NEAR EAST UNIVERSITY

GRADUATE SCHOOL OF APPLIED AND SOCIAL SCIENCES

NETWORK TRAFFIC OPTIMIZATION

Shadi Jundi

Master Thesis

Department of Computer Engineering

Nicosia-2005

Shadi Jundi: Network Traffic Optimization

Approval of the Graduate School of Applied and Social Sciences

[illegible]

Prof. Dr. Fakhreddin Sadikoğlu
Director

We certify this thesis is satisfactory for the award of the Degree of Master of Science in Computer Engineering

Examining Committee in charge:

Prof. Dr. Perviz Alizade, Committee Chairman, Electrical and Electronic Engineering Department, NEU

[illegible]

Assis. Prof. Dr. Erdal Onurhan, Committee Member, Mechanical Engineering Department, NEU

[illegible]

Assis. Prof. Dr. Firuddin Muradov, Committee Member, Computer Engineering Department, NEU

[illegible]

Assoc. Prof. Dr. Rahib Abiyev, Supervisor, Computer Engineering Department, NEU

[illegible]

ACKNOWLEDGMENTS

First I want to thank Assoc. Prof. Dr. Rahib Abiyev for his help and valuable comments during the preparation of this thesis. Under his guidance, I successfully overcame many difficulties and learned a lot about network optimization. He provided all the help from his excellent resources as well as from his superb background on the subject. I over came this and many other difficulties in analyzing the various mathematical problems discussed in this thesis.

Thanks to my friends in Cyprus, Mohammad Ali from DAU's computer engineering, Haider Abbas and Mohammad Ashfaq from NEU's computer engineering. We had great times in graduate and in undergraduate life that I will always remember.

Thanks to the Near East University, for having good instructors and for having a great library. The support I got from instructors for my academic and non-academic life is greatly appreciated. I would like to specially thank the following Instructors: Assist. Prof. Dr. Erdal Onurhan, Assoc. Prof. Dr. Adnan Khashman, Prof. Dr. Dogan Ibrahim, and finally Assoc. Prof. Dr. Rahib Abiyev.

I want to thank my family, especially my parents, for supporting my education. Without their support and love for me, I would have never achieved my current position.

All trademarks used in this research belong to their respective owners.

ABSTRACT

Using computer networks, huge information from one side of the world to the other can be sent and shared. In such conditions, reliable network paths are needed to route network traffic. The network paths which have the lowest cost metric are granted permission to route data packets. For this procedure, some routing algorithms are used. The use of those algorithms allows a significant decrease of congestion in the network. This thesis is devoted to one of the important issues in network optimization. Which is the optimization of network traffic in TCP/IP networks. To solve this problem, an overview of various optimization techniques that are important for the planning and the design of many networks and network optimization algorithms are reviewed and applications to network traffic are analyzed. Network optimization algorithms such as network maximum flows, shortest path routing, and network analysis techniques are considered. The network traffic optimization problem based on a given network topology is formulated. The applications of Dijkstra's algorithm to find the shortest path to route network traffic with computer simulation are discussed. For this reason a developed program to generate network data is developed, and using Dijkstra's algorithm, the shortest path is found.

The limitation of conventional network routing protocols that uses Dijkstra's algorithm compared to optimization techniques on a given network is demonstrated showing the results of both approaches. The topological optimization of computer network has been considered. The minimizations of objective function that include costs of network traffic under reliability constraints have been formulated. The solution of this problem has been analyzed.

The simulation of Dijkstra's Algorithm was developed using MS-Visual Basic™ programming language, and the simulation of the network optimization model was accomplished using the optimization package WinQSB™.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Network Traffic And Its Optimization Problem

Computer Networks are a key characteristic of modem-day society. It affects the lifestyle of individuals, the structure of the society, and gives people convenience and freedom of sharing and accessing information around the world. However, it also poses a problem of frustrating congestion and delays, especially in Wide Area Networks as The Internet, with its 530 Million users and 8 Million Site daily additions and with more applications in integrating the internet with more applications such as home appliances, cellular phones, banking, video conferencing and many others [15]. Traffic demands are exponentially increasing. The Internet infrastructure, on the other hand, has been unable to some extent to expand at the same pace. These facts lead to the so-called Network Traffic Congestion Problem.

For this reason it obtains importance to design reliable network and define its optimal structure such that maximally to avoid from congestion of network traffic.

Network traffic congestion itself has directly cost millions of dollars of wasted uptime and millions of hours of delayed important information, and has also resulted in increased packet loss, which in tum increase congestion, and aggravate the problem of network congestion. This problem costs heavily and deserves serious attention. The magnitude and seriousness of traffic congestion problems have been noticed by the Internet authorities, the Internet engineering task force, as well as other different research agencies [17].

Great deal of effort is being put into practice, from packet transportation planning to packet management. Some proposed solutions include: introduction of new optimized mathematical solutions based on current protocols such as improving current routing protocols such OSPF to better handle congestion, Packet Discard Policies, Manually altering the Maximum\Minimum link utilization, Improving packet management strategies (bit control, real-time diversion planning, etc.), and more recently the development of

MPLS (Multi Protocol Label Switching), which reroute congested traffic across other less congested routes and routers. However most network transportation problems are very complicated and cost heavily in practice.

Also, some chain reactions may be associated with these problems. For example, increasing supply could attract more users and in tum increase the demand, resulting in enhanced congestion problems. In addition, there are many problems combined with the packet transportation planning, operation and management. A careful investigation on the effect of relevant demand and supply is necessary before any decision is implemented.

1.2 Research Goals And Outline Of This Thesis

An Optimized route is a mathematical solution based on network route congestion level compared to demand needs with emphasis on available less congested links, by introducing a convex "cost" function of the link (route) as the objective function. This cost function the link utilization (traffic), the heavier the cost to send a unit of traffic via that link. This by can be interpreted as the penalty cost of sending an additional unit of traffic via a link, the heavier it self has two main advantages:

1. The use of such a function helps deter routing traffic via those links that are already heavily loaded.

2. Encourage routing traffic via lightly loaded links.

The intention is that the resulting optimal routing will therefore result in more even load distribution across the network.

The goal of this research is to develop an integer programming-based approach for solving the network traffic congestion problem using traffic costs on links. As discussed in later chapters, some similar topics have been previously studied. The difference between the approaches provided here with others is mainly based on the fact that the model given in this research places more emphasis on the nature of cost in the data and in the problem. Further more, for the purpose of real-time application, the algorithmic procedure is more tuned toward quick execution time and graphical user interface interactions.

This thesis consists of five chapters, conclusion, references and appendices. It is organized as follows.

An introduction is provided in the chapter one, this is aimed on providing the reason why this research was carried out. Chapter two, provides a entrance to optimization methods beginning with a discussion of general concepts used on this subject, and focusing on using an integer-programming model, on which this research is based. Different optimization techniques, such as linear programming and integer programming with its great flexibility is discussed and its methods are reviewed. Starting with the graphical method and how it can be used to solve network related problems, then simplex method and how it can be used to solve more complex network problems. And finally how can complex linear programming problems be used in integer programming. Illustrative examples showing this are reviewed. The chapter is also extended into non-linear problems with the Newton method and how it can be used to iterate solutions of linearly unsolved problems. We then go to the Lagrangian method and how it can reformulate a non-linear problem subject to constraints into a model solvable by linear programming. This method emphasize the fact that some systems may not be "optimized", unless a modification in the system is necessary. This is particularly relevant to avoid the complexity of real network optimization problems.

