TOPOLOGY DESIGN IN COMMUNICATION NETWORKS

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

İlktuğ Çağatay Kepek

July 2003

ABSTRACT

TOPOLOGY DESIGN IN COMMUNICATION NETWORKS

İlktuğ Çağatay Kepek
M.S. in Industrial Engineering
Supervisor: Assoc. Prof. Mustafa Ç. Pınar
July 2003

In this thesis, we study the topology design problem in communication networks. It is the problem of a Virtual Private Network(VPN) provider. Given a set of customer nodes and a set of commodities, we aim to locate links between customer nodes and route the commodities over these links. The cost to be minimized is the sum of location and routing costs. The problem has capacity, degree and delay constraints. An important characteristic of the problem is that the commodities cannot be split, therefore they must be routed over single paths.

We present an integer programming formulation of the problem and introduce two sets of valid inequalities. The problem has two parts: locating links and routing commodities. We first analyze the commodity routing problem and propose an efficient heurisric for it. Finally we propose a heuristic method of generating good feasible solutions to our problem. The final heuristic is a Tabu Search which uses the first heuristic proposed for routing problem as a subroutine. Our results prove to be closer to the lower bounds we generate than previously proposed heuristics.

Keywords: Network Topology Design, Integer Multicommodity Flow Problem, Tabu Search, Capacitated Network Design.

ÖZET

İLETİŞİM AĞLARINDA YERLEŞKE TASARIMI

İlktuğ Çağatay Kepek
Endüstri Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç.Dr. Mustafa Ç Pınar
Temmuz 2003

Bu tezde İletişim Ağlarıda Yerleşke Problemi üzerinde çalışıldı. Çalıştığımız problem Sanal Özel Ağ(VPN) sağlayıcısının problemidir. Müşteri düğümleri kümesi ve bu düğümler arasındaki trafik verildiği halde, yerleştirme ve akım gönderme maliyetlerinin toplamını en azlamayı amaçladık. Problemimizin kapasite, derece ve gecikme kısıtları vardır. Düğümler arasındaki trafiğin farklı yollara dağıtılamaması da problemimizin bir başka özelliğidir.

Problemin tamsayı programlama modelini verdikten sonra iki farkli geçerli eşitsizlik sunduk. Trafiğin yollanması için etkili bir sezgisel yöntem önerdik. Ana problemimiz içinse Tabu Araması yapan bir sezgisel yöntem geliştirdik. Sonuçlarımız ürettiğimiz alt sınırlara yakındır ve daha önce önerilen metodlara göre daha iyidir.

Anahtar sözcükler: Ağ Yerleşke Tasarımı,Tamsayı Çoklu Akım Problemi, Tabu Araması, Kapasiteli Ağ Tasarımı.

Aileme...

Acknowledgement

I would like to express my deepest gratitude to Assoc. Prof. Mustafa Ç. Pınar for his encouragement and instructive comments in the supervision of the thesis.

I am grateful to Asst. Prof. Oya Karaşan for her guidance, patience and great helps.

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

I am also indebted to Asst. Prof. Nail Akar for his suggestions and remarks.

I would like to express my deepest thanks to Halil Şekerci for his countinuous support and friendship. He always brings new ideas and forces me to think. I feel very lucky for having such a brillant friend.

I would like to extend my sincere thanks to Aykut Özsoy for his keen friendship and helps. I can not forget his patience and support during this study.

I am also grateful to Savaş Çevik and Güneş Erdoğan for their technical supports.

I would like to express my special thanks to Erdoğan Solak, Bilgin Kalebek and Ozan Çakır for their keen friendship.

Finally, I would like to express my gratitude to my family; Efraim Sami Kepek, Şükran Kepek and Müge Kepek for their patience, love and understanding not only for the time of this study but for my lifetime.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Background

In the age of knowledge, “Internet” is an ideal platform for information retrieval and exchange. Because it is cost effective, easy to use, and most importantly, highly accessible. All of these attributes of Internet, make it the information infrastructure of our age. Besides its advantages, Internet is vulnerable to practises that endanger security of data exchanges. Hence, several companies are now offering ”Virtual Private Network” solution to enterprises.

The term “Virtual Private Network” (VPN) refers to the communication between the customer nodes over a shared network structure of a “Service Provider”(SP). Service Provider is the firm that offers VPN solutions to enterprises. If customer nodes(customer devices) know only the customer edge devices, i.e, they don’t know the infrastructure of the SP, then we call it an overlay VPN.

VPNs are being built on public backbones and are one of the hottest areas of international networking market. A company offering VPN service, must create a virtual network over the public network. Thus, geographically dispersed customer nodes can communicate securely over the untrusted public networks. Hence, SPs are building virtual networks over public networks in order to satisfy the requirements of their customers.

In this thesis, the topology design problem of a VPN provider is studied. Given a set of customer nodes and an associated traffic matrix for every pair of these customer nodes, our job is to locate tunnels on the network and route the traffic over those tunnels. A tunnel is simply a link between a pair of nodes. Note that tunnels can transport traffic in two ways, i.e, they act as edges. Besides flow conservation constraints, the problem has some specific constraints that have to be satisfied.

The first one is the "bandwidth capacity constraint" which expresses that total traffic flow on a link can not exceed the bandwidth capacity, i.e, the summation of the flow amounts of commodities on a link must be less than or equal to the bandwidth capacity, which is given a priori.

The second one is the "degree constraint". Due to hardware/software constraints, only a limited number of tunnels can be located on a customer node. Therefore, the number of tunnels(links) belonging to a customer node can not exceed a certain number. That is, the degree of a node in the network is bounded above by a predetermined number.

A third constraint to our problem is the "delay constraint". This constraint arises from the time requirements of each traffic(commodity). We have to route every commodity in the time no longer than a maximum delay value. Given a delay matrix representing delay values between every node pair, the delay value of a commodity must be less than equal to the maximum delay value. Delay value of a commodity is simply the sum of delay values of every link in the path of that commodity.

Finally, an important characteristic of our problem is that every traffic(commodity) must use a single path from its source to its destination. Therefore, it is not possible to split the traffic value of a commodity and route them on several paths. All of the traffic value of a commodity must be sent through a single path. Because every commodity represents a communication package between customer nodes, we have to route them without splitting.

Under these constraints, we have to find a feasible topology and route the commodities on this topology while minimizing the cost value. The cost has two components: location cost and routing cost. Location cost is simply the cost of locating a link(tunnel) between two customer nodes. Routing cost is the cost of routing the commodities on the network.

We assume that location cost of every possible link is identical, say it is C. Therefore location cost of a topology is the number of links times C. Routing cost of a commodity is the traffic value times the number of links in the path that commodity uses. We also need to multiply this value with a scalar, say α, in order to convert routing cost into the same terms of location cost. The routing cost of a topology is the sum of routing costs of every commodity.

It is easy to verify that, as the number of links used in a topology decreases location cost decreases. But then the path lengths of commodities can increase and this yields an increase in the routing cost. Similarly as we increase the number of links, it is possible to route the traffic in shorter paths but at the expense of an increase in the location cost. Therefore, there exists a tradeoff between the location cost and the routing cost. One must find a topology design and a corresponding routing of commodities that minimizes the total cost which is the summation of location and routing costs.

Although Network Design in Computer Networks is studied by many researchers, Topology Design is not studied much. An important study on Topology Design in Computer Networks is of Karaşan et al. [21]. They studied "mesh topology design problem" in overlay VPNs. The problem they study is similar to our problem. Their topology design problem starts with a set of customer nodes and an associated traffic matrix. The goal of the problem is to determine where to locate links while minimizing the cost value. They have a degree number that must be equal to the number of links assigned to any customer node in the network. Therefore, the number of links of every node in the network must be equal to a constant degree number. In our case, the degree of a node can be less than the maximum degree number. Also, they do not have a bandwidth capacity constraint for links. They assume that the links in the network are uncapacitated, i.e, there is no capacity limit for the links. But in our problem, every link can transport only a limited amount of traffic, that is, the maximum bandwidth capacity of a link. Another difference between the two problems is the delay constraint. They do not have any delay requirement for commodities, which we incorporate. Given a delay matrix denoting the delay amounts between nodes, we have to route commodities over the network such that all of the commodities arrive at their destinations within a desired delay time. Finally, their cost is only the routing cost. They do not take location cost into account. Their aim is to design a feasible topology that minimizes the routing costs of commodities. But our cost value has two components: location and routing cost. Locating a link between customer nodes has a cost as well, and our model takes this cost in to account and tries to find a feasible topology that minimizes total cost, not only the routing cost.

Karaşan et al. [21] showed that the problem is large and extremely hard. They stated that it takes many hours to find an optimal solution by CPLEX already for networks with 10 customer nodes. It is also given that the LP Relaxation value is very far from the optimal value. Therefore, they attacked to the problem by finding valuable upper and lower bounds. By squeezing these bounds one can evaluate the quality of the feasible solution(upper bound) found. Two types of valid inequalities are generated: flux inequalities and distance inequalities. Flux inequalities are first introduced by Bienstock and Günlük [9]. By adding them to the LP Relaxation , the lower bound is strengthened. In order to find an upper bound they applied a Tabu Search based heuristic. Starting with an initial connected graph where each node has the same degree, say p, they generate neighborhood solutions and move to the one with smallest cost at each iteration. Experimental results for $|N| = 20, p = 3, 4$ for their approach is given. The gap between upper and lower bounds is found to be around %2.

