sum_{j} Y_{ij} && \leq && p \quad \forall i \in N && (3.10)\\
&\sum_{(i,j) \in A} d_{ij} \delta_{ij}^p X_p^k && \leq && D \quad \forall p \in \mathcal{P}^k \quad \forall k \in K && (3.11)\\
&Y_{ij} && = && Y_{ji} \quad \forall \{i,j\} \in E && (3.12)\\
&X_p^k && \in && \{0,1\} \quad \forall p \in \mathcal{P}^k \quad \forall k \in K && (3.13)\\
&Y_{ij} && \in && Z^+ \cup \{0\} \quad \forall (i,j) \in A && (3.14)
\end{aligned}$$

Constraints 3.8 state that each commodity k is assigned to one path $p \in \mathcal{P}^k$.

Constraints 3.9 are the capacity constraints with the new definition of the variables. Given a solution to Path Formulation(PF) we can recover a solution to the IP formulation by using:

$$X_{ij}^k = \sum_{p \in P^k} \delta_{ij}^p X_p^k \quad \forall (i,j) \in A \quad \forall k \in K \tag{3.15}$$

Since δ_{ij}^p indicator parameters are given apriori, we can weed out the path variables that are not feasible because of delay constraints. Thus constraints 3.11 become redundant.

The number of variables is enormous, growing exponentially in the size of the network, since PF has a variable for every path connecting a source and destination node for each of the commodities. However, the number of constraints reduces considerably if we remove delay constraints. They become $|K|+3|E|+|N|$ in our formulation. The number of constraints required for the 10 node example above becomes 235.

Though PF has an exponential number of variables, only one of the paths for each commodity will carry flow in the optimal solution to the problem. Thus column generation method is appropriate for solving PF.

3.4 Conclusion

In this chapter, we presented two formulations of the problem. The first one is the IP formulation of the problem which contains a large number of constraints and variables. One can not hope to solve this IP with off-the-shelf softwares. The second one is the PF of the problem. It contains a moderate number of constraints, but a huge number of variables. It shows that we can not find an optimal solution to the problem in a reasonable time. Since LP relaxation of the problem is too weak, valid inequalities to tighten the lower bound will be the next focus of this thesis.

Chapter 4

Lower Bound

In this chapter, we attempt to obtain a 'good' lower bound to our problem. Since we can not solve our problem to optimality, we need a good lower bound to be able to evaluate our heuristic solution.

First, we will adapt valid inequalities from the literature to our problem and try to strengthen the LP relaxation with these valid inequalities. Then we will give a relaxation of the problem which is easier to handle according to the original formulation. Finally, at the end of this chapter, computational results about effects of valid inequalities and the relaxation for test problems will be presented.

4.1 Valid Inequalities

In this section, four sets of valid inequalities are considered that will be applied to LP relaxation of our original formulation. These are cutset inequalities, link inequalities, flux inequalities and partition inequalities.

4.1.1 Cutset Inequalities

The first set of valid inequalities is known as the cutset inequalities in the literature [14]. Whenever we partition the node set into two, the commodities whose sources belong to a set and destinations belong to the other set must use the links between these two node sets. Therefore, the bandwidth capacity installed on these links must be greater than or equal to the total flow amounts of those commodities.

Let S be a subset of N. Furthermore, let $T = N \setminus S$ and K_{ST},(K_{TS}) be the sets of commodities whose sources are in $S(T)$ and destinations are in $T(S)$ respectively. Then we have:

$$\sum_{i \in S, j \in T} c * Y_{ij} \geq \sum_{k \in K_{ST}} f_k \quad \forall S \subset N$$
$$\sum_{i \in S, j \in T} c * Y_{ji} \geq \sum_{k \in K_{TS}} f_k \quad \forall S \subset N$$

Since Y_{ij}'s are integer variables and $Y_{ij} = Y_{ji}$, we can strengthen these inequalities as follows:

$$\sum_{i \in S, j \in T} Y_{ij} \geq max \left\{ \lceil \frac{\sum_{k \in K_{ST}} f_k}{c} \rceil, \lceil \frac{\sum_{k \in K_{TS}} f_k}{c} \rceil \right\} \quad \forall S \subset N \tag{4.1}$$

The cutset inequalities are exponential in number. Every subset S of N gives a probable cut. Adding this many constraints expands the problem much beyond tractability. Thus, we select subsets with one element for cutset inequalities. In other words, we select $S \subset N$ such that $|S| = 1$. We also tried for sets with cardinality higher than one, but observed that adding such inequalities does not improve the lower bound.

4.1.2 Link Inequalities

The second set of inequalities is the link inequalities [10]. They state that whenever a commodity is routed over a link, there must be at least one cable located on that link.

Since X_{ij}^k are binary variables, they each can take value at most 1. Therefore the number of cables on arc (i, j) should be greater or equal to X_{ij}^k. Then we have,

$$Y_{ij} \geq X_{ij}^k \quad , \quad \forall (i,j) \in A \quad , \forall k \in K \tag{4.2}$$

4.1.3 Flux Inequalities

The third set of inequalities is the flux inequalities [8]. Flux inequalities express that if there is not a direct link between source and destination nodes of a commodity than the commodity must use at least 2 arcs to be routed. Although Y_{ij}'s are integer variables, the flux inequalities can be stated as follows:

$$\sum_{(i,j)\in A} X_{ij}^k + Y_{s_k d_k} \geq 2 \quad , \forall k \in K \tag{4.3}$$

This inequality is always valid. Because, if $Y_{s_k d_k} = 0$ than it means that there is no link between s_k and d_k, thus commodity k must use at least two arcs. If $Y_{s_k d_k} = 1$ than it becomes $X_{ij}^k \geq 1$ which means that commodity must be routed. If $Y_{s_k d_k} \geq 2$, than there is not any restriction upon flow variables.

4.1.4 Partition Inequalities

Another set of inequalities are the partition(multicut) inequalities [3]. Partition inequalities is a generalization of the cutset inequalities. Let $N_1, \ldots, N_p$ be a

partition of N with nonempty sets . Denoted by $\delta(N_1, \ldots, N_p)$ is the set of edges having their endnodes in different sets of the partition. Then, the partition inequality induced by $N_1, \ldots, N_p$ is defined as:

$$\sum_{ij \in \delta(N_1,\ldots,N_p)} Y_{ij} \geq p - 1 \tag{4.4}$$

This inequality is valid for every partition of N since our network must be connected. Otherwise, our problem can be reduced to small sized problems and can be solved by solving these subproblems.

4.2 Aggregated Relaxation

Since IP formulation of the problem has a large number of variables, to overcome this problem, we used aggregation. We assumed all demands with the same source as constituting a single commodity. So the number of commodities becomes at most $|N|$.

Once aggregation is done, we can not use our original binary variables for routing of commodities anymore. We define new continuous flow variables for routing of commodities. X_{ij}^k is now described as amount of flow routed over arc (i, j) originating from node k.

Let F^k equal to $\sum_{i \in N} f_{ki}$. We replace the value of parameter r_i^k as follows:

$$r_i^k = \begin{cases} F^k & \text{if } i = k \\ -f_{ki} & \text{otherwise} \end{cases}$$

According to these modifications, the aggregated formulation is as follows:

$$\begin{array}{ll} \text{Minimize} & \sum_{\{(i,j)\,:\,i<j\}} Y_{ij} \\ \text{subject to} & \end{array}$$

$$\sum_{j\in N,\, j\neq i} X_{ij}^k - \sum_{j\in N,\, j\neq i} X_{ji}^k = r_i^k \quad \forall i,k \in N \tag{4.5}$$

$$\sum_{k\in K} X_{ij}^k \leq c * Y_{ij} \quad \forall (i,j) \in A \tag{4.6}$$

$$\sum_{j} Y_{ij} \leq p \quad \forall i \in N \tag{4.7}$$

$$\sum_{ij\in A} d_{ij} X_{ij}^k \leq D * F^k \quad \forall k \in N \tag{4.8}$$

$$Y_{ij} = Y_{ji} \quad \forall \{i,j\} \in E \tag{4.9}$$

$$X_{ij}^k \geq 0 \quad \forall k \in N \quad \forall (i,j) \in A \tag{4.10}$$

$$Y_{ij} \in Z^+ \cup \{0\} \quad \forall (i,j) \in A \tag{4.11}$$

We have $|A|$ location variables and $|A||N|$ routing variables, and the number of constraints is $|N|^2 + 3|E| + 2|N|$.

