2005

An improved routing strategy for wavelength routed WDM networks.

M. Abul Kalam
University of Windsor

An improved Routing strategy for wavelength routed WDM Networks

By

M. Abul Kalam

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through the School of Computer Science
In Partial Fulfillment of the Requirements for
The Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2005

Your file Votre référence
ISBN: 0-494-09830-9
Our file Notre référence
ISBN: 0-494-09830-9

Abstract

The routing problem in Wavelength-division multiplexing (WDM) networks, on a given logical topology, is to find an optimum scheme for data communication so that the network may handle all traffic requirements in an efficient manner and minimize the network congestion. This problem is typically solved using a Linear Programming (LP) formulation using the node-arc representation and a LP solver such as the CPLEX. As the network size increases, the time taken by a tool like CPLEX to find a solution to the routing problem becomes unacceptable.

This thesis investigates a novel technique, using the arc-chain formulation for representing the logical topology, to speed up the solution of the routing problem. We use *eta-factorization* with the generalized upper bound (GUB) technique to solve such problems. Experiments show that our approach significantly outperforms the standard LP techniques based on a node-arc formulation, in terms of the time required to generate an optimal solution.

Keywords: WDM, multi-commodity, congestion, Linear Programming, Generalized upper bounding, eta-factorization.

to my mother and family……

Acknowledgments

I would like to take this opportunity to thank my advisors, Dr. Subir Bandyophadhyay and Dr. Yash Aneja for their constant support and guidance through out this journey. I am grateful to Dr. Kobti, Dr. Das for their appreciated suggestions. Special thanks to Dr. Jaekel for her suggestion and guidance to write this thesis.

It also gives me pleasure to thank all of my friends and colleagues whom I have worked with during my graduation study. Finally, I want to express my gratitude to my family for their support and love over these years.

Table of Contents

List of Figures and Tables

Chapter 1: Introduction

With the explosive growth of Internet as well as of data traffic, there is a growing demand for huge bandwidth. Optical networks employing the Wavelength-division multiplexing (WDM) technology represent the most promising candidate to meet this increasing demand. The estimates show that today's Asynchronous Transfer mode Networks (ATM) do not have the capacity to fulfil the exponential growth in data flow or traffic demand by user's applications [Mu00]. WDM technology allows a single optical fiber to send out many light beams of different wavelengths simultaneously with tremendous transmissible bandwidth. Because, a single link carries tens of Terabits per second with extremely low loss [Mu00], a huge amount of data is affected when network failure occurs. A WDM network consists of a set of nodes, physically interconnected by optical fiber (the physical topology), upon which a logical topology is overlaid by establishing lightpath [Mu00] interconnections between the nodes. WDM networks are considered to be the future wide-area backbone networks.

WDM network design usually is divided in two sub-problems: Network design and Routing and Wavelength Assignment (RWA) [XY02]. The Network design involves physical topology and configuration design [XY02]. The physical topology of a WDM network is defined by

- Network access station/ end-nodes, typically an access station is equipped with number of transmitters and receivers to transmit data to or to receive data from multiple data sources.
- Optical Cross Connects (OXS) can route the optical signal coming on a wavelength towards their respective destinations and
- Fiber links, that provides the physical medium for optical communication.

A Logical topology of a WDM network is the topology viewed by the higher layer such as SONET, ATM, IP [HA00]. It is often defined by a Graph of end nodes and the lightpaths between them. A lightpath is a logical all-optical connection established to satisfy data communication requests between a source node and a destination node. In a WDM network, logical topology is represented by a graph ***LG,*** where the nodes in ***LG*** are the end-nodes in the network and, and there is a directed edge $i \rightarrow j$ in ***LG*** (often called a logical edge) from node ***i*** to node ***j***, if the node pair (***i, j***) are connected by a lightpath. [AJB04]. Congestion of a WDM network is defined as the maximum load offered in a logical link.

The next step of the network design is to determine the optimal routing strategy considering the traffic requirements between the end nodes and the logical topology. Minimizing network congestion is a way to optimize routing strategy. If a traffic load t is given for a source destination (s, d) pair, then the challenge of the routing strategy is to find a suitable path/ paths to route it in an optimal manner. In general there are many path from **s** to d; one may be $s = x_1 \rightarrow x_2 \rightarrow x_3 \ldots \rightarrow x_p = d$. The routing strategy determines how much traffic should be carried by this path while taking into account about each of the path from source s to destination d. As the objective of the routing strategy is to minimize congestion, we have to consider all the node pairs in the network that have some traffic demand.

This thesis studies techniques for minimizing the congestion in wavelength routed WDM networks by adopting some techniques used in the Operation Research community for multi-commodity network flows.

1.2 Visualization of Routing Problem in a WDM network as a Multi-Commodity Network Flow Model

The multi-commodity network flow model, a well known optimization model arises in the areas of transportation, production and communication. The model concerns flow routing of a number of commodities (e.g. message, vehicles) through a capacitated network at minimal cost. In the basic model it is assumed that for each commodity, the flow can be routed on any path connecting its origin and destination. Linear programming (LP) formulation is generally used to solve this type of problems. The shared resources of the network are expressed using linear constraints in an LP.

In a WDM network, each source destination node pair(s, t) with traffic demand t sends t amount data from **s** to **d**. The data communication from all sources to destination shares the network resources. If we view the data between each node pair with traffic demand ($t(s, d) \neq 0$) as a distinct commodity, the routing problem in wavelength routed WDM networks may be viewed as a multi commodity network flow model.

1.3 Motivation

If the physical topology and traffic matrix of a WDM network is given, then the routing problem may be formulated as LP [RS96]. A straight-forward LP formulation can be solved by a

commercial LP solver package, like CPLEX [BM00] to give optimum routing with minimum congestion. For small network, such an approach is feasible.

Visualising the WDM network as multi-commodity flow problem shows that the number of commodities with N nodes is very close to *N(N-1)*, since most nodes communicate with each other. Thus for a 30-node network, the number of commodities is nearly 870. In operation-research multi-commodity flow normally deals with a small number of commodities. So, dealing with large number of commodities becomes a challenging problem, as the number of constraints increases rapidly with the size of the network. A previous experiment shows that CPLEX solver takes long time to give a solution for larger network and moreover, it can't handle networks with more than 40 nodes.

Thus we introduce a different algorithm to solve the problem. We use arc-chain representation to formulate the problem, explore the advantage of the Generalized Upper Bounding (GUB) structure of the basis and then use eta-factorization to avoid direct inversion of the matrix. This process has a dramatic effect on the time to perform an iteration of the revised simplex method.

1.4 Solution Outline

The network flow problem is formulated in a number of ways in the operation-research field. Arc-chain representation has been profitably used for solving minimum-cost network flow problems [To66][FF58]. This representation can be readily adapted to the problem of minimizing congestion of wavelength-routed WDM network. To solve the problem using arc-chain representation, by the revised simplex method the most expensive operation is that of inverting a matrix called the basis [Ta82]. For a network with m number edge in logical topology and q number commodity the size of basis is $(m+q) \times (m+q)$. So, the time needed to invert a matrix limits the size of the network that can be safely handle.

At each iteration of the revised simplex method we need to find an entering column to improve the solution. In our approach, instead of generating all possible columns and storing them before starting the iteration process, we generate the entering column at each iteration. This implicit column generation saves time and memory space.

It is established that the constraints in our problem have a special structure called, Generalized Upper Bounding (GUB) structure [DS67]. It is well known that if a LP satisfies the GUB

structure, we can avoid the invention of the entire basis $(m+q)$ and can achieve the same result by inverting the matrix of size $m \times m$, which provides a significant improvement on time to perform an iteration of the revised simplex method. A new technique to represent matrices S and T is used with GUB structure which reduces the computational cost for vector-matrix and matrix-vector multiplication.

Though the GUB structure gives a significant improvement in time, inverting the matrix of size $m \times m$ takes most of the time to obtain a solution in the revised simplex method. So, we introduce eta-factorization to eliminate direct matrix inversion. Eta-factorization is used in the GUB structure to obtain the solution and find the simplex multiplier at each iteration of the revised simplex method. Use of eta-factorization on the GUB structure dramatically improves the time to perform a iteration of the revised simplex method.

Using these approaches we have formulated our algorithm for routing in a WDM network and implemented the algorithm in C language. We have tested our program on various size networks.

1.5 Organization of thesis

In chapter two of this thesis, we have given a literature review and basic terminology in the fields of WDM network and operations research. Chapter three describes our formulation for the routing problem. In chapter four, we have given the experimental setup and the results of our algorithm for some small to large size networks along with a critical summary of our work. Conclusion and future direction of work is also included in chapter four.

Chapter 2: Back ground review of related techniques

2.1 WDM Optical Networks

Wavelength-division multiplexing (WDM) is an approach that can exploit huge opto- electronic bandwidth mismatch by requiring that each end-user's equipment operate only at electronic rate, but multiple WDM channels from different end-users may be multiplexed on the same fiber. Under WDM, the optical transmission spectrum is carved up into a number of non-overlapping wavelength (or frequency) bands, with each wavelength supporting a single communication channel operating at whatever rate one desires, e.g., peak electronic speed [M00]. Thus, by allowing multiple WDM channels to coexist on a single fiber, one can tap into huge fiber bandwidth, with corresponding challenges being the design and development of appropriate network architectures, protocols, and algorithms. Also, WDM devices are easier to implement since, generally, all components in a WDM device need to operate only at electronic speed; as a result, several WDM devices are available in the marketplace today, and more are emerging. Optical cross-connects (OXCs), Wavelength division multiplexing (WDM) and demultiplexing (WDDM) are among them.

This chapter will review optical network technology, multiplexing/demultiplexing techniques; brief description of some commonly used devices in fiber technology and some terminology

2.1.1 Optical fiber

Optical fiber is essentially a thin filament of glass/plastic, which acts as a wave-guide [M00]. An optical fiber consists of two concentric layers termed the core and the cladding. These are shown on the right side of Figure 2-1. The core and cladding have different indices of refraction with the core having n_1 and the cladding n_2. Light is piped through the core. A fiber optic cable has an additional coating around the cladding called the jacket. Core, cladding and jacket are all shown in the three dimensional view on the left side of Figure 2-1. The jacket usually consists of one or more layers of polymer. Its role is to protect the core and cladding from shocks that might affect their optical or physical properties [KS]

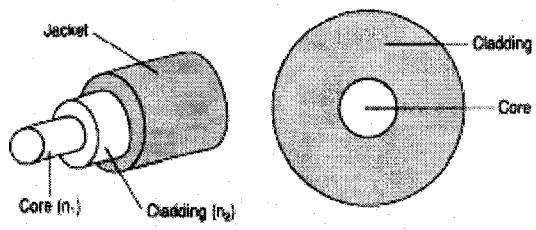


Figure 2.1: Fiber Optic Cable, 3 dimensional view and basic cross section [KS]

Because of the total internal reflection phenomena of light, light can travel the length of a fiber with very little loss. This occurs because the core and cladding have different indices of refraction with the index of the core, n_1, always being greater than the index of the cladding, n_2. Figure 2-2 shows how this is employed to effect the propagation of light down the fiber optic cable and confine it to the core. If the light ray is injected and strikes the core-to-cladding interface at an angle greater than the critical angle then it is reflected back into the core. Since the angle of incidence is always equal to the angle of reflection, the reflected light will again be reflected. The light ray will then continue this bouncing path down the length of the fiber optic cable. If the light ray strikes the core-to-cladding interface at an angle less than the critical angle then it passes into the cladding where it is attenuated very rapidly with propagation distance. This angle is fixed by the indices of refraction of the core and cladding and is given by the formula:

θ_c = arc *cosine* (n_2 / n_1).

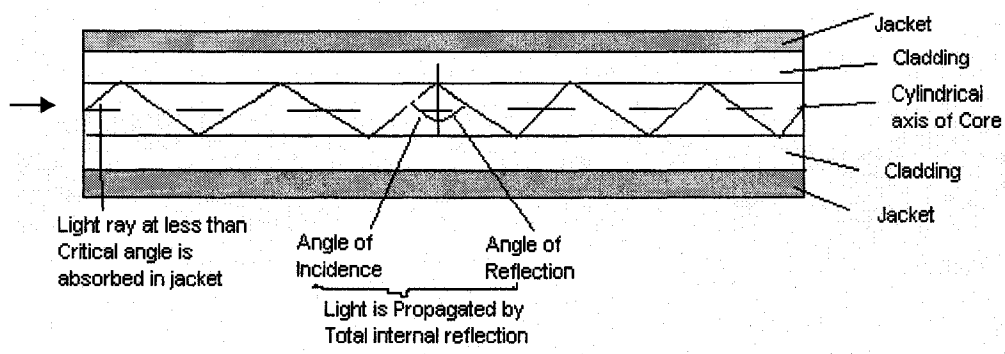


Figure 2.2: Propagation of a light ray down a fiber optic cable [RS]

The critical angle is measured from the cylindrical axis of the core. By way of example, if n_1 = 1.446 and n_2= 1.430 then a quick computation will show that the critical angle is 8.53 degrees, a fairly small angle.

2.1.2 Wavelength Division Multiplexing (WDM) Technology

"Wavelength-division multiplexing (WDM) is an approach that can exploit the huge optoelectronic bandwidth mismatch by requiring that each end-user's equipment operate only at electronic rate, but multiple WDM channels from different end-users may be multiplexed on the same fiber"[Mu00]. This technique divides the huge bandwidth of a fiber into many non-overlapping bands of wavelength, each operating at a desirable speed [Mu00]. These channels can be modulated to accommodate dissimilar data formats, including analog and digital. This technique improves the transmission capacity of a fiber having multiple channels at different carrier wavelength.

2.1.3 Components of optical network

In order to develop appropriate network architecture, it is required to have protocols, algorithms and hardware components for real world implementation. Research for the development of optical hardware is ongoing for decades. It is anticipated that next generation Internet will employ WDM optical backbone. A Four channel point-to-point WDM transmission system using optical amplifiers, multiplexers and demultiplexers is shown in figure 2.3

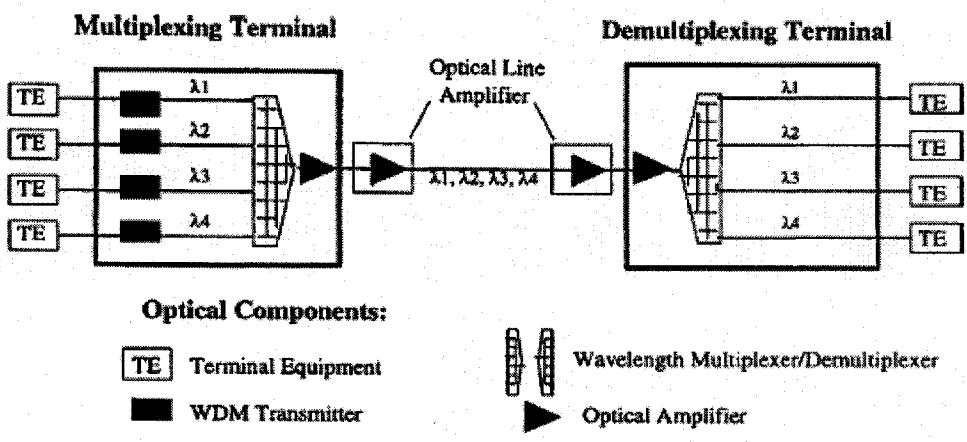


Figure 2.3: A Four-channel point-to-point transmission channel with amplifiers [M00]

Evolution of the WDM network is successful due to stable lasers, erbium doped fiber amplifiers and optical low cross talk, stable and compact filters. The tuning bandwidth and speed, and temperature stability are essential characteristic of lasers are suitable for WDM networks [AV +2001].

OXC: An optical cross connect (OXC) has several input port and output ports. Each OXC can switch the optical signal coming on a wavelength of an input fiber link to the same wavelength in an output fiber link, without requiring the signal to undergo any optoelectronic conversion. If an OXC is equipped with converters, it can switch the optical signal on an incoming wavelength of an input fiber to any wavelength in output fiber. A typical OXC with optical switches is shown in figure 2.4.