Chapter three is devoted to the network optimization algorithms. A brief introduction to terminologies of algorithms, its complexities and other fundamental notations are described. The explanations of the algorithms used in network routing, the discussion of Dijkstra's algorithm and how it is used in real networks to find the shortest path based on the path's costs. Also a description of maximum flow algorithms and how its used to generate the highest amol1nt of traffic in a given network based on the link capacity, the Ford-Fulkerson method are discussed and how it applies in a given network.

Chapter four introduces a modified version of the "Network Optimization Problem". Based on the fact that congested link information may contain some data. The objective cost function is formulated under the constraints of the overall network reliability transported via each link. Related research is reviewed and a network optimization problem is solved.

Chapter five focuses on the implementation of the network optimization model. Note that the artificial random variables used in the model give a convenient starting feasible solution for this kind of integer programming model. It is also noticed that a standard simplex method application is not suitable for this kind of a problem because of limitations in computer memory. Hence, the integer-programming module will be used to improve network performance by maintaining reliable links in the network using an optimization package. We also implemented the single source shortest path problem using Dijkstra's algorithm in MS-Visual Basic™, and demonstrate the working of this algorithm, with costs assigned as distances from one point to another, and how the algorithm find the shortest path visually.

In conclusion, the obtained results achieved from the thesis are given and the solution is simulated.

2. STATE OF APPLICATION PROBLEM OF OPTIMIZATION MODELS FOR NETWORK TRAFFIC

2.1 Overview

Optimization is a branch of science dealing with techniques for optimizing the performance of systems. It is a scientific method that provides executives a quantitative and rational basis for taking decisions, especially those dealing with the allocation of resources. The focus of the subject is on scientific methods of decision making that seek to understand the complex operations of any system to predict it's behavior and improve it's perfot~ance. Optimization applications deal with making decisions or taking actions that are optimal in the sense of helping any department, organization, or entity such as a manufacturing company making and selling a variety of products, a hospital delivering health services, a university educating students, etc.

2.1.1 An Introduction To Optimization

The optimization approach for improving any system typically takes the following six steps discussed below.

1. Identifying the Decision Variables.

This means observing the operation of the system carefully and identify the parameters whose values can be controlled, and which affect its performance.

2. Construct a Mathematical Model of System Operation and Objectives.

Here we identify measures of effectiveness of system performance, and express each of them as a mathematical function of the decision variable. To optimize an objective function means either maximize or minimize it as desired. The objective function is a mathematical model for the system.

3. Solve the model for an Optimum Solution.

A solution refers to a numerical vector giving values to each decision variable, A solution is said to be a feasible solution if it satisfies all the constraints in the model. An optimum solution is a feasible solution that has the most durable value(s) for the objective function(s) among all feasible solutions. Solving the model means finding an optimum solution for it.

4. Perform Sensitivity Analyses.

These analyses determine the sensitivity of the optimum solution to the model specifications. They determine how robust the optimum solution is under inaccuracies in input data and structural assumptions.

5. Implementing the findings and updating the model.

In this final phase, the optimum solution is implemented. This requires checking it for practical feasibility, making necessary modifications in the model and solving it again if it is found to be impractical for some reason, and repeating the whole process as needed.

Optimization of system performance is the basic aim of Operation Research (OR). The performance of most systems can be improved through intelligent use of optimization algorithms [12].

Many years ago, Ancient humans worried about problems like, "What is the shortest route to the river from my home? " And they worked out solutions by trail and error. Since there are no constraints on the route, this is called unconstrained optimization model.

On the other hand, the unconstrained shortest route between the house and the river may pass through risky areas. If passing through tiger infested areas, the new question will be modified as " What is the shortest route from my home to the river that avoids areas infested with tigers?" This is called constrained optimization model.

Unconstrained optimization continues to be extremely important in optimization theory. Because its often advantageous to solve constrained optimization models using unconstrained optimization techniques, either by ignoring the constraints in the model, or by transforming the constrained problem into an unconstrained one using penalty function methods or Lagrangian methods. Such transformations are most commonly used for solving nonlinear models in continuous variables [12].

And as humans started to rely on farming, questions like" How much of my 100 acres of farmland should I allocate for wheat, com, and oats to maximize my income? " began to rise. The amount of land allocated to a crop can be continuous variable, which is one that can assume all possible real values within the bounds on it imposed by the constraints in the model. Problems of this type are called continuous variable optimization models, which can be linear or nonlinear programs. Also when a question assumes only a value from a specified discrete set as " How many cows and lambs should I grow to maximize my income? " type of questions. The problems involving such variables are called discrete optimization problems or integer programs. And when a problem involves some continuous variables and some discrete variables its known as a mixed discrete optimization problem or a mixed integer program [14).

Questions like " What is the optimum farming policy for my farm for each year over the next five year horizon? "are called multi-period or dynamic models in which they involve decisions for each period over a multi-period horizon. In the same manner problems that involve decision for a one period problem are called static models.

In a single objective static optimization model, the objective function can be interpreted as the yield or profit that is required to be maximized. The objective function expresses the yield as a function of the various decision variables. In real world applications, the yield is almost never known with certainty; typically it's a random variable subject to many random fluctuations that are not under our control. For example the yield may depend on the unit profit coefficients of the various goods manufactured by the company (data elements in the model) and these things fluctuate randomly. To analyze the problem treating the yield as a random variable requires the use of complicated stochastic programming models. Instead one normally analyses the problem using a deterministic model in which the random variable in the yield function are replaced by their most likely values or expected values.

2.2 Structure of Optimization Problems

Optimization problems are made up of three basic ingredients:

• An objective function, which we want to minimize or maximize. For instance, in a manufacturing process, we might want to maximize the profit or minimize the cost. In fitting experimental data to a user-defined model, we might minimize the total deviation of observed data from predictions based on the model. In designing an automobile panel, we might want to maximize the strength.

• A set of unknowns or variables, which affect the value of the objective function. In the manufacturing problem, the variables might include the amounts of different resources used or the time spent on each activity. In fitting-the-data problem, the unknowns are the parameters that define the model. In the panel design problem, the variables used define the shape and dimensions of the panel.

• A set of constraints that allow the unknowns to take on certain values but exclude others. For the manufacturing problem, it does not make sense to spend a negative amount of time on any activity, so we constrain all the "time" variables to be non-negative. In the panel design problem, we would probably want to limit the weight of the product and to constrain its shape [11].

The optimization problem is summarized as: Find values of the variables that minimize or maximize the objective function while satisfying the constraints. There are however, important terms that will be repeatedly used in almost all optimization problems those terms are:

Objective Function: Almost all optimization problems have a single objective function. (When they don't they can often be reformulated so that they do!) The two interesting exceptions are:

• No objective function. In some cases (for example, design of integrated circuit layouts), the goal is to find a set of variables that satisfies the constraints of the model. The user does not particularly want to optimize anything so there is no reason to define an objective function. This type of problems is usually called a feasibility problem.