Since our flow variables are not binary anymore, the flow of a commodity in the original problem may split and be routed via several paths. Thus aggregated formulation of the problem is a relaxation of our original problem. Another relaxation is done at the constraints 4.8. In the original problem, every commodity must be routed such that the delay amount is less than the maximum delay value. But here we only require that average weighted delay value of an aggregated commodity originating from a node must be less than the maximum delay value.

Aggregation is quite useful for reducing the number of variables, for example, number of variables required for the 10 node example above becomes 990 where 900 of them are continuous variables and 90 of them are integer variables. The number of constraints is 255. For small size problems, aggregated formulation is tractable with commercially available optimization softwares. However the number of variables increases drastically with the number of nodes since the number of flow variables is of $O(|N|^3)$.

By aggregated formulation, we reduce the number of variables reasonably. Still the problems with moderate sizes are not tractable for commercially available optimization softwares. To make the problem easier, we drop constraints 4.8 since

a relaxation is already done at these constraints.

4.3 Test Problems

In this thesis, we tackled with different sets of problems. In all problems, traffic values come from a uniform distribution between 0 and 200. For each set of problems, we consider high capacity($c = 2488$) or low capacity($c = 622$) cases. Another parameter that we varied is the number of commodities. We consider three cases: when there is traffic between all node pairs, when there is traffic between 30% of the node pairs and when there is traffic between 60% of the node pairs. For each set, we consider 6 different problems as a result of the combination of these two parameters.

The first set of our problems consists of the U.S. cities. We choose three city sets with sizes of 10, 15, 20 and take d_{ij} values as the air distances between the cities in miles. The distance between any two cities is less than 2596 miles. So we take 4000 as the maximum delay value for these sets of problems. Input characteristics of these 18 problems are given in Table 4.1.

The second set of the problems consists of Turkish cities. We choose one city set with size of 15 and take d_{ij} values as the driving distances between the cities in kilometers. The distance between any two cities is less than 1457 kilometers. So we take 2000 as the maximum delay value for this set of problems. Input characteristics of these 6 problems are given in Table 4.2.

Our third sets uses European cities. We choose three city sets with sizes of 10, 15, 24 and take d_{ij} values as the driving distances between the cities in kilometers. The distance between any two cities is less than 4452 kilometers. So we take 6000 as the maximum delay value for these sets of problems. Input characteristics of these 18 test problems are given in Table 4.3.

Totally, we have 42 test problems with different characteristics.

Problem Name	$\|N\|$	$\|K\|$	c	$Maxdelay$	f_k
US1	10	90	622	4000	U[0,200]
US2	10	90	2488	4000	U[0,200]
US3	10	52	622	4000	U[0,200]
US4	10	52	2488	4000	U[0,200]
US5	10	29	622	4000	U[0,200]
US6	10	29	2488	4000	U[0,200]
US7	15	210	622	4000	U[0,200]
US8	15	210	2488	4000	U[0,200]
US9	15	113	622	4000	U[0,200]
US10	15	113	2488	4000	U[0,200]
US11	15	60	622	4000	U[0,200]
US12	15	60	2488	4000	U[0,200]
US13	20	380	622	4000	U[0,200]
US14	20	380	2488	4000	U[0,200]
US15	20	245	622	4000	U[0,200]
US16	20	245	2488	4000	U[0,200]
US17	20	115	622	4000	U[0,200]
US18	20	115	2488	4000	U[0,200]

Table 4.1: Characterization of Test Problems of U.S. cities

Problem Name	$\|N\|$	$\|K\|$	c	$Maxdelay$	f_k
TR1	15	210	622	2000	U[0,200]
TR2	15	210	2488	2000	U[0,200]
TR3	15	124	622	2000	U[0,200]
TR4	15	124	2488	2000	U[0,200]
TR5	15	65	622	2000	U[0,200]
TR6	15	65	2488	2000	U[0,200]

Table 4.2: Characterization of Test Problems of T.R. cities

Problem Name	$\|N\|$	$\|K\|$	c	$Maxdelay$	f_k
EU1	10	90	622	6000	U[0,200]
EU2	10	90	2488	6000	U[0,200]
EU3	10	59	622	6000	U[0,200]
EU4	10	59	2488	6000	U[0,200]
EU5	10	26	622	6000	U[0,200]
EU6	10	26	2488	6000	U[0,200]
EU7	15	210	622	6000	U[0,200]
EU8	15	210	2488	6000	U[0,200]
EU9	15	124	622	6000	U[0,200]
EU10	15	124	2488	6000	U[0,200]
EU11	15	69	622	6000	U[0,200]
EU12	15	69	2488	6000	U[0,200]
EU13	24	552	622	6000	U[0,200]
EU14	24	552	2488	6000	U[0,200]
EU15	24	332	622	6000	U[0,200]
EU16	24	332	2488	6000	U[0,200]
EU17	24	168	622	6000	U[0,200]
EU18	24	168	2488	6000	U[0,200]

Table 4.3: Characterization of Test Problems of E.U. cities

4.4 Computational Experiments for Lower Bound

In this section, we present the computational experiments for lower bound. We first analyzed the effect of the each valid inequality to the LP relaxation of the original IP formulation independently, then applied them together. For the Cutset inequalities, we select the subsets with one elements, i.e. $S \subset N$ such that $|S| = 1$. It is observed that Cutset Inequalities do not improve the LP relaxation value so much. Secondly, we add Link inequalities to the LP relaxation. Link Inequalities increase the lower bound very much when compared to the Cutset Inequalities. Thirdly, we add Flux Inequalities to the LP relaxation. For most instances, Flux Inequalities increase the lower bound slightly worse than Link Inequalities do. However, it performed badly when high capacity ligthpaths are used and number of commodities is low. For the Partition Inequalities, we select only one partition where size of each set in partition is 1. Formally;

$$\sum_{ij \in \delta(N_1,\ldots,N_{|N|})} Y_{ij} \geq |N| - 1 \tag{4.12}$$

By Partition Inequalities, we ensure that our final network is connected and has at least $|N| - 1$ nodes. Finally, we add all of four inequalities to the LP relaxation. The results can be categorized into two groups, one for which Link Inequalities dominate and the other where Partition Inequalities dominate. In the first case, the results are the same as only adding Link Inequalities for nearly all of the cases. For only a few cases, the results are slightly better than only adding Link Inequalities. In the latter case, the lower bounds we obtained are equal to $|N| - 1$.

Aggregated Relaxation of the problem is also very hard. Finding an optimum solution to this problem within reasonable time is also outside the capabilities of off-the-shelf software even for small instances. So we set a time limit of 6 hours and take the lower bound found to the this problem in 6 hours as a lower bound to

our original problem. Since Aggregated Relaxation is a relaxation of our original problem, any lower bound to Aggregated Relaxation is also a lower bound to our original problem. In some small instances, relaxation of Aggregated Formulation improved the lower bound found by adding all of four inequalities.

In Tables 4.4-6, we present the computational results for each set of test problems. In the second column, LP relaxation values of the test problems are represented. From the third column to the sixth column, we give lower bounds attained after adding Cutset(C), Link(L), Flux(F) and Partition Inequalities(P) separately. Finally, all of the four inequalities are added(All) and results are presented. In $|AR|$ column, we give last lower bound value of Aggregated Relaxation attained within the allowed time of 6 hours. In the last column, we rounded up to the smallest integer greater than the best lower bound found for each test problem since our objective function is sum of links used and all link variables are integer.

The lower bounds presented here will be the ones that we will use to evaluate the upper bounds found by heuristics in the next chapter.