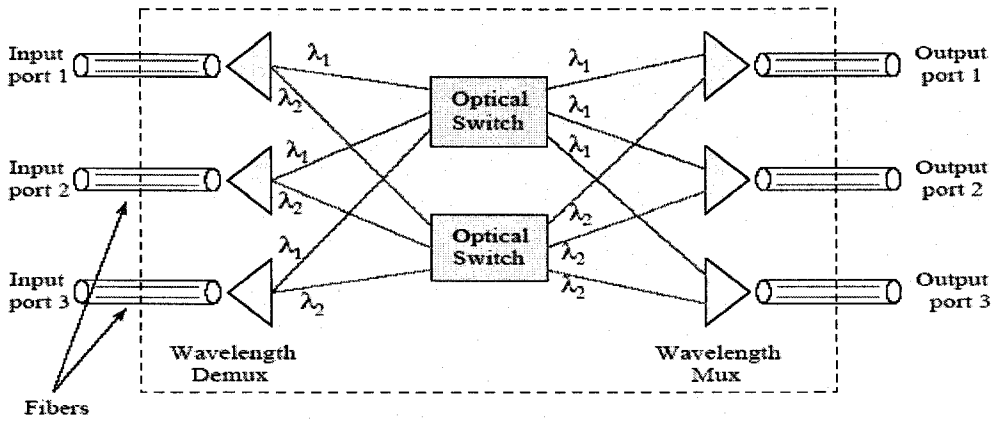


Figure 2.4: OXC [G92]

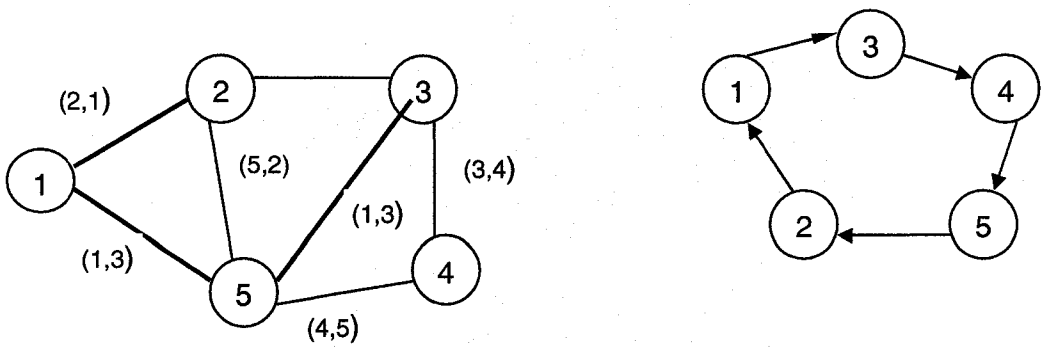
2.1.4 Terminology

Lightpath: A *lightpath* in an optical network is a point-to-point communication path that connects a transmitter at a source node to a receiver at a destination node through a number of router nodes where no optoelectronic conversion is needed at any intermediate node. [AJB04]. Two lightpath in the fiber link must be of different wavelength to prevent interference of the optical signals [RM99].

Link: A link is a point-to-point optical fiber connection. Each link is able to carry multiple wavelengths.

Physical topology: The physical topology of an optical network consists of a set of end-nodes (capable of generating data for transmission, receiving data and having a number of optical transmitters and receivers), router nodes and the optical fibers interconnecting these nodes [AJB04]. It is the network topology seen by the optical layer.

Logical topology: The logical topology is the network topology seen by the higher layer e.g. the layer above the optical layer. In a WDM network, logical topology is represented by a graph ***LG,*** where the nodes in ***LG*** are the end-nodes in the network and, and there is a directed edge $i \rightarrow j$ in ***LG*** (often called a logical edge) from node i to node j, if the node pair (i, j) are connected by a lightpath. [AJB04]. A physical topology and corresponding logical topology is shown in figure 2.5. Where lightpath is established between the nodes 2→1, 1→5→3, 5→2, 4→5, 3→4 of physical topology. All the connected edge (1-3. 2-5 etc.) of logical topology are called link.

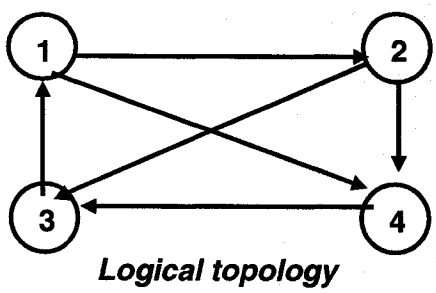


a) Physical topology *b) Logical topology*

Figure 2.5: physical and logical topology

Congestion: The congestion of the network is defined as the load on the logical edge, which carries maximum amount of data. Our target is to minimize congestion. [AJB04]

Traffic Matrix: Traffic in a network is the amount of data that must be transmitted from each source node to desired destination. It is a square matrix of size equal to number of node in the network. The $(i,j)^{th}$ entry in the traffic matrix determines the amount of data that flows from node i to node j.



Node	1	2	3	4
1	0	15	0	40
2	0	0	18	22
3	10	0	0	0
4	0	0	20	0

Figure 2.6: 5-Node logical topology and its traffic matrix

Figure 2.6 shows a logical topology of a network with four nodes and corresponding traffic matrix. The diagonal elements of the traffic matrix are zeros, as data can't flow from and to same node. In the figure it shows that 15 units of data are sent from node 1 to 2. Usually the

traffic matrix is expressed as a percentage of the capacity of light path. For example, the traffic shown in fig 2.6, node 1 to 2 is using 15% of its capacity.

2.1.5 Routing in WDM Network

Routing strategy in a WDM network is the policy of sending data from all source nodes to the corresponding destination nodes using appropriate paths. Traffic can be send from source node to its destination using one or more paths, where each path consists of one or more logical edge or lightpaths. In the figure 2.6, the 40 units of data need to be sent from node 1 to 4. This amount can be sent via following two paths:
Using logical link 1 → 4 and using logical link 1→2→4. We can use either one of the two paths or can distribute the traffic over both paths. For every pair of source destination nodes that has some traffic to route we have to consider every possible route in the logical topology. Our objective is to route all the traffic in a way that minimize the congestion in the network and thus optimize the use of optical resources in network.

2.1.6 WDM Network Architecture:

WDM network technology makes it possible for end users to communicate via all-optical WDM channels, which may span multiple fiber links. The architecture of a wavelength-routed WDM network is studied in several papers [RM99], [CHMR98], [G91], [GD02], [IMG96], [RSM03]. "The WDM architecture consists of wavelength cross connects interconnected by fiber links [RM99]." The optical cross-connect (OXC) can route optical signal without optoelectronic conversion. Each network link has a pair of unidirectional fiber links. The network has access station associated with each optical OXC. An access station helps to transmit signals on different wavelengths, which are coupled into the fiber using wavelength multiplexers. A wavelength-routed optical network is shown in figure 2.7.

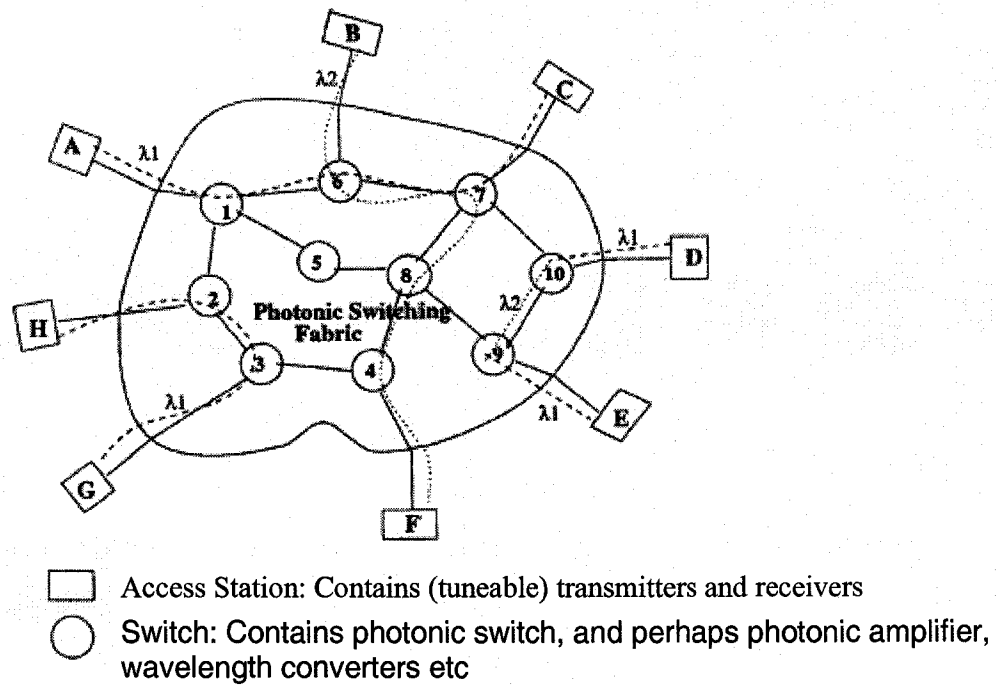


Figure 2.7: A wavelength-routed (wide-area) optical WDM networks. [M00]

2.2 Introduction to Linear Programming (LP) and Revised Simplex Method

In mathematics, linear programming (LP) problems are optimization problems in which the objective function and the constraints are all linear [Taha]. Linear programming is the process of solving LP problems.

Linear programming is an important field of optimization for several reasons. LP problems are the easiest kind of optimization problems, since everything is linear. Furthermore, many practical problems in operations research can be expressed as linear programming problems. Finally, many algorithms for other optimization problems work by solving LP problems [Taha75].

An Integer Linear Programming (ILP) problem is a linear-programming problem with the additional constraint that variables must take on integral values [ILOG]. In general, it is NP-Complete Problem. The general form of an ILP problem is written as follows:

$$
\begin{array}{lll}
(LP\ Parts:) & \max & C^T x \\
 & such\ that & Ax \le b \\
(ILP\ Part:) & & x \in Z
\end{array}
$$

Where c is a column vector, A is a matrix representing coefficient of x, b is the constraint values and Z is the boundary value or acceptable range of values for x. We can also formulate a problem in the form:

Minimize (or maximize) $\quad c_1x_1 + c_2x_2 + \ldots + c_nx_n$

subject to
$$
\begin{aligned}
& a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \sim b_1 \\
& a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \sim b_2 \ldots \\
& a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \sim b_m
\end{aligned}
$$

with these bounds
$$
\begin{aligned}
& l_i \le x_i \le u_i \\
& l_n \le x_n \le u_n
\end{aligned}
$$

Where ~ can be ≤, ≥ or =, and the upper bounds u_i and lower bounds *li* may be positive infinity, negative infinity or any real number [CPLEX01].

In this example, all the constraints are of the form $\sum_j a_{ij}x_j \le b_i$, where b_i is a scalar. Here c_1, c_2,..., *cn* are the *cost coefficients*, $z = \sum_{j=1}^{n} c_j x_j$ is the *objective function* to be minimized, x_1, x_2, ..., x_n are the *decision variables*. There are m_c *inequality constraints* and these constraints can be represented as a constraints matrix A such that

$$
A = \begin{bmatrix}
a_{11} & a_{12} & .... & a_{1n} \\
a_{21} & a_{22} & .... & a_{2n} \\
... & ... & ... & ... \\
a_{m_c1} & a_{m_c2} & .... & a_{m_cn}
\end{bmatrix}
$$

To solve any LP problem, all the inequality constraints in the problem must be transformed into equality constraints. In case of the problems where all constraints are in the form $\sum_j a_{ij}x_j \le b_i$ we add a non-negative slack variable to transform the constraints to equations. Thus the constraint equation becomes

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + x_s^1 = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + x_s^2 = b_2$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{m_c1}x_1 + a_{m_c2}x_2 + \ldots + a_{m_cn}x_n + x_s^{m_c} = b_{m_c}$$

Where $x_1, x_2, x_3, \ldots x_n \geq 0$

Since there are m number of constraints, m number of slack variables are added to the matrix. The new equation becomes $Ax = b$ where A is a matrix of size $m \times (n+m)$ and b is column vector of m elements, x is the vector consisting of the n decision variables $x_1, x_2, x_3, \ldots x_n$ followed by m slack variables $x_s^1, x_s^2, \ldots x_s^{m_c}$. In explaining the simplex algorithm we will use notation $\hat{a}_j$ to represent the j^{th} Column of A and if p is a vector of m we will use $p(i)$ to represent the i^{th} element of p.

When the constraints of the problem are of the form $\sum_j a_{ij}x_j = b_i$, we will add an artificial variable to create equation of form $\sum_j a_{ij}x_j + x_a^i = b_i$ and the objective function will be changed to $z = \sum_i c_i x_i + Mx_a$, where M is a vary large constant[BJ90]. When the constraints are of form $\sum_j a_{ij}x_j \geq b_i$, we will subtract a slack variable x_s^i to convert the constraint into equality constraint $\sum_j a_{ij}x_j - x_s^i = b_i$. Then using artificial variable converts it to equality equation. Thus equation will be in the form $\sum_j a_{ij}x_j + x_a^i - x_s^i = b_i$ and the objective function will be $z = \sum_i c_i x_i + Mx_a - x_s^i$.

When the equations of a problem are in the form $Ax = b$ and $x \geq 0$, and there exist a solution vector x that satisfies all the constraints of the problem, it is called a *feasible solution.* We can partition the components of the x in such a way that there will be m basic variables and n non-basic variables, where all the non-basic variables will have the value of 0. A matrix B having the columns of matrix A corresponding to the set of basic variables is called a *basis*. We call the vector $x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$ a basic solution of the system where $x_B = B^{-1}b$ and $x_N = 0$. When $x_B \geq 0$, then the vector x_B of basic variables is called a *basic feasible solution* of the system [BJ90]. The corresponding value of the objective function is $\hat{c}_B B^{-1}b$ where $\hat{c}_B$ is a vector of the cost coefficients, corresponding to the basic variables in x_B.

The simplex method is the first method introduced by G.B Dantzig [DOW55] to solve linear programming formulation in which one basic feasible solution is replaced by adjacent solution. The revised simplex method is an efficient implementation of the standard simplex method when the non basic variable is much greater then the number of constraints. If we are given a basic feasible solution with basis B, basic variables x_B and cost vector $\hat{c}_B$, then we carry out the following steps [BJ90] iteratively to get optimal solution:

Step 1: Calculate vector $\hat{b} = B^{-1}b$ and the objective function $z = \hat{c}_B\hat{b}$ where $\hat{c}_B$ is the vector of cost of basis.

Step 2: Calculate vector $\hat{w} = \hat{c}_B B^{-1}$, where $\hat{w}$ is called the vector of simplex multipliers.

Step 3: For each non-basic variable x_j, calculate $z_j - c_j = \hat{w}\hat{a}_j - c_j$. Let k be such that the value of $z_k - c_k \geq z_j - c_j, \forall j,\ 1 \leq j \leq n + m_c$. If $z_k - c_k \leq 0$, stop; otherwise go to step 4.

Step 4: Calculate vector $\hat{d}_k = B^{-1}\hat{a}_k$, if $\hat{d}_k \leq 0$ stop; otherwise go to step 5.

Step 5: Calculate $r_{\min} = \text{minimum}\{\hat{b}(i)/\hat{d}_k(i) : \hat{d}_k(i) > 0\}, \forall i, 1 \leq i \leq m$ and let b_r be the index of $\hat{b}$ and $\hat{d}_k$ corresponding to $r_{\min}$. This gives us the variable x_{br} to leave the basis.

Step 4: Update the basis by replacing $\hat{a}_{Br}$ column with $\hat{a}_k$.

2.3 Flow of Commodities in a Network

Optimizing the flow of goods or messages in a network is an important research topic. The target is to use a transportation system to move a single entity or multiple entities from a source to its destination in an efficient way that makes the best use of the system resources. This transportation system could be a roadway, railway or for instance could be a computer network, where data need to be transported. We can visualize the transportation system by a graph where nodes are the actual/ potential sources and destinations and arcs correspond to the transportation routes. We will represent an arc from node i to node j by $i \rightarrow j$. The entity being transported is called a *commodity*. The arc (route) for carrying commodities may have limitations. Say for a given arc $i \rightarrow j$, the maximal carrying capacity can be represented by u_{ij}.

The generic models representing the practical applications can be classified as single commodity flow models and multi commodity flow models. The single commodity network problem sends one commodity from the source to the destination in some optimal way. For example a shipper

wants to ship one type of goods from a railway station to different location (rail station) by train. All the locations have different demands for the goods. The objective is to meet the demand with a minimum possible transportation cost. To deal with this problem, network flow model can be used. Here the rail stations are nodes and the train line joining the stations are the arcs. Available capacity of the carrying wagon in a route restricts the total carrying capacity.

In reality, a number of commodities, regulated by their own network flow constraints, share the network and the capacity of an arc limits the total sum of flows of all the commodities using the arc. To optimize such situation we use special mathematical model – the *multi-commodity network flow* model [AMO93]. In this model several commodities share the arc capacity with their own flow constraints. For example, we want to ship different types of goods to various destinations using a railway transportation system. Each destination has a different demand for goods. This is a case of multi-commodity flow problem as different goods will share the wagon and the wagon has capacity limitations.