Gendron and Crainic [7], studied the Multicommodity Capacitated Network Design Problem. The differences of their model from ours are as follows: In their model commodities can be split and routed over several paths, but in our case a commodity can not be split and must be routed through a single path. For a commodity they have multiple origins and destinations however we have a single source-destination pair for every commodity. They also don't have degree and delay constraints.

They presented three formulations of the problem: Weak formulation, strong formulation and extended formulation. Then, they proposed some bounding procedures based on classical relaxation approaches. Several methods are compared both theoretically and empirically, and the results are discussed at the end of the paper. One of the perspectives that they pointed out was to apply Tabu Search based heuristics to improve the upper bounds which is one of the outcomes of this thesis.

Holmberg and Yuan [13], studied a problem which is very similar to the problem of Gendron and Crainic [7]. The only difference is that in Holmberg and Yuan, commodities have single origin-destination pairs. They proposed a method based on a Lagrangian heuristic within a branch-and-bound framework. The heuristic uses a Lagrangian relaxation to obtain subproblems and solves the Lagrangian dual by subgradient optimization. Finally, they presented the computational results which show that the proposed method generates good solutions within reasonable time.

Ramaswami and Sivarajan [10] and Ahn et al. [6] studied a different version of topology design problem. They are locating virtual links over physical links. In other words, their objectives are to generate a topology consisting of both virtual and physical links, and the physical link topology is given priori.

Ramaswami and Sivarajan [10] studied the problem of designing a logical topology over a wavelength-routed all-optical physical topology which consists of the nodes and fiber links in the network. For a given network physical topology and a traffic pattern, their objective is to set up links(lightpaths) between nodes. Located lightpaths constitudes the logical topology. The cost to be minimized is the network congestion. Therefore, they formulated the problem in terms of minimizing congestion subject to the restriction that the delay of a commodity can not be more than a predetermined value. Additional constraints are degree constraints and wavelength number constraints. Their formulation allows traffic to be split across multiple possible paths. The problem is solved for a 6-node network. For larger networks, they proposed some lower bounding techniques and introduced 5 different heuristics for upper bounding. Finally, a discussion of the quality of these heuristics is given.

Ahn et al. [6] studied Virtual Path Layout Design Problem. For a given graph $G = (V, E)$ where V is the union of switch nodes and user nodes, and a traffic, they seek to find a Virtual Path(VP) layout which satisfies flow and logical connectivity constraints. Their aim is to assign the maximal percentage of offered traffic bandwidth on a VP-Layout such that call broking probability and the setup, switching and transmission costs are minimized. They introduced an iterative heuristic and analyzed some of its properties analytically. Then, they present simulation results of their approach.

A different version of designing a network over a physical network is due to Dahl et al. [19]. They studied the pipe selection and routing problem. Given a physical network, a demand matrix and a pipe network, the problem is to select pipes that are to be used and to determine on which path of the selected pipe set the demands should be routed. The cost to be minimized is the summation of selection and routing costs. After giving an integer programming formulation, they introduced several facet defining valid inequalities to the problem. Then, they described a cutting plane algorithm with separation algorithms and a primal heuristic algorithm. Finally, the computational results for some realistic problems are reported.

Kenington et al. [22] studied the problem of routing and assignment in a survivable WDM network. In order to simplify the problem, they partitioned it into three components: Layout Subnetworks, Optical Cycles and Optimization Model. In the Optimization Model, they aimed to select paths and layered subnetworks to satisfy the demand and assign wavelengths to these paths while minimizing the construction cost. Hence, it can be seen as a path selection problem. Then, they introduced an MIP model of the optimization model. Chlamtac et al. [4] showed that this problem is NP-hard even if there is only one layered subnetwork. Therefore, Kenington et al. [22] introduced a heuristic approach which is based on fixing binary variables iteratively to obtain feasible solutions. After applying this procedure multiple times, it returns the one with minimum cost. At the end of the paper, they presented experimental results for 4 types of problems, each type containing 5 networks. It is shown that the heuristic results are very fast and slightly different from optimal values.

Bienstock et al. [17] studied the Minimum Cost Capacity Installation problem, the objective of which is to obtain a minimum cost installation of capacities to the arcs to ensure that all commodities can be routed simultaneously. They described two different formulations of the problem: Capacity Formulation and Multicommodity Formulation. Then, they introduced two classes of strong valid inequalities and came up with solution procedures. By the computational experiments, they concluded that the two formulations are comparable and yield effective solution algorithms.

Magnanti et al. [8], studied a version of capacitated network design problem, which is two-facility capacitated network loading problem. The problem is to determine the number of facilities to be loaded on each arc of the network to meet given communication demand at minimum cost. They assumed that only two types of facilities are available: low capacity and high capacity facilities. They introduced three facet defining valid inequalities: Arc Residual Capacity Inequality, Cut Set Inequality and 3-partition Inequality. It is shown that adding arc residual capacity inequality to the LP formulation, guarantees a lower bound equal to the Lagrangian lower bound. Then, they presented a two-phase cutting plane algorithm which is quite effective in reducing the integrality gap.

Another reference that studied network loading problem is Barahona [11]. The paper described relaxation based cut inequalities and introduced a cutting plane algorithm based on these inequalities. The main difference of the algorithm from others is that it uses an exact algorithm for the separation problem and does not use the flow variables in the first approximation.

Günlük [16], studied a variant of Network Loading Problem, which is called Capacity Expansion Problem(CEP). The difference of CEP from Network Loading Problem is that, there exists initial capacities on edges in CEP. Polyhedral structure of the MIP formulation of CEP is analyzed and several valid inequalities are introduced. Note that, commodities can be split, and therefore routed over several paths in CEP. Then, a new branching method which is called as ”knapsack branching” is applied in the proposed branch-and-cut algorithm. Finally, the results of computational experiments are discussed and it is concluded that branch-and-cut is effective in difficult problems.

Polyhedral structure of the MIP formulation of CEP is first studied by Bienstock and Günlük [12]. Using facet defining inequalities, they developed a cutting plane algorithm which produces good lower bounds and a starting point for branch-and-bound procedure.

Alevras et al. [14] is another reference that studied a version of Network Loading Problem with the exception that their problem is defined on two graphs on the same node set V; the supply graph and the demand graph. Their aim is to install capacities on each edge of the network in order to satisfy the traffic requirement, while minimizing the building cost. After giving an MIP formulation of the problem, they developed a cutting plane algorithm and several heuristics. Then, computational results for real world data are reported.

A comprehensive survey of models and algorithms for capacitated network design problems can be found in Magnanti and Wong [2], Minoux [3] and Gendron et al. [18].

1.2 Approach to Problem

As mentioned before our problem is to locate links on the given node set and to route the given traffic over these links while satisfying the constraints. The constraints are bandwidth capacity constraint, degree constraint and delay constraint. Additionally we can not split the traffic flow of any commodity, therefore every commodity must be routed over a single path, which makes our problem harder. The objective of the problem is to minimize the total cost which is the summation of location and routing costs.

It is clear that one can not hope to find an optimum solution to this problem within reasonable time even for moderate cases with 10 nodes. Therefore we have to find a feasible solution that has a "reasonable cost value". Here reasonable cost value refers to a cost which is close enough to the optimal value. To have an idea about how close we are to the optimal value, we must have a lower bound. Hence, we need to find good lower and upper bounds to the problem. Since our problem is an integer program, LP relaxation of it is a lower bound to the optimal value. Also, every feasible solution is automatically an upper bound to the optimal value. But the quality of a solution to problem increases as the gap between upper and lower bounds decreases. Therefore, in this thesis, our aim is to find a feasible solution the cost of which is minimized as much as we can. In order to test how far we are from optimal value we aim to find a lower bound which is maximized as much as we can.

In Chapter 2, we give a mathematical formulation of the problem. The formulation is an Integer Programming formulation, in which we have both integer and binary variables. Then, we introduced 2 sets of valid inequalities to the IP formulation. At the end of Chapter 2, we present the effects of these cuts to the lower bounds on 10 test problems.

In order to find a good feasible solution we proposed a Tabu Search based heuristic. The heuristic is based on a local search, at each iteration of the algorithm, it selects the feasible move with smallest cost. The heuristic determines how many links to be located between every pair of nodes and how to route traffic over these links. Therefore, we have a two-step problem: Link Location and Traffic Routing. In each iteration of the Tabu Search we have to find the cost of each possible move, therefore we need an algorithm that routes traffic over a given network topology. This Routing problem is a very hard problem by itself. It is called the Integer Multicommodity Flow Problem in the literature.

In Chapter 3, we analyzed the Integer Multicommodity Flow Problem independently, and developed a simple and efficient heuristic for it. The main idea of the algorithm is to route commodities one by one over the network. After each routing, edge capacities are updated by lowering the capacities of the edges in the path of routing by the traffic amount of that commodity. Then, the next commodity is routed with these new edge capacities. The algorithm adopts a greedy approach and routes the commodities in descending order of their flow amounts. The IP formulation of the problem, a detailed description of the proposed algorithm and computational experiments are given in Chapter 3.