	$\|LP\|$	$\|LP+C\|$	$\|LP+L\|$	$\|LP+F\|$	$\|LP+P\|$	$\|LP\|$ +All	$\|AR\|$	LB
US1	8.58	10	12.22	12.04	9	12.24	14	14
US2	2.15	5	5	3.48	9	9	6.99	9
US3	6.91	8.5	9.33	8.98	9	9.35	11	11
US4	1.73	5	5	2.58	9	9	6.61	9
US5	3.59	5	5.44	4.64	9	9	9	9
US6	0.9	5	5	1.3	9	9	6.29	9
US7	19.84	22.5	28.46	28.22	19.84	28.46	28.51	29
US8	4.96	7.5	8.35	8.16	14	14	8.44	14
US9	13.45	15	18.1	17.5	14	18.13	18.36	19
US10	3.36	7.5	7.5	4.98	14	14	6.31	14
US11	7.97	10	10.66	9.75	14	14	11.19	14
US12	2	7.5	7.5	2.71	14	14	4.87	14
US13	37.22	39	52.71	52.53	37.22	52.72	52.7	53
US14	9.3	10	15.55	15.32	19	19	15.22	19
US15	27.92	28	37.36	36.73	27.92	37.36	37.33	38
US16	6.98	10	10.98	10.53	19	19	10.64	19
US17	14.1	16	18.58	17.51	19	19	18.6	19
US18	3.53	10	10	4.85	19	19	5.86	19

Table 4.4: Computational Results for U.S. cities

	$\|LP\|$	$\|LP+C\|$	$\|LP+L\|$	$\|LP+F\|$	$\|LP+P\|$	$\|LP\|$ +All	$\|AR\|$	LB
TR1	21.12	22.5	29.47	29.29	21.12	29.48	29.55	30
TR2	5.28	7.5	8.68	8.52	14	14	8.73	14
TR3	14.91	15.5	20.11	19.51	14.91	20.11	20.33	21
TR4	3.73	7.5	7.5	5.55	14	14	6.71	14
TR5	8.13	9.5	10.85	10.02	14	14	11.36	14
TR6	2.03	7.5	7.5	2.78	14	14	4.15	14

Table 4.5: Computational Results for T.R. cities

	$\|LP\|$	$\|LP+C\|$	$\|LP+L\|$	$\|LP+F\|$	$\|LP+P\|$	$\|LP\|$ +All	$\|AR\|$	LB
EU1	8.31	10	11.9	11.72	9	11.91	13	13
EU2	2.08	5	5	3.37	9	9	6.79	9
EU3	6.92	9	9.25	8.88	9	9.27	10.3	11
EU4	1.73	5	5	2.54	9	9	6.36	9
EU5	3.46	6	5.47	4.37	9	9	9	9
EU6	0.87	5	5	1.22	9	9	5.75	9
EU7	20.34	22.5	28.33	28.04	20.34	28.33	28.38	29
EU8	5.09	7.5	8.39	8.16	14	14	8.54	14
EU9	14.41	16	19.37	18.8	14.41	19.38	19.47	20
EU10	3.6	7.5	7.5	5.35	14	14	6.23	14
EU11	7.1	9	9.83	8.93	14	14	10.27	14
EU12	1.77	7.5	7.5	2.46	14	14	4.44	14
EU13	51.91	53.5	74.53	74.34	51.91	74.53	74.38	75
EU14	12.98	17.5	21.97	21.69	23	23	21.5	23
EU15	33.68	35	45.96	45	33.68	45.96	45.72	46
EU16	8.42	12	13.39	12.86	23	23	12.81	23
EU17	20.9	23.5	27.29	25.89	23	27.3	27.03	28
EU18	5.23	12	12	7.15	23	23	7.59	23

Table 4.6: Computational Results for E.U. cities

Chapter 5

Upper Bound

In this chapter, we present two heuristics to find an upper bound to the optimal value. The first one is a Tabu Search based heuristic which starts with an initial solution and finds neighborhood solutions by a local search. The second one is branch-and-price algorithm. In the beginning of the heuristic, we start with the path formulation which contains only one variable for each commodity. At the nodes of the branch-and-bound tree, by pricing of nonbasic variables, new path variables are generated. At the end of this chapter, computational experiments are presented.

5.1 Tabu Search Algorithm

The idea behind the Tabu Search Algorithm is to solve the problem in two steps. First we generate a network topology, and then solve the routing problem for the fixed network topology. The heuristic starts with an initial network topology, and finds neighborhood topologies by a local search. At each iteration, we try to find a feasible routing of commodities for each neighborhood topology and move to the topology with the smallest number of links. In order to prohibit cycling, we put a move in a Tabu list whenever it is applied and we prevent that move to be withdrawn for a specified number of iterations. The algorithm works for

a minimum prespecified number of T iterations. After T iterations it only stops if the best objective value has not improved for U(again a prespeficied number) iterations, else it keeps going until no improvement occurs in the last U iterations.

5.1.1 Initial Topology

First, we have to find some initial topologies to start the algorithm. We have two different procedures to generate initial topologies. The first one is the greedy procedure, and the second one is the random procedure.

In the greedy procedure, first we list the commodities in descending order of their traffic flow. Then, starting from the first element of this list, we try to put a link between the source and destination nodes of the commodity. If the degree of both nodes is less than maximum degree number then we put a link between them. If not, we do not put a link. Clearly, with this procedure, we get a network in which commodities with higher traffic values can be routed using small number of links. Therefore, we may decrease the number of links that have to be used. Following is the formal description of the procedure:

Procedure *Initial Greedy Topology*

Step 1. Order commodities in descending order of traffic value in $DescendingList$

Step 2. Set $Link^{ij} \leftarrow 0 \quad \forall\, (i,j) \in A$

Step 3. Set $degree[i] \leftarrow 0 \quad \forall\, i \in N$

Step 4. Set $counter \leftarrow 1$

Step 4.1. Set $a \leftarrow DescendingList[counter]$

Step 4.2. Set $i \leftarrow s_a$ and $j \leftarrow d_a$

Step 4.3. If $degree[i] < MaxDegree$ and $degree[j] < MaxDegree$ then set $Link^{ij} \leftarrow Link^{ij} + 1$; $Link^{ji} \leftarrow Link^{ji} + 1$; $degree[i] \leftarrow degree[i] + 1$;$degree[j] \leftarrow degree[j] + 1$

Step 4.4. Set $counter \leftarrow counter + 1$. If $counter \leq |K|$, go to Step 4.1. Otherwise terminate the procedure.

Secondly, we generate random initial solutions. We randomly select two nodes each with degree less than the maximum degree number and connect these two nodes by a link. Here, the number of links that will be used is predetermined. If all of the predetermined number of links are used, we stop the procedure. If there is not any pair left whose degrees are less than the maximum degree number, then we also stop the procedure without reaching the predetermined link limit. During the tabu search, we try different predetermined number of links. Thus we can generate random initial topologies with different edge densities. Here is the random procedure:

Procedure *Initial Random Topology* (Input, $NumberOfLinks$)

Step 1. Set $Link^{ij} \leftarrow 0 \quad \forall\,(i,j) \in A$

Step 2. Set $degree[i] \leftarrow 0 \quad \forall\, i \in N$

Step 3. Set $counter \leftarrow 0$

Step 4. Set $S \leftarrow N$

Step 4.1. If $S \leftarrow \emptyset$, go to Step 8. Else, randomly pick an element i from S

Step 4.2. If $degree[i] = MaxDegree$, set $S \leftarrow S \setminus \{i\}$
and go to Step 4.1. Else, go to Step 5.

Step 5. Set $S \leftarrow N \setminus \{i\}$

Step 5.1. If $S \leftarrow \emptyset$, go to Step 8. Else, randomly pick an element j from S

Step 5.2. If $degree[j] = MaxDegree$, set $S \leftarrow S \setminus \{j\}$
and go to Step 4.1. Else, go to Step 6.

Step 6. Set $Link^{ij} \leftarrow Link^{ij} + 1$; $Link^{ji} \leftarrow Link^{ji} + 1$; $degree[i] \leftarrow degree[i] + 1$; $degree[j] \leftarrow degree[j] + 1$

Step 7. Set $counter \leftarrow counter + 1$. If $counter < NumberOfLinks$, go to Step 4. Otherwise go to Step 8.

Step 8. Terminate the procedure

5.1.2 Integer Multicommodity Flow Problem

When we fix the network topology, our problem becomes Capacitated Integer Multicommodity Flow Problem(CIMFP). CIMFP is also a very hard problem.

Since we have to solve routing problem for every neighborhood topology, our approximation algorithm for CIMFP must be very fast.

The main idea of the routing algorithm is to route the commodities one by one over the network. After routing of a commodity, capacities of edges that are in the path of commodity are lowered by the specific flow amount of that commodity. Then, the next commodity is routed with these new edge capacities. In each routing, a shortest path is found only using the edges which have capacity as much as flow of the commodity. While finding shortest paths, we use delay values of edges as distance values. By this way, we ensure that the commodity is routed within the maximum delay value.