• Multiple objective functions. Often, the user would actually like to optimize a number of different objectives at once. For instance, in the panel design problem, it would be nice to minimize weight and maximize strength simultaneously. Usually, the different objectives are not compatible; the variables that optimize one objective may be far from optimal for the others. In practice, problems with multiple objectives are reformulated as single-objective problems by either forming a weighted combination of the different objectives or else replacing some of the objectives by constraints. These approaches and others are described in our section on multi-objective optimization.

Variables: These are essential. If there are no variables, we cannot define the objective function and the problem constraints.

Constraints: Constraints are not essential. In fact, the field of unconstrained optimization is a large and important one for which a lot of algorithms and software are available. It's been argued that almost all problems really do have constraints. For example, any variable denoting the "number of objects" in a system can only be useful if it is less than the number of elementary particles in the known universe! In practice though, answers that make good sense in terms of the underlying physical or economic problem can often be obtained without putting constraints on the variables [10].

Figure 2.1 in the next page shows the optimization tree and its major branches:

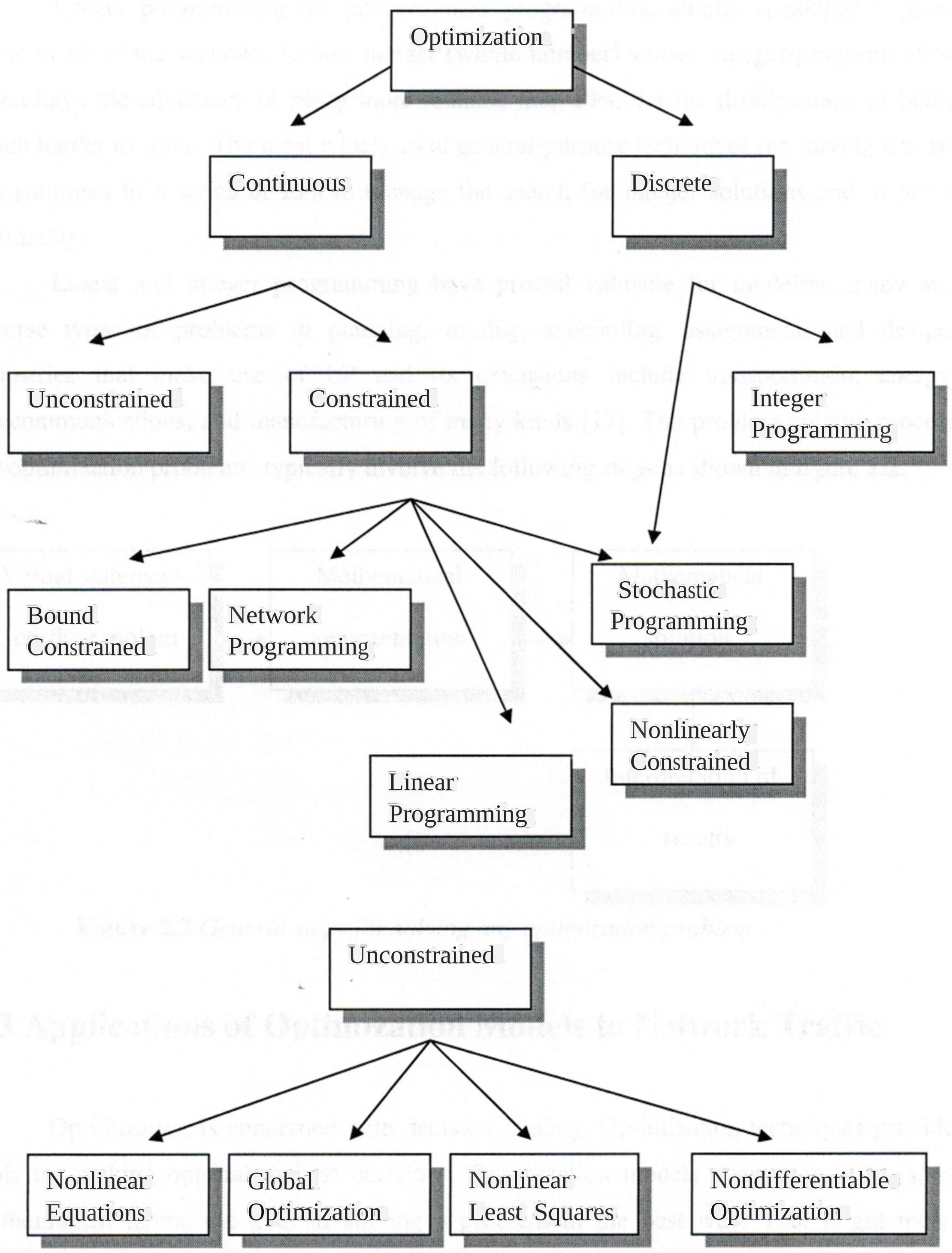
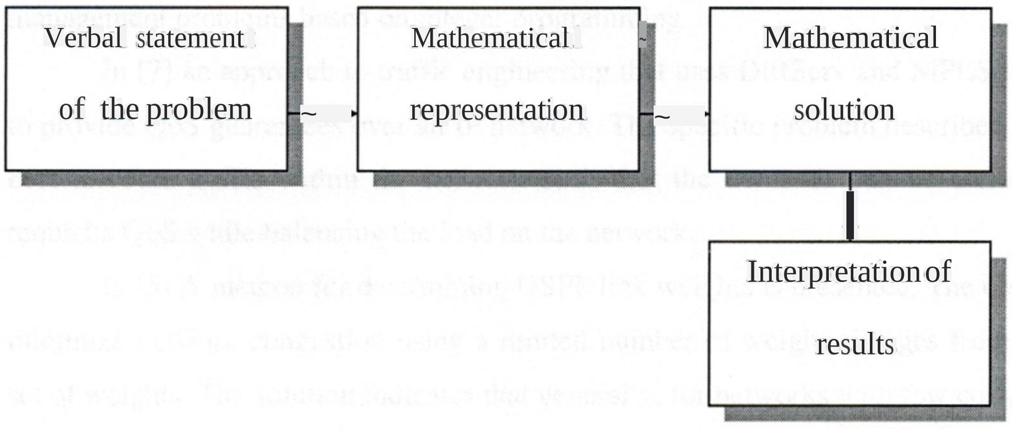


Figure 2.1 *The Optimization tree structure [10]*

Integer programming (or integer linear programming, strictly speaking) requires some or all of the variables to take integer (whole number) values. Integer programs (IPs) often have the advantage of being more realistic than LPs, but the disadvantage of being much harder to solve. The most widely used general-purpose techniques for solving IPs use the solutions to a series of LPs to manage the search for integer solutions and to prove optimality.

Linear and integer programming have proved valuable for modeling many and diverse types of problems in planning, routing, scheduling, assignment, and design. Industries that make use of LP and its extensions include transportation, energy, telecommunications, and manufacturing of many kinds [17]. The problem solving process for optimization problems, typically involve the following steps as shown in figure 2.2.