In Chapter 4, the Tabu Search algorithm is described. How to find neighborhood solutions, iteration procedures and tabu criteria are explained in detail. Computational experiments with several kinds of problems are given at the end of the chapter. One can find more about Tabu Search and its application areas in Glover and Laguna [15]

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

Chapter 2

Problem Formulation

In this chapter we give a mathematical formulation of the problem. Although it is very difficult to obtain an optimal solution by the mathematical formulation, most of the times it is a useful tool to have a better understanding of the problem. Since, one of the objectives of this thesis is to find a good lower bound, we have to study the polyhedral properties of the mathematical model and develop some valid inequalities. Before giving the formulation, it will be useful to state the notation that will be used through the thesis.

2.1 Notation

Let N =$1, ..., |N|$, be the set of customer nodes where $|N|$ is the cardinality of node set. Let K be the set of commodities, with cardinality $|K|$. Each element in the set K has a triplet $(s_k, d_k, flow(k))$ associated, where s_k denotes the source of commodity k, d_k denotes the destination of commodity k and $flow(k)$ denotes the flow amount(traffic) of commodity k. We can also represent the commodities by a traffic matrix T, where t_{ij} represents the traffic between customer node i and j. Without loss of generality, it is assumed that the traffic should be routed from smaller indexed node to the greater indexed one. For example we take the traffic between node 1 and node 3 as to be routed from 1 to 3, in other words 1 is the source and 3 is the destination of this commodity. Therefore the traffic matrix T is an upper triangular matrix and for every commodity in K, s_k is smaller than d_k.

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

The constants of the problem are as follows: C is interface cost, i.e, the cost of locating a link between a node pair. It is assumed that location cost of a link is independent from where it is located. Location cost of every possible link is equal to C.

Let α be the scalar that is used to convert routing cost in to the same terms of location cost.

We use W to represent the bandwidth capacity of a link. It is used in the bandwidth capacity constraint which states that total flow between a node pair can not exceed the total bandwidth capacity of the links between these nodes.

We use d_i to represent the interface number limit for router i. It is the maximum degree of node i. In this thesis we assume that all of the nodes in N have the same interface limit number, say d.

Finally, we are given a value D which is the maximum delay. Every commodity in the set K must be routed such that the delay amount of that routing is less than maximum delay value, D.

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

Let a_{ij} be the location variable. It takes the number of links between node i and j. Since we can put an integer number of links between nodes, a_{ij} takes integer values, hence it is an integer variable. It is clear that a_{ij} and a_{ji} take the same value, because they are representing the same parameter, which is the number of links between nodes i and j. In the model we set them equal to each other by a constraint.

Let b_{ij}^k be the binary variable used for routing. It takes value 1 if commodity k is routed over link ij from i to j. It takes value 0 otherwise.

We have $|N|^2$ location variables and $|N|^2|K|$ routing variables.

2.2 Integer Programming Formulation

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

$$\begin{aligned}
\text{Minimize} \quad & C' \textstyle\sum_{ij} a_{ij} + \quad \alpha \sum_{k\in K} flow(k) \sum_{ij} b_{ij}^k \\
\text{subject to} \quad & \\
\sum_{j\in N} b_{ij}^k - \sum_{j\in N} b_{ji}^k \quad &= \quad r_i^k \quad \forall i \in N, \quad \forall k \in K && (2.1)\\
\sum_{k\in K} flow(k)(b_{ij}^k + b_{ji}^k) \quad &\leq \quad (a_{ij} + a_{ji})W' \quad \forall i,j \in N && (2.2)\\
\sum_{j} a_{ij} \quad &\leq \quad d_i \quad \forall i \in N && (2.3)\\
\sum_{ij\in N} b_{ij}^k l_{ij} \quad &\leq \quad D \quad \forall k \in K && (2.4)\\
a_{ij} \quad &= \quad a_{ji} \quad \forall i,j \in N && (2.5)\\
b_{ij}^k \quad &\in \quad \{0,1\} \quad \forall i,j \in N, \forall K \in K && (2.6)\\
a_{ij} \quad &\in \quad Z \setminus Z^- \quad \forall i,j \in N && (2.7)
\end{aligned}$$

where r_i^k takes value 1 if i is the source of commodity k, takes value -1 if i is the destination of commodity k and takes value 0 otherwise. In the model $C' = C/2$ in order to eliminate double counting of links. We add up all a_{ij} values and since both a_{ij} and a_{ji} represent the number of links between nodes i and j, the sum of all a_{ij}'s is twice the number of links located. Therefore we have to multiply this sum with half of the cost value. For similar reasons we use $W' = W/2$ in the model. Because the total bandwidth capacity of a link is equal to the number of links times bandwidth capacity of a single link.

Constraints (2.1) are the usual flow conservation constraints. Every commodity must leave its source and arrive at its destination. The other nodes (i.e, the transshipment nodes) must have a balance on entering and leaving commodities.

Constraints (2.2) are the bandwidth capacity constraints. They express that the flow amount between two nodes can not exceed the bandwidth capacity between these nodes. Total flow between two nodes is found by summing up the flows of commodities that are routed in both directions. The bandwidth capacity is the number of links times the bandwidth capacity of a single link. Since we sum both a_{ij}'s and a_{ji}'s, we need to multiply this sum with the half of the bandwidth capacity in order to eliminate double counting.

Constraints (2.3) express that there can be at most d_i links connected to node i. We call them degree constraints, and they are coming from the hardware/software restrictions of routers. During our computational experiments we assume that this limit number is the same for all nodes and is 8.

Constraints (2.4) are delay constraints and express that the delay value of a commodity can not exceed a maximum delay value. The delay value of a commodity is found by summing the delay values of edges used in the routing of that commodity.

We set a_{ij} equal to a_{ji} for every i and j in Constraints (2.5). We need such a constraint because we represent an edge by 2 arcs, and whenever a link is located between nodes i and j, both a_{ij} and a_{ji} values increase by 1. For example if we have 2 links located between nodes 2 and 5, then both $a_{2,5}$ and $a_{5,2}$ must be equal to 2: $a_{2,5} = a_{5,2} = 2$.

Constraints (2.6) express that routing variables are binary, and constraints (2.7) express that the location variables are integer.

2.3 Valid Inequalities

Computational experiments show that we can not find an optimal solution to the problem within 10 hours for $|N| = 20$. Also LP relaxation values are very far from the optimal values(about %50), so that they can not be good lower bounds. Therefore we have to develop some valid inequalities and solve the LP relaxation after adding these inequalities to the problem. In this section, two sets of valid inequalities are developed and the computational results for several problems are presented in the next section.

2.3.1 Cut Inequalities

The first set of valid inequalities is the cut inequalities. The main idea behind is as follows: Whenever we divide the node set into two parts, the commodities whose sources and destinations belong to different sets must use the links between these two node sets. Therefore, the bandwidth capacity between these two node sets must be greater than or equal to the flow amounts of those commodities.

Say S is a subset of N. Then let $\bar{S} = N \setminus S$ and K_S be the set of commodities whose sources and destinations are in different sets, one is in S and the other one is in $\bar{S}$. Then we have:

$\sum_{i \in S, j \in \bar{S}} (a_{ij} + a_{ji}) W' \geq \sum_{k \in K_S} flow(k)$

Sice a_{ij}'s are integer variables we can strengthen these inequalities as follows:

$$\sum_{i \in S, j \in \bar{S}} (a_{ij} + a_{ji}) \geq \lceil \frac{\sum_{k \in K_S} flow(k)}{W'} \rceil \quad , \qquad \forall S \subset N \tag{2.8}$$

2.3.2 Link Inequalities

Secondly, we have link inequalities. It expresses that whenever a commodity is routed between two nodes, there must at least one link located between them.

Since b_{ij}^k's are binary variables, they can take at most 1. Therefore, a_{ij}'s are greater than or equal to b_{ij}^k's. Then we have:

$$a_{ij} \geq b_{ij}^k \quad , \qquad \forall i, j \in N \quad , \forall k \in K \tag{2.9}$$

2.4 Computational Experiments for Lower Bound

In this section we present the computational experiments of the proposed valid inequalities. We first analyzed the effect of each valid inequality independently, and then applied them together. For the Cut Inequalities, we select subsets with one element, i.e, we select $S \subset N$ such that $|S| = 1$, that is we look for individual nodes. The results are obtained after adding these inequalities to the LP Relaxation. Then we look for the sets with cardinality 2. But it is observed that adding cut inequalities for the sets with cardinality 2 does not improve the lower bound. Then in our computational experiments, we only add cut inequalities of the sets with cardinality 1. In other words, for every node in the network, we add a cut inequality to the LP Relaxation. It is observed that Cut Inequalities do not improve the LP Relaxation lower bound so much.

Secondly, we add Link Inequalities to the LP Relaxation. Unlike Cut Inequalities, Link Inequalities increases the lower bound very much, and give good results. Finally, we add both inequalities to the LP Relaxation. The results are slightly better than only adding Link Inequalities.

We have 10 test problems with different characteristics. But, for all of the problems the traffic values come from a uniform distribution between 0 and 100. The delay values are also uniformly distributed between 0 and 50. Bandwidth capacity of a link is 2000, maximum degree limit is 8 for every node and maximum delay value is 75. For determining maximum delay value, we take the 1.5 multiple of the maximum delay valued link. Since delay values for links are uniformly distributed between 0 and 50, we take 1.5x50=75 as the maximum delay value. With this setting we assure that every commodity is routed within the 1.5 times of the maximum delayed link in the network. Input characterictics of these problems are given in Table 2.1.

We take location cost(C) equal to 100 and α equal to 1.