2.3.1 Single Commodity Maximal Flow Problem

Single commodity maximal flow problem is a network optimization problem, where the objective is to maximize the flow of a commodity from a designated node called the *source* node to another designated node called the *destination* node (sometimes called a *sink* node), without exceeding the capacity of any arc [FF56]. The source has to supply sufficient amount of flow to meet the demand of the destination. There may exist some node in the network that does not supply or receive flows, are called intermediate nodes. The incoming and outgoing flow in an intermediate node must be equal.

In Figure 2.8 a 4 nodes network is shown with nodes numbered 1 to 4. We associate a label to each of the arc of the network to describe the current flow and the maximal flow capacity of that arc. For example an arc $i \rightarrow j$ with label (x_{ij}, u_{ij}) denotes that the arc has capacity u_{ij} and the current flow is x_{ij} .

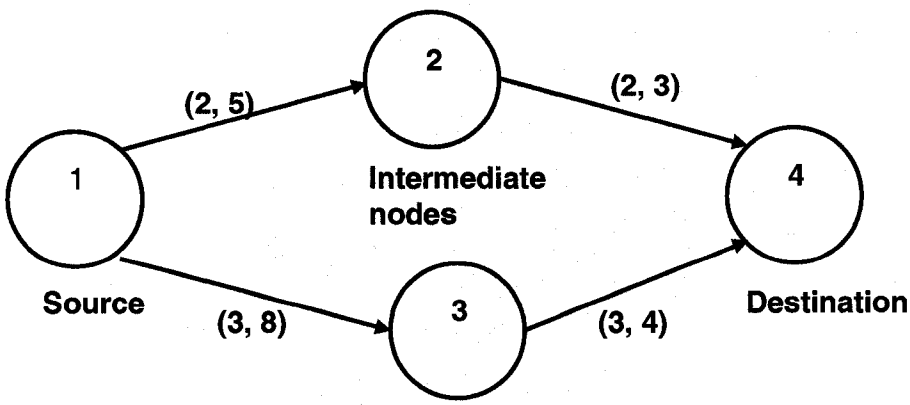


Figure 2.8: Single commodity network

In the network of Figure 2.8, node 1 is the source node and node 4 is the destination, and the demand to ship is 5 units. In this case the node 1 is sending 2 units of flow through arc 1→2 and 2→3 and 3 units of flow through arcs 1→3.and 3→4 to node 4. Thus 5 units of flow are sent to node 4, using two different routes, without exceeding the arc capacity. In a capacitated network G = (V, E), we can model a single commodity maximum flow problem where V is the set of nodes and E is the set of directed arcs, with a non-negative capacity u_{ij} associated with each arc $(i \rightarrow j) \in E$.

The problem formulation to find the maximum flow from source ***s*** to destination ***d*** satisfying the arc capacity and the flow conservation constraints is as follows [AMO 93]:

Maximize v

Subject to

$$\sum_{\{j:(i\rightarrow j)\in E\}} x_{ij} - \sum_{\{j:(j\rightarrow i)\in E\}} x_{ji} = \begin{cases} v & i = s \\ 0 & i \in V - \{s,d\} \\ -v & i = d \end{cases} \quad \ldots \quad (2.1)$$

$$0 \le x_{ij} \le u_{ij} \quad \forall\, i \rightarrow j \in E$$

This formulation refers to a flow vector $x = (x_{ij})$ which corresponds to the value of flow(scalar variable v). The total outflow from node i is the first term ($\sum_{\{j:(i\rightarrow j)\in E\}} x_{ij}$) of the flow conservation constraints and the second term ($\sum_{\{j:(j\rightarrow i)\in E\}} x_{ji}$) is total inflow into node i. Flow conservation constrains state that for all nodes, except the source and destination nodes, the outflow must be

equal to inflow. These constraints are also known as the *mass balance constraints*. Capacity constraints impose restrictions on the total flows on each arc.

2.3.2 Single Commodity Minimum Cost Network flow (MCF) Problem

In this problem the objective is to make the most economic and best use of the network resources. A directed graph G = (*V, E)* is used to represent the network with *N* number of nodes and *m* number of directed arcs, where *V* is a set of nodes and *E* is a set of directed arcs. The cost per unit of flow, denoted by c_{ij} is associated with each arc $i \rightarrow j \in E$. Like before, the maximum flow capacity of each arc $i \rightarrow j \in E$ *is* limited by u_{ij}. Each node $i \in V$ has a number r_i associated with it, which represents the requirement (i.e., supply/ demand) at that node. $r_i > 0$ denotes that a source node is supplying r_i units of flow, while $r_i < 0$ denotes a destination node with a demand of r_i, for all intermediate nodes $r_i = 0$. The decision variable x_{ij} is the flow on arc $i \rightarrow j$. The minimum cost flow problem [AMO93] can be formulated as

Minimize $\sum_{i \rightarrow j \in E} c_{ij} x_{ij}$

Subject to

$$\sum_{\{j:(i \rightarrow j) \in E\}} x_{ij} - \sum_{\{j:(j \rightarrow i) \in E\}} x_{ji} = r_i, \quad i \in V \qquad \ldots \quad (2.2)$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall\, i \rightarrow j \in E$$

2.3.4 Multi-Commodity Network flow Problem

In this problem a number of distinct commodities with their sources and corresponding set of destinations share a capacitated network. The network is represented by a directed graph G = (*V, E*) with each arc $i \rightarrow j$ having a capacity (cost/unit flow) $u_{ij}(c_{ij})$. Figure2.9 shows a 5 node network with a cost and a capacity (u_{ij}, c_{ij}) associated with every arc $i \rightarrow j$.

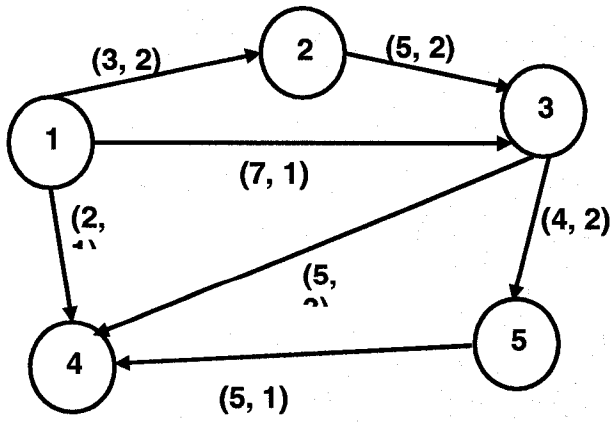


Figure 2.9: Multi-commodity Network flow

We consider a case where there are two commodities K^1 and K^2 flowing from node 1 to 5 and node 3 to 4 with a demand of 3 and 6 units respectively. Commodity K^1 may be sent using the paths 1 → 3 → 5 and or 1 → 2 → 3 → 5. Commodity K^2 can be sent using one or both the paths 3→ 4, 3 → 5 → 4. In this case for commodity K^2 we have to use both paths due to capacity constraints. As commodity K^1 has to use the arc 3 → 5 to send its 3 units of flow so only 1 (4 minus 3) unit capacity is available for commodity K^2 in arc 3 → 5. So, the remaining of the 5 (6 minus 1) units of commodity K^2 have to use the path 3→ 4. This situation is shown in figure 2.10. The total cost for all the commodities is the sum of the cost of all flows for each commodity. The cost for commodity K^1 and K^2 will be $3\times2+3\times2+3\times2=18$ and $1\times2+1\times1+5\times3=18$ respectively, thus total cost of the network flows become 36 units.

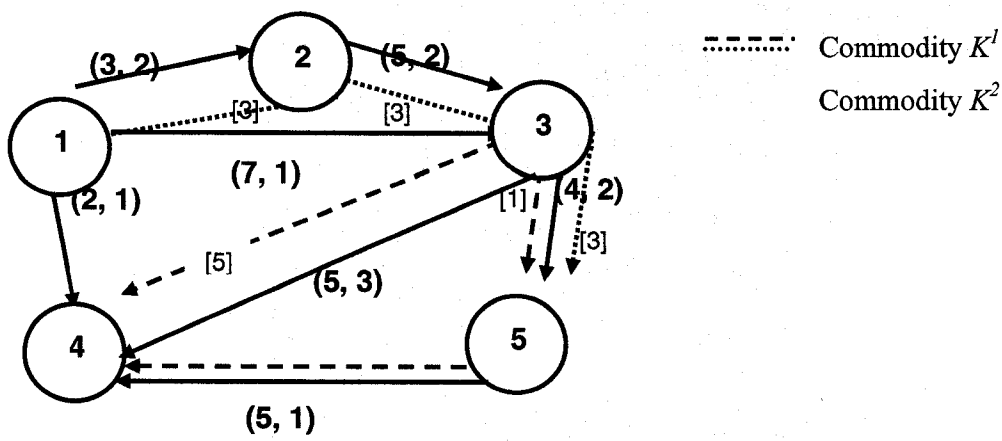


Figure 2.10 Multi-commodity Network flow

Mainly two types of multi-commodity problems are studied in literature [AMO93]. These are the maximum multi-commodity network flow problem and the minimum-cost multi-commodity network flow problem. The objective of the maximum multi-commodity network flow problem is to maximize the sum of flows for all commodities satisfying the capacity constraint for all arcs and for the minimum-cost multi-commodity network flow problem is to determine the demands of all commodities at a minimum cost without violating capacity constraint.

2.4 Representations for multi-commodity network flow problems

The two possible ways of representing the multi-commodity network flow problems, the node-arc representation and arc-chain representation are described here.

2.4.1 Node-Arc Incidence Matrix

A node-arc *incidence matrix* $N \times m$ represents a network with N nodes and m arcs, where the i^{th} row corresponds to the i^{th} node and the j^{th} column corresponds to the j^{th} arc. The column corresponding to the arc $i \rightarrow j$ (flow from node i to node j) has

- +1 in the row corresponding to node i,
- -1 in the row corresponding to node j and
- 0 in all other rows.

Figure 2.12 is the node-arc representation for the network shown in figure 2.11

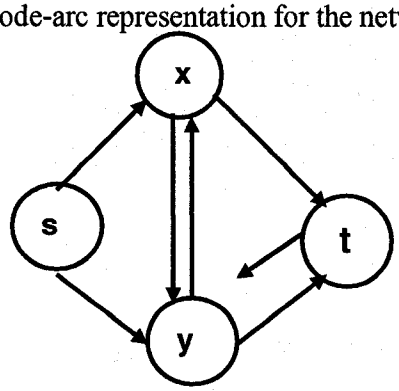


Figure 2.11: 4-Node network

	(s,x)	(s,y)	(x,y)	(y,x)	(x,t)	(y,t)	(ty)
s	1	1	0	0	0	0	0
x	-1	0	1	-1	1	0	0
y	0	-1	-1	1	0	1	-1
t	0	0	0	0	-1	-1	1

Figure 2.12: Node-Arc incidence matrix

2.4.2 Node-Arc Formulation for Multi-commodity network flow

Consider a network represented by directed graph G = (V, E) of N nodes and m arcs. Let a $N \times m$ matrix (N_a) denote the node-arc incidence matrix for this graph. To formulate the minimum cost multi-commodity (MMCF) network flow [AMO93] with q commodities in the network we will use the following notation:

source (k) = the source of the k^{th} commodity

destination (k) = the destination of the k^{th} commodity

r_i = the total amount of flow of commodity i from source(i) to destination(i)

x_{ij}^k = the flow for commodity k on arc $i \rightarrow j$

x^k = the entire flow vector for commodity k

$\hat{c}^k$ = the cost vector for x^k

u_{ij} = a capacity associate with each arc $i \rightarrow j$.

With this notation the MMCF can be formulated as

$$\text{Minimize} \sum_{k=1}^{q} \hat{c}^k x^k$$

Subject to :

The arc capacity constraints that restricts the total flow of all commodities on arc $i \rightarrow j$ to at most

$$u_{ij} \text{ is } \sum_{k=1}^{q} x_{ij}^k \le u_{ij} \quad \forall\, i \rightarrow j \in E \quad \ldots (2.3)$$

The mass balance constraint which state that each commodity has its own supply vector is

$$N_a x^k = b^k \quad \forall k : 1 \le k \le q \ \ldots (2.4)$$

Here b^k is a supply vector for commodity k, and $b^k(i)$ is defined as follows:

$b^k(i) = r_k$, if i = source(k)

$b^k(i) = -r_k$, if i = destination(k)

= 0, otherwise.

To restrict the flow of commodity k on arc $i \rightarrow j$ within u_{ij}^k is

$$0 \le x_{ij}^k \le u_{ij}^k \quad \forall\ k : 1 \le k \le q \quad \forall (i \rightarrow j) \in E \quad \ldots (2.5)$$

2.4.3 Arc-Chain Incidence Matrix

A *chain* [BJ90] is a sequence of arcs $[(i_0 \rightarrow i_1), (i_1 \rightarrow i_2), \ldots, (i_{p-1} \rightarrow i_p)]$ from a source s to a destination d, where $s = i_0$ and $d = i_p$.

In a network with m arcs numbered 1, 2, … , m, a chain can be represented by a vector of m 1's and 0's. In the network, if an arc (i) appears in the chain then the corresponding i^{th} element in the vector becomes 1 otherwise it is 0 $(1 \leq i \leq m.)$.

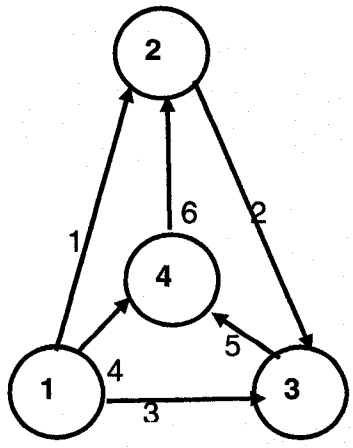


Figure 2.13: 4-node network

A simple networks with 4 node and 6 arcs, is shown in the figure 2.13, the network carries four commodities K^1, K^2, K^3 and. K^4. The nodes are numbered 1 to 4 and the arcs are numbered 1 to 6 as shown. The source and destination of commodity K^1 is node 1 and 2. There are three chains in this network for commodity K^1. These are arc 1→ 2 and the sequence of arcs [(1 → 4), (4 → 2)] and [(1 → 3), (3 → 4), (4 → 2)]. The chain 1→ 2 may be represented by the vector [1, 0, 0, 0, 0, 0] since the arc 1→ 2 corresponds to arc number 1 in the network. In this way the chain [(1 → 4), (4 → 2)] may be represented by the vector [0, 0, 0, 1, 0, 1] since arcs 1 → 4 and 4 → 2 correspond to arcs numbered 4 and 6 in the network, and third chain as [0, 0, 1, 0, 1, 1]. Commodity K^2 has source 3 and destination 4, commodity K^3 has source 2 and destination 4 and commodity K^4 has source 1 and destination 3. Commodity K^2 and K^3 each have a single chain represented by the vector [0, 0, 0, 0, 1, 0] and [0, 1, 0, 0, 1, 0] respectively. Commodity K^4 has three chains represented by vector [0, 0, 1, 0, 0, 0], [1, 1, 0, 0, 0, 0] and [0, 1, 0, 1, 0, 1].

To represent a network with m arcs and q commodities with N_k chains for the k^{th} commodity, the arc-chain incidence matrix uses a matrix $\mathbb{C}$ of size $m \times N_c$ where $N_c = N_1 + N_2 + \ldots + N_q$.

We will use C_j^k to represent the j^{th} chain of the k^{th} commodity, $1 \leq j \leq N_k$. C_j^k is a vector of

length m containing 0's and 1's only that represents the existence of arcs in that chain. Each chain of a commodity corresponds to a column in matrix $\mathbb{C}$, so that the first N_1 columns correspond to chains for commodity 1, the next N_2 columns correspond to chains for commodity 2, and so on. We will use a_{ij}^k to denote the i^{th} element of the j^{th} chain of the k^{th} commodity and $\mathbb{C}^k$ to denote the $m \times N_k$ sub-matrix of $\mathbb{C}$ corresponding to commodity k. In other words, if edge i is in chain j of commodity k, a_{ij}^k is 1; otherwise it is 0.

For example, Figure 2.14 shows the arc-chain matrix for the network shown in Figure 2.13 for commodities K^1, K^2, K^3, and K^4.