Verbal statement of the problem
Mathematical representation
Mathematical solution
Interpretation of results

Figure 2.2 *General steps for solving any optimization problem*

2.3 Applications of Optimization Models to Network Traffic

Optimization is concerned with decision-making. Optimization techniques provide tools for making optimal or best decisions. Optimization models attempt to express, in mathematical terms, the goal of solving a problem in the best way. That might mean running a business to maximize the profit, minimize loss, maximize efficiency, or minimize risk. It might mean designing a bridge to minimize weight or maximize strength. It might mean selecting a flight plan for an aircraft to minimize time or fuel use.

The desire to solve a problem in an optimal way is so common that optimization models arise in almost every area of application. Optimization models have been used for centuries, since their purpose is appealing. One of application area of optimization models is network traffic optimization. There are number of research works about application of optimization models to network traffic.

In [6] the effectiveness of the application of optimization techniques to traffic management in computer networks is demonstrated. The target technology consists of computer networks based on the IP Protocol over Multi-protocol Label Switching (MPLS), which plays a key role by providing services unsupported by the IP Protocol, a number of traffic management tasks have been cast as mixed-integer programming problems, whose optimal solution entails the solution of computationally hard problems. To this end, that paper applied Lagrangean relaxation methods to approximately solve one of these traffic management problems based on integer programming.

In [7] an approach to traffic engineering that uses DiffServ and MPLS technologies to provide QoS guarantees over an IP network. The specific problem described here is how best to route traffic within the network such that the demands can be carried with the requisite QoS while balancing the load on the network.

In [5] A method for determining OSPF link weights is presented. The objective is to minimize network congestion using a limited number of weight changes from an original set of weights. The solution indicates that generally, for networks with few congested links, the network performance may be significantly improved with weight changes to just a few links. That method is very useful for autonomous system operators to improve the network performance in response to growing or changed traffic demands without necessarily having to upgrade the network resources based on linear programming.

In (54] A new approach to fuel-optimal path planning of multiple vehicles using a combination of linear and integer programming is presented. The basic problem formulation is to have the vehicles move from an initial dynamic state to a final state without colliding with each other, while at the same time avoiding other stationary and moving obstacles. It is shown that this problem can be rewritten as a linear program with mixed integer/ linear constraints that account for the collision avoidance. A key benefit of this approach is that the path optimization can be readily solved using the CPLEX optimization software with an AMPUMatlab interface. An example was worked out to show that the framework of mixed integer/linear programming is well suited for path planning and collision avoidance problems. Implementation issues are also considered. In particular, we compare receding horizon strategies with fixed arrival time approaches.

In (8) in order to improve network reliability, a heuristic search algorithm based on genetic algorithms was presented to optimize the design of large-scale network topologies subject to a reliability constraint. The search was implemented with an improved Monte-Carlo simulation technique to estimate the system reliability of a network topology.

In (52) an asynchronous team algorithm was proposed in a parallel heterogeneous asynchronous environment, to optimize the design of reliable communication networks given the set of nodes and possible links. That proposed team combines parallel genetic algorithms with different reliability calculation approaches in a network of personal computers.

In (53) A machine learning approach to the topological optimization of computer networks is presented. Traditionally formulated as an integer program, this problem is well known to be a very difficult one, only solvable by means of heuristic methods. This paper addresses the specific problem of inferring new design rules that can reduce the cost of the network, or reduce the message delay below some acceptable threshold. More specifically, it extends a recent approach using a rule-based system in order to prevent the risk of combinatorial explosion and to reduce the search space of feasible network topologies. This extension essentially implements an efficient inductive learning algorithm leading to the refinement of existing rules and to the discovery of new rules from examples, defined as network topologies satisfying a given reliability constraint. The contribution of this paper is the integration of learning capabilities into topological optimization of computer networks.

In [1] Its stated that network optimization problems can be approached in many different ways, e.g. using linear programming, operations research theory, discrete simulation, and using algorithmic approaches from the computer science field.

Those research works and many others, point out to the effective use of optimization in the field of networks.

2.4 Linear Programming (LP)

Linear programming is a widely used model type that can solve decision problems with many thousands of variables. Generally, the feasible values of the decisions are delimited by a set of constraints that are described by mathematical functions of the decision variables. The feasible decisions are compared using an objective function of the decision variables. For a linear program the constraints and objective functions are required to be linearly related to the variables of the problem. A Linear Program (LP) is a problem that can be expressed as follows:

$$\begin{aligned} &\textit{Minimize}\ \ ex \\ &\textit{Subject to}\ Ax \equiv b \\ &x{\sim}O \end{aligned}$$

Where x is the vector of variables to be solved for, A is a matrix of known coefficients, and c and b are vectors of known coefficients. The expression *"ex"* is called the objective function, and the equations *"Ax=b"* are called the constraints (11].

Basically, a linear programming problem can be solved in either the graphical method or the simplex method. Graphical method is basically applied to problems where the decision variables can be reduced to two, where as the simplex method can handle more than two, tens and even hundreds of variables. The simplex method can also be applied to computer calculations easily, which will greatly aid in getting an optimal solution, due to the high computing speed of the computer. Next we discuss the dual problem, and how it can be used to change a maximization problem into a minimization problem. Which can aid in problems that get stuck on some solution.

A deeper demonstration of linear programming is explained in the graphical method as well as in the simplex method sub chapters, along with applied examples, in networks for each method. The examples in the coming sections illustrate that linear programming can be used in a wide variety of practical situations. With them, we illustrate how situations can be translated into a mathematical model.

2.4.1 The Fundamental Theorem of Linear Programming

The Fundamental Theorem of Linear Programming: The maximum (or minimum) value of the objective function is achieved at one of the vertices of the feasible set. Generally, such problems involve finding the values of x and y which maximize (or minimize) a particular linear expression in x and y and where x and y are chosen so as to satisfy one or more restrictions in the form of linear inequalities. The expression that is to be maximized (or minimized) is called the objective function. We can summarize the steps to be followed in approaching any linear programming problem.

Step 1, Translate the problem into mathematical language.

A. Organize the data.

B. Identify the unknown quantities and define corresponding variables.

C. Translate the restrictions into linear inequalities.

D. Form the objective function.

Step 2, Graph the feasible set.

A. Put the inequalities in standard form.

B. Graph the straight line corresponding to each inequality.

C. Determine the side of the line belonging to the graph of each inequality. Cross out the other side. The remaining region is the feasible set.

Step 3, Determine the vertices of the feasible set.

Step 4, Evaluate the objective function at each vertex. Determine the optimum point.

Linear programming can be applied to many problems. The US Army Corps of Engineers has used linear programming to plan the location of a series of dams so as to maximize the resulting hydroelectric power production. The restrictions were to provide adequate flood control and irrigation [9].

Public transit companies have used linear programming to plan routes and schedule buses in order to maximize services. The restrictions in this case arose from the limitations on labor, equipment, and funding. The petroleum industry uses linear programming in the refining and blending of gasoline. Profit is maximized subject to restrictions on availability of raw materials, refining capacity, and product specifications. Some large advertising firms have used linear programming in media selection, the problem consist of determining how much to spend on each medium in order to maximize the number of consumers reached. The restrictions come from limitations on the budget and the relative costs of different media. Psychologists used linear programming to design an optimum battery of tests. The problem is to maximize the correlation between test scores and the characteristics that to be predicted [12].

The restrictions are imposed by the length and cost of testing. Dieticians also use linear programming, in planning of meals for large numbers of people [14]. The objective is to minimize the cost of the diet, and the restrictions reflect the minimum daily requirements of the various nutrients considered in the diet. In short linear programming has numerous uses in industry and in practice.