Since adding Cut Inequalities for $|S| > 2$ does not affect the lower bound, we only add the Cut Inequalities for single nodes. For every node in the network, we add up the flow amounts of commodities whose source or destination is that node. We call this sum as the traffic that must use the links connected to that node, therefore total bandwidth capacity of the links connected to that node must be greater than or equal to that sum. In Table 2.2 we give the LP Relaxation values of 10 test problems and the lower bounds attained after adding Cut Inequalities to LP Relaxation. In the last column we present the percent increase in the lower bounds after adding Cut Inequalities.

In Table 2.3, we present the effect of adding Link Inequalities to the LP Relaxation. Last column represents the increase amount of lower bound after adding Link Inequalities. It is observed that Link Inequalities are more powerful than Cut Inequalities.

Finally, in Table 2.4, both inequalities are added and the results are presented. Again last column stands for the percent increase in the lower bound. The lower bounds presented in Table 2.4 are the ones that we will use to evaluate the upper bounds found by Tabu Search Heuristic. The closer heuristic results to the lower bounds, the closer they are to the optimal value.

Table 2.1: Characterization of Test Problems

	$\|N\|$	$\|K\|$	$traffic(i,j)$	$delay(i,j)$
Problem1	20	190	U[0,100]	U[0,50]
Problem2	20	190	U[0,100]	U[0,50]
Problem3	20	190	U[0,100]	U[0,50]
Problem4	20	190	U[0,100]	U[0,50]
Problem5	20	190	U[0,100]	U[0,50]
Problem6	25	300	U[0,100]	U[0,50]
Problem7	25	300	U[0,100]	U[0,50]
Problem8	25	300	U[0,100]	U[0,50]
Problem9	30	435	U[0,100]	U[0,50]
Problem10	30	435	U[0,100]	U[0,50]

Table 2.2: Adding Cut Inequalities

	LP	$LP+(1)$	$Improvement(\%)$
Problem1	9192	9961	8.4
Problem2	9878	10625	7.6
Problem3	9611	10367	7.9
Problem4	9570	10326	7.9
Problem5	9717	10470	7.7
Problem6	14912	15765	5.7
Problem7	15693	16523	5.3
Problem8	15542	16381	5.4
Problem9	22144	23060	4.1
Problem10	22666	23577	4.0

Table 2.3: Adding Link Inequalities

	LP	$LP+(2)$	$Improvement(\%)$
Problem1	9192	16233	76.6
Problem2	9878	17392	76.1
Problem3	9611	16934	76.2
Problem4	9570	16629	73.8
Problem5	9717	16896	73.9
Problem6	14912	26299	76.4
Problem7	15693	27256	73.7
Problem8	15542	26964	73.5
Problem9	22144	38896	75.7
Problem10	22666	39926	76.1

Table 2.4: Adding Link and Cut Inequalities

	LP	$LP + (1) + (2)$	$Improvement(\%)$
Problem1	9192	16247	76.8
Problem2	9878	17406	76.2
Problem3	9611	16949	76.4
Problem4	9570	16651	74.0
Problem5	9717	16904	74.0
Problem6	14912	26310	76.4
Problem7	15693	27480	75.1
Problem8	15542	27208	75.1
Problem9	22144	39115	76.6
Problem10	22666	40136	77.1

Chapter 3

Integer Multicommodity Flow Problem

This chapter is devoted to analyze the Integer Multicommodity Flow Problem. Our problem has two components: locating links and routing flows. In the Tabu Search based heuristic that will be discussed in detail in the next chapter, we need an efficient procedure to find a routing for any given network topology. Since we must route every commodity in a single path, the problem of routing becomes extremely difficult. Therefore, we have to find a fast and good approximation algorithm to find a feasible routing. In this chapter, we study the Integer Multicommodity Flow Problem independently, and develop an efficient heuristic. The computational experiments for several kinds of test problems are given at the end of the chapter.

3.1 Introduction

In this chapter, we investigate a simple and efficient heuristic for the integer multicommodity flow problem($IMCFP$). $IMCFP$ is a constrained version of the linear multicommodity flow problem. In $IMCFP$ commodities can not be splitted, therefore they must be routed over a single path, also there exists a commodity for every pair of nodes. The problem is to route all of the commodities over the network with the minimum cost while satisfying edge capacity constraints.

One can find a detailed analysis of linear multicommodity flow problems in Ahuja *et.al*. [5] and Assad [1]. Besides linear multicommodity flow problems, integer multicommodity flow problem is not studied much in the literature. Barnhart et al. [20] give a column-generation model and a branch-and-price-and-cut algorithm for the integer multicommodity flow problems, and they test their algorithm on several networks. Their algorithm is a variant of branch-and-bound, with bounds provided by solving LP's using column-and-cut generation at nodes of the branch-and-bound tree. Since there does not exist a commodity for every pair of nodes in their cases, they called the problem as origin-destination integer multicommodity flow problem. In our case, we have a commodity for every pair of nodes, hence our problem is called full integer multicommodity flow problem. It is certain that as the number of commodities increases the problem becomes harder to solve.

The main idea of our algorithm is to route the commodities one by one over the network. After each routing, edge capacities are updated by lowering the capacities of the edges in the path used by that commodity by the flow amount. Then the next commodity is routed with these new edge capacities. In each routing subproblem, the commodity is simply routed over the shortest path from its source to its destination.

In the algorithm, we adopt a greedy approach and route the commodities in the descending order of their flow amounts. It is clear that the effect of a decrease in the shortest path distance of a commodity on the objective function is more for the commodities with higher flow values. Therefore we generate a list called $OrderedList$, in which commodities are listed in the descending order of flow amounts. We also generate two other lists called $List1$ and $List2$. $List1$ contains commodities in the order of lexicographically smaller principle and $List2$ contains commodities in the reverse order of $List1$. In lexicographically smaller principle, we first look at the origins of the commodities and then the destinations. If the index of the source of a commodity is smaller than another commodity then it is lexicographically smaller, if those index values are the same then the one with the smaller index valued destination is lexicographically smaller. As an example commodity i with source-destination pair (2,5) is lexicographically smaller than commodity j with (3,4), but it is lexicographically bigger than commodity k with (2,3). These three lists, $OrderedList$, $List1$ and $List2$ are routed and the one with the minimum cost is found. Therefore we route the commodities in three different order.

The rest of the chapter is organized as follows: In Section 3.2, we give the definition and a mathematical model of the problem. In Section 3.3, the solution approach to the problem and the proposed algorithms are given. We present the computational results of the algorithm over several networks in Section 3.4. Finally, we give conclusions in Section 3.5.

3.2 Problem Definition

The integer multicommodity flow problem is defined over the network $G = (N, E)$, where N is the node set and E is the edge set. $IMCFP$ has the commodity set K in which there exists a commodity for every pair of nodes. Our problem is to route all of the commodities in the set K with the minimum cost. The cost of assigning commodity k to edge ij equals the flow amount of commodity k which is q^k times the unit flow cost for edge ij, denoted c_{ij}^k.

Throughout the chapter, we will show an edge by two arcs, i.e, if there is an edge between node i and node j, then we have two arcs: arc_{ij} and arc_{ji}. Therefore, we have an arc set A such that, for every edge $\{i, j\}$ in E we have two arcs, (i, j) and (j, i) in A.

An integer programming formulation for the problem is the following:

$$\text{Minimize} \qquad \sum_{k \in K} \sum_{ij \in A} c_{ij}^k q^k x_{ij}^k$$

subject to

$$\sum_{k\in K} q^k(x_{ij}^k + x_{ji}^k) \leq capacity_{ij} \quad \forall (i,j) \in E \tag{3.1}$$

$$\sum_{ij\in A} x_{ij}^k - \sum_{ji\in A} x_{ji}^k = b_i^k \quad \forall i \in N,\ \forall k \in K \tag{3.2}$$

$$x_{ij}^k \in \{0,1\} \quad \forall ij \in A, \forall k \in K \tag{3.3}$$

where x_{ij}^k takes value 1 if the k^{th} commodity uses arc (i,j), and 0 otherwise. We let q^k values represent the flow amounts of commodities, c_{ij}'s are the the arc lengths and $capacity_{ij}$'s are the edge capacities. We use K to represent the set of all possible commodities. Node i has supply of commodity k, denoted b_i^k, equal to 1 if i is the source node for k, equal to -1 if i is the destination node for k, and equal to 0 otherwise. Constraints (3.1) express that the flow value over edge(i,j) can be at most its capacity which is $capacity_{ij}$. Constraints (3.2) are the flow conservation constraints, and constraints (3.3) are the binary constraints for variables x_{ij}^k.

3.3 Solution Approach

It is clear that $IMCFP$ is hard to solve optimally. In section 3.4 we observe that for moderate cases like $|N| = 60$ it takes nearly an hour to solve it and for networks with $|N| = 80$ and $|N| = 100$ we are unsuccessful to solve it optimally. In the heuristic we route the commodities one by one. A commodity is defined by three attributes: its source, destination and flow amount. In the routing of a commodity we are finding a shortest path from its source to its destination. Since the path must route the flow amount, we are considering the edges, capacities of which are greater than or equal to the flow amount. In the first algorithm we generate a list of the commodities in the descending order of their flow amounts. This list is called $OrderedList$. The commodities in the $OrderedList$ are routed one by one. Second algorithm is an extension of the first one. In this algorithm we generate two more lists, $List1$ and $List2$, and we repeat the algorithm three times for three lists, $OrderedList$, $List1$ and $List2$. Then we take the minimum cot of those three.