	K^1			K^2	K^3	K^4		
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1	1	0	0	0	0	0	1	0
2	0	0	0	0	1	0	0	1
3	0	0	1	0	0	1	0	0
4	0	1	0	0	0	0	0	1
5	0	0	1	1	1	0	0	0
6	0	1	1	0	0	0	1	1

Figure 2.14: The arc-chain matrix

It may be noted that even a small network with a modest number of commodities may have a large number of chains, which may results in a large arc-chain matrix.

2.5 Tomlin's Approach

Arc-chain formulation was adopted in revised simplex method by Ford and Fulkerson to compute the maximal flow multi-commodity network flows [FF58]. Tomlin used both the node-arc and the arc-chain formulation to solve Minimum-cost multi-commodity network flow problem [To66]. We will use Tomlin's arc-chain approach with some modification to solve our routing problem.

In a formal arc-chain formulation we have to handle a very large arc-chain matrix $\mathbb{C} = (a_{ij}^k)$ as the number of chains for all commodities becomes extremely large for a non-trivial network. By exploiting the special structure of the problem we can avoid dealing directly with the entire

matrix (creating all the chains). We assign numbers 1, 2, ……,m to the m arcs of the network, u_i and c_i are used to denote the capacity and cost of the i^{th} arc, and x_j^k is used to represent the flow of commodity k in chain C_j^k. Capacity vector $u_1, u_2, \ldots u_m$ for the edges 1, 2,…., m can be represent by $\hat{u}$. While the network is handling q commodities, r_k represent the flow requirement of the k^{th} commodity, $1 \leq k \leq q$. In arc-chain representation a_{ij}^k has a value 1 if the chain j of commodity k uses arc i, and a value of 0, otherwise.

The capacity constraint for the i^{th} arc, for all i, $1 \leq i \leq m$ is

$$\sum_{k=1}^{k=q} \sum_{j=1}^{j=N_k} a_{ij}^k x_j^k \leq u_i \quad \ldots \quad (2.6)$$

The flow requirement is expressed as:

$$\sum_{j=1}^{j=N_k} x_j^k = r_k \quad \ldots(2.7)$$

$\sum_{i=0}^{i=m} c_i a_{ij}^k$ is the cost associated with the chain C_j^k so the total cost for all flows with all commodities becomes $\sum_{k=1}^{k=q} \sum_{j=1}^{j=N_k} \sum_{i=1}^{i=m} c_i a_{ij}^k x_j^k$

Presentation of the problem in matrix notation is:

Minimize $c^{'}A^1x_1 + c^{'}A^2x_2 + \ldots\ldots + c^{'}A^qx_q$

Subject to $A^1x_1 + A^2x_2 + \ldots\ldots\ldots\ldots + A^qx_q + x_s = \hat{u}$ … (2.8)

$$\begin{aligned} e_1'x_1 &= r_1 \\ e_2'x_2 &= r_2 \\ &\ldots\ldots \\ e_q'x_q &= r_q \end{aligned} \quad \ldots \quad (2.9)$$

Here e_k represent the vector N_k for commodities $k = 1, \ldots q$ and x_s is a vector of m slack variables. As we are dealing with m arcs and q commodities, the formulation will have $m + q$ rows and a very large number of columns (one for each chain) in constraint matrix $\mathbb{C}$. To solve this problem we don't need to generate all these chains (column) explicitly. Instead we find only shortest chain for each commodity and check if this chain satisfies the simplex condition to be a part of an entering column as outlined in step 3 of the revised simplex algorithm. If the chain

satisfies the condition then the algorithm creates the chain and proceeds to the next step of the revised simplex algorithm.

Steps to find an entering column are as follows:

- We have a basic feasible solution and $m + q$ simplex multipliers in each iteration,
- Simplex multipliers for arc i are $\pi_1, \ldots\ldots, \pi_m$ and for commodity j are $\alpha_1, \ldots\ldots, \alpha_q$.
- If $\pi_i > 0$ for some i, then the slack variable x_s corresponding to i^{th} constraint in equation (2.8) will improve the objective function. So, x_s will enter into the basis.
- If $\pi_i < 0$ for some i, x_j^k will enter into basis if

$$\sum_{i=1}^{i=m} c_i a_{ij}^k - \sum_{i=1}^{i=m} \pi_i a_{ij}^k - \alpha_k < 0 \quad \ldots \quad (2.10)$$

$$\text{i.e., if } \sum_{i=1}^{i=m} (c_i - \pi_i) a_{ij}^k < \alpha_k \quad \ldots \quad (2.11).$$

As $\pi_i \leq 0$, for all values of i $(1 \leq i \leq m)$, cost $(c_i - \pi_i)$ can be assigned to arc i $(1 \leq i \leq m)$ to get non-negative weights for all arcs. Then $\sum_{i=1}^{i=m} (c_i - \pi_i) a_{ij}^k$ represents the length of the j^{th} chain for commodity k. This is the shortest chain for commodity k, because, if the shortest chain cannot satisfy equation 2.11, then no other chain will be able to satisfy it.

Tomlin's technique to find an entering variable can be summarised as.

a) for any positive simplex multiplier π_j, enter the slack variable x_s and stop.

b) assign cost $(c_i - \pi_i)$ to attached edge i $(1 \leq i \leq m)$. Enter the variable x_j^k into the basis if it satisfies the equation 2.11 and stop.

In this approach, the important point is that we only need to keep track of the basis matrix and not the chains corresponding to the non-basic variables (which are huge in number).

The initial basic feasible solution is obtained by introducing artificial variables in the last q equations and then minimizing the sum of all artificial variables [To66].

2.6 Generalized Upper-Bounding (GUB)

Generalized upper bounding (GUB) [DS57] is a technique, used to make the revised simplex method more efficient for linear programming problems when the problems have a special structure. A matrix of size $(m+q)\times(m+q)$ for some m and q, exhibits a GUB structure, if

- q is relatively large compared to m and
- each column of the last q rows has at most one nonzero entry equal to 1.

The following figure 2.15 shows a constraint matrix A having GUB structure where blank spaces has zero (0) entry.

$$A = \begin{bmatrix} 3 & 2 & 4 & 3 & 5 & 9 & 7 & 4 & 8 & 2 & 5 & 3 \\ 2 & 5 & 1 & 3 & 7 & 4 & 6 & 2 & 9 & 3 & 2 & 5 \\ 1 & 1 & 1 & & & & & & & & & \\ & & & 1 & 1 & & & & & & & \\ & & & & & 1 & 1 & 1 & 1 & & & \\ & & & & & & & & & 1 & 1 & \end{bmatrix} \begin{matrix} \} m \\ \\ \} q \end{matrix}$$

Figure 2.15: An arc-chain incidence matrix with special structure

For the revised simplex method explained in section 2.2, we are required to find $\hat{w}$ and $\hat{d}_k$, where $\hat{w} = \hat{c}_B B^{-1}$ and $\hat{d}_k = B^{-1}\hat{a}_k$, to carry out the ratio test in step 2 – 4 at each iteration. That means we have to solve a system of $(m + q)$ equations with $(m + q)$ variables. When the matrix size is very large then it may become computationally intractable. But, if the matrix satisfies the GUB structure, then we can largely reduce the computational expense. The idea is to update the basis matrix of size $m + q$, and compute $\hat{w}$ and $\hat{d}_k$, in successive iterations of the revised simplex method without inverting the whole matrix of size $m + q$, but to invert the sub-matrix of size m only. When q is much larger compared to m, then the GUB technique can dramatically improve the time to compute $\hat{w}$ and $\hat{d}_k$.

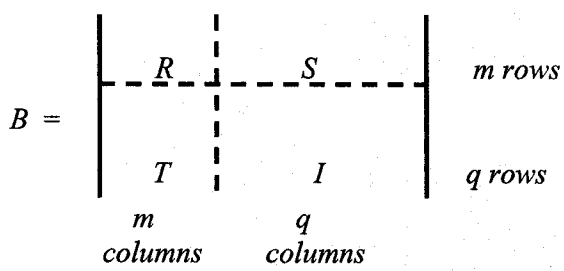


Figure 2.16: Basis satisfying GUB structure after permutation

In the revised simplex method, if we have a basis that satisfies the GUB structure, we can always apply column permutation to that basis matrix so that the resulting matrix has an identity sub-matrix of size *m x m* in its lower right corner. Figure 2.16 shows a basis matrix after column permutation. Here, S and T are matrices of size $(m \times m)$, $(m \times q)$ and $(q \times m)$, respectively. So, we can visualise the basis *B* as the composition of four sub-matrices *R, S, T* and *I*. In order to find the value of $\hat{w}$ and $\hat{d}_k$ in our problem we need to invert the matrix (*R - ST*) of size $m \times m$ as explained below. In our problem the basis *B* is of size $(m+q)$ and we need to solve the equations $\hat{w} \cdot B = \hat{c}_B$ and $B \cdot \hat{d}_k = \hat{a}_k$ at each iteration of the simplex algorithm. Let the basis *B* at iteration *k* after performing the permutation to get GUB structure is as follows

$$B = \begin{bmatrix} R_k & S_k \\ T_k & I \end{bmatrix}$$

Let the simplex multiplier vector $\hat{w}$ of size $(m+q)$ be divided into two parts, so that $\hat{w}'$ be the vector of the first *m* simplex multipliers and $\hat{w}''$ be the vector of the last *q* simplex multipliers. Similarly, the cost vector $\hat{c}_B$ is divided into $\hat{c}'_B$ and $\hat{c}''_B$ corresponding to the first *m* and last *q* elements. So the equation $\hat{w} \cdot B = \hat{c}_B$ can be written as

$$[\hat{w}' \quad \hat{w}''] \cdot \begin{bmatrix} R_K & S_K \\ T_K & I \end{bmatrix} = [\hat{c}'_B \quad \hat{c}''_B] \quad \ldots. (2.12)$$

we can re-express the equation (2.12) as

$$\hat{w}' R_k + \hat{w}'' T_k = \hat{c}'_B \ldots (2.13)$$

and

$$\hat{w}' S_k + \hat{w}'' = \hat{c}''_B \ldots (2.14)$$

From equation (2.13) and (2.14) substituting the value of $\hat{w}'$ in equation (2.13) we get

$$\hat{w}'(R_k - S_k T_k) = \hat{c}'_B - \hat{c}''_B T_k \ldots (2.15)$$

that implies $\hat{w}' = (\hat{c}'_B - \hat{c}''_B T_k) \cdot (R_k - S_k T_k)^{-1} \ldots (2.16)$

To calculate the value of $\hat{w}'$ we need to invert matrix $(R_k - S_k T_k)$ of size $m \times m$, which is much less expensive than inverting a matrix of size $(m+q) \times (m+q)$. Once we have the $\hat{w}'$ value we can calculate $\hat{w}''$ using equation (2.14) and can get the simplex multipliers.

Let $\hat{a}'_k$ and $\hat{a}''_k$ be the first *m* and last *q* elements of the entering column $\hat{a}_k$, $\hat{d}'_k$ and $\hat{d}''_k$ be the first *m* and last *q* elements of $\hat{d}_k$. We can write the equation $B \cdot \hat{d}_k = \hat{a}_k$ as

$$\begin{bmatrix} R_k & S_k \\ T_k & I \end{bmatrix} \cdot \begin{bmatrix} \hat{d}'_k \\ \hat{d}''_k \end{bmatrix} = \begin{bmatrix} \hat{a}'_k \\ \hat{a}''_k \end{bmatrix}.$$

This can be expressed as

$R_k \hat{d}'_k + S_k \hat{d}''_k = \hat{a}'_k \ldots(2.17)$

$T_k \hat{d}'_k + \hat{d}''_k = \hat{a}''_k \quad \ldots(2.18)$

Eliminating the value of $\hat{d}''_k$ from the equation (2.18) and (2.17), we get

$(R_k - S_k T_k)\hat{d}'_k = \hat{a}'_k - S_k \hat{a}'' \ldots(2.19)$

From this equation (2.19), we may get the value of $\hat{d}'_k$ as

$\hat{d}'_k = (\hat{a}'_k - S_k \hat{a}'') \cdot (R_k - S_k T_k)^{-1} \ldots(2.20)$

Using the equation (2.18) we can get the value of $\hat{d}''_k$ as

$\hat{d}'' = \hat{a}''_k - T_k \hat{d}'_k \ldots(2.21)$

To get the value of $\hat{d}_k$ using equation (2.20) and (2.21), the expensive operation is to invert the matrix $(R_k - S_k T_k)$, which is done earlier while calculating the value for $\hat{w}$. So, we can readily use that result.

2.7 Updating the matrices(R, S, T)

To take advantage of the GUB technique, we have to maintain the GUB structure in each iteration of the revised simplex algorithm after updating (removing a column and inserting a column) the basis. Here is the technique that is used to ensure that the requisite form $B = \begin{bmatrix} R & S \\ T & I \end{bmatrix}$ is always maintained. This approach is described in [Ch83]. $\hat{d}'_k = [d'_1, d'_2, \ldots, d'_m]$ is used to denote a vector of size m consisting of the first m elements from $\hat{d}_k$. As the basis B changes due to update at each iteration, the matrix R, S and T also changes. Let B_k, R_k, S_k and T_k be the matrices at iteration K and R_{k+1}, S_{k+1} and T_{k+1} be the matrices at iteration k+1. Our objective is to compute R_{k+1}, S_{k+1} and T_{k+1} from R_k, S_k and T_k.

Depending on the position of the entering column and the leaving column three different situations arise. The different cases are described below:

Case 1: The leaving column h is one of the first m columns of the basis (i.e. $h \leq m$). In this case we just replace the leaving column with the entering column to update basis. Since $h \leq m$, this update will only affect the matrix R and T. So, updating R and T will update the entire basis $B = \begin{bmatrix} R_{k+1} & S_k \\ T_{k+1} & I \end{bmatrix}$ and the new basis still satisfies the GUB structure. The expression for $(R_{k+1} - S_{k+1}T_{k+1})$ will be

$$(R_{k+1} - S_{k+1}T_{k+1}) = (R_k - S_kT_k)F_{k+1} \quad \ldots (2.22)$$

Here F_{k+1} is called an *eta* matrix [Ch83], which is an identity matrix of size $m \times m$ with inserting vector $\hat{d}'_k$ at position h.

$$F_{k+1} = \begin{bmatrix} 1 & 0\ldots & d'_1 & \ldots 0 \\ 0 & 1\ldots & d'_2 & \ldots 0 \\ \vdots & \vdots & \vdots & \ldots \vdots \\ 0 & 0\ldots & d'_m & \ldots 1 \end{bmatrix}$$

Case 2 and 3 arises when the leaving column is one of the last q column of the basis (i.e. $m < h$). We will use f' to denote a vector consisting of the first m elements of f and f'' to denote last q element of f. We will use g to denote any other first m column in the basis where g' and g'' denote a vector consisting of the first m and last q elements of g.

Case 2: If there exist another column g such that $f'' = g''$. In this case, we can interchange the leaving column f and the column g. Now we can update basis like case 1. Steps are follows:

Step1. Interchange the two columns f and g of the existing basis. So basis $B = \begin{bmatrix} R_k & S_k \\ T_k & I \end{bmatrix}$ becomes $\tilde{B}_{\text{intermidiate}} = \begin{bmatrix} \tilde{R}_k & S_{k+1} \\ T_K & I \end{bmatrix}$.

Step2. Insert the entering column replacing the column f to obtain the final basis.

$\tilde{B}_{\text{intermidiate}} = \begin{bmatrix} \tilde{R}_k & S_{k+1} \\ T_K & I \end{bmatrix}$ becomes $B_{new} = \begin{bmatrix} R_{k+1} & S_{k+1} \\ T_{k+1} & I \end{bmatrix}$.

Let f originally be the $(m + i)^{\text{th}}$ column of B_k and let $\dot{r}$ denote the i^{th} row of T_k. If J_{k+1} is the $m \times m$ identity matrix whose h^{th} row has been replaced by $-\dot{r}$ then

$(\widetilde{R}_k - S_{k+1}T_k) = (R_k - S_kT_k)J_{k+1}$ and to obtain the final basis we need to multiply it with eta matrix F_k.

$(R_k - S_kT_k) = (R_0 - S_0T_0)J_1F_1J_2F_2.....J_kF_k$ ... (2.23)

The eta matrix F_{k+1} is a $m \times m$ identity matrix whose h^{th} column has been replace by a vector $\hat{z}$ of m elements, where $\hat{z} = \begin{cases} J_{k+1}d' & if\ a'' \neq f'' \\ J_{k+1}d' + \hat{e}_h & if\ a'' = f'' \end{cases}$ and $\hat{e}_h$ is the h^{th} column of the $m \times m$ identity matrix [Ch83].