In the algorithm, it is easy to see that the order we route the commodities is very crucial. We adopt a greedy approach and route the commodities in the descending order of their flow amounts. It is expected that the commodities with higher flow values will use less number of links in comparison to the commodities with lower flow values in this algorithm. Thus, this will help route all of the commodities with given capacities.

Before giving the description of the algorithm, we state a definition that will be used in the algorithm.

Definiton 1 *Given the network* $G=(N, A)$, $G(f)$ *represents the network* $G'=(N, A')$ *where* A' *is a subset of* A *such that* $\forall(i,j) \in A$ *if* $capacity_{(i,j)} \geq f$ *then* $(i,j) \in A'$, *if* $capacity_{(i,j)} < f$ *then* $(i,j) \notin A'$, *the length of arc* (i,j) *in* A', *is again equal to* d_{ij}.

The algorithm for routing is as follows:

Procedure *Greedy CIMFP*

Step 1. Order commodities in descending order of traffic value in *DescendingList*

Step 2. Set $counter \leftarrow 1$

Step 2.1. Set $a \leftarrow DescendingList[counter]$

Step 2.2. Set $i \leftarrow s_a$; $j \leftarrow d_a$ and $f \leftarrow f_a$

Step 3. Find the shortest path from i to j in $G(f)$, say $dist(i, j)$

Step 4. If no path exists, go to Step 9.

Step 5. If $dist(i, j) \geq MaxDelay$, go to Step 9.

Step 6. Decrease the capacities of edges on the shortest path by f.

Step 7. Set $counter \leftarrow counter + 1$. If $counter > |K|$ then go to Step 8. Else go to Step 2.1.

Step 8. Set $Feasible \leftarrow 1$ and terminate

Step 9. Set $Feasible \leftarrow 0$ and terminate

After generating initial topologies, we check whether there exists a feasible routing of commodities in this topology or not by our Greedy CIMFP procedure. If there is a feasible routing, that means we have an initial solution to our problem. Now, we can start Tabu Search with this feasible solution.

5.1.3 Neighborhood Search

Whenever we generate an initial solution, the main procedure of Tabu Search is started. In the main procedure, we simply do a local search. We find neighborhood topologies to the current topology and check whether there is a feasible routing of commodities. Among the feasible neighborhood topologies, we move to one with minimum number of links. We used four different methods to find neighborhood topologies. These are Link Addition, Link Deletion, Link Interchange and Triangle Interchange.

5.1.3.1 Link Addition

First method of generating neighborhood topology is the Link Addition Method. In this method, we find the nodes whose degrees are less than the maximum degree number. Every pair of nodes in this list is a candidate to join with a link. If adding a link between these pairs is not in the Tabu List of Link Addition, we add the link and find a neighborhood topology. Since we select the nodes whose degrees are not at maximum degree number, in the network generated

after addition of the link, the degree constraint is satisfied for all nodes. For every new network generated by Link Addition is added to Candidate Move List. We do not have to solve routing problem from scratch. Routing solution of the original topology is also feasible for these generated topologies.

5.1.3.2 Link Deletion

The second method that is proposed to find neighborhood topology is the Link Deletion Method. Every link in the current network topology is a candidate to delete. For every link in the network, first we control that whether this move is in the Tabu List of Link Deletion or not. If not, we delete the link and find a neighborhood topology. Since we delete the links, the degree constraint is satisfied. For every new network generated by Link Deletion, we call Greedy CIMFP procedure to find a feasible routing. If such routing is found, then this move is added to Candidate Move List.

5.1.3.3 Link Interchange

The third method is the Link Interchange Method. In this method, we first find a pair of disjoint links. Two links are called disjoint if their endpoints do not intersect. Let L_1 and L_2 be a pair of disjoint links, L_1 connects node h_1 and node t_1 and L_2 connects node h_2 and node t_2. Then $h_1 \neq h_2$, $h_1 \neq t_2$, $t_1 \neq h_2$ and $t_1 \neq t_2$. Link interchange method is applied in two ways. In the first one, we delete the given pair of the links and add two links that connects the heads and tails of L_1 and L_2. In other word, we put links on edges (h_1, h_2) and (t_1, t_2). In the second one, we delete the given pair of the links and add two links that connects the head(tail) of L_1 and the tail(head) of L_2. In other words, we put links on edges (h_1, t_2) and (t_1, h_2). Both of these interchanges are candidates of Link Interchange for L_1 and L_2 pair. If this pair of disjoint links is not in the Tabu List of Link Interchange, we apply these two interchange methods and generate two neighborhood topologies. After deleting L_1 and L_2, we put two new edges. Note that degrees of nodes h_1, t_1, h_2 and t_2 do not change. So degree constraint

is automatically satisfied. For every new network generated by Link Interchange, we call Greedy CIMFP procedure to find a feasible routing. If such a routing is found, then this move is added to Candidate Move List.

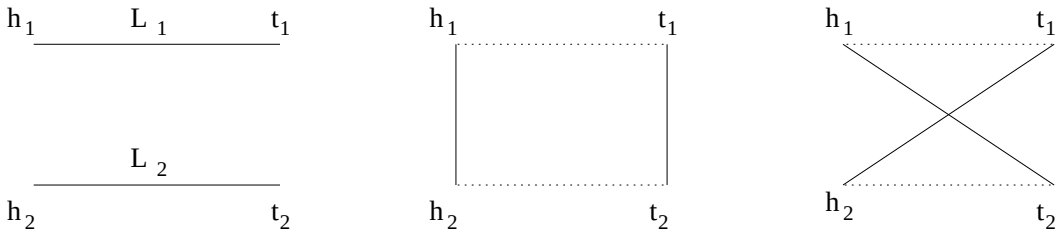

Figure 5.1: Link Interchange

5.1.3.4 Triangle Interchange

The final method of generating neighborhood topologies is the Triangle Interchange Method. In this method, we first find two links which share a common node. Let L_1 and L_2 be a pair of links, such that L_1 connects node h and node t_1 and L_2 connects node h and node t_2. Every (L_1,L_2) pair is a candidate for Triangle Interchange. In Triangle Interchange method, we delete these two links and we add a link that connects the tails of these links. In other words, we delete links L_1 and L_2 and add a link that connects t_1 and t_2. By the Triangle Interchange Method, the degrees of tail nodes do not change and the degree of the common head node decreases by 2. For every new network generated by Triangle Interchange, we call Greedy CIMFP procedure to find a feasible routing. If such a routing is found, then this move is added to Candidate Move List.

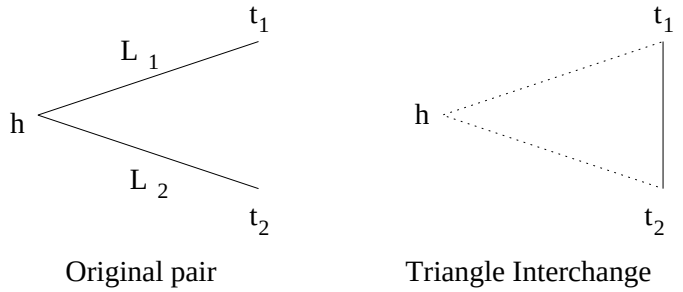

Figure 5.2: Triangle Interchange

5.1.4 Tabu Search

Whenever we have an initial feasible solution, $BestFound$ is set to the objective value of this solution and then Tabu Search procedure is started. For each iteration, neighborhood search is made. After all possible moves found are added to the Candidate Move List, we select the move from this list with the minimum number of links, $CostOfMove$, (ties are breaked randomly) and apply this move to the current topology permanently. If the $CostOfMove$ of the applied move is less than the $BestFound$, than $BestFound$ is updated and counter of improvement is reset. After applying this move, we put it into the Tabu List. We have 3 tabu lists. These are Tabu List of Link Addition, Tabu List of Link Deletion and Tabu List of Link Interchange.