3.3.1 Algorithm 1

First start with a definition that will be used in the algorithm.

Definition 1 *Given the network G=(N, E), $G(f)$ represents the network G'=(N, E') where E' is a subset of E such that $\forall(i, j) \in E$ if $capacity_{ij} \geq f$ then $(i, j) \in E'$, if $capacity_{ij} < f$ then $(i, j) \notin E'$*

The first algorithm proposed to find a solution to $IMCFP$ is as follows:

Step 1. Order the commodities in K by descending order of flow amounts, in $OrderedList$

Step 2. Set $counter = 1$

Step 3. Set $i = source[OrderedList(counter)]$,

set $j = destination[OrderedList(counter)]$ and set $f = Flow[OrderedList(counter)]$

Step 4. Find the shortest path from i to j in $G(f)$, say $dist(i, j)$

Step 4.1. If shortest path is found go to Step 5
Step 4.2. else go to Step 9.

Step 5. Find the cost of routing $commodity(counter)$ by multiplying the shortest path distance with the flow amount of $commodity(counter)$

Step 6. Decrease the capacities of edges in the shortest path by f, update G.

Step 7. Increase $counter$ by 1.

Step 7.1. If $counter > |K|$ then go to Step 8
Step 7.2. else go to Step 3.

Step 8. Sum up the routing costs of all commodities. Terminate.

Step 9. No feasible solution is found. Terminate.

3.3.2 Algorithm 2

The second algorithm proposed to solve $IMCFP$ is a modified version of Algorithm 1. In Algorithm 1, the commodities are routed over residual network in the order of $OrderedList$. Now, we have three lists. First one is $OrderedList$ and the other two are lexicographic lists, $List1$ and $List2$. $List1$ contains commodities in the order of lexicographically smaller principle. It starts with the commodities originated from node 1, then from node 2 and continues. $List2$ is in the reverse order of $List1$.

In Algorithm 2 the commodities in these three lists are routed as in Algorithm 1, and the smallest one is chosen.

3.3.3 Complexity Issues

Theorem 1 *The computational complexity of the proposed algorithms is* $O(n^4)$.

Proof:
The routing of a commodity requires finding a shortest path from its source to its destination and then updating the edge capacities in that path. Complexity of finding the shortest path is $O(n^2)$ and complexity of update process is $O(n)$. Therefore complexity of a routing is $O(n^2)$. Since we have a commodity for every pair of nodes the cardinality of the lists, $OrderedList$, $List1$ and $List2$, is in $O(n^2)$. Since we do a routing process for every commodity in the lists, the computational complexity of the proposed algorithms is $O(n^4)$.

3.4 Computational Results

In order to test our heuristic we generate 15 problems varying in node size and arc density. In the first 9 problems commodity flows come from a uniform distribution between 50 and 150. For the last 6 problems commodity flows are distributed uniformly between 0 and 100.

Each problem is solved for several edge capacity values. Optimum solutions found by CPLEX and heuristic solutions by Algorithm 1 and Algorithm 2 and their deviation from the optimum are presented in Tables 1-15. The CPU column expresses the computation time for each solution method in seconds. "Time Limit" expression in some CPLEX Solution columns means that CPLEX could not find a solution within 3 hours.

In the experiments, optimum values are obtained by Cplex 7.1 and heuristic values are obtained on a Pentium II PC. Our programs were coded in Delphi 6.0.

It is observed that proposed heuristics generate feasible solutions that are slightly different from optimal values in very small cpu times. Cplex could not find a solution within three hours for the networks with $|N| = 80$ and $|N| = 100$. Therefore we can not evaluate the solutions found by our heuristics for these networks. LP relaxation values of these networks are around %60 of the heuristic solutions.

Table 3.1: Network1 n=20 nodes , arcdensity=0.1 , Flow∼U[50,150]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
4500	59503	4.62	59503	0.00	0.00	59503	0.00	0.00
4000	59503	5.97	59503	0.00	0.00	59503	0.00	0.00
3800	59540	6.61	59603	0.00	0.11	59603	0.00	0.11
3700	59640	8.27	60009	0.00	0.62	60009	0.00	0.62
3600	59740	16.82	infeasible	0.00	X	60712	0.00	1.63
3500	infeasible	10.96	infeasible	0.00	X	infeasible	0.00	X

Table 3.2: Network2 n=20 nodes , arcdensity=0.2 , Flow∼U[50,150]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
2500	45177	8.43	45492	0.00	0.70	45382	0.00	0.45
2400	45177	9.36	45622	0.00	0.99	45540	0.00	0.80
2300	45177	29.12	45711	0.00	1.18	45653	0.00	1.05
2200	45177	12.99	45711	0.00	1.18	45653	0.00	1.05
2100	45177	16.67	45813	0.00	1.41	45615	0.00	0.97
2000	infeasible	17.12	infeasible	0.00	X	infeasible	0.00	X

Table 3.3: Network3 n=20 nodes , arcdensity=0.3 , Flow∼U[50,150]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
1500	34989	13.91	34989	0.00	0.00	34989	0.00	0.00
1200	34989	37.48	35419	0.00	1.23	34989	0.00	0.00
1100	34989	76.44	35393	0.00	1.15	35130	0.00	0.40
1000	34989	130.94	36525	0.00	4.39	35274	0.00	0.81
800	34989	236.98	infeasible	0.00	X	36298	0.05	3.74
700	Time Limit	X	infeasible	0.05	X	infeasible	0.05	X

Table 3.4: Network4 n=30 nodes , arcdensity=0.1 , Flow∼U[50,150]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
12000	149517	337.96	153274	0.11	2.51	152238	0.16	1.82
10000	151517	438.93	157120	0.11	3.70	156202	0.17	3.09
9000	Time Limit	X	164076	0.11	X	161522	0.16	X
8000	Time Limit	X	172132	0.11	X	172132	0.16	X
7500	Time Limit	X	infeasible	0.11	X	infeasible	0.16	X

Table 3.5: Network5 n=30 nodes , arcdensity=0.2 , Flow∼U[50,150]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
2000	85922	30.9	85922	0.11	0.00	85922	0.16	0.00
1800	85922	28.99	86222	0.11	0.35	86153	0.16	0.27
1500	85922	38.52	86950	0.11	1.20	86766	0.16	0.98
1400	85922	32.21	87379	0.11	1.70	87379	0.16	1.70
1200	infeasible	33.16	infeasible	0.11	X	infeasible	0.17	X

Table 3.6: Network6 n=30 nodes , arcdensity=0.3 , Flow∼U[50,150]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
2000	83059	30.24	83125	0.14	0.08	83059	0.16	0.00
1500	83059	31.47	83900	0.16	1.01	83294	0.16	0.28
1200	83059	34.16	85373	0.15	2.79	83991	0.16	1.12
1000	83059	36.42	87697	0.17	5.58	86059	0.22	3.61
800	83059	36.44	infeasible	0.16	X	infeasible	0.16	X
500	infeasible	35.48	infeasible	0.16	X	infeasible	0.16	X

Table 3.7: Network7 n=40 nodes , arcdensity=0.1 , Flow∼U[50,150]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
7000	218923	114.94	218923	0.66	0.00	218923	0.76	0.00
6000	218923	115.78	219106	0.66	0.08	219106	0.76	0.08
5000	218923	125.41	221003	0.66	0.95	220435	0.76	0.69
4000	218923	126.54	infeasible	0.72	X	infeasible	2.14	X
3000	infeasible	125.48	infeasible	0.66	X	infeasible	2.14	X

Table 3.8: Network8 n=40 nodes , arcdensity=0.2 , Flow∼U[50,150]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
5000	160700	123.17	160700	0.66	0.00	160700	0.77	0.00
4000	160700	122.85	160700	0.66	0.00	160700	0.77	0.00
3000	160700	124.00	160968	0.66	0.17	160849	0.77	0.09
2500	160700	123.06	161754	0.66	0.66	161236	0.76	0.33
2000	infeasible	123.48	infeasible	0.66	X	infeasible	1.93	X

Table 3.9: Network9 n=40 nodes , arcdensity=0.3 , Flow∼U[50,150]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
2000	129657	143.53	129657	0.66	0.00	129657	0.82	0.00
1500	129657	143.06	129840	0.65	0.14	129657	0.82	0.00
1000	129657	151.82	130799	0.71	0.88	129866	0.77	0.16
750	129657	175.10	infeasible	0.66	X	131943	0.82	1.76
500	Time Limit	X	infeasible	0.66	X	infeasible	1.92	X

Table 3.10: Network10 n=50 nodes , arcdensity=0.1 , Flow∼U[0,100]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
5000	155931	456.38	155931	3.07	0.00	155931	3.24	0.00
4000	155931	463.23	155948	3.08	0.01	155931	3.24	0.00
3000	155931	458.01	156314	3.08	0.25	156314	3.24	0.25
2500	155931	452.24	157328	3.08	0.90	157328	3.24	0.90
2000	155931	472.42	infeasible	3.07	X	infeasible	3.13	X
1000	infeasible	470.12	infeasible	3.02	X	infeasible	3.13	X

Table 3.11: Network11 n=50 nodes , arcdensity=0.2 , Flow∼U[0,100]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
3000	113008	496.51	113008	2.96	0.00	113008	3.13	0.00
2000	113008	493.76	113008	3.02	0.00	113008	3.18	0.00
1000	113008	507.35	113354	2.97	0.31	113092	3.18	0.07
750	113008	514.42	114140	3.02	1.00	114139	3.13	1.00
500	113008	1862.61	infeasible	2.91	X	infeasible	3.02	X
400	Time Limit	X	infeasible	2.91	X	infeasible	3.02	X