Case 3. No column of g of basis B satisfies the condition $g'' = f''$. In this case simply replace the leaving column by the entering column. In this case

$R_{k+1} - S_{k+1}T_{k+1} = R_k - S_kT_k$ ... (2.24)

After k iteration $R_k - S_kT_k$ may be represent as

$(R_k - S_kT_k) = (R_0 - S_0T_0)J_1F_1J_2F_2.....J_kF_k$

possibly with missing some of the J_1 and F_1 matrices.

To summarize, at each iteration of the revised simplex method, we use formulae (2.22) – (2.24) to update the basis, depending on the situation, and keep the new basis in the required form $B = \begin{bmatrix} R & S \\ T & I \end{bmatrix}$.

2.8 Eta-factorization

The efficiency of the revised simplex method described in section 2.2 depends on the implementing of step 2-4, where we need to invert the matrix called basis, to calculate the vectors $\hat{w}$ and $\hat{d}_k$ where $\hat{w} = \hat{c}_B B^{-1}$ and $\hat{d}_k = B^{-1}\hat{a}_k$ to carry out the ratio test. Actually we need to solve two systems $yB_k = c_B$ and $B_{k-1}d = a$. Without solving the system from scratch, we can use some device to facilitate their solutions and can update the matrices/ vectors at the end of iteration.

Let B_{k-1} be the basis matrix at step *K-1* and B_k be the basis matrix at step *k*, so that each B_k differs from the preceding B_{k-1} in only one column (figure 2.17).

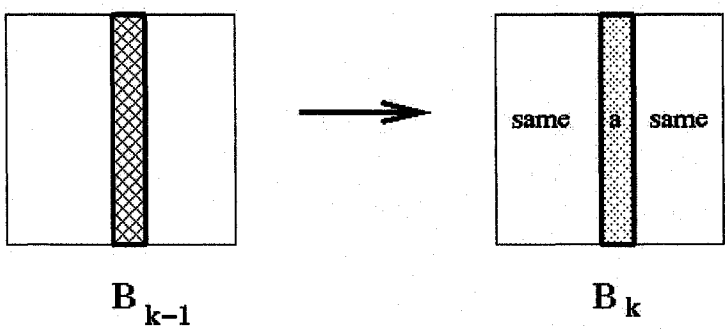


Figure 2.17: Basis before and after iteration

Where a is the column calculated from B_{k-1} d = a

Let E_{k-1} be the Eta matrix (an identity matrix with the column corresponding to the leaving variable replaced by d) then $B_k = B_{k-1}E_{k-1}$ (Fig 2.18).

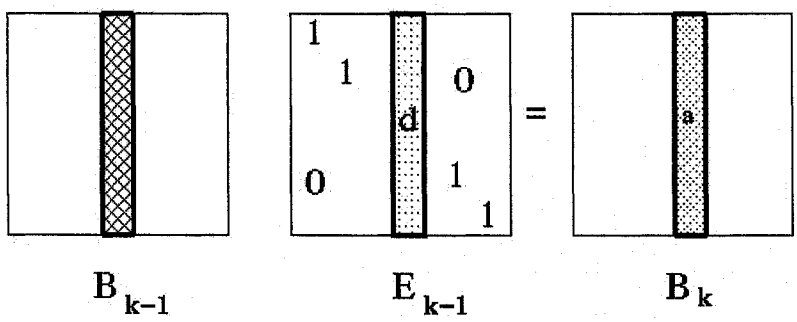


Figure 2.18: Basis before and after iteration with eta matrix

When the initial basis consists of slack variables then it is an identity matrix in that case

$B_0 = I$

$B_1 = E_1$

$B_2 = E_1 * E_2$

:

:

$B_k = E_1 * E_2 * ..... * E_k$

This is called *eta-factorization* of B_k.

This gives a convenient way of solving two systems of equations: the system $yB_k = c_B$ can be viewed as,

$(((yE_1)E_2).....)E_k = c_B$

and the system $B_k d = a$ can be viewed as

$E_1(E_2(.....(E_k d))) = a$

So by solving the equation iteratively we can get the value of y and d. As E is an eta matrix, these systems of equations are easy to solve.

So far we have consider the basis as identity matrix but when the basis is not a identity matrix then we can write

$B_k = B_0 * E_1 * E_2 * ..... * E_k$

Where B_0 is the initial basis.

We can solve the two system $yB_k = c_B$ and $B_k d = a$ as $(((yU_m)U_{m-1}).....)E_k = c_B$ and $B_0 E_1(E_2(.....(E_k d))) = a$ respectively. Now we can do a triangular factorization of the initial basis B_0 (using lower triangular matrix L and permutation matrix P) before the first iteration to use them with the growing sequence of $E_1, E_2, ... E_k$ to solve the systems

$L_m P_m .....L_1 P_1 B_0 = U$ where, $U = U_m U_{m-1} .....U_m$

U_i is the eta matrix obtained by replacing j^{th} column of I by j^{th} column of U. So the equation to solve d becomes

$U_m(U_{m-1}(.....(E_k d))) = (L_m P_m(.....(L_1 P_1 a)))$

In this way the system $yB_k = c_B$ can be solved by first solving $(((yU_m)U_{m-1}).....)E_k = c_B$ and then calculating y from $((yL_m P_m).....L_1 P_1)$. Similarly the system $B_k d = a$ may be solved by first calculating $(L_m P_m(.....(L_1 P_1 a)))$ and then solving $U_m(U_{m-1}(.....(E_k d))) = a$ where a is replaced by $(L_m P_m(.....(L_1 P_1 a)))$.

As the number of eta matrix grows with the iteration, solving the systems $yB_k = c_B$ and $B_k d = a$ become more laborious and may take longer time than to solve the systems from scratch. To avoid this we may compute a fresh triangular factorization of the basis treating B_k as B_0 and start with a new sequence of $E_1, E_2, ... E_k$. This is known as refactorizations of the basis.

Chapter 3: Efficient Routing In Wavelength-Routed WDM Networks

3.1 The Routing Problem

In designing a WDM network an important task is to find a logical topology, given the physical topology and expected traffic patterns. The next problem is to route that traffic over the logical topology in an "optimal" fashion. One widely used method of optimization is to minimize congestion in the network [Rs02]. In this chapter we will discuss our proposed techniques and algorithms for minimizing congestion in medium to large sized wavelength-routed WDM networks.

The routing problem in WDM networks can be viewed as a multi-commodity flow problem. The multi-commodity flow problems considered in the Operation Research community typically consider a small number of commodities [Me95]. One important aspect of viewing the routing problem in WDM networks as multi-commodity network flow problem is that the number of commodities becomes quite large. If we consider a network with N nodes and m edges, the number of commodities is $O(N^2)$, since each source destination pair of the network with some traffic is considered as a commodity. If we use standard node-arc formulation for minimum cost multi-commodity network flow, the number of constraints is $O(N^3)$, i.e. the number of constraints increases rapidly as the number of nodes in the network increases [BJ90]. Table 3.1 shows the growth rate of arcs, commodities and constraints with node number.

Nodes(N)	**Arcs(m)**	**Commodities(q)**	**Constraints ($m+Nq$)**
10	30	90	930
20	60	380	7660
40	120	1560	62520
100	300	9900	990300

Table 3.1: Constraints size increase rapidly with nodes

This work focuses on the techniques for solving the routing problem efficiently, considering the large number of commodities in WDM networks. In this chapter we have described our approach for fast routing in WDM networks. An initial version of this approach was presented

in [Sr04]. We have further modified this technique to incorporate eta-factorization, which leads to significant improvement in speed.

The node-arc formulation is not a good way to handle large networks, as the number of constraints grows rapidly. Using an arc-chain formulation to represent the routing problem can significantly reduce the number of constraints. But, a straight forward arc-chain formulation is not possible since the number of possible chains for each commodity is very high, even for a medium size network. To address this drawback, we follow Tomlin's approach [To66] of implicitly keeping track of the constraints and generating a chain only when it is established that the chain will enter into the basis. Using this approach, the basis size becomes $O(N^2)$ for a N nodes networks. Inverting a basis of this size, at each iteration, is still very expensive. However, since our problem satisfies the structure required for GUB method, we can adopt this technique to perform the operation more efficiently. Using the GUB technique, instead of inverting the basis of size $(m+q)\times(m+q)$ (where m is the number of arcs in the network and q is the number of commodities), it is sufficient to invert a matrix of size $m \times m$. This leads to an improvement in the time required to inverse a matrix, since m is $O(N)$ and q is $O(N^2)$.

But experiments [Sr04] show that after applying GUB, inversion of the basis is still the most costly operation, in each iteration. So, it is important to try to eliminate the matrix inversion operation, as far as possible.

At each iteration of the revised simplex method, we need to solve two systems of equations. This requires inversion of the basis matrix. In this chapter we discuss the steps for finding an optimal routing over the logical topology, to handle the specified traffic requirements. We also show how eta-factorization can be incorporated to eliminate matrix inversions in each iteration, and significantly reduce the time required to find a solution.

3.2 Overview of Routing Strategy

We are given a logical topology and the corresponding traffic matrix. Our problem is to route the traffic over the given topology, in an optimal way. We will use arc chain formulation to represent our problem.

As discussed in chapter 2, let, $A^k = (a_{ij}^k)$ denote the $(m x N_k)$ arc-chain incidence sub-matrix for commodity k. We use the following notation in our LP formulation.

- $a_{ij}^k = 1$ if the j^{th} chain for the k^{th} commodity uses the i^{th} arc; otherwise it is 0.
- x_j^k denotes the flow for the k^{th} commodity on the j^{th} chain (C_j^k),
- $x_k = (x_1^k, \ldots, x_{N_k}^k)$
- r_k is the required flow for commodity k, obtained from the traffic matrix.

Now, the LP formulation for the routing problem can expressed as follows:

Minimize λ_{max}

Subject to

$$A^1 x_1 + A^2 x_2 + \cdots + A^q x_q \le \lambda_{max} \quad (3.1)$$

$$\left.\begin{array}{rl} e_1 x_1 & = r_1 \\ e_2 x_2 & = r_2 \\ & \vdots \\ e_q x_q & = r_q \end{array}\right\} \quad (3.2)$$

$x_k \ge 0$ *(k = 1 ...q)*

The objective is to minimize the congestion λ_{max}. Constraint (3.1) states that the total flow, for all commodities, on an arc cannot exceed the maximum load λ_{max}. In other words, the sum of all flows, x_j^k through chains C_j^k that uses arc i (i.e., a_{ij}^k = 1,) for all commodities k cannot exceed λ_{max}. Equation (3.2) ensures that the total flow for a commodity k in network meets the demand r_k for that commodity, i.e.,

$$\sum_{j=1}^{N_k} x_j^k = r_k, \forall k, \; 1 \le k \le q \quad \ldots \quad (3.3)$$

In equation (3.2) e_k represents a row vector with N_k 1's, and x_k is a vector of flow variables for the different chain of commodity k where $1 \le k \le q$.

These equations lead to $m + q$ constraints, where m rows coming from equation (3.1) and q rows from equation (3.2). The size of the basis for revised simplex becomes $(m+q)\times(m+q)$. Adding slack variables ($\hat{x}_s$) to remove inequality constraints, the LP for minimizing congestion can be stated as follows.

Minimize λ_{max}

Subject to

$$A^1 x_1 + A^2 x_2 + \cdots + A^q x_q + \hat{x}_s = \lambda_{max} \quad \ldots (3.4)$$

$$\left.\begin{aligned} e_1 x_1 &= r_1 \\ e_2 x_2 &= r_2 \\ &\vdots \\ e_q x_q &= r_q \end{aligned}\right\} \quad \ldots (3.5)$$

$$x_k \geq 0, k = 1, \ldots, q; \hat{x}_s \geq 0$$

Below is the standard revised simplex algorithm to process this LP. For our routing problem, we will be working with the basis B, basis variable vector x_B and the right-hand side b. Now our constraint can be expressed as $Bx_B = b$ [Sr04].

Step 1: Find an initial basic feasible solution ($Bx_B = b$).

Step 2: Do step 3-7 until no entering column is found

Step 3: Find, (if possible), an entering column that improves the objective function.

Step 4: If no entering column is found, stop.

Step 5: Find the leaving column.

Step 6: Replace the leaving column by the entering column and update the basis B and right hand side b.

Step 7: Compute the value for the objective function and go to step 2.

3.3 Finding Initial Feasible Solution

In order to start the revised simplex process, we need to have an initial feasible solution, i.e. a solution that satisfies all the constraints. This solution is specified by a set of chains used by each commodity, and the corresponding flows on these chains such that all constraints are satisfied. In an LP, when we have a constraint of the form $\sum_{1 \leq j \leq n} a_{ij} x_j \leq b_i$, we can introduce a slack variable to change the constraint in equality form $\sum_{1 \leq j \leq n} a_{ij} x_j + x_s^i = b_i$. If all constraints are in this form, the initial feasible solution is simply an identity matrix with basis variables $x_s^1, x_s^2, \ldots, x_s^{m_c}$. In our case, the constraints described in (3.1) are in the form $\sum_{1 \leq j \leq n} a_{ij} x_j \leq b_i$, but the constraints specified in (3.2) have the form $\sum_j a_{ij} x_j = b_i$ which cannot be handled in this way. We have used

the following algorithm to create a basis B of size $(m+q)\times(m+q)$, compute the corresponding flows x_B and obtain the initial feasible solution. This approach was first proposed in [Sr04].

Step 1: for each arc $i \rightarrow j$, assign flow $v_{ij} = 0$.

Step 2: Repeat steps 3 to 8 for all k, $1 \leq k \leq q$.

Step 3: Let ***s(d)*** = source(destination) for commodity k.

Step 4: Using Dijkstra's shortest path algorithm, after assigning unit length to each arc, find the shortest path $\boldsymbol{P_{sd}}$ from ***s*** to ***d.***

Step 5: for all arcs $i \rightarrow j$ in the path $\boldsymbol{P_{sd}}$, repeat step 6.

Step 6: $v_{ij} = v_{ij} + r_k$

Step 7: Create the chain C_1^k corresponding to the path $\boldsymbol{P_{sd}}$.

Step 8: Create the k^{th} column of basis B using C_1^k to define the first m elements. The remaining q elements will be all 0, except for the k^{th} element which will be 1. x_1^k denotes the basis variable corresponding to the k^{th} column since it denotes the flow in C_1^k. The value of x_1^k will be r_k.

Step 9: Find λ_{max} = maximum value of $\{v_{ij} : \forall ij, (i \rightarrow j) \in E_L\}$. Let e be the arc number of the arc carrying λ_{max}. If more than one arc has the same flow λ_{max}, arbitrarily pick any one of these arcs and use the arc number of the selected arc for e.

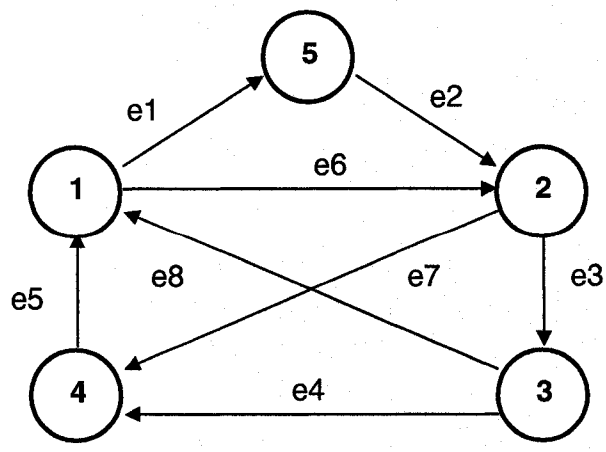
Step 10: Create column (m+1) of basis B containing -1 in the first m positions and 0 in all remaining q positions. The corresponding basis variable will be λ_{max} having the value computed in step 9.

Step 11: Repeat steps 12 for all i, $1 \leq i \leq m$, $i \neq e$.

Step 12: Create column (m+1+i) of basis B containing a 0 in all positions except in position e. The basis variable corresponding to this will be x_s^i having a value $\lambda_{max} - r_i$.

3.3.1 An Example

In this section, we illustrate the above approach through a simple example. Figure 3.1(a) shows a 5 node network with arcs numbered as e_1, e_2, e_3, …, e_8 and figure 3.2(b) shows the corresponding traffic matrix.



3.1(a)

$$\textit{Traffic matrix} = \begin{bmatrix} 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

3.1(b)

Figure 3.1: (a) A Logical Topology (b) A corresponding traffic matrix

In the above traffic matrix there are four non-zero entries, which means there are four commodities in the networks. These are

K^1 representing 5 units of traffic from node 1 to node 3

K^2 representing 4 units of traffic from node 2 to node 4

K^3 representing 2 units of traffic from node 3 to node 5 and

K^4 representing 3 units of traffic from node 5 to node 4

According to the step-1 of the algorithm, flow $v_{ij} = 0$ is initialized, for all node pair (i, j) where there is an arc $i \rightarrow j$. We have 8 arcs in this example.