Tabu Lists work as follows: Whenever a new link is added to network, this link is put into Tabu List of Link Deletion. So deletion of this link is prohibited. Whenever a new link is deleted from network, this link is put into Tabu List of Link Addition. So addition of this link is prohibited. Whenever a link interchange is made, these two links are put into Tabu List of Link Interchange. And, a link interchange including all the 4 corresponding nodes of these two links is prohibited. Whenever a triangle interchange is made, both of the deleted links are put into Tabu List of Link Addition and added link is put into Tabu List of Link Deletion. Otherwise, algorithm may add the deleted links, delete the added link and apply this Triangle Interchange again which causes cycling.

For how many iterations a move will be held in the Tabu Lists, i.e. Tabu Tenure, is important for efficiency of the algorithm. A move is keep in a Tabu List, to prevent cycling, otherwise same moves will take place repeatedly, and this blocks us from finding other potentially good solutions. If we take a large Tabu Tenure, then we block some moves and decrease the chance of finding a better solution. A small Tabu Tenure will increase the risk of cycling. During our computations, we observed that taking Tabu Tenure equal to 10 is a good choice.

Another parameter is the number of iterations that is made before stopping Tabu Search. Number of iterations increases the chance of finding a better solution at the expense of increasing the running time of the algorithm. In our algorithm, Tabu Search makes T iterations and after T iterations it still continues if $BestFound$ value has improved in the past U iterations.

In Figure 5.3, the flowchart of the Tabu Search procedure can be seen and the formal description of the procedure is as follows:

Procedure *Tabu Search*

Step 1. Set $counter \leftarrow 1$

Step 2. Disjoint Pair List

Step 2.1. Look at every link pair. If this pair is a candidate for Link Intercange and not in the $TabuListOfLinkInterchange$ than put this pair into Disjoint Pairs List. If List is empty go to Step 3.

Step 2.2. Take a pair from Disjoint Pairs List and remove this pair from the list.

Step 2.3. Apply Link Interchange 1.

Step 2.4. Call Greedy CIMFP

Step 2.5. If $Feasible = 1$, set $CostOfMove$ to the number of links and add this move to $CandidateMoveList$.

Step 2.6. Reverse the changes done in Step 2.3.

Step 2.7. Apply Link Interchange 2.

Step 2.8. Call Greedy CIMFP

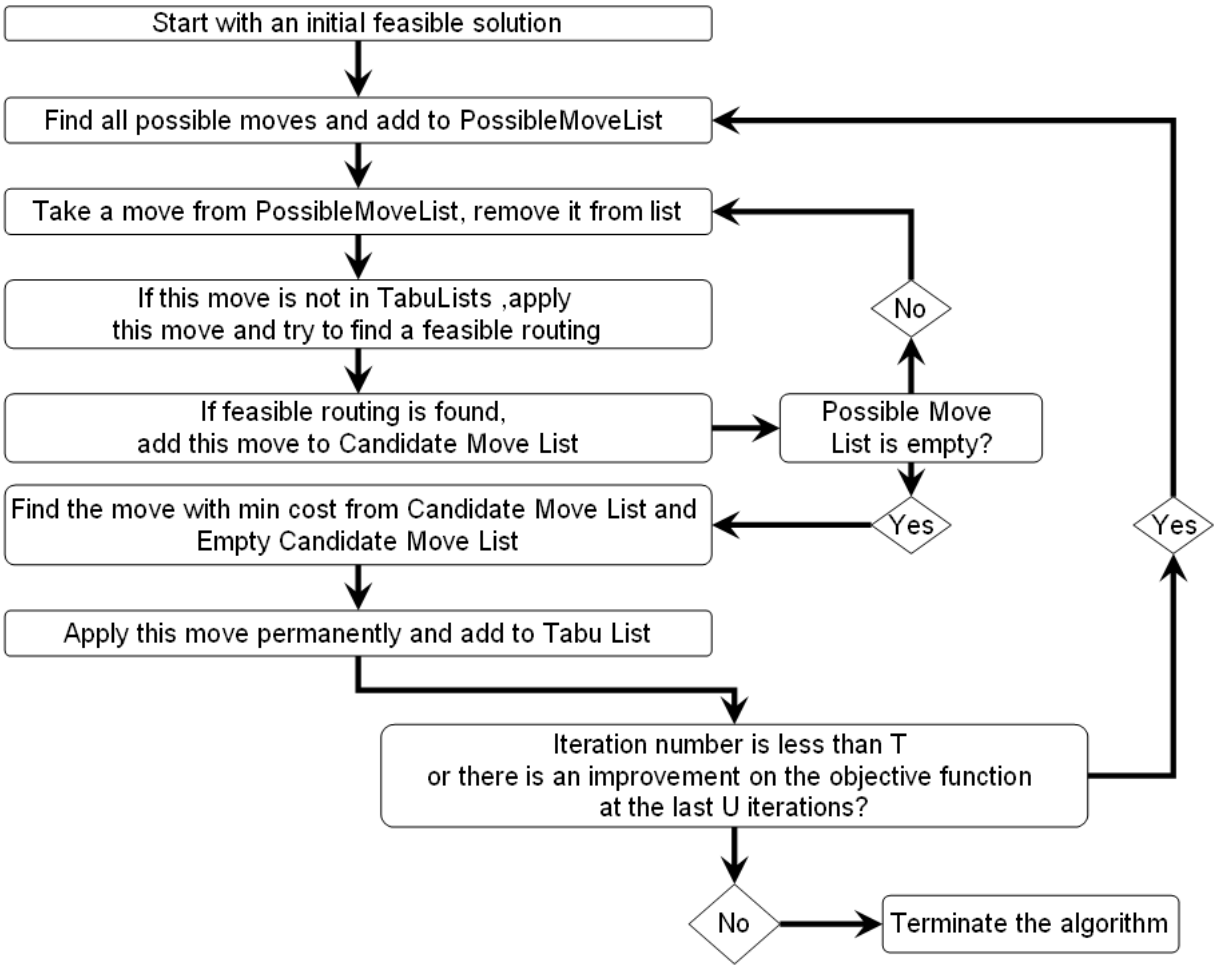

Figure 5.3: Tabu Search Procedure

Step 2.9. If $Feasible = 1$, set $CostOfMove$ to the number of links and add this move to $CandidateMoveList$.

Step 2.10. Reverse the changes done in Step 2.7.

Step 2.11. If Disjoint Pairs List is empty then go to Step 3. Else go to 2.2.

Step 3. Addition List

Step 3.1. Look at every node pair. If this pair is a candidate for Link Addition and not in the $TabuListofLinkAdditon$ than put this node pair into Addition List. If List is empty go to Step 4.

Step 3.2. Add all of the moves in Addition List to the $CandidateMoveList$, set $CostOfMove$ of each move to the number of links used.

Step 3.7. Empty the Addition List and go to Step 4.

Step 4. Deletion List

Step 4.1. Look at every edge. If this edge is a candidate for Link Deletion and not in the $TabuListofLinkDeletion$ than put this node pair into Deletion List. If List is empty go to Step 5.

Step 4.2. Take a link from Deletion List and remove this link from the list.

Step 4.3. Delete the link.

Step 4.4. Call Greedy CIMFP

Step 4.4. If $Feasible = 1$, set $CostOfMove$ to the number of links and add this move to $CandidateMoveList$.

Step 4.5. Reverse the changes done in Step 4.3.

Step 4.6. If Deletion List is empty then go to 5. Else go to 4.2.

Step 5. Triangular Pairs List

Step 5.1. Look at every link pair. If this pair is a candidate for Link Interchange, than put this pair into Triangular Pairs List. If List is empty go to Step 6.

Step 5.2. Take a pair from Triangular Pairs List and remove this pair from the list.

Step 5.3. Apply Triangle Interchange

Step 5.4. Call Greedy CIMFP

Step 5.5. If $Feasible = 1$, set $CostOfMove$ to the number of links and add this move to $CandidateMoveList$.

Step 5.6. Reverse the changes done in Step 5.3.

Step 5.7. If Triangular Pairs List is empty then go to 6. Else go to 5.2.

Step 6. Select the move with smallest $CostOfMove$ from $CandidateMoveList$ (ties are breaked randomly), apply this move to the current topology permanently.
Step 7. Add this to the corresponding $TabuList$ and update the $TabuList$s.
Step 8. If $CostOfMove < BestFound$ then set $BestFound \leftarrow CostOfMove$ and $ImproveCounter \leftarrow 0$
Step 9. Set $counter \leftarrow counter+1$ and $ImproveCounter \leftarrow ImproveCounter+1$
Step 10. Empty the $CandidateMoveList$.
Step 11. If $counter < T$, then go to Step 2. Else go to Step 12.
Step 12. If $ImproveCounter < U$, then go to Step 2. Else go to Step 13.
Step 13. Terminate the procedure.