Table 3.12: Network12 n=50 nodes , arcdensity=0.3 , Flow∼U[0,100]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
2000	105983	541.48	105983	3.13	0.00	105983	3.30	0.00
1000	105983	546.58	106090	3.13	0.10	105983	3.30	0.00
500	105983	591.52	107971	3.07	1.88	106742	3.30	0.72
400	105983	2267.98	infeasible	3.07	X	108194	3.24	2.09
300	Time Limit	X	infeasible	3.03	X	infeasible	3.08	X

Table 3.13: Network13 n=60 nodes , arcdensity=0.1 , Flow∼U[0,100]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
3000	209697	3379.09	209780	8.94	0.04	209780	9.55	0.04
2500	209697	3344.01	210654	9.07	0.45	210425	9.66	0.35
2000	209697	3413.62	213351	9.09	1.74	213351	9.61	1.74
1500	209697	3368.38	infeasible	9.86	X	infeasible	9.34	X
1000	infeasible	3313.34	infeasible	8.82	X	infeasible	9.12	X

Table 3.14: Network14 n=80 nodes , arcdensity=0.1 , Flow∼U[0,100]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
4000	Time Limit	X	354473	42.62	X	354473	43.67	X
3000	Time Limit	X	354682	42.51	X	354473	44.77	X
2000	Time Limit	X	357095	42.73	X	356579	43.17	X
1500	Time Limit	X	infeasible	40.84	X	infeasible	42.00	X
1000	Time Limit	X	infeasible	41.04	X	infeasible	42.00	X

Table 3.15: Network15 n=100 nodes , arcdensity=0.1 , Flow∼U[0,100]

	Cplex Solution		Heuristic 1			Heuristic 2		
capacity(i,j)	Optimum	CPU	Solution	CPU	Gap(%)	Solution	CPU	Gap(%)
4000	Time Limit	X	551852	209.76	X	551803	211.90	X
3000	Time Limit	X	552367	199.43	X	552107	212.28	X
2000	Time Limit	X	556681	203.66	X	555521	211.64	X
1500	Time Limit	X	infeasible	200.26	X	infeasible	234.97	X
1000	Time Limit	X	infeasible	201.46	X	infeasible	205.42	X

3.5 Conclusion

We presented a simple, easy to program and very efficient heuristic algorithm for the integer multicommodity flow problem in which there exists a commodity for every pair of nodes. The main idea of the heuristic is to route the commodities one by one. We reported results of computational experiments using randomly generated networks with different characteristics.

The results showed that the proposed heuristic is indeed very efficient in delivering a reasonably high quality approximate solution to problems. We believe that the idea of routing commodities one by one over the network can be applied to several network flow problems.

Chapter 4

Finding an Upper Bound

In this chapter, we present a Tabu Search based heuristic to find an upper bound to the optimal value. The heuristic starts with an initial solution and finds neighborhood solutions by a local search. Then, at each iteration it moves to the solution with smallest cost value. Hence, the proposed algorithm has a greedy approach. In order to eliminate cycling, whenever a move is applied, it is put in a Tabu List and it can not be applied in some of the future iterations. For how many iterations a move is kept in Tabu List is a problem parameter, and it is determined experimentally.

We find the cost of a possible move in two steps. First, we find the location cost, simply multiplying the number of links used by the location cost C. The second cost, which is the routing cost, is found by calling the heuristic algorithm proposed in the previous chapter. This algorithm returns either the cost of routing, or says that the routing subproblem is infeasible. If it returns a cost value, it is added to the location cost, and the total cost of the move is calculated. If it returns that the routing subproblem is infeasible, then the move is taken to be infeasible.

4.1 Proposed Algorithm

The algorithm proposed to find an upper bound for our problem is as follows:

```
  Step 1. Read the Traffic Values and Delay Values from Infile
Step 2. Make necessary Initializations
Step 3. Generate Greedy Solution
    Step 3.1. Order commodities in descending order of Traffic Value
    Step 3.2. Set $counter = 1$
        Step 3.2.1. Set $i = source[OrderedList(counter)]$ , set $j = destination[OrderedList(counter)]$
        Step 3.2.2. If $degree[i] < MaxDegree$ and $degree[j] < MaxDegree$ then put an edge between $i$ and $j$
        Step 3.2.3. Increase $counter$ by 1, if $counter \leq |K|$ then go to Step 3.2.1, else go to Step 3.3.
    Step 3.3. Check whether this network is connected, if not go to Step 3.2. by increasing counter by 1.
    Step 3.4. Call Integer Multicommodity Flow Routine. Check whether every commodity is routed within maximum delay value, if not go to Step 3.2. by increasing counter by 1.
    Step 3.5. Go to Step 5., PROCESS
Step 4. Generate Random Solution
    Step 4.1. For every node $i$ in the network do
        Step 4.1.1. Set $counter = 0$
        Step 4.1.2. Increase $counter$ by 1
        Step 4.1.3. Generate a number $j$ between 1 and $|N|$, randomly
        Step 4.1.4. If $i \neq j$ and $degree[i] < P$ then put an edge between $i$ and $j$
        Step 4.1.5. If $counter < |N|$ then goto Step 4.1.1
    Step 4.2. Check whether this network is connected, if not go to Step 4.1.
    Step 4.3. Call Integer Multicommodity Flow Routine. Check whether every commodity is routed within maximum delay value, if not go to Step 4.1.
    Step 4.4. Go to Step 5., PROCESS
Step 5.(PROCESS) Set $counter = 1$
    Step 5.1. Find the $DisjointPairList$
        Step 5.1.1. Look at every edge pair in the network, if sources and destinations
```

of two edges are different and if it is not in the *TabuList* of *DisjointPairs* then put this edge pair into the *DisjointPairList*

Step 5.2. Find the *AdditionPairList*

Step 5.2.1. Look at every node pair in the network, if the degrees of both nodes are less than maximum degree and if it is not in the *TabuList* of *AdditionPairs* then put this node pair into the *AdditionPairList*

Step 5.3. Find the *DeletionPairList*

Step 5.3.1. For every edge in the network if it is not in the *TabuList* of *DeletionPairs* put it into the *DeletionPairList*

Step 5.4. For every element in *DisjointPairList* do

Step 5.4.1. Delete these two edges, and add two edges: one connecting tail of edge1 with tail of edge2 and another one connecting head of edge1 and head of edge2

Step 5.4.2. Call Integer Multicommodity Flow Routine, to find a solution to the routing. It returns the *RoutingCost*.

Step 5.4.3. Find the *LocationCost* by multiplying number of links in the network with the cost of a lint, that is C.

Step 5.4.4. Find $TotalCost = RoutingCost + LocationCost$ and record this move into the *SearchList*

Step 5.4.5. Reverse the changes done in Step 5.4.1.

Step 5.4.6. Delete the two edges in the *DisjointPirList*, and add two edges: one connecting tail of edge1 with head of edge2 and another one connecting head of edge1 and tail of edge2

Step 5.4.7. Call Integer Multicommodity Flow Routine, to find a solution to the routing. It returns the *RoutingCost*.

Step 5.4.8. Find the *LocationCost* by multiplying number of links in the network with the cost of a lint, that is C.

Step 5.4.9. Find $TotalCost = RoutingCost + LocationCost$ and record this move into the *SearchList*

Step 5.4.10. Reverse the changes done in Step 5.4.6.

Step 5.5. For every element in *AdditionPairList* do

Step 5.5.1. Add an edge connecting the two nodes in *AdditionPairList*

Step 5.5.2. Call Integer Multicommodity Routine, to find a solution to the routing. It returns the *RoutingCost*.

Step 5.5.3. Find the *LocationCost* by multiplying number of links in the network with the cost of a lint, that is C.

Step 5.5.4. Find $TotalCost = RoutingCost + LocationCost$ and record this move into the $SearchList$

Step 5.5.5. Reverse the changes done in Step 5.5.1.

Step 5.6. For every element in $DeletionPairList$ do

Step 5.6.1. Delete the edge in the $DeletionPairList$

Step 5.6.2. Call Integer Multicommodity Routine, to find a solution to the routing. It returns the $RoutingCost$.

Step 5.6.3. Find the $LocationCost$ by multiplying number of links in the network with the cost of a lint, that is C.

Step 5.6.4. Find $TotalCost = RoutingCost + LocationCost$ and record this move into the $SearchList$

Step 5.6.5. Reverse the changes done in Step 5.6.1.

Step 5.7 Find the move with the minimum cost in the $SearchList$ and apply it

Step 5.8 If $Cost < BestValueFound$ then update $BestValueFound$

Step 5.9 Increase *counter* by 1.

Step 5.10 If $counter < ReplicationNumber$ then go to Step 5.1.

4.2 Description of the Algorithm

The algorithm given in the last section works as follows. First of all we have to find some initial solutions. We have two different procedures to generate initial solutions. The first one is the greedy solution, and the second one is the random solution.