We have to repeat steps 3-8 for all the commodities K^1, K^2, K^3, *and* K^4. In step 4 we find the shortest path $\boldsymbol{P_{13}}$ (1→2→3) from node 1 to node 3 for commodity K^1, using Dijkstra's algorithm. For each arc in the shortest path $\boldsymbol{P_{13}}$, we update the flow on the arc (step 6).

Thus, $v_{12} = 5$, and $v_{23} = 5$. Similarly, we repeat steps 3-8 for the remaining commodities K^2, K^3, and K^4. Thus for commodity K^2, $v_{24} = 4$; for K^3, $v_{31} = 2$ and $v_{15} = 2$; for K^4, $v_{52} = 3$ and $v_{24} = 3$. The commodities K^2 *and* K^4 both use arc 2→4 (arc no 7) and the total flow on this arc is (4 + 3 = 7 units). This is the maximum flow among all arcs. So, $\lambda_{max} = 7$.

After finishing the above steps, the basis is created as follows

		Commodity					Slack variables									
		K^1	K^2	K^3	K^4	λ_{max}	x_s^1	x_s^2	x_s^3	x_s^4	x_s^5	x_s^6	x_s^8			
Arcs	1	0	0	1	0	-1	1	0	0	0	0	0	0	x_1^1		0
	2	0	0	0	1	-1	0	1	0	0	0	0	0	x_1^2		0
	3	1	0	0	0	-1	0	0	1	0	0	0	0	x_1^3		0
	4	0	0	0	0	-1	0	0	0	1	0	0	0	x_1^4		0
	5	0	0	0	0	-1	0	0	0	0	1	0	0	λ_{max}		0
	6	1	0	0	0	-1	0	0	0	0	0	1	0	x_s^1	=	0
	7	0	1	0	1	-1	0	0	0	0	0	0	0	x_s^2		0
	8	0	0	1	0	-1	0	0	0	0	0	0	1	x_s^3		0
Commodity	9	1	0	0	0	0	0	0	0	0	0	0	0	x_s^4		5
	10	0	1	0	0	0	0	0	0	0	0	0	0	x_s^5		4
	11	0	0	1	0	0	0	0	0	0	0	0	0	x_s^6		2
	12	0	0	0	1	0	0	0	0	0	0	0	0	x_s^8		3

Figure 3.2: Basis from Initial feasible solution

$$X_B = \begin{bmatrix} 5 \\ 4 \\ 2 \\ 3 \\ 7 \\ 2 \\ 3 \\ 5 \\ 7 \\ 2 \\ 0 \\ 5 \end{bmatrix}$$

This x_B *value* (solution from fig 3.2) satisfies the constraint of the basis vector.

3.4 Finding an entering column

In the revised simplex method, we need to find the simplex multipliers for each non-basic variable x_j, and compute the value of $z_j - c_j = \hat{w}\hat{a}_j - c_j$, where $\hat{w}$ is the vector of $(m + q)$ simplex multipliers, $\hat{a}_j$ is the j^{th} column from the constraints matrix and c_j is the cost coefficient corresponding to basic variable x_j. If the value of $z_j - c_j = \hat{w}\hat{a}_j - c_j$ is positive, then we know that inclusion of x_j in the basis can improve the objective value. So x_j is a potential candidate to enter the basis. In our case, we do not explicitly generate the set of non-basic variables, as the numbers of possible chains are very large. Instead, we will follow the approach given in [To66], and create a chain when we are sure that the chain may be a part of an entering column.

The first step for finding an entering column is to calculate the simplex multiplier vector $\hat{w}$. Let $\pi_1 \ldots\ldots \pi_m$ be the first m simplex multipliers (corresponding to the m arcs of the network) and $\alpha_1 \ldots\ldots \alpha_q$ be the remaining q simplex multipliers (corresponding to the q commodities) in vector $\hat{w}$. Theorem 1 states the rules for finding an entering column. The complete proof for Theorem 1, is given in [Sr04]. In this thesis, we simply use the theorem to find a suitable entering column.

Theorem 1:

a) If $\pi_i > 0$, for any i, $1 \le i \le m$, slack variable , x_s^i is a candidate to enter the basis.

b) If the sum of the first m simplex multipliers is less than -1, $\lambda_{\max}$ is a candidate to enter the basis.

c) If, for chain j of commodity k, $\sum_{i=1}^{m}(-\pi_i)a_{ij}^k < \alpha_k$, then the variable x_j^k corresponding to this chain C_j^k is a potential candidate to enter the basis.

Theorem 1 states that, if any π_i is positive, the corresponding slack variable can be entered into the basis. If all the π_i values are non-negative, then $(-\pi_i)$ is assigned as the length of arc i, and $\sum_{i=1}^{m}(-\pi_i)a_{ij}^k$ is the length of chain C_j^k. If this length is less than α_k, then we can use this chain to create the entering column. The steps of the procedure are follows:

Step 1: initialize i to 1.

Step 2: if $\pi_i > 0$, create an entering column consisting of all 0's except in position i and stop.

Step 3: if $(i < m)$, $i = i + 1$ and go back to step 2.

Step 4: Considering all commodities, find, if possible, chain j of some commodity k, such that $\sum_{i=1}^{m}(-\pi_i)a_{ij}^k < \alpha_k$.

Step 5: If no chain satisfying $\sum_{i=1}^{m}(-\pi_i)a_{ij}^k < \alpha_k$ is found in step 4, then no entering column exists, and the current solution is an optimal one. Otherwise, create an entering column with the a_{ij}^k as the i^{th} entry of the column, for all i, $1 \leq i \leq m$. In the remaining positions, only the element in position $(m + k)$ will be 1. The other elements will be 0.

To calculate the value of $\sum_{i=1}^{m}(-\pi_i)a_{ij}^k < \alpha_k$ we have used Dijkstra's method [Di59] to find the shortest path P_{sd} from a source destination node pair ***s*** and ***d.***

An example illustrating how we use the above theorem is given below.

For the logical topology shown figure 3.3, let there be two commodities K^1 and K^2, where K^1 has source node 1 and destination node 4, K^2 has source node 1 and destination node 5. Let the simplex multipliers $\hat{w}$ at some stage be [-0.1, -0.2, -0.2, -0.2, -0.5, 0.5, 0.1].

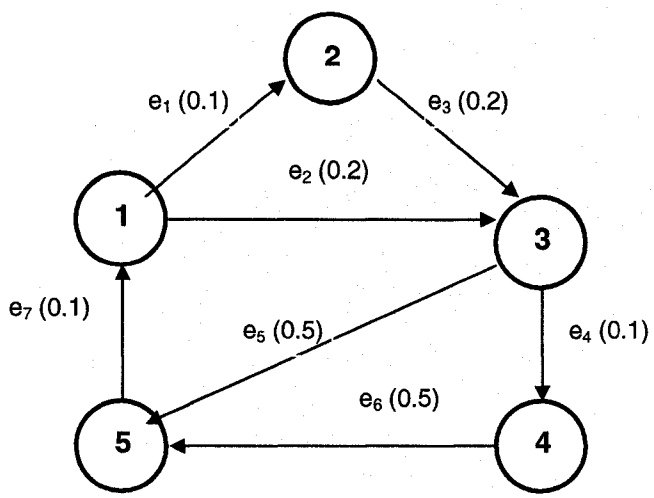


Figure: 3.3: A four node logical topology

In Figure 3.4, we have labelled the edges with the appropriate simplex multipliers. Now the shortest path for commodity K^1 is 1→3→4, and the length of the shortest path $P_{14} = 0.3$. Similarly, shortest path for commodity K^2 is 1→3→5, and the length of $P_{15} = 0.5$.

Let the values of the simplex multipliers α_1 (for commodity K^1) be 0.5 and the value of α_2 (for commodity K^2) be 0.1. Therefore, the condition $\sum_{i=1}^{m}(-\pi_i)a_{ij}^k < \alpha_k$ to enter the basis is satisfied for K^1 but not for K^2. The next step is to create the entering column for K^1 as follows:
The path is P_{14} = 1→3→4, where the arc e_2 is 1→3 and the arc e_4 is 3→4. So, the first m(= 7) entries in the entering column for this chain will be [0, 1, 0, 1, 0, 0, 0]. Since the column is for the commodity K^1, there will be a 1 at position 1 (corresponding to commodity K^1) in the entering column and a 0 at position 2 (corresponding to commodity K^2). Thus the entering column will be [0, 1, 0, 1, 0, 0, 0, 1, 0].

3.5 Finding the leaving column

We use the standard revised simplex method to find the leaving column. The algorithm is briefly described below:
Given an entering column $\hat{a}_j$,

$B^{-1}\hat{a}_j(i)$ denotes the i^{th} element of $B^{-1}\hat{a}_j$

Step 1) If $\hat{a}_j$ is the column to enter to basis, calculate $B^{-1}\hat{a}_k$ using the current basis B.

Step 2) Set *MinimumRatio* = 9999.00.

Step 3) Repeat step 4 for all i, for all i, $1 \le i \le (m+q)$ such that $B^{-1}\hat{a}_k(i) > 0$.

Step 4) If $MinimumRatio > b(i) / B^{-1}\hat{a}_k(i)$, set $MinimumRatio = b(i) / B^{-1}\hat{a}_k(i)$ and $leavingcolumn = i$.

Let $b(i)$($b_{new}(i)$) to denote the i^{th} element of $b(b_{new})$. The steps are used to update the basis:

Step 1) Replace the leaving column with the entering column

Step2) Update the right hand side b to b_{new} using the following formula:

$$b_{new}(i) = b(i) - \hat{d}_k(i) \times MinimumRatio \text{ for all } i,\ 1 \le i \le (m+q),\ i \ne leavingcolumn,$$

$$b_{new}(leavingcolumn) = MinimumRatio.$$

3.6 GUB and Representation of matrix S and T

A basis B of size $(m+q)\times(m+q)$ satisfies the GUB structure when

- q is relatively large compared to m and
- each column of the last q rows has at most one nonzero entry, the nonzero entry being equal to 1.

By exploiting this GUB structure we can calculate the value of $\hat{w}$ ($\hat{w}=\hat{c}_B B^{-1}$) and $\hat{d}_k$ ($\hat{d}_k = B^{-1}\hat{a}_k$) without directly computing B^{-1}. These values are required to find the entering column and the leaving column.

In our problem formulation, equation $A^1x_1 + A^2x_2 + \cdots + A^qx_q + \hat{x}_s = \lambda_{\max}$ (3.4) gives m constraints and equation $e_i x_i = r_i$, $1 < i <= q$ (3.5) gives q constraints where m = O(N), and q = O(N^2). Each column of the last q rows (equation 3.5) has exactly one nonzero element having 1 in position k. So, our basis satisfies the GUB structure.

From our experiments, we have found that after the decomposition of the basis in the form $B = \begin{bmatrix} R & S \\ T & I \end{bmatrix}$, the matrices S and T have the following property

- In each column of the matrix S there is a very small number of 1's (<<5%)
- Satisfying GUB structure, matrix T has at most one 1 in each column

So, it is possible to store and represent these two matrixes in a compressed form, which only stores information about the position of the 1's. Thus, matrix T can be represented as a vector of integers, where the size of the vector is the number of edges (m) in the logical topology. Matrix S can be represented as an integer matrix of size ($f \times q$). Here, q is the number of commodities and f is the maximum number of 1's in any column of matrix S. In the example given below, S is the initial (uncompressed) matrix of size 6×9.

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

Here $f = 2$, and $q = 9$. So, the information in matrix S can be represented by the compressed matrix S_{new} of size 2×9, as shown below.

$$S_{new} = \begin{bmatrix} 3 & 2 & 1 & 2 & 0 & 3 & 0 & 0 & 3 \\ 4 & -1 & 4 & 4 & 2 & 4 & 1 & 4 & -1 \end{bmatrix}$$

Here S_{new}[i][j]gives the position of the $(i+1)^{th}$ 1 in the j^{th} column of S. If there are n_j 1's in the j^{th} column of S, where $n_j < f$, then $S_{new}[i][j] = -1$, for $i \geq n_j$. For example, let us consider the j^{th} column ($j = 0$) in S_{new}. We get $S_{new}[0][0]=3$ and $S_{new}[1][0] = 4$. This means that there are two 1's in column 0 of matrix S, and they are in rows 3 and 4, i.e., $S[3][0]=1$ and $S[4][0]=1$. If we consider column j ($j=1$) in S_{new}, we get $S_{new}[0][1]=2$ and $S_{new}[1][1] = -1$. This means that $S[2][1] = 1$ and all remaining values in the column are 0.

In the case of matrix T, we know that it contains at most one 1 in each column. Therefore the compressed representation can be expressed simply as a vector (T_{vector}), where the value of the i^{th} element specifies the position (row number) of the 1 in the i^{th} column of T. If there are no 1's in the i^{th} column of T, a value of –1 is entered in the corresponding position of T_{vector}.

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \Rightarrow \; T_{vector} = [5,\ 0,\ 2, -1,\ 5,\ 7]$$

For example, $T_{vector}[0] = 5$ means that there is a row 5 of column 1 in matrix T *i.e.,* T[0][5] = 1.

The value of f becomes very small (around 1%) in case of large networks (more than 40 nodes). Therefore these compressed representations of the S and T matrices save significant amount of time in performing the matrix-vector multiplication (equation 2.21) and vector-matrix multiplication (equation 2.14 & 2.16) required to calculate the simplex multipliers and value of d. Table 3.2 shows the savings achieved by this technique for some typical networks.

Number of nodes	Number of commodities	Number of edges	Max number of 1's in a column		Size of matrix S and T			
			Matrix S	Matrix T	Old S	New S	Old T	New T
14	175	50	7	1	50×175	7×175	50×175	1×50
20	370	130	8	1	130×370	8×370	370×130	1×130
25	580	230	7	1	230×580	7×580	580×230	1×230
30	825	270	7	1	270×825	7×825	825×825	1×270
40	1490	560	8	1	560×1490	8×1490	1490×560	1×560
50	2300	730	8	1	730×2300	8×2300	2300×730	1×730

Table 3.2: Savings in Matrix S and T with new representation

3.8 Generalized Upper-Bounding with Eta Factorization

We have discussed the eta factorization technique in chapter 2. Eta factorization provides an efficient way to eliminate matrix inversion in the revised simplex algorithm. The use of the Generalized Upper-Bounding technique reduces the time required to invert the basis. But even after using GUB, matrix inversion remains the most expensive operation of this technique. For a matrix $B = \begin{bmatrix} R & S \\ T & I \end{bmatrix}$, we need to invert the matrix R, in each iteration, to get the value of simplex multiplied $\hat{w}$ ($\hat{w} = \hat{c}_B B^{-1}$) and $\hat{d}_k$ ($\hat{d}_k = B^{-1}\hat{a}_k$). As the number of nodes (N) increases, the size of matrix R also increases and inverting the matrix R takes a substantial amount of time. Table 3.3 shows how the average size of matrix R increases with number of nodes. The size of the matrix R is the number of edges (m) in the logical topology.

Number of nodes	Size(m) of matrix R	Size of basis (m + q)
10	36	124
20	135	499
30	262	1099
40	557	2060
50	732	3033

Table 3.3: Sample (average) Size of matrix R with node number

To make the revised simplex algorithm faster, we integrate eta factorization along with GUB technique for our routing problem. This allows us to eliminate the costly matrix operations. The technique has been described in [Ch83] which we are giving below for completeness.

Since our basis is in the form $B = \begin{bmatrix} R & S \\ T & I \end{bmatrix}$, we need to solve the equation

$\hat{w}'(R_k - S_k T_k) = \hat{c}'_B - \hat{c}''_B T_k$ (2.15) to find the simplex multiplier $\hat{w}'$ and equation

$(R_k - S_k T_k)\hat{d}'_k = \hat{a}'_k - S_k \hat{a}''$ (2.19) to get the value of $\hat{d}'_k$ at each iteration of the revised simplex algorithm. Let B_k be the basis after k iterations. Since B_k and B_{k+1} differ in only one column, it is possible to express the new basis (B_k) in terms of B_{k-1} as follows: $B_k = B_{k-1} * J_k F_k$, where J_k, F_k are eta-matrices as described in section 2.7.