5.1.5 Main Algorithm

At the begining of the algorithm, $TotalBestFound$ is set to a big number, $BigM$. For each initial solution, Tabu Search procedure is applied and a $BestFound$ value is obtained. If $BestFound$ value found is less than the $TotalBestFound$, than $TotalBestFound$ is updated. At the end of the algorithm, $TotalBestFound$ gives the result of the Tabu Search heuristic.

The number of initial random solutions to be generated is also a parameter of our algorithm. The more the number of initial random solutions, the more the chance of finding a better solution, but again at the expense of increasing the running time of the algorithm. In our algorithm, we generate random initial topologies with different edge densities. $NumberOfLinks$, that is used during generating random initial topologies, is calculated by multiplying number of nodes, $|N|$, by the edge density, number of links per node($AverageLink$). $AverageLink$ takes values between $MinAverageLink$ and $MaxAverageLink$. For each $AverageLink$ value, we generate $TotalRepNumber$ random initial solutions.

The main algorithm is as follows:

Procedure *Main Algorithm*

Step 1. Read the Traffic Values and Delay Values from files

Step 2. Make initializations and set $TotalBestFound \leftarrow BigM$

Step 3. Initial Greedy Topology

Step 3.1 Call Initial Greedy Topology and obtain $Link^{ij}$ values

Step 3.2. Set $capacity_{ij} \leftarrow c * Link^{ij} \quad \forall\,(i,j) \in A$

Step 3.3. Call Greedy CIMFP

Step 3.4. If $Feasible = 1$, set $BestFound$ to the number of links and call Tabu Search.

Step 3.5. If $BestFound < TotalBestFound$ then set $TotalBestFound \leftarrow BestFound$

Step 4. Initial Random Topology

Step 4.1. Set $AverageLink \leftarrow MinAverageLink$

Step 4.2. Set $NumberOfLinks \leftarrow AverageLink * |N|$

Step 4.3. Set $RepNumber \leftarrow 1$

Step 4.4. Call Initial Random Topology and obtain $Link^{ij}$ values

Step 4.5. Set $capacity_{ij} \leftarrow c * Link^{ij} \quad \forall\,(i,j) \in A$

Step 4.6. Call Greedy CIMFP

Step 4.7. If $Feasible = 1$, set $BestFound$ to the number of links and call Tabu Search. Else go to Step 4.4.

Step 4.8. If $BestFound < TotalBestFound$ then set $TotalBestFound \leftarrow BestFound$

Step 4.9. Set $RepNumber \leftarrow RepNumber + 1$

Step 4.10. If $RepNumber < TotalRepNumber$ go to Step 4.4.

Step 4.11. Set $AverageLink \leftarrow AverageLink + 1$

Step 4.12. If $AverageLink < MaxAverageLink$ go to Step 4.3.

Step 5. Terminate Algorithm

5.2 Branch-and-Price Algorithm

In the Branch-and-Price Algorithm, we present a column generation model using path formulation. Branch-and-price, a generalization of branch-and-bound with LP relaxations, allows column generation to be applied throughout the branch-and-bound tree. Since only one path will be chosen for each commodity, most of the path variables will be equal to the zero in the optimal solution. Sets of columns are left out of the LP relaxation because there are too many columns to handle efficiently and they can be easily generated by the pricing problem.

5.2.1 Pricing

In the pricing problem, LP relaxation of the problem is solved to try to identify columns to enter the basis. In other words, we try to find whether there are any columns not included in the current LP with negative costs. If such columns exist, they are added to the formulation. The pricing problem is actually a separation problem for the dual of LP. Sets of columns, that are left out of the LP, correspond to the constraints of the dual of LP.

For the path formulation, let $-\pi_{ij}$ represent the nonnegative dual variables associated with constraints 3.9 and σ^k represent the unrestricted dual variables associated with constraints 3.8. Then, the reduced cost of column p for commodity k, denoted $\bar{c}_p^k$ is:

$$\bar{c}_p^k = \sum_{(i,j)\in A} f^k \delta_{ij}^p \pi_{ij} - \sigma^k \qquad \forall p \in P^k \quad \forall k \in K \tag{5.1}$$

Columns that price out can be identified by solving one shortest path problem for each commodity $k \in K$ over a network with arc costs equal to the π_{ij} for each $(i,j) \in A$. Denote the cost of shortest path $p*$ for any commodity k as c_{p*}^k. Then, if for all $k \in K$,

$$f^k c^k_{p*} - \sigma^k \geq 0$$

the LP relaxation is optimal. Otherwise, the LP relaxation is not optimal and for each $k \in K$ with

$$f^k c^k_{p*} - \sigma^k < 0$$

path $p* \in P^k$ has a negative reduced cost and path $p*$ is added to the formulation. At the end of the pricing procedure new variables are generated by implicit pricing of nonbasic variables or LP optimality is proved.

5.2.2 Branching on link variables

At the nodes of the branch-and-bound tree, first the pricing problem is solved to identify new columns to enter the basis. Whenever we can not find any path variable that prices out and enters the basis, branching occurs if the LP solution does not satisfy the integrality conditions. First, we check the integrality of link variables. If there are some link variables that are not integral, we start branching on the one having the biggest value in the optimum solution of the LP relaxation. We use conventional integer branching on link variables. Let $Y_{ij}*$ be the value of link variable that has the biggest nonintegral optimum value in the LP relaxation. One branch forces $Y_{ij} \geq \lceil Y_{ij}* \rceil$, and the other branch forces $Y_{ij} \leq \lfloor Y_{ij}* \rfloor$. The first branch is easy to enforce because we increase the capacity installed on the edge $\{i, j\}$. But the latter branch can not be enforced if some commodities that use this link have no other path variables in the current formulation. This may cause infeasibility due to not having enough path variables in the formulation. We solve this problem by adding new path variables for such commodities. To do this, we find the shortest path using the current dual variables π_{ij} without requiring the cost of this path to be less than σ^k.

In the branching procedure, whenever we set a link variable equal to the 0, we delete all of the path variables that use this corresponding edge. Since those path variables will not be in the basis anymore. By this way, we ensure that the number of path variables does not increase exponentially and maintain the rapidness of the algorithm.

5.2.3 Branching on path variables

If all of the link variables are integral then we start branching on the path variables. Conventional integer branching on the path variables may not be effective because fixing variables can destroy the structure of the pricing problem. To illustrate, consider branching based on variable dichotomy in which one branch forces $X_p^k = 1$ and the other branch forces $X_p^k = 0$. The first branch is easy to enforce because commodity k is assigned to a path p and no additional paths need to be generated anymore. The latter branch cannot be enforced if shortest path problem is used for pricing. There is no guarantee that shortest path problem will find a path other than p. Otherwise we have to solve a next shortest path procedure for pricing new path variable. In general, pricing problem involving a set of branching decisions must be solved using a kth shortest path procedure.

Our branching rule works as follows: we look for all split commodities, in other words all commodities that are assigned to more than one path. We choose the one having the highest traffic value, say commodity k. Let $p1$ and $p2$ be the paths that have greatest fractions of the flow of commodity k. Then these two paths differ on at least two arcs. Assume path $p1$ contains arc $a1$ and $p2$ does not and in the same way, path $p2$ contains arc $a2$ and $p1$ does not. We branch creating two subproblems in which on the first branch we do not allow commodity k to use the arc $a1$ and on the second branch we do not allow commodity k to use the arc $a2$. By setting the commodity's cost on the forbidden arc to a very high value, the pricing problem can still be solved using the shortest path algorithm.

5.2.4 Main Algorithm

The algorithm works as follows: We start with an initial feasible solution to our problem and write the path formulation with only corresponding path variables of the initial feasible solution. Thus, our model contains only one variable for each commodity at the root node. In the formulation, constraints (3.11) are left out of the LP since we start with a feasible solution in which each of the path variables satisfies the delay constraint. During the pricing, when a path is found which does not satisfy the delay constraints it will not be added to the formulation. Therefore all the path variables used during the algorithm automatically satisfies the delay constraint. So constraints (3.11) become redundant.