2.4.2 A Network Linear Programming Problem Solved By The Graphical Method

Suppose that a network of some type (computers, locations, factories-markets, warehouse-retail stores) has demand nodes (receivers) in Node 1 and Node 2 and supply nodes (senders) in Node3 and Node4. The cost of sending a unit from Node 3 to Node 1 is \$6. From Node3 to Node2, \$3. From Node4 to Nodel, \$9. And from Node 4 to Node 2, \$5. Suppose that Node 1, requests 25 units and Node 2, 30 units. Suppose further that Node 3 has a limited supply of 45 units and the Node 4, 40 units. What is the most economical way to supply the requested units to the two demand nodes? [9]

The first step in solving a linear programming problem is to translate it into mathematical language. And the first part of this step is to organize the information given preferably in the form of a chart. We drew a schematic diagram, as in Fig 1.2 which shows the flow of units between supply nodes and demand nodes. By each route, we have written the cost. Below each supply node, we have written down its available units and below each demand node the number of units it requested.

Next, we determine the variables. It appears initially that four variable are required, namely the number of units to be sent over each route. However, a closer look shows that only two variables are required. For if x denotes the number of units to be sent from Node 3 to Node 2. Since Node 2 requested 30 units. The number sent from Node 4 to Node 2 is 30. similarly, if y denotes the number of units sent from Node 3 to Node 1, then the number sent from Node 4 to Node 1, is 25- y. We have written the appropriate unit sending values beside the various routes.

As the third part of the translation process let us write down the restrictions on the variables. Basically, there are two kinds of restrictions: none of x, y, 30 - x, 25 - y can be negative, and a supply node cannot send more units than it has. Referring to Fig2.4. We see that Node 3 ships x + y sets, so that x + y $ 45. Similarly. Node 4 sends (30 - x) + (25 - y) units

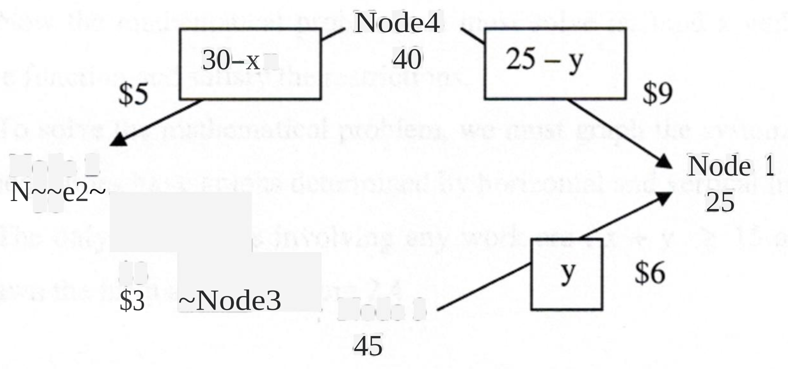


Figure 2.3 *Solving a network optimization problem using the graphical method [9]*

Now we have (30 - x) + (25 - y) $ 40. Simplifying this inequality, we get

55 - x - y $ 40

- x -y $ 15

x + y ~ 15

The inequality 30 - x ~ 0 can be simplified to x $ 30. and the inequality 25 - y ~ 0 can be written y $ 25. So the restriction inequalities are these:

xZ 0, y ~ 0

x ~ 30, y s 25

x + y ~ 15

x + y ~ 45

The final step in the translation process is to form the objective function. x sets In this problem we are attempting to minimize cost, so the objective function must express the cost in terms of x and y. Referring again to Fig 2.3. There are x sets going from Node 3 to Node 2, and each costs \$3 to transport, so the cost of delivering these x sets is 3x. Similarly, the costs of making the other deliveries are 6y, 5(30 - X), and 9(25 - y), respectively. Thus the objective function is

[Cost] = 3x + 6y + 5(30 - x) + 9(25 - y) = 3x + 6y + 150 - 5x + 225 - 9y =375 - 2x - 3y.

Now the mathematical problem we must solve is: Find x and y that minimize the objective function and satisfy the restrictions.

To solve the mathematical problem, we must graph the system of inequalities. Four of the inequalities have graphs determined by horizontal and vertical lines.

The only inequalities involving any work are . x + y ~ 15 and x + y ~ 45 . We have drawn the inequalities in figure 2.4

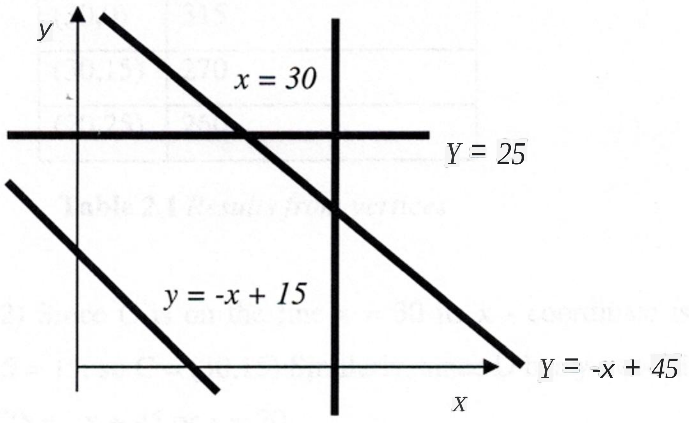


Figure 2.4 *Drawing the lines of equations to find the feasible set*

In Figure 2.5, we have drawn the feasible set and have labeled each boundary line with its equation. The vertices A to F are now simple to determine. First, A and F are the intercepts of the line y = - x + 15. Therefore. A = (15,0) and F = (0,15). Since B is the x-intercept of the line x = 30. We have B = (30,0).

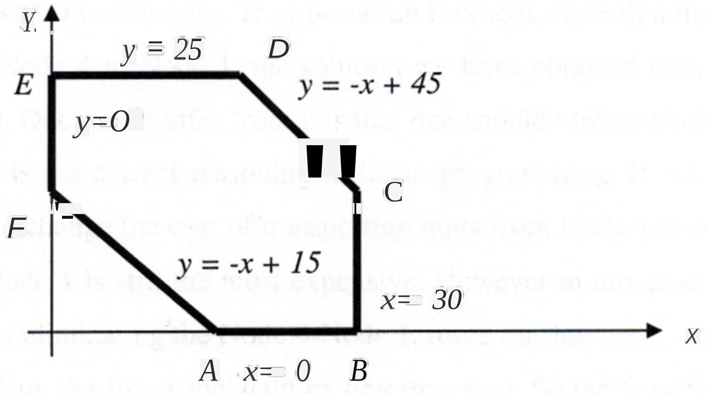


Figure 2.5 *Finding the feasible set and it boundaries*

Vertex	Cost= 375 - 2 x - 3 y
(0,25)	300
(0,15)	330
(15,0)	345
(30,0)	315
(30, 15)	270
(20,25)	260

Table 2.1 *Results from vertices*

Similarly E = (0,2) Since C is on the line x = 30 its x - coordinate is 30, its y - coordinate is y = - 30 + 45 = 15, so C = (30,15) Similarly, since D has y - coordinate 25, its x -coordinate is given by 25 = - x + 45 or x = 20.