So, if the initial basis is B_0, then

$B_1 = B_0 * J_1 F_1$,

$B_2 = B_1 * J_2 F_2 = B_0 * J_1 F * J_2 F_2$, and

$B_k = B_{k-1} J_k F_k = B_0 * J_1 F * J_2 F_2 * \ldots\ldots * J_k F_k$

In our case the initial basis for eta-factorization is $B_0 = R_0 - S_0 T_0$ (applying GUB technique).

After k iteration the basis is

$B_k = (R_0 - S_0 T_0) J_1 F_1 J_2 F_2 \ldots\ldots\ldots J_k F_k \ldots\ldots$ (3.6)

We get the value of $J_k F_k$ during update operation of matrices [Ch83].

Now a triangular factorization of $R_0 - S_0 T_0$ gives the following equation

$L_m P_m \ldots\ldots L_1 P_1 (R_0 - S_0 T_0) = U = U_m U_{m-1} \ldots\ldots\ldots U_1 \ldots\ldots$ (3.7)

where, U is the upper triangular matrix and U_i is the eta matrix obtained by replacing j^{th} column of identity matrix by j^{th} column of U

After k iterations, the equation (3.7) becomes (using equation 3.6)

$L_m P_m \ldots\ldots L_1 P_1 (R_k - S_k T_k) = U_m U_{m-1} \ldots\ldots\ldots U_1 J_1 F_1 J_2 F_2 \ldots\ldots\ldots J_k F_k$ (3.8)

From the equation (2.15) and (3.8) we can write

$Z U_m U_{m-1} \ldots\ldots\ldots U_1 J_1 F_1 J_2 F_2 \ldots\ldots\ldots J_k F_k = \hat{c}'_B - \hat{c}''_B T_k \ldots\ldots\ldots$ (3.9)

where $\hat{w}' = Z L_m P_m \ldots\ldots L_1 P_1$

3.8.1 Calculating simplex multipliers $\hat{w}$

Now solving the equation (3.9) for Z and then computing $ZL_mP_m.....L_1P_1$ will give the value of $\hat{w}'$.

The algorithm for calculating $\hat{w}'$ is as follows:

Step 1: calculate the value of $\hat{c}'_B - \hat{c}''_B T_k$,

Step 2: set $i = k$ and $Z = \hat{c}'_B - \hat{c}''_B T_k$,

Step 3: if $i \geq 1$ do steps 4 and 5,

Step 4: set $v = Z$, replace y by the solution of $ZF_i = v$,

Step 5: set $v = Z$, and replace y by the solution of $ZJ_i = v$. Replace i by i -1.

Step 6: Set $j = 1$

Step 7: if $j \leq m$, then set $v = Z$, replace Z by the solution of $ZU_j = v$, replace j by j +1 and repeat this step.

Step 8: set $j = m$.

Step 9: if $j \geq 1$, then replace Z by ZL_jP_j replace j by j -1, and repeat this step.

The final value is the simplex multiplier $\hat{w}'$. We can get the value of $\hat{w}''$ using equation $\hat{w}'S_k + \hat{w}'' = \hat{c}''_B$.

3.8.2 Calculating value of *d* ($B^{-1}a$)

To calculate the value of $\hat{d}_k$ we have the equation (2.19)

$(R_k - S_kT_k)\hat{d}'_k = \hat{a}'_k - S_k\hat{a}''$ this can be written as

$L_mP_m.....L_1P(R_k - S_kT_k)\hat{d}'_k = L_mP_m.....L_1P(\hat{a}'_k - S_k\hat{a}'')$

from equation (3.8) we can write

$U_mU_{m-1}.....U_1J_1F_1J_2F_2.....J_kF_k\hat{d}'_k = L_mP_m.....L_1P_1(\hat{a}'_k - S_k\hat{a}'')$ .....(3.10)

Now solving the equation (3.10) gives us the value of $\hat{d}'$.

The procedure is as follows:

Step 1: calculate the value of $(\hat{a}'_k - S_k\hat{a}'')$

Step 2: set $j = 1$ and $\hat{d}' = (\hat{a}'_k - S_k\hat{a}'')$

Step 3: if $j \leq m$, then replace $\hat{d}'$ by $L_jP_j\hat{d}'$, replace j by $j + 1$ and repeat this step.

Step 4: Set $j = m$;

Step 5: if $j \geq 1$, then set $v = \hat{d}'$, replace $\hat{d}'$ by the solution of $U_j \hat{d}' = v$, replace j by j - 1 and repeat this step.

Step: 5 set $i = 1$.

Step 3: if $i \leq k$ do step 4 and 5

Step 4: set $v = \hat{d}'$, replace y by the solution of $\hat{d}' F_i = v$

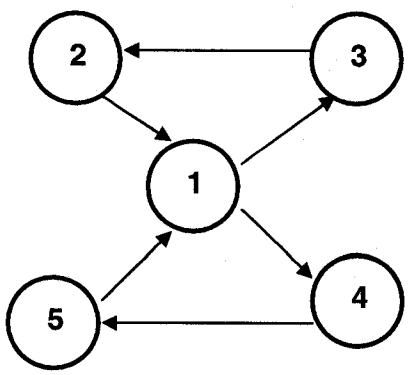
Step 5: set $v = \hat{d}'$, and replace y by the solution of $\hat{d}' J_i = v$. Replace i by i -1. go to step 3.

Thus we get the value of vector $\hat{d}'$. And using equation $\hat{d}'' = \hat{a}_k'' - T_k \hat{d}_k'$ we can get the $\hat{d}''$ and thus the vector d. As the matrices F, J, U are identity matrices, except for only one column or row, it is easy to get the solution for simplex multiplier and vector d.

As the iteration number grows, the number of J and K matrices become larger and the solution using J and K matrices may take more time than the time needed to invert the matrix R - ST. To avoid this situation, we will refactorize the matrix $(R_k - S_k T_k)$ at appropriate intervals, replace $(R_0 - S_0 T_0)$ by $(R_k - S_k T_k)$ and start the process all over again.

3.8.3 An Example

Here is an illustrated example of the algorithm for eta factorization. We consider the simple 5 node network and traffic matrix as shown in figure 3.4



Node	1	2	3	4	5
1	0	8	0	0	6
2	0	0	0	0	1
3	0	0	0	0	5
4	9	0	0	0	0
5	0	9	0	0	0

Figure 3.4: A five node network and traffic matrix

This network has 6 edges (m=6) and 6 commodities (q=6) and the basis B is of size 12 (m+q).

$$B_0 = \begin{matrix} \\ \text{m} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \text{q} \end{matrix} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

As this basis satisfies the GUB structure, we can express B_0 in the form $B_0 = \begin{bmatrix} R_0 & S_0 \\ T_0 & I \end{bmatrix}$

$$R_0 = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad S_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$T_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

From the above matrices, we can calculate the value of $R_0 - S_0 T_0$. We can do so as follows:

$$R_0 - S_0T_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Now to calculate the simplex multipliers, we need to find the upper triangular matrix (U) of $R_0 - S_0T_0$ as follows:

$$U = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{So } U_1 = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$ with the 1st column of I replaced by the 1st column of U.

$$\text{Similarly } U_m = U_6 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$ where 6th column of I is replaced by 6th column of U.

We know that U_i is an identity matrix where the i^{th} column is replaced by the i^{th} column of U matrix. During the triangulation process, we get the matrices $L_6, P_6, \ldots\ldots L_1$ *and* P_1 (equation 3.7).

$$L_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$L_6 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \qquad P_6 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Let the cost of the basis (calculated while creating initial feasible solution) be C_B = [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]^T. (We use v^T before the a vector to indicate the transpose of vector v is a column vector). The right hand side for the equation gives $\hat{c}'_B - \hat{c}''_B T_k$ = [1 0 0 0 0 0]^T.

When we calculate the simplex multipliers for the first time, we don't have matrices F and J which are created when the basis is updated. Therefore, for the first iteration, we have to solve the equation:

$ZU_6U_5.....U_1 = \hat{c}'_B - \hat{c}''_B T_k$

We will first consider $ZU_6U_5.....U_2$ as a vector V and solve the equation $VU_1 = \hat{c}'_B - \hat{c}''_B T_k$ for U_1. After obtaining the value of V, we have the equation $ZU_6U_5.....U_2$= [1 0 0 0 0 0]^T.

After solving all the U matrix iteratively the Z vector is [-1 -1 -1 -1 -1 0]^T and after calculating $ZL_mP_m.....L_1P_1$ to get the simplex multiplier, we get the vector $\hat{w}'$=[0 0 0 0 1 0]^T. Using equation (2.14) we get the value of $\hat{w}''$ and thus the simplex multiplier becomes [0 0 0 0 -1 0 1 0 0 1 0 1]^T. In this way, we get the value of simplex multipliers without doing the expensive inversion operation in a matrix.

To show how we can calculate vector $\hat{d}_k$ using eta factorization, let us have an entering column vector a = [0 0 1 0 0 0 0 0 0 0 0 0 0]^T. So, vector $(\hat{a}'_k - S_k\hat{a}'')$ = [0 0 1 0 0 0]^T. The value of the right-hand side of equation (310) becomes $L_mP_m.....L_1P_1(\hat{a}'_k - S_k\hat{a}'')$ = [0 0 1 0 0 0]^T. Now we need to solve the equations

$U_6U_5.....U_1\hat{d}'_0$ = [0 0 1 0 0 0]^T

We can solve it iteratively as before and the solution is $\hat{d}_0' = [-1 \quad -1 \quad -1 \quad 0 \quad 0 \quad 0]^T$

Now using equation (2.20) and (2.21) we can get the vector $\hat{d}_k = [1 \quad -1 \quad -1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$. Using this value we can get the leaving column number and update the matrices as described in section 3.6. For this small network, we obtain the optimal value in single step but for larger networks, we may have the value of F and J matrices and, in that case, we need to solve the equation (3.9) for simplex multipliers and equation (3.10) for the value of vector $\hat{d}_k$.

Chapter 4: Experiment and Results

In chapter three we outlined our approach and formulation for traffic routing to minimize congestion in a logical topology. In this chapter, we will present and analyze the results of our experiments to test and evaluate the proposed approach. Our algorithm augments the approach introduced in [sr04]. We compare our results with those presented in [sr04], as well as standard LP formulations, solved using CPLEX. All experiments were carried out on a SUN 1.2 GHz platform and CPLEX version 9.0 was used to solve the standard LP formulations.

4.1 Methodology

Each of our experiments requires a logical topology and its corresponding traffic matrix to develop an optimum routing strategy that gives the minimum congestion for that topology. To design a logical topology, the underlying physical topology and traffic matrix must be specified. We have used an existing C program presented in [Hou03] to generate the logical topologies. Solutions were generated using the three different approaches, based on the same data sets (i.e. logical topology and traffic matrix). A C program was used to generate the input file for CPLEX in standard lp format.

4.2 Experimental Results

We have carried out experiments on a number of medium to large size networks, ranging in size from 10 to 50 nodes. For each network size, we tested several different logical topologies and traffic matrices. The values reported in Tables 4.1 and 4.2 represent average values, based on the results of the different experiments for a given network size. The results for the individual experiments are given in Appendix A. To compare our result and examine the advantages of eta-factorization, experiments were carried out using the following three approaches:

i) Our scheme

ii) Implicit column generation, without eat-factorization [Sr04] and

iii) Standard LP formulation, using CPLEX-9.

Table 4.1 shows the average time (in seconds) to obtain an optimal solution and average number of iterations for each of the three approaches.

Size of Network	Approach					
	Our Scheme		CPLEX		Scheme without eta-factorization	
	Average Time (secs)	Average # of iterations	Average Time (secs)	Average # of iterations	Average Time (secs)	Average # of iterations
10	0.1863	174.1579	0.4321	1212.1579	1.24	148
14	0.91181818	448	2.39418182	4240	13.15	370
20	46.4717	8,081	76.5456	42374	2753.45	7567
30	1282.3770	57988	3264.3403	337127	10677.67	41055
40	36031.7913	378270	56667.9297	1505692	**	**
50	159139.416	835952	**	**	**	**

Table 4.1: Summarized data of different approach

Node number	Time ratio (ours /CPLEX)	Iteration ratio(ours /CPLEX)
10	0.43115	0.143564
14	0.380848	0.10566
20	0.607111	0.190707
30	0.392844	0.172006
40	0.635841	0.251227
50	**	**
Average	0.466831	0.158893

Table 4.2: Comparison ratio of our approach with CPLEX

** results not found

In Table 4.2, we have compared the solution time and number of iterations for our approach with that of CPLEX. We see that our method provides an improvement of 40%-60% over standard LP techniques. The average improvement is 46%. The average number of iterations required in our approach is $1/6^{th}$ of that of CPLEX. Furthermore, our approach can be used for larger networks of 50 nodes, where CPLEX fails to find a solution.

Figure 4.1 (semi log graph) compares the growth rate of the algorithm, to that of CPLEX.

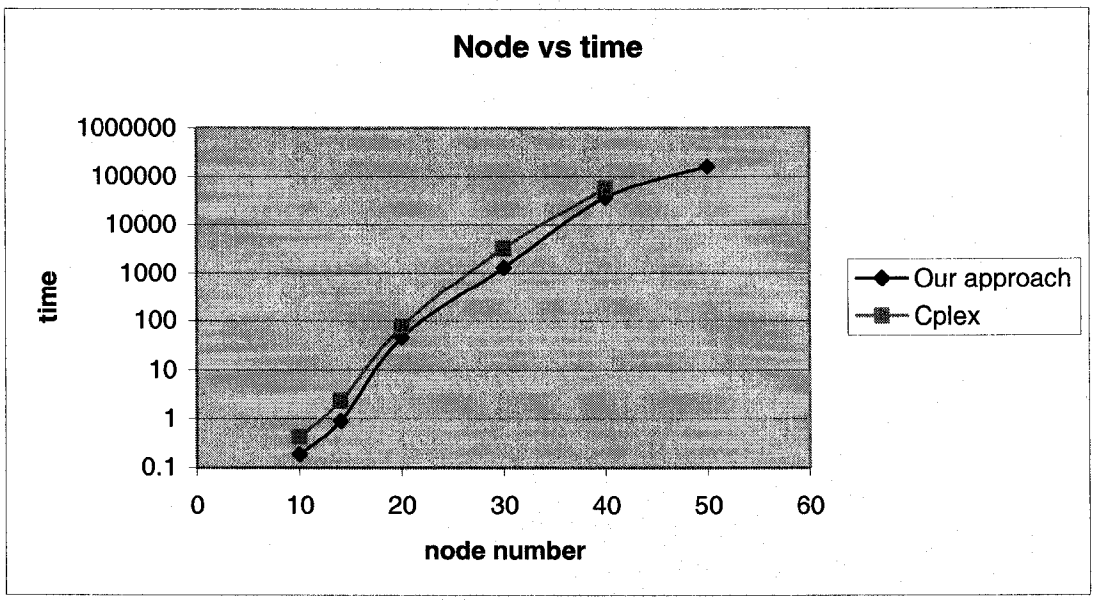


Figure 4.1: Growth rate with size of network

4.3 Analysis of the experiments

From the experimental results, we observe that our approach with eta-factorization performs significantly better than other approaches, for all network sizes. Use of eta-factorization speeds-up the algorithm significantly. Our scheme can handle networks of 50 nodes which cannot be handled using CPLEX.

To track the development of λ_{max} value, we recorded the (sub-optimal) objective values at certain intervals. This λ_{max} values for a 30 node network is shown in Figure 4.2. From Figure 4.3, we can see that initial development in revised simplex method is very fast and when it approaches to the optimal value, development becomes slower. The majority of the time is spent to obtain very little improvement (< 10%). So, if we can fix a threshold value for λ_{max} to terminate the process, we can get a "near-optimal" solution in a very short time. For example if the threshold value was set to 60 then we could reach that value within 420 second, less than 50% of the fully optimized time.