After writing the initial formulation of the problem, we will start the pricing procedure. If such columns are found, the LP is reoptimized. Pricing process is repeated until such columns do not exist. Then we check whether the LP relaxation solution is integral. If not, we start the branching procedure. First we branch on link variables. If all of the link variables are integral then we start branching on the path variables.

In Figure 5.4, the flowchart of the Branch-and-Price algorithm can be seen and the formal description of the algorithm is as follows:

Procedure *Branch-and-Price Algorithm*

Step 1. Formulate initial path formulation using a given feasible solution

Step 2. Call pricing procedure

Step 3. Check Integrality of link variables. If all of the link variables are integral go to Step 4. If not branch on the variable with has the maximum nonintegral value. Then go to Step 2.

Step 4. Check Integrality of path variables. If all of the path variables are integral go to Step 5. If not, call the branching procedure for path variables. Then go to Step 2.

Step 5. State this solution as a feasible solution. If objective value is less than the $BestValueFound$, update the $BestValueFound$. If no nodes are left for branching, i.e., all variables are integral, terminate the algorithm, else go to Step 2.

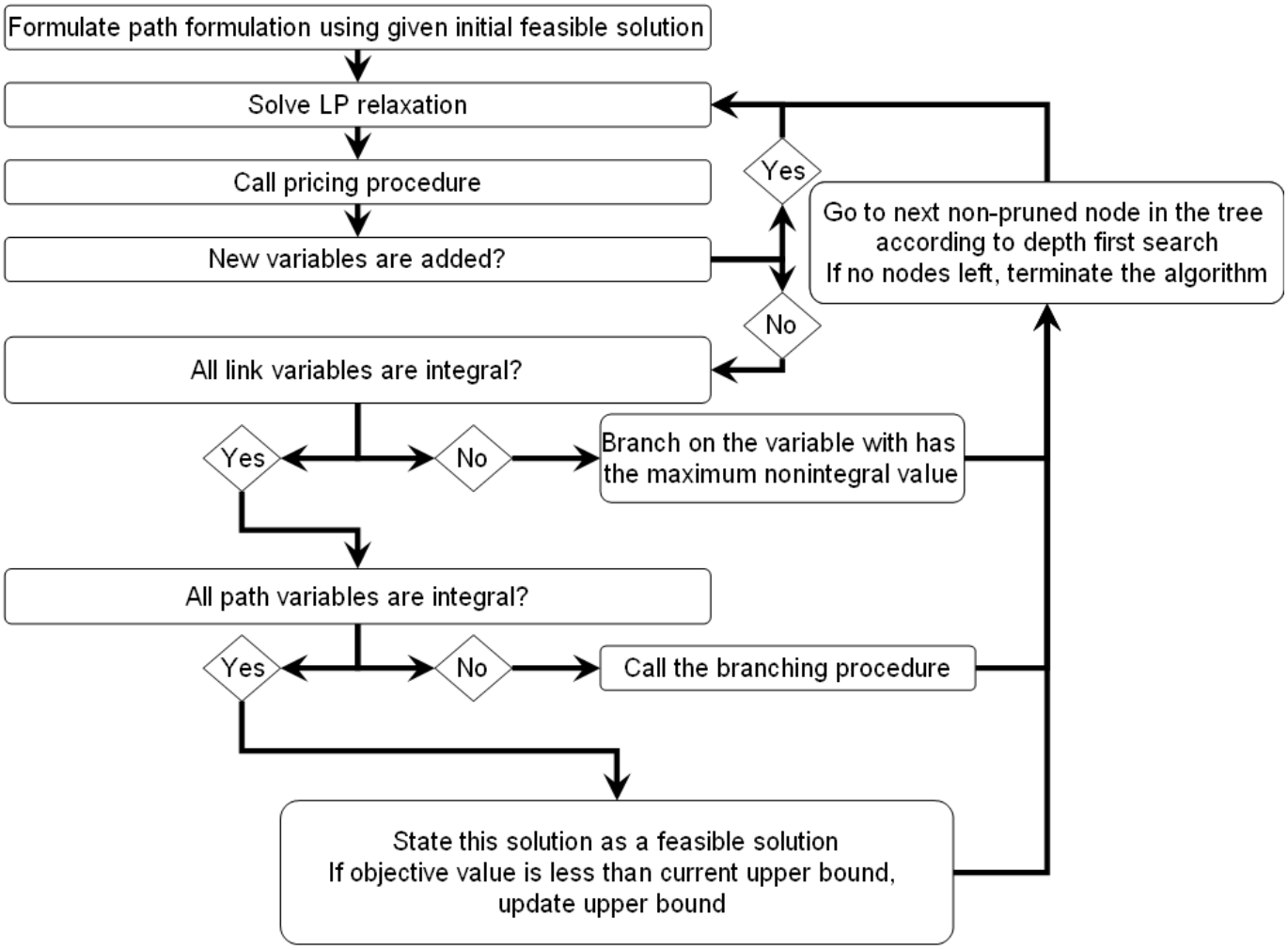

Figure 5.4: Branch-and-Price Algorithm

5.3 Computational Experiments for Our Upper Bound

In our computational experiments, we first apply Tabu Search algorithm for each test problem. Then, the solution of Tabu Search algorithm is used as an initial solution to the Branch-and-Price algorithm. The objective function of this solution is also used as an upper bound for the Branch-and-Price algorithm to prune the nodes. The Branch-and-Price algorithm is very time consuming, we expect to lower this running time by pruning some nodes with this upper bound value. If the solution of Tabu Search algorithm happens to be optimal(has the same objective with the corresponding lower bound), then we do not run Branch-and-Price algorithm for this test problem.

In Tabu Search algoritm, we take $MinAverageLink$ value equal to 2, $MaxAverageLink$ value equal to 4 and $TotalRepNumber$ value equal to 10. Thus, we generate 10 initial solutions for $AverageLink = 2$, 10 initial solutions for $AverageLink = 3$ and 10 initial solutions for $AverageLink = 4$. With 1 greedy initial solution, we start Tabu Search procedure 31 times for each test problem. But in some instances, we can not find a feasible routing for greedy initial topologies. In the random initial topologies, we continue to generate new random topologies until finding a feasible routing for that topology. But for some instances, it is infeasible to find a feasible routing when we set $AverageLink$ equal to 2 or 3. For example, lower bound we found is 53 for the test problem US13 which has 20 nodes. The initial topology we want to generate can have at most 40 nodes if we set $AverageLink$ equal to 2, which is less than lower bound we found. So for these instances we set $MinAverageLink$ equal to 3 or 4. But we ensure that we have at least 10 initial solutions for each test problem.

We set T equal to 100 and U equal to 30. Thus the algorithm works for a minimum number of 100 iterations. After 100 iterations it only stops if the best objective value has not improved for the last 30 iterations, else it keeps going until no improvement occurs in the last 30 iterations.

Our solution attempts showed that the running time of Branch-and-Price algorithm is not reasonable. So we set a time limit for the running time of the algorithm and this value is 12 hours. If the algorithm does not end in 12 hours, we stop the algorithm and take the feasible solution with minimum objective value. In most of cases, Branch-and-Price algorithm can not improve the value found by the Tabu Search algorithm. But for a few instances, best found objective value is improved and this further reduced the gap between lower and upper bound. For only one test problem(EU13), both of the algorithms could not find a feasible solution. So we doubt that this problem may be infeasible.

The computational results are presented in Tables 5.1-3 below. The first column expresses the lower bound attained in Chapter 4. Tabu column represents the results of the Tabu Search algorithm, and B&P column represents the results of the Branch-and-Price algorithm. And the last column shows the percentage difference between the best value found and the lower bound.

Our algorithms were implemented in C and executed on a computer equipped with Intel Celeron 2.80 GHz processor and Red Hat Linux 3.2.2.5. CPU times of algorithms in seconds are given at the columns following the result columns of algorithms, respectively. In the Branch-and-Price algorithm, LP relaxations are solved by Cplex 8.1. Also the lower bounds are obtained by Cplex 8.1.

It is observed that, the gap between lower and upper bound generally increases as the number of nodes increases. Because as the node size increases, the problem gets harder and Aggregated Relaxation does not improve the lower bound found by adding inequalities to the LP relaxation of the original formulation. Nearly in all cases the gap between lower and upper bounds when low capacity lightpaths are used is greater than the gap when high capacity lightpaths are used. This is expected since low capacity lightpaths result in more number of lightpaths when compared with high capacity ones.