Thus, $D = (20,25)$. In Table 2.1 the vertices A to F are listed, as well as the cost corresponding to each one. The minimum cost of $260 occurs at the vertex (20,25). So $x = 20$, $y = 25$ yields the minimum of the objective function. In other words 20 Units should be sent from Node 3 to Node 2, and 25 from Node 3 to Node 1, $30 - x = 10$ from Node 4 to Node 2, and $25 - y = 0$, from Node 4 to Node 1. And this solves the problem.

Remarks Concerning the Transportation Problem, Note that the highest cost route is the one from Node 4 to Node 1, the solution we have obtained eliminates any shipments over this route. One might infer from this that one should always avoid the most expensive route, but this is not correct reasoning in linear programming. If we reconsider the above example except change the cost of transporting units from Node 4 to Node 1, from $9 to $7 The Node 4-Node 1 is still the most expensive. However in this case the minimum cost is not obtained by eliminating the Node 4-Node 1, route for the

Revised problem, the linear inequalities stay the same. So the feasible set and the vertices remain the same. The only change is in the objective function, which now is given by

$[\text{Cost}] = 3x + 6y + 5(30 - x) + 7(25 - y) = 325 - 2x - y.$

Therefore, the costs at the various vertices are as given in Table 1.4. The minimum cost of $250 is achieved when $x = 30$, $y = 15$, $30 - x = 0$. And $25 - y = 10$. Note that 10 sets are being shipped from Node 4 to Node 1, even though this is the most expensive route. It is even possible for the cost function to be optimized simultaneously a two different vertices For example if the cost from Node 4 to Node 1 is $8 and all other data are the same as in the example, then the optimum cost is $260 and is achieved at both vertices (30,15) and (20,25).

The graphical method is a good way to find an optimized solution for problems that has two variables or can be reduced to two variables. In the next we will introduce the simplex method, which is a method capable of solving linear problems with literary hundreds of variables.

2.5 The Simplex Method

In the previous section, we introduced a graphical method for solving linear programming problems. This method, although simple, is of limited usefulness, since it applies only to problems, which involve (or can be reduced to) two variables.

On the other hand, linear programming applications in business and industry can involve dozens or even hundreds of variables. In this section we describe a method for handling such applications. This method, called the simplex method (or simplex algorithm). Was developed by the mathematician George B. Dantzig in the late 1940s and today is the principal method used in solving complex linear programming problems. The simplex method can be used for problems in any number of variables and is easily adapted to computer calculations [17].

2.5.1 The Simplex Method for Problems in Standard Form

The Simplex method of the standard form can be solved using the following steps:

1. Introduce slack variables and state the problem in terms of a system of linear equations.

2. Construct the simplex tableau corresponding to the system.

3. Determine if the left part of the bottom row contains negative entries. If none are present, the solution corresponding to the tableau yields a maximum and the problem is solved.

4. If the left part of the bottom row contains negative entries, construct a new simplex tableau.

(a) Choose the pivot column by inspecting the entries of the last row of the current tableau, excluding the right-hand entry. The pivot column is the one containing the most negative of these entries.

(b) Choose the pivot element by computing ratios associated with the positive entries of the pivot column. The pivot element is the one corresponding to the smallest nonnegative ratio.

(c) Construct the new simplex tableau by pivoting around the selected element.

5. Return to step 3. Steps 3 and 4 are repeated as many times as necessary to find a maximum.

2.5.2 The Simplex Method, Minimum Problems.

In the preceding sections we have developed the simplex method and applied it to a number of problems, however throughout we restricted the problems to linear-programming problems in standard form. Such problems satisfied three properties: The objective function is to be maximized. Each variable must be ~0. And all constraints other than those implied by (2) must be in the form:

[Linear polynomial] ~ [Nonnegative constant].

In this section, we do what we can to relax these restrictions.
The simplex method for problems in nonstandard form:

1. If necessary, convert all inequalities (expect x ~ 0, y ~ 0), into the form [Linear polynomial] ~ [Constant].

2. If a negative number appears in the upper part of the last column of the simplex tableau, remove it by pivoting

(a) Select one of the negative entries in its row. The column containing the entry will be the pivot column.

(b) Select the pivot element by determining the least of the positive ratios associated with entries in the pivot column (except the bottom entry)

(c) Pivot.

3. Repeat step 2 until there are no negative entries in the upper part of the right hand column of the simplex tableau.

4. Proceed to apply the simplex method for tableaus in standard form.

The methods we have just developed can be used to solve minimum problems as will as maximum problems, minimizing the objective function f is the same as maximizing (-1). f This is so since multiplying an inequality by (-1) reverses the direction of the inequality sign. Thus, to apply this method to a minimum problem, we merely multiply the objective function by (-1) and tum the problem into a maximum problem.

2.5.3 Solving Network Problems Using The Simplex Method.

Now, we rework an applied network problem previously treated, this time using the simplex method. For easy reference we restate the problem, Suppose that a network of some type (computers, locations, factories-markets, warehouse-retail stores) has demand nodes (receivers) in: Node 1 and Node 2, and supply nodes (senders) in Node 3 and Node 4. The cost of sending a unit from Node 3 to Node 1 is $6: from Node 3 to Node 2, $3. From Node 4 to Node 1, $9: and from Node 4 to Node 2, $5. Suppose that the Node 1 requests 25 units and the Node 2, 30 units. Suppose further that the Node 3 has a limited supply of 45 units and the Node 4, 40 units. What is the most economical way to supply the requested units to the two demand nodes?

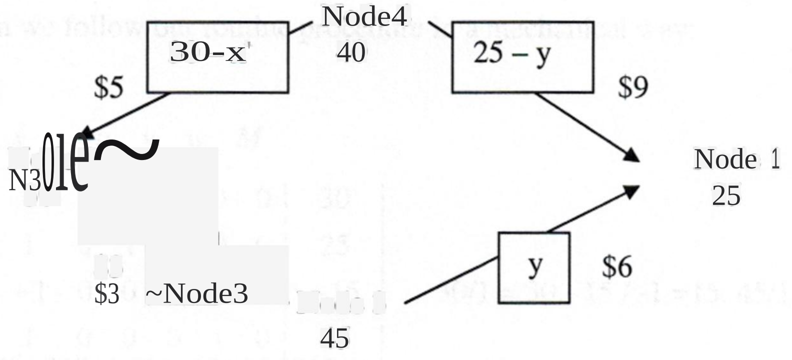


Figure 2.6 *The same network problem to be solved by the simplex method*

Solution via the Simplex method: As in the previous solution, let x be the number of sets sent from node 3 to node 2 and y the number sent from node 3 to node 1, The flow of sets is shown in Figure 2.6. Exactly as in the previous solution, we reduce the problem to the following algebraic form: Minimize 375 - 2x - 3y, subject to the constraints

$$x \sim 0,\ y \sim 0$$
$$x \sim 30,\ y \sim 25$$
$$x + y \sim 15$$
$$x + y \sim 45$$