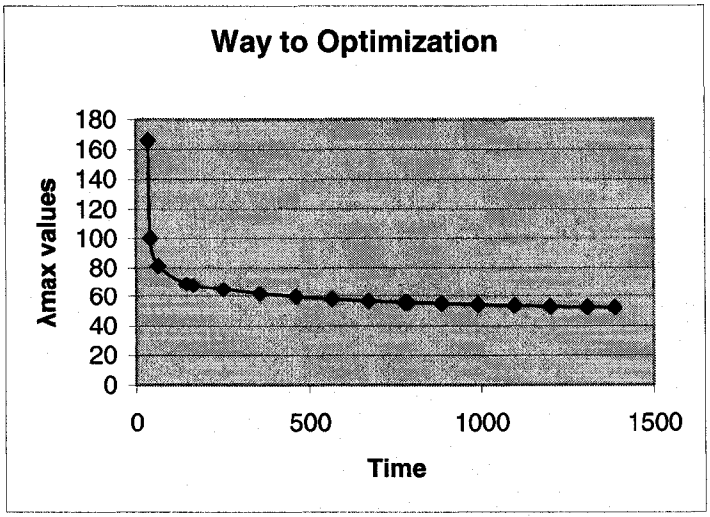


*Figure 4.2: **Improvement of** λ_{max} value with time*

4.4 Conclusion and Future work

In this thesis we have introduce eta-factorization to determine the optimum routing in a wavelength routed WDM network. Adoption of eta factorization with GUB technique produce better results in time than the solution from CPLEX-9. The same optimised value from our approach and from CPLEX ensure the correctness of the solution. During test process we found that a significant amount of the solution time is consumed by the multiplication operation of matrix-vector and vector-matrix. In arc-chain formulation the matrices have the values of 0's and 1's only and most of them are zeros. (We have exploited this feature for matrix T and S by vertical (column) compression). If we can totally exploit this feature to store the matrix and can find a easy way to calculate the value of multiplication, then it will be possible to get a better solution in time. Use of data-compression technique to store the matrix may take less time to calculate multiplication value. Another future area of research is to combine this technique with the problem of finding a logical topology for further improvement in network congestion.

.

Selected Bibliography

[AMO93] R.Ahuja, T.LMagnanti and J.BOrlin, "Network Flows: Theory Algorithms and Applications", Prentice Hall, Englewood Cliffs, NJ 1993.

[BG95] D.Bienstock and O.Gunluk, "Computational experience with a difficult mixed-integer multicommodity flow problem", Mathematical Programming vol.68, pp 213-237, 1995.

[BGBK99] S.Baroni, R.J.Gibbens, P.Bayvel and S.K.Korotky, "Analysis and design of resilient multifiber wavelength routed optical transport networks", IEEE Journal of Lightwave Technology, vol.17, pp.743-758, May 1999.

[BJ90] M.S.Bazaraa and J.J Jarvis, "Linear Programming and Network Flows", Wiely, New York, 1990.

[BM2000] D.Banerjee and B.Mukherjee "Wavelength Routed optical network: linear formulation, resource budgeting tradeoff and reconfiguration study" IEEE/ACM Transaction on Networking, vol.8, Issue: 5 pp.598-607, October. 2000.

[Ch83] V.Chvatal, "Linear Programming", W.H Freeman, November 1983.

[CHMR98] V.W.S Chan, K.L.Hall, E.Modiano and K.A.Rauschenbach "Architecture and Technologies for High-Speed Optical Data Networks" IEEE Journals of Lightwave Technology, vol.16, No.12, December 1998.

[CWH92] Y.J Chang, J.L.C Wu and H.J Ho, "Optimal virtual circuit routing in computer networks" IEEE Proceeding 1, vol.139, Issue 6, December 1992.

[Di59] E.W.Dijkstra, "A note on two problems in connexion with graphs" Numerische Mathematic, vol. 1, pp 269-271, 1959.

[DOW55] G.B Dantzig, A.Orden and P.Wolfe, "The Generalized Simplex Method for Minimizing a Linear form under Linear Inequality Constraints", Pacific Journal of Mathematics, vol.5, pp.183-195, 1955.

[DR2000] R.Dutta and G.N.Rouskas, "A survey of virtual topology design algorithms for wavelength routed optical networks", Optical Networks Magazine, pp. 73-89, January 2000.

[DS67] G.B. Dantzig and R.M. Van Slyke, "Generalized Upper Bounding Technique" Journal of computer and System Sciences, vol.1, pp 213-226, 1967.

[FF58] L.R Ford and D.R.Fulkerson, "A Suggested Computation for maximal Multi-Commodity Network Flows", Management Science, Vol.5, Issue 1, pp. 97-101, October 1958.

[FF62] Lestor R.Ford and D.R. Fulkerson, "Flows in Networks" Princeton University Press, 1962.

[FG99] A.Frangioni and G.Giorgio "A Bundle type Dual ascent Approach to Linear Multi-commodity Min Cost Flow problems", INFORMS journal of computing, vol.11, Issue 4, pp370-393, 1999.

[GBHC2000] S. Gangxiang, S. K.Bose, C. T. Hiang and L.Chao "Designing WDM Optical Network for Reliability: Routing Light Paths Efficiently for Path protection". IEEE Optical Fiber Communication Conference, 2000, VOL.3, pp. 50-52, March 2000.

[Gr2001] P.Green "Progress in Optical Networking", IEEE Communication magazine, vol.39, Issue: 1, pp. 54-61, January 2001.

[Ho2003] M.Hou "A Heuristic for WDM Path Protection", MSc. Thesis , School of Computer Scienec, University of Windsor, 2003.

[Hu63] T.C.Hu, "Multi-Commodity Network Flows", Operation Research, vol. 11, pp 344-360, 1963.

[MBLN93] M.A Marson, A.Bianco, E.Leonardi and F.Neri, "Topologies for wavelength-routing all-optical networks", IEEE/ACM Transaction, Networking, vol.1, pp 534-546, October, 1993.

[Me95] E.A Medova, "Network flow algorithms for routing in networks with wavelength division multiplexing", IEEE Proceeding- Communication, vol.142, Issue 4, August 1995.

[Mu2000] B.Mukherjee, "WDM Optical Communication Networks: Progress and Challenges" IEEE Journals on areas in communications, vol.18, Issue 10, pp. 1810-1824, October 2000.

[Mu97] B.Mukherjee, "Optical Communication Networks", McGraw-Hill, New York, 1997.

[RS2002] R. Ramaswami and K. N.Sivaranjan, "Optical Networks – A Practical Perspective" Morgan Kaufman Publishers, Optical Network Magazine, vol.3, May 2002.

[RS96] R.Ramaswami and K.N Sivarajan, "Design of logical topologies for wavelength-routed optical networks", IEEE JSAC, vol.14, pp.840-851, June 1996.

[Sr04] Srabanti Dey "A Heuristic for WDM Path Protection", MSc. Thesis , School of Computer Scienec, University of Windsor, 2004.

[Ta82] H.A.Taha, "Operations Research: An Introduction", Macmillan, New York, 1982.

[To66] J.A Tomlin "Minimum-Cost Multicommodity Network Flows", Operation Research, vol.14, Issue 1, pp. 45-51 (January-February, 1966).

[W2003] I.L. Wang, "Shortest Paths and Multi-commodity network flows" , PhD dissertation, Georgia Institute of Technology, Department of Industrial and System Engineering, April, 2003.

Appendix A

Data Table for 10-nodes networks

	Our approach			CPLEX		
Case Number	value	Time	Iteration	value	Time	Iteration
logical_top_10_11	78.7333	0.1800	170	78.7333	0.3410	978
logical_top_10_12	78.4167	0.1500	105	78.4167	0.4070	1191
logical_top_10_13	73.2272	0.1500	115	73.2273	0.4600	1356
logical_top_10_14	77.4167	0.2000	190	77.4167	0.5570	1445
logical_top_10_15	69.1510	0.2400	241	69.1509	0.4140	1147
logical_top_10_21	84.4285	0.1600	145	84.4286	0.3190	994
logical_top_10_22	79.4286	0.1300	91	79.4286	0.4150	1187
logical_top_10_23	72.5714	0.2000	189	72.5714	0.4330	1227
logical_top_10_24	77.4167	0.2000	190	77.4167	0.5580	1445
logical_top_10_25	69.1510	0.2200	241	69.1509	0.4150	1147
logical_top_10_32	79.4286	0.1300	91	79.4286	0.4190	1187
logical_top_10_33	71.3077	0.2200	203	71.3077	0.5020	1433
logical_top_10_35	69.1510	0.2200	241	69.1509	0.4150	1147
logical_top_10_45	69.1510	0.2200	241	69.1509	0.4140	1147
logical_top_10_51	84.4285	0.1600	145	84.4286	0.3180	994
logical_top_10_52	79.4286	0.1300	91	79.4286	0.4160	1187
logical_top_10_53	72.5714	0.2100	189	72.5714	0.4330	1227
logical_top_10_54	77.4167	0.2000	190	77.4167	0.5580	1445
logical_top_10_55	69.1510	0.2200	241	69.1509	0.4150	1147
Average	75.3671	0.1863	174	75.3671	0.4321	1212

Data Table for 14-nodes networks

	Our approach			CPLEX		
Case Number	value	Time	Iteration	value	Time	Iteration
logical_top_14_11	59.8889	0.9700	475	59.8889	2.6020	4581
logical_top_14_12	72.0001	0.7600	397	72.0000	1.9640	3347
logical_top_14_14	64.0555	0.8500	356	64.0556	2.5290	4600
logical_top_14_15	64.3126	0.7700	354	64.3125	2.1280	4351
logical_top_14_22	72.0001	0.7600	397	72.0000	2.0000	3347
logical_top_14_24	64.0001	0.8600	402	64.0000	3.2620	5440
logical_top_14_25	64.3125	0.7200	302	64.3125	1.4290	2556
logical_top_14_31	59.8889	0.9600	475	59.8889	2.5710	4581
logical_top_14_32	72.0001	0.7600	397	72.0000	1.9950	3347
logical_top_14_33	54.6487	1.4600	812	54.6486	3.1390	4805
logical_top_14_34	64.0555	0.8500	356	64.0556	2.5000	4600
logical_top_14_35	64.3126	0.7600	354	64.3125	2.1450	4351
logical_top_14_41	59.8889	0.9600	475	59.8889	2.5530	4581
logical_top_14_42	72.0001	0.7500	397	72.0000	1.9600	3347
logical_top_14_43	54.6487	1.4600	812	54.6486	3.0510	4805
logical_top_14_44	64.0555	0.8600	356	64.0556	2.4990	4600
logical_top_14_45	64.3126	0.7600	354	64.3125	2.1220	4351

logical_top_14_51	59.8889	0.9600	475	59.8889	2.5600	4581
logical_top_14_52	72.0001	0.7500	397	72.0000	1.9820	3347
logical_top_14_53	54.6487	1.4600	812	54.6486	3.0650	4805
logical_top_14_54	64.0555	0.8500	356	64.0556	2.5040	4600
logical_top_14_55	64.3126	0.7700	354	64.3125	2.1120	4351
Average	63.88	0.9118	448	63.88	2.3942	4240

Data Table for 20-nodes networks

	Our approach			CPLEX		
Case Number	value	Time	Iteration	value	Time	Iteration
logical_top_20_11	55.5587	54.6700	9394	55.5585	81.0890	44635
logical_top_20_12	61.7007	54.0700	9661	61.7001	80.5370	46236
logical_top_20_14	62.8714	29.9800	5863	62.8710	51.8490	33358
logical_top_20_15	56.7091	50.3200	7878	56.7090	84.8670	41931
logical_top_20_21	55.5587	54.6800	9394	55.5585	79.8420	44635
logical_top_20_23	59.3862	44.2000	7887	59.3852	86.8630	47510
logical_top_20_24	62.8714	30.0400	5863	62.8710	51.7310	33358
logical_top_20_25	56.7091	50.3400	7878	56.7090	84.7800	41931
logical_top_20_31	55.5587	55.0700	9394	55.5585	81.7620	44635
logical_top_20_32	61.7007	54.3600	9661	61.7001	81.5630	46236
logical_top_20_33	59.3862	44.2200	7887	59.3852	87.5300	47510
logical_top_20_34	62.8714	30.2000	5863	62.8710	52.9770	33358
logical_top_20_35	56.7091	50.3200	7878	56.7090	84.5680	41931
logical_top_20_41	55.5587	54.6600	9394	55.5585	83.8000	44635
logical_top_20_42	61.7007	53.9000	9661	61.7001	81.2620	46236
logical_top_20_43	59.3862	44.2200	7887	59.3852	87.2620	47510
logical_top_20_44	62.8714	29.9600	5863	62.8710	52.7610	33358
logical_top_20_45	56.7091	50.3000	7878	56.7090	84.0260	41931
logical_top_20_51	55.5587	54.6700	9394	55.5585	80.8040	44635
logical_top_20_52	61.7007	53.9200	9661	61.7001	80.0870	46236
logical_top_20_53	59.3862	44.2200	7887	59.3852	85.5070	47510
logical_top_20_54	62.8714	30.2400	5863	62.8710	52.2250	33358
logical_top_20_55	56.7091	50.2900	7878	56.7090	82.8570	41931
Average	59.1323	46.4717	8,081	59.1319	76.5456	42374

Data Table for 30-nodes networks

	Our approach			CPLEX		
Case Number	value	Time	Iteration	value	Time	Iteration
logical_top_30_11	52.1681	1385.2300	63702	52.1667	3692.1210	368030
logical_top_30_12	53.2002	1289.1499	56994	53.2000	3508.3380	362333
logical_top_30_13	49.8365	1395.9800	60621	49.8365	3059.4520	312422
logical_top_30_14	50.8270	1254.2699	55914	50.8261	3246.7290	317945
logical_top_30_15	53.2588	1140.5500	53744	53.2579	2993.4890	330895
logical_top_30_21	52.1681	1380.4900	63702	52.1667	3541.4570	368030
logical_top_30_22	53.2002	1287.7600	56994	53.2000	3183.8580	362333

logical_top_30_23	49.8365	1387.6100	60621	49.8365	3151.4360	312422
logical_top_30_24	50.8270	1252.1700	55914	50.8261	3065.1950	317945
logical_top_30_25	53.2588	1138.1200	53744	53.2579	2962.4710	330895
logical_top_30_31	52.1681	1377.1901	63702	52.1667	3580.8890	368030
logical_top_30_32	53.2002	1280.9999	56994	53.2000	3735.5280	362333
logical_top_30_33	49.8365	1382.6699	60621	49.8365	2940.2870	312422
logical_top_30_34	50.8270	1246.6000	55914	50.8261	2984.0380	317945
logical_top_30_35	53.2588	1134.9301	53744	53.2579	3087.8280	330895
logical_top_30_43	50.8047	1335.9200	60170	50.8046	3486.0260	338573
logical_top_30_44	50.8270	1246.0701	55914	50.8261	3105.4100	317945
logical_top_30_45	53.2588	1133.3600	53744	53.2579	2920.8680	330895
logical_top_30_51	52.1681	1379.9399	63702	52.1667	4234.1740	368030
logical_top_30_52	53.2002	1283.0001	56994	53.2000	3543.1150	362333
logical_top_30_53	49.8365	1392.4700	60621	49.8365	3043.0760	312422
logical_top_30_54	50.8270	1248.9800	55914	50.8261	3121.4440	317945
logical_top_30_55	53.2588	1141.2100	53744	53.2579	2892.5970	330895
Average	51.8284	1282.3770	57988	51.8277	3264.3403	337127

Data Table for 40-nodes networks

	Our approach			CPLEX		
Case Number	value	Time	Iteration	value	Time	Iteration
logical_top_40_11	53.1734	33837.0625	361258	53.1600	61331.9230	1468652
logical_top_40_12	52.2614	36789.8516	392455	52.2500	57489.1920	1492830
logical_top_40_13	51.3814	38710.7305	390599	51.3700	53611.8890	1207017
logical_top_40_14	50.4012	34040.3008	345165	50.3909	56915.1350	1461725
logical_top_40_15	51.8380	36516.6680	374574	51.8246	59445.9280	1553953
logical_top_40_21	53.2544	36317.1836	391385	53.2423	54139.6890	1514260
logical_top_40_22	52.2614	36010.7422	392455	52.2486	53741.7520	1492830
Average	52.0816	36031.7913	378270	52.0695	56667.9297	1505692

Data Table for 50-nodes networks

	Our approach			CPLEX		
Case Number	value	Time	Iteration	value	Time	Iteration
logical_top_50_11	46.0532	166088.969	842210	**	**	**
logical_top_50_12	46.3313	148754.781	785464	**	**	**
logical_top_50_13	45.6885	159690.172	823782	**	**	**
logical_top_50_14	45.8053	159522.531	846865	**	**	**
logical_top_50_15	46.5233	161640.625	881440	**	**	**
Average	46.08032	159139.416	835952	**	**	**

** results not found

Vita Auctoris

M Abul Kalam was born in Dhaka, Bangladesh. He received his first Bachelor degree in civil engineering from Bangladesh University of Engineering and technology Dhaka. He also received bachelor degree in Computer Science from University of Windsor, Canada in 2002. He joined University of Windsor in January 2003 as a graduate student under the supervision of Dr. Subir Bandyopadhayay and completed Masters in Computer Science in August 2005.