	LB	Tabu	CPU	B&P	CPU	Gap(%)
US1	14	15	25	X[1]	710	7.14
US2	9	9	4	OPT[2]	-	0.00
US3	11	12	10	X	71	9.09
US4	9	9	2	OPT	-	0.00
US5	9	9	2	OPT	-	0.00
US6	9	9	2	OPT	-	0.00
US7	29	35	673	X	43200	20.69
US8	14	16	81	X	910	14.29
US9	19	24	2184	23	43200	21.05
US10	14	14	29	OPT	-	0.00
US11	14	17	38	16	43200	14.29
US12	14	14	15	OPT	-	0.00
US13	53	64	19494	X	43200	20.75
US14	19	25	815	X	43200	31.58
US15	38	49	1120	X	43200	28.95
US16	19	22	295	X	43200	15.79
US17	19	28	466	X	43200	47.37
US18	19	19	89	OPT	-	0.00

[1]No improved solution found
[2]Tabu Search is optimal. B&P algorihtm is not executed.

Table 5.1: Algorithm Results of U.S. cities

	LB	Tabu	CPU	B&P	CPU	Gap(%)
TR1	30	36	1174	X	43200	20.00
TR2	14	16	64	X	2318	14.29
TR3	21	26	141	25	43200	19.05
TR4	14	14	31	OPT	-	0.00
TR5	14	17	38	16	43200	14.29
TR6	14	14	16	OPT	-	0.00

Table 5.2: Algorithm Results of T.R. cities

	LB	Tabu	CPU	B&P	CPU	Gap(%)
EU1	13	15	26	X	5779	15.38
EU2	9	9	4	OPT	-	0.00
EU3	11	12	12	X	95	9.09
EU4	9	9	3	OPT	-	0.00
EU5	9	9	2	OPT	-	0.00
EU6	9	9	2	OPT	-	0.00
EU7	29	35	605	X	43200	20.69
EU8	14	15	79	X	886	7.14
EU9	20	26	174	25	43200	25.00
EU10	14	14	38	OPT	-	0.00
EU11	14	16	37	X	9002	14.29
EU12	14	14	19	OPT	-	0.00
EU13	75	X	-	X	-	-
EU14	23	35	2780	X	43200	52.17
EU15	46	61	3293	X	43200	32.61
EU16	23	27	921	X	43200	17.39
EU17	28	41	1263	39	43200	39.29
EU18	23	24	260	X	43200	4.35

Table 5.3: Algorithm Results of E.U. cities

Chapter 6

Conclusion

In this thesis, we studied the problem of logical network design in telecommunications networks. The problem contains location of lightpaths and routing of commodities over these lightpaths. Our aim is to find a feasible network topology and routing of commodities over that topology while minimizing the number of lightpaths used. Besides the usual flow conservation constraints, we have bandwidth capacity constraints, degree constraints and delay constraints.

First we introduced three different formulations of the problem, namely Integer Programming Formulation, Aggregated Formulation and Path Formulation. After that, we considered four sets of valid inequalities: Cutset Inequalities, Link Inequalities, Flux Inequalities and Partition Inequalities. To obtain a lower bound to our problem, we give a relaxation of Aggregated Formulation. Then, some computational experiments are done in order to observe the effects of these inequalities and relaxation of Aggregated Formulation to lower bound of original formulation. Link Inequalities are found to be more powerful than other inequalities. Partition Inequalities dominate when capacities are high and number of commodities is less. In small instances, relaxation of Aggregated Formulation improved the lower bound found by adding all of four inequalities.

The problem is very difficult to solve to optimality. Therefore we developed

two heuristic approaches to the problem. The first one is the Tabu Search algorithm. Tabu Search tries to solve the problem in two steps. First a network topology is generated, and then the commodities are routed over this fixed topology. The algorithm starts with initial feasible solutions and at each iteration it makes a local search to find neighborhood topologies and moves to a neighboring topology with minimum number of links. For a given network topology, finding the optimum routing of commodities is also a very hard problem. This problem is named as Integer Multicommodity Network Flow Problem. A fast heuristic solution procedure is applied to find a feasible routing for a fixed topology. This procedure is used as a subroutine for our Tabu Search Algorithm.

The second algorithm we proposed is a Branch-and-Price algorithm. As a column generation model, path formulation of the problem is used and sets of columns are left out of the formulation. A pricing procedure is applied throughout the Branch-and-Bound tree to generate new columns. In the pricing procedure, LP relaxation of the problem is solved and the columns that are not included in current LP with negative costs are found and added to the LP relaxation. After adding new variables, LP is reoptimized and pricing procedure is repeated. Whenever pricing procedure stops finding any new columns, branching is applied. First we branch on link variables. If link variables are integral, we apply a special branching procedure for path variables that preserves the structure of the pricing procedure.

The proposed algorithms are tested on several test problems with different characteristics. First we apply Tabu Search algorithm for test problems. The solution of the Tabu Search algorithm is used an initial solution for the Branch-and-Price algorithm, and the objective value of this solution is used in the pruning of nodes. The results are compared with the lower bounds found.

A further research avenue can be to use cutting plane techniques or Lagrangian heuristics. Rounding techniques and heuristics that use the optimal solution of the LP may be investigated. For example, locating only a few lightpaths by rounding optimal solution of LP relaxation and reoptimizing the LP may be an alternative way of generating feasible solutions. In the Tabu Search algorithm, a

better method of routing commodities may be used for IMNFP. In the Branch-and-Price algorithm, new branching rules that provide faster convergence may be found. In this thesis, we assumed only one type of technology can be installed on the edges. A further research topic can be studying network design problems with two types of technologies. In this thesis, we take the location cost of every lightpath as the same regardless of which node pair is connected. But, in some cases cost of locating a ligthpath may depend on the distance between node pair, therefore we may have different location costs for possible links.

Bibliography

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, N.J., 1993.

[2] F. Alvelos and J. V. de Carvalho. Comparing branch-and-price algorithms for the unsplittable multicommodity flow problem. In *International Network Optimization Conference*, 2003.

[3] F. Barahona. Network design using cut inequalities. *SIAM Journal on Optimization*, 6:823–837, 1996.

[4] C. Barnhart, C. Hane, E. Johnson, and G. Sigismondi. A column generation and partitioning approach for multicommodity flow problems. *Telecommunications Systems*, 3:239–258, 1995.

[5] C. Barnhart, C. Hane, and P. H. Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48:318–326, 2000.

[6] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[7] C. Barnhart and Y. Sheffi. A network-based primal-dual heuristic for the solution of multicommodity network flow problem. *Transportation Science*, 27(2):102–117, 1993.

[8] D. Bienstock and O. Günlük. Computational experience with a difficult mixed integer multicommodity flow problem. *Mathematical Programming*, 68:213–237, 1995.

[9] G. Dahl, A. Martin, and M. Stoer. Routing through virtual paths in layered telecommunications networks. *Operations Research*, 47:693–702, 1999.

[10] B. Gendron and T. G. Crainic. Relaxations for multicommodity capacitated network design problems. In *Publication CRT-965*, 1994.

[11] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[12] İ.Ç. Kepek. Topology design in communication networks. Master's thesis, Bilkent University, 2003.

[13] E. Karasan, O. Ekin-Karasan, N. Akar, and M. Pinar. mesh topology design in overlay virtual private networks. *Electronics Letters*, 38(16):939–941, 2002.

[14] T. L. Magnanti, P. Mirchandani, and R. Vachani. Modeling and solving the two facility capacitated network loading problem. *Operations Research*, 43(1):142–157, 1995.

[15] T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, 18(1):1–55, Feb. 1984.

[16] M. Minoux. Network synthesis and optimum network design problems: Models, solution methods and applications. *Network*, 19:313–360, 1989.

[17] R. Ramaswami and K. Sivarajan. Design of logical topologies for wavelength-routed optical networks. *IEEE Journal on Selected Areas in Communications*, 14:840–851, 1996.

[18] T. J. V. Roy and L. A. Wolsey. Valid inequalities and seperation for uncapacitated fixed charge networks. *Operations Research Letters*, 4:105–112, 1985.