Two changes are needed, First instead of minimizing 375 - 2x - 3y, we maximize - (375 -2x - 3y). Second we write the constraint, x + y ~ 15 in the form- x -y~ 15 With these changes made, we can write down the lineal system:

$$\begin{array}{lllllllll}
x & & +t & & & & & = & 30 \\
 & y & & +u & & & & = & 25 \\
-x & -y & & & +v & & & = & -15 \\
x & +\ y & & & & +w & & = & 45 \\
-2x & -3y & & & & & +M & = & -375.
\end{array}$$

From here on we follow our routine procedure in a mechanical way:

	x	y	t	u	v	w	M	
t	1	0	1	0	0	0	0	30
u	0	1	0	1	0	0	0	25
v ==4	-1	-1	0	0	1	0	0	-15
w	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
M	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

30/1 = 30, -15 / -1 =15, 45/1 = 45

$$
\begin{array}{c|ccccccc:c|}
 & x & y & t & u & v & w & M & \\
t & 0 & -1 & 1 & 0..1, & 0 & 0 & & 15 \\
u & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 25 \\
x & 1 & 1 & 0 & 0 & -1 & 0 & 0 & 15 \\
w & & & & & & & & \\
M & \text{[illegible]} & & & & & & & 5
\end{array}
\qquad 15/1 = 15\ ,\ 30/1 = 30
$$

$$
\begin{array}{c|ccccccc:c|}
 & x & y & t & u & v & w & M & \\
v & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 15 \\
u & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 25 \\
x & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 30 \\
w & 0 & 1 & -1 & 0 & 0 & 1 & 0 & 15 \\
\hdashline
M & 0 & -3 & 2 & 0 & 0 & 0 & 1 & -315
\end{array}
\qquad 25/1 = 25\ ,\ 15/1 = 15
$$

t

$$
\begin{array}{c|ccccccc:c|}
 & x & y & t & u & v & w & M & \\
v & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 30 \\
u & 0 & 0 & 1 & 1 & 0 & -1 & 0 & 10 \\
x & 1 & 0 & T & 0 & 0 & 0 & 0 & 30 \\
y & 0 & 1 & -1 & 0 & 0 & 1 & 0 & 15 \\
\hdashline
M & 0 & 0 & -1 & 0 & 0 & 3 & 1 & -270
\end{array}
\qquad 10/1 = 10\ ,\ 30/1 = 30
$$

t

$$
\begin{array}{c|ccccccc:c|}
 & x & y & t & u & v & w & M & \\
v & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 30 \\
t & 0 & 0 & 1 & 1 & 0 & -1 & 0 & 10 \\
x & 1 & 0 & 0 & -1 & 0 & 1 & 0 & 20 \\
y & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 25 \\
\hdashline
M & 0 & 0 & 0 & 1 & 0 & 2 & 1 & -260
\end{array}
$$

The last tableau corresponds to a maximum. So $2x + 3y - 375$ has a maximum value of -260, and therefore $375 - 2x - 3y$ has a minimum value 260. This value occurs when $x = 20$ and $y = 25$. This is an agreement on our previous graphical solution.

The simplex method is so mechanical in its execution that it is much easier to program for a computer. For another, our previous method was restricted to problems in two variables; however suppose that the two supply nodes were to deliver their units to three or four or perhaps even 100 demand nodes. Our previous method could not be applied however; the simplex method although yielding very large matrices and very tedious calculations is applicable. This is the method used by many industries to optimize distribution of their products [9].

2.6 Integer Programming

When formulating LP's, it's often found that, strictly, certain variables should have been regarded as taking integer values but, for the sake of convenience, we let them take fractional values reasoning that the variables were likely to be so large that any fractional part could be neglected. While this is acceptable in some situations, in many cases it is not, and in such cases we must find a numeric solution in which the variables take integer values [21].

Problems in which this is the case are called integer programs (IP's) and the subject of solving such programs is called integer programming (also referred to by the initials IP). IP's occur frequently because many decisions are essentially discrete (such as yes/no, go/no-go) in that one (or more) options, must be chosen from a finite set of alternatives.

Problems in which some variables can take only integer values and some variables can take fractional values are called mixed-integer programs (MIP's).
As for formulating LP's the key to formulating IP's is practice. Although there are a number of standard procedures available to cope with situations that often arise in formulating IP's it is probably true to say that formulating IP's is a much harder task than formulating LP's [21].

2.6.1 Solving Integer Programs

For solving LP's we have general purpose (independent of the LP being solved) and computationally effective (able to solve large LP's) algorithms (simplex or interior point). For solving IP's no similar general purpose and computationally effective algorithms exist. Theory suggests that no general purpose computationally effective algorithms will ever be found. This area is known as computational complexity and concerns NP-completeness. It was developed from the early 1970's onward and basically is a theory concerning "how long it takes algorithms to run". This means that IP's are a lot harder to solve than LP's [38].

Solution methods for IP's can be categorized as:

• General purpose (will solve any IP) but potentially computationally ineffective (will only Solve relatively small problems)

• Special purpose (designed for one particular type of IP problem) but potentially computationally more effective.

Solution methods for IP's can also be categorized as:

• Optimal

• Heuristic

An optimal algorithm is one, which (mathematically) guarantees to find the optimal solution. It may be that we are not interested in the optimal solution, because the size of problem that we want to solve is beyond the computational limit of known optimal algorithms within the computer rime we have available; or we could solve optimally but feel that this is not worth the effort (time, money, etc) we would spend in finding the optimal solution.

In such cases we can use a heuristic algorithm, which is an algorithm that should hopefully find a feasible solution, which in objective function terms, is close to the optimal solution. In fact it is often the case that a well-designed heuristic algorithm can give good quality (near-optimal) results [10].

For example a heuristic for a capital budgeting problem would be:

1. Consider each project in turn

2. Decide to do the project if this is feasible in the light of previous decisions

Applying this heuristic we would choose to do for example 1 project or 2, giving a total return, which may (or may not) be the optimal solution.

Hence, we have four categories that we potentially need to consider:

- General purpose, optimal
- General purpose, heuristic
- Special purpose, optimal
- Special purpose, heuristic

The methods presented below are suitable for solving both IP's (all variables integer) and MIP's (mixed-integer programs - some variables integer, some variables allowed to take fractional values).

- General-purpose optimal solution algorithms:

We will deal with just two general purpose (able to deal with any IP) optimal solution algorithms for IP's:

1. Enumeration (sometimes called complete enumeration)
2. Branch and bound (tree search).

We consider each of these later in this section.

- Enumeration:

Unlike LP (where variables took continuous values (≥ 0)) in IP's (where all variables are integers) each variable can only take a finite number of discrete (integer) values. Hence, the obvious solution approach is simply to enumerate all these possibilities by calculating the value of the objective function at each one and choosing the (feasible) one with the optimal value. IP nowadays is often called "combinatorial optimization" indicating that we are dealing with optimization problems with an extremely large (combinatorial) increase in the number of possible solutions as the problem size increases [16].

- Branch and bound (tree search)

The most effective general purpose optimal algorithm is LP-based tree search (tree search also being called branch and bound). This is a way of systematically enumerating feasible solutions such that the optimal integer solution is found.

Where this method differs from the enumeration method is that not all the feasible solutions are enumerated but only a fraction (hopefully a small fraction) of them. However, we can still guarantee that we will find the optimal integer solution. The method was first put forward in the early 1960's by Land and Doig [12].