In the greedy solution, we sort the commodities in the descending order of traffic flow. Then, starting from the first element of this list, we put a link between the source and destination nodes of the commodity. When putting a link, we check whether the degrees of these nodes is less than maximum degree or not. If the degree of both nodes is less than maximum degree then we put a link between them. Otherwise, we do not put a link. Hence, we check every element in the ordered list of commodities, and locate links with the above criteria. Then we check two things, first whether the network is connected or not, if not we start with the second element in the orderedlist, and continue with the same principles. Secondly, we call the algorithm for integer multicommodity flow problem, and find the routing of every commodity. Then, we check whether every commodity is routed within the maximum delay value or not. If not, we again start with the second element in the orderedlist, and continue with the same principles. At the end of this procedure, we get a connected network topology, where every commodity is routed within the maximum delay value and the nodes with higher traffic are linked to each other. Clearly, with this procedure, we get a network in which the commodities with higher traffic amounts can be routed with in small distances, therefore we may decrease the routing cost. This procedure is very similar to the heuristic proposed by Ramaswami and Sivarajan [10]. They called this procedure as HLDA, and proposed it as a heuristic for network topology design problems. Therefore one of the initial solutions that we start Tabu Search algorithm is the HLDA of Ramaswami and Sivarajan [10].

Secondly, we generate random initial solutions. We randomly select two nodes, and if the degrees of these nodes are less than a predetermined number we connect them by a link. Here, we use a predetermined number rather than the maximum degree. This is a flexibility that we bring to the algorithm. Thus we can generate initial random solutions with different edge densities. At the end of the random initial solution procedure, we check whether the network is connected or not. If it is not connected, we generate a new one. Then, we call the algorithm for integer multicommodity flow problem. After finding the routing of every commodity, we check whether they are routed within the maximum delay value or not. If not, we generate a new one. In the computational experiments we generate several random initial solutions with different edge densities. For example, we take maximum degree as 8. But when generating random initial networks, we generate networks where maximum degree is 3, 5 and 8, respectively. This method increases our chances of obtaining better solutions.

Both of the initial solution generation algorithms return a network that satisfies bandwidth capacity, degree and maximum delay constraints. Therefore, we start Tabu Search with feasible solutions.

Whenever we generate an initial solution, either by greedy approach or randomly, we send this network to "Process" procedure. In the "Process" procedure we simply do a local search, i.e, find the neighborhood solutions. The cost of each neighborhood is calculated, and then we move to the one with smallest cost. Another important part of the proposed algorithm is how to find neighborhood solutions. We developed three different methods to find neighborhood solutions: Edge Interchange, Edge Addition and Edge Deletion. Let us analyze each of them separately.

Edge Interchange method is borrowed from Karaşan et al.(2002). In this method we first find pair of disjoint edges. Two edges are called a disjoint edge pair if the nodes they are connected to are different from each other. Assume we have two edges E_1 and E_2. Let E_1 connects nodes i_1 and j_1 and let E_2 connects nodes i_2 and j_2. Then, E_1 and E_2 are called disjoint if tails and heads of E_1 and E_2 are different, in other words $i_1 \neq i_2$, $i_1 \neq j_2$ and $j_1 \neq i_1$, $j_1 \neq i_2$. After finding all disjoint edges, we apply edge interchange in two ways. In the first one, we delete these edges and put edges on (i_1, i_2) and (j_1, j_2), called *type*1 interchange. In the second one, we again delete edges E_1 and E_2 and put edges on (i_1, j_2) and (j_1, i_2), called *type*2 interchange. Both of these new networks are neighborhood solutions. But we do not know whether these two neighborhoods are feasible solutions or not. The only constraint that is automatically satisfied during this change is the degree constraint. Because after deleting E_1 and E_2, we put two new edges and the number of links adjoint to the corresponding nodes i_1, j_1, i_2, j_2 does not change. The only thing we have to do is to call integer multicommodity flow routine. This routine returns us either a routing of commodities, or it says that it can not find a feasible routing of commodities. If it can not find a routing we do not take this move into account. If it finds a routing then we check whether all the commodities are routed within the maximum delay value or not. Again if not, we do not take this move into account. If it returns a feasible routing such that all of the commodities are routed within the maximum delay value and this move is not in the Tabu List for Edge Interchange, we calculate the cost of this move by summing the routing cost with location cost. Then this move is stored in the Search List. For every element in the Disjoint Edge List, we do this process twice, one for the *type*1 and another for the *type*2 change. After calculating the cost of a possible move, we reverse the change that we do in order to return to the current network that we are processing.

Another method of generating neighborhood solutions is the Edge Addition Method. In this method, we first find the nodes whose degrees are not at maximum degree. Then, we put these nodes into the Edge Addition List in pairs. Every pair of node in this list is a candidate to put a new link between them. Then, we add a link between these pairs to find neighborhood solutions. Again the only constraint that is automatically satisfied is the degree constraint because we select the nodes whose degrees are not at maximum degree limit. For every new network generated by Edge Addition, we call integer multicommodity flow routine. Again, if it does not find a routing or the commodities are not routed within the maximum delay value in the returned routing then this move is not taken into account. If a routing is found such that every commodity is routed within maximum delay value in this routing and this move is not in the Tabu List of Edge Addition, then we calculate the total cost and store this move in the Search List. After every move, we reverse the change that we do.

The final method that is proposed to find neighborhood solutions is the Edge Deletion Method. Every link in the current network topology is a candidate link to delete. Therefore, deletion of every link generates a neighborhood solution to the current network. Edge Deletion List consists of the links in the current network. For every element in this list, we delete the corresponding link and generate a neighborhood solution. The degree constraint is still satisfied by this change, because we are decreasing the degrees of the nodes connected to the deleted link. In order to check whether bandwidth capacity and delay constraints are satisfied by this network, we call integer multicommodity flow routine. If it does not find a routing or the commodities are not routed within the maximum delay value in the routing or this move is in the Tabu List for Edge Deletion, then we don't take this move into account. Otherwise, we calculate the cost of move, and store this move in the Search List. Then we reverse the changes we do, in other words add the deleted edge in order to return to the current network.

After finding all possible moves, we select the move in the Search List with minimum cost and apply this move permanently. After application of the move we put it into the Tabu List. We keep a tabu list for every neighborhood generation method. In other words we have a Tabu List for Edge Interchange, a Tabu List for Edge Addition and a Tabu List for Edge Deletion.

For how many replications a move will stay active in the Tabu List, i.e, the Tabu Number, is an important parameter of the algorithm. We keep a Tabu List in order to prevent cycling, otherwise the same move may take place repeatedly, and the efficiency of algorithm decreases. Determining an optimum Tabu Number is an open question. There is no method to find it explicitly. The only method we can approximate it is to make some computational experiments. If we take a large Tabu Number, then we block some moves and decrease our chance to find a better solution. However, if we take a small Tabu Number, then we increase the risk of cycling. During our computations, although we can not get a definite pattern, it is observed that taking Tabu Number equal to 5 is a good choice. Therefore, we take Tabu Number equal to 5 in our computational experiments for testing the proposed algorithm.

One of the parameters of the proposed the algorithm is the Replication Number which is the number of times we process an entering initial solution. As the Replication Number increases we apply more moves for an initial solution, therefore our chance to find a better solution increases. The expense of increasing Replication Number is the increase in running time of the algorithm.

Another parameter is how many initial random solutions to generate. As it is mentioned earlier, we can generate initial random solutions with different edge densities. If we generate more initial random solutions again we increase the chance of finding a better solution, but still at the expense of an increase in the running time of the algorithm. Therefore, we have to find a balance between the number of random solutions to be generated and the replication number. This is an algorithmic issue, and finding an optimum balance between these two parameters is an open question to researchers.

4.3 Computational Experiments

In our computational experiments, we take replication number equal to 100 and we generate 15 initial random solutions, 5 with maximum degree 3, 5 with maximum degree 5 and 5 with maximum degree 8. Therefore we start with 16 initial solutions, 15 random and 1 greedy. Total number of iterations is 16x100=1600.

We have 10 test problems characteristics of which are presented in Table(2.1). Experiments are done for three different α values. Problem parameters are as defined in Chapter 2. We also run a modification of HLDA heuristic proposed by [10]. The modified version of HLDA works as follows. It first sorts the commodities in descending order of flow amounts and connects the sources and destinations of commodities in this list until a feasible solution is attained. This algorithm is simply the greedy initial solution generation procedure that is used in our Tabu Search heuristic.

The computational results are presented in Tables 5.1-3 below.

First column expresses the lower bounds attained by adding the valid inequalities that are introduced in Chapter 2. Heuristic column represents the results of our heuristic, and HLDA column represents the results of the modified HLDA heuristic. Gap columns show the percentage gap between the heuristics and the lower bound.

Our heuristic programs are coded in Delphi 6.0 and run on a Pentium II PC. In all of the experiments we made 1600 iterations and the computing times for the heuristics are around 2 hours. The lower bounds are obtained by Cplex 7.1.

It is observed that our heuristic gives better results than HLDA in all of the problems. We also note that heuristic results are closer to lower bounds for higher α values. The gaps between our heuristics and lower bounds are around %17 for $\alpha = 1$, %34 for $\alpha = 0.2$ and %72 for $\alpha = 0$. As the percentage of routing cost in the total cost decrases the effect of our heuristic decreases too. If the percentage of routing cost is smaller in a problem, then link location decisions become more critical. Our Tabu Search heuristic is simply a local search, at each iteration it evaluates some neighborhood solutions and moves to the one with minimum cost. But, when the percentage of routing cost in total cost decrases then the percentage of location cost in total cost automatically increases. It means that the cost of a solution depends more on the topology than routing. Since we visit only the neighborhood topologies of the initial solutions, the effect of local search in finding good solutions decreases as α decreases.

It is also observed that the percentage of Routing Cost in Total Cost is around %80 for $\alpha = 1$, around %50 for $\alpha = 0.2$ and finally it is %0 for $\alpha = 0$.

Table 4.1: Experiments with $\alpha = 1$

	$LP + (1) + (2)$	Heuristic	*Gap*(%)	$HLDA$	*Gap*(%)
Problem1	16247	18899	16,3	20928	28,8
Problem2	17406	21049	20,9	22824	31,1
Problem3	16949	19829	17,0	22172	30,8
Problem4	16651	19640	18,0	21278	27,8
Problem5	16904	19958	18,1	21816	29,1
Problem6	26310	30641	16,5	33160	26,1
Problem7	27480	32011	16,5	35984	30,9
Problem8	27208	31727	16,6	35268	29,6
Problem9	39115	45162	15,5	50958	30,3
Problem10	40136	46653	16,2	51878	29,3

Table 4.2: Experiments with $\alpha = 0.2$

	$LP + (1) + (2)$	Heuristic	*Gap*(%)	$HLDA$	*Gap*(%)
Problem1	4593	6186	34,7	7849	70,9
Problem2	5013	6852	36,7	8498	69,5
Problem3	4787	6335	32,3	8146	70,2
Problem4	4711	6311	34,0	8286	75,9
Problem5	4802	6334	31,9	8378	74,5
Problem6	7264	9592	32,0	11832	62,9
Problem7	7786	10259	31,8	13012	67,1
Problem8	7668	10250	33,7	12544	63,6
Problem9	10788	14821	37,4	17362	60,9
Problem10	11096	15142	36,5	17486	57,6

Table 4.3: Experiments with $\alpha = 0$

	$LP + (1) + (2)$	Heuristic	$Gap(\%)$	$HLDA$	$Gap(\%)$
Problem1	1266	2200	73,8	2800	121,2
Problem2	1278	2200	72,1	2800	119,1
Problem3	1306	2200	68,5	2700	106,7
Problem4	1296	2200	69,8	2800	116,0
Problem5	1318	2200	66,9	2800	112,4
Problem6	1778	3100	74,4	3700	108,1
Problem7	1808	3100	71,5	3800	110,2
Problem8	1785	3100	73,7	3700	107,3
Problem9	2375	4200	76,8	5100	114,7
Problem10	2502	4400	75,9	5100	103,8

Chapter 5

Conclusion

In this thesis, we analyzed the problem of topology design in Virtual Private Networks(VPNs). The problem has two parts: location of links and routing of commodities over these links. Therefore, our aim is to find a feasible network topology and a feasible routing over that topology while minimizing the cost. The cost also has two components: location cost of links and routing cost of commodities. The constraints of the problem are bandwidth capacity constraint, degree constraint, maximum delay constraint and finally the usual flow conservation constraint.

After introducing the Integer Programming Formulation of the problem, two sets of valid inequalities are introduced: Cut Inequalities and Link Inequalities. Then, some computational experiments are done in order to observe the effects of these inequalities. Link Inequalities are found to be more powerful than Cut Inequalities. After adding both of the inequalities to the LP Relaxation of the IP formulation, it is observed that lower bounds are improved by around %75.

The problem is very difficult to solve optimally. The optimum solution time of small size problems (like, $|N| = 20$) takes more than 10 hours. Therefore we developed a heuristic approach to the problem. It is a Tabu Search algorithm. The algorithm starts with initial feasible solutions and at each iteration it finds neighborhood solutions and moves to the one with minimum cost. In order to find the cost of each neighborhood we have to find both location cost and routing cost. Location cost is simply the link cost times the number of links used in the topology. But finding the routing cost is not that simple. Because given a network topology, traffic pattern and edge capacities finding the optimum routing is a very difficult problem. This problem is named as Integer Multicommodity Flow Problem. Since we have to find routing costs for every possible move, we need a fast solution procedure. But in our computational experiments it is observed that finding the optimum solution to $IMCFP$ is a time consuming process. Therefore we introduced a fast and efficient heuristic algorithm that finds feasible solutions very close to optimum in very small cpu times. Chapter 3 discussed the Integer Multicommodity Flow Problem independently and described the proposed algorithm. Hence, another outcome of this thesis is an efficient heuristic algorithm for Integer Multicommodity Flow Problem.

The proposed algorithm for $IMCFP$ is used as a subroutine in the main algorithm for topology design problem. At each iteration, we call this algorithm and find the cost of routing. After calculating the cost of each possible move we select the move with the minimum cost, and this move is taken to be "tabu" for next iterations in order to prevent cycling. The proposed algorithm is tested on several problems with different characteristics. The results are compared with the lower bounds found by adding cut and link inequalities to the LP relaxation. We also compare these results with the HLDA heuristic proposed by Ramaswami and Sivarajan [10]. It is found that our algorithm works better than HLDA.

A further research avenue can be to use branch-and-bound or cutting plane techniques. Also a problem with nonlinear cost structure can be an interesting future research topic. In this thesis we take location and routing cost to be linear. But, in some cases cost of locating a link may depend on where we are locating that link, therefore we may have different location costs for possible links. Also, routing cost may be in nonlinear form. In such cases finding the cost of possible moves becomes more complex and we may need new ideas for finding good solutions.

Bibliography

[1] A.Assad, ,Multicommodity Network Flows: A Survey, *Networks*, Vol.8, No.1, 37–82, 1978.

[2] T.L.Magnanti and R.T.Wong, Network Design and Transportation Planning: Models and Algorithms, *Transportation Science*, vol.18, no.1, pages 1-55, 1984.

[3] M.Minoux, Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications, *Networks*, vol.19, pages 313-360, 1989.

[4] I.Chlamtac, A.Ganz and G.Karmi, Lightpath Communications: An Approach to High Bandwidth Optical WAN's, *IEEE Transactions on Communications*, vol.40, no.7, pages 1171-1182, 1992.

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

[6] S.Ahn, R.P.Tsang, S.R.Tong and D.H.C.Du, Virtual Path Layout Design on ATM Networks, *Tecnical Report, CS Department, University of Minnesota*, TR94-29, 1994.

[7] B.Gendron and T.G.Crainic, Relaxations for Multicommodity Capacitated Network Design Problems, *Publication CRT-965, Centre de recherche sur les transports, Universite de Montreal*, 1994.

[8] T.L.Magnanti, P.Mirchandani and R.Vachani, Modeling and Solving the Two Facility capacitated Network Loading Problem, *Operations Research*, vol.43, no.1, pages 142-157, January-February 1995.

[9] D.Bienstock and O.Günlük, Computational Experience with a difficult mixed-integer multicommodity flow problem, *Mathematical Programming*, vol.68, pages 213-237, 1995.

[10] R.Ramaswami and K.N.Sivarajan, Design of Logical Topologies for Wavelength-Routed Optical Networks, *IEEE Journal on Selected Areas in Communications*, vol.14, no.5, pages 840-8541, 1996.

[11] F.Barahona, Network Design Using Cut Inequalities, *SIAM Journal on Optimization*, vol.6, no.3, pages 823-837, August 1996.

[12] D.Bienstock and O.Günlük, Capacitated Network Design-Polyhedral Structure and Computation, *INFORMS Journal on Computing*, vol.8, no.3, pages 243-259, 1996.

[13] K.Holmberg and D.Yuan, A Lagrangian Heuristic Based Branch-and-Bound Approachfor the Capacitated Network Design Problems, *Research Report LITH-MAT-R-1996-23, department of Mathematics, Linkoping Institude of Technology*, 1996.

[14] D.Alevras, M.Grotschel and W.Wessaly, Cost-Efficient Network Synthesis from Leased Lines, *Preprint SC 97-22, konrad-Zuse-Zentrum fur Informationstechnik*, Berlin, 1997.

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

[16] O.Günlük, A Branch-and-Cut Algorithm for Capacitated Network Design Problems, *Mathematical Programming*, Series A, vol.86, no.1, pages 17-39, 1999.

[17] D.Bienstock, S.Chopra, O.Günlük and C-Y.Tsai, Minimum Cost Capacity Installation for Multicommodity Network Flows, *Mathematical Programming*, Series B, vol.81, no.2-1, pages 177-199, July 1998.

[18] B.Gendron, T.G.Crainic and A.Fragioni, Multicommodity Capacitated Network Design, *Telecommunications Network Planning*, pages 1-19, 1998.

[19] G.Dahl, A.Martin and M.Stoer, Routing through Virtual Paths in Layered Telecommunication Networks, *Operations Research*, vol.47, no.5, pages 693-702, 1999.

[20] C. Barnhart, C. A. Hane, P. H. Vance, Using Branch-and-Bound-and-Cut to Solve Origin-Destination Integer Multicommodity Flow Problems, *Operations Research* vol.48, no.2, pages 318-326, March-April 2000.

[21] E.Karaşan, O.Ekin-Karaşan, N.Akar and M.Ç.Pınar, Mesh Topology Design in Overlay Virtual Networks, *Electronics Letters*, vol.38, no.16, pages 939-941, August 2002.

[22] J.Kenington, E.Olinick, A.Ortynski and G.Spride, Wavelength Routing and Assignment in a Survivable WDM Mesh Network, *Operations Research*, vol.51, no.1, pages 67-79, January-February 2003.