2.6.2 General Purpose Heuristic Solution Algorithms

If we are dealing with one specific type of IP then we might well be able to develop a special purpose solution algorithm (designed for just this one type of IP) that is more effective computationally than the general-purpose branch and bound method given earlier. These are typically tree search approaches based upon generating bounds via:

- Dual ascent
- Lagrange-an relaxation
- Sub gradient optimization
- Multiplier adjustment.

Such algorithms draw upon the concepts, such as branch and bound, outlined previously and they often use linear programming via LP relaxation. Or take advantage of the structure of the constraints of the IP they are solving. Different methods have different computational performance and their general behavior is that each method is computationally effective up to a çertain size of problem and then becomes computationally ineffective (as the effort needed to obtain an optimal solution begins to increase exponentially in terms of the size of problem considered).

2.7 Non-Linear Programming (NLP)

In general a Nonlinear Program (NLP) is a problem that can be put into the form:

$$\begin{aligned} &\textit{Minimize } f(x) \\ &\textit{Subject to } g_i(x) = 0 \quad \textit{for } i = 1, \ldots, m1 \quad \textit{where } m1 \geq 0 \\ &\qquad\qquad\quad h_j(x) \geq 0 \quad \textit{for } j = m1+1, \ldots, m \quad \textit{where } m \geq m1 \end{aligned}$$

That is, there is one scalar-valued function!, of several variables (x here is a vector), that we seek to minimize subject to one or more other such functions that serve to limit or define the values of these variables. *f* is called the "objective function", while the various other functions are called the "constraints". (If maximization is desired, it is trivial to do so, by multiplying *f* by -1.)

Because NLP is a difficult field, researchers have identified special cases for study. A particularly well-studied case is the one where all the constraints *g* and *hare* linear. The name for such a problem, unsurprisingly, is "linearly constrained optimization". If, as well, the objective function is quadratic at most, this problem is called Quadratic Programming (QP). An even more special case of great importance is where the objective function and the constraints are entirely linear; this is called Linear Programming (LP). Another important special case, called unconstrained optimization, is where there are no constraints at all [1],[19].

One of the greatest challenges in NLP is that some problems exhibit "local optima"; that is, spurious solutions that merely satisfy the requirements on the derivatives of the functions. Think of a near-sighted mountain climber in a terrain with multiple peaks, and you'll see the difficulty posed for an algorithm that tries to move from point to point only by climbing uphill. Algorithms that propose to overcome this difficulty are termed "Global Optimization".

Sequential Linear Programming (SLP) is solving a nonlinear program by a sequence of linear approximations and using linear programming to solve each one. The linear approximations are usually done by using the first-order Taylor expansion. Sequential Unconstrained Minimization Technique (SUMT). This is the penalty function approach. Other methods used to solve NLP are, (IP) Interior-Point, (GRG) Generalized Reduced Gradient, (SQP) Successive (Sequential) Quadratic Programming [21].

2.7.1 The Newton's Method (Newton-Rap son method)

Newton's method is a technique for obtaining successive approximations (iterations) to the solution of an equation, each more accurate than the preceding one. The equation in a variable x is written in the form $f(x)=0$, and the general formula or algorithm is: $x_{n+1} = x_n - f(x_n) / f'(x_n)$

Applied where x_n is the nth approximation. Newton's method can be thought of as repeated estimates of the position on a graph of $f(x)$ against x at which the curve crosses the x-axis, by extrapolation of the tangent to the curve. The slope of the tangent at $(x_1, f(x_1))$ is df/dx at $x = x_1$, that is $f'(x_1) = f(x_1) / (x_2 - x_1)$, $x_2 = x_1 - f(x_1) / f'(x_1)$ and therefore the point where the tangent crosses the x-axis, and is a closer approximation to x at $f(x) = 0$ than x_1 is, Similarly, $x_3 = x_2 - f(x_2) / f'(x_2)$ is a better approximation still. For example, if $f(x) = x^2 - 3 = 0$, then $f'(x) = 2x$ and we obtain the algorithm:

$$x_{n+1} = x_n - (x_n^2 - 3) / 2x_n = \tfrac{1}{2}(x_n + 3/x_n)$$

Now, considering a method for solving
$f(x) = 0$

We first consider the one-dimensional case where x is a scalar and f is a real-valued function. Later we look at the n-dimensional case where $x = (x_1, \ldots, x_n)^T$ and $f(x) = (f_1(x), \ldots, f_n(x))^T$ where both x and $f(x)$ are vectors of the same length n.

Throughout this section we assume that the functionfhas two continuous derivatives If $f(x)$ is a linear function, it is possible to find a solution if the system is nonsingular.

The cost of finding the solution is predictable; it is the cost of applying Gauss Ian - elimination.

Except for a few isolated special cases, such as quadratic equations in one variable, in the nonlinear case, it is not possible to guarantee that a solution can be found, nor is it be possible to predict the cost of finding a solution However, the situation is not totally bleak. There are effective algorithms that work much of the time, and that are efficient on a wide variety of problems. They are based on solving a sequence of linear equations As a result, if the function! is linear they can be as efficient as the techniques for linegr systems[19],[37],[1].

Also, we can apply our knowledge about linear systems in the nonlinear case. The methods discussed are based on Newton's method. Given an estimate of the solution x_k, the function f is approximated by the linear function consisting of the first two terms of the Taylor series for the function! at the point x_k, The resulting linear system is then solved to obtain a new estimate of the solution x_{k+1}. To derive the formulas for Newton's method, we first write out the Taylor series for the function! at the point x_k:

$$f(x_k+p) \approx f(x_k) + pf'(x_k).$$

If $f'(x_k) \neq 0$ then we can solve the equation

$f(x^*) \approx f(x_k) + pf'(x_k) = 0$

For p to obtain $\quad p = -f(x_k)/f'(x_k)$

The new estimate of the solution is then $x_{k+1} = x_k + p$ or

$x_{k+1} = x_k - f(x_k)/f'(x_k)$. This is the formula for Newton's method.

As an example, consider the one-dimensional problem:

$f(x) = 7x^4 + 3x^3 + 2x^2 + 9x + 4 = 0$. Then

$J'(x) = 28x^3 + 9x^2 + 4x + 9$ and the formula for Newton's method is

$$X_{k+J} = X_k - \frac{7x^4+3x^3+2x^2+9x+4}{28x^3 + 9x^2 + 4x + 9}$$

If we start with the initial guess $x_0 = 0$ then

$$X_, = X_0 - \frac{7x^4+3x^3+2x^2+9x+4}{28x^3 + 9x^2 + 4x + 9}$$

$$= 0 - \frac{7(0) + 3(0) + 2(0) + 9(0) + 4}{28(0) + 9(0) + 4(0) + 9}$$

$$= 0 - 4/9 = -4/9 = -0.4444\ldots$$

At the next iteration we substitute $x_, = -4/9$ into the formula for Newton's method and obtain $x_2 = -0.5063$. The complete iteration is given in the table shown below.

K	X_k	$f(x_k)$	$X_k - X^*$
0	0	4×10^{u}	5×10^{-I}
1	− 0.4444444444	4×10^{-I}	7×10^{-7}
2	− 0.50632557489	4×10^{-2}	5×10^{-3}
3	− 0.51100924286	3×10^{-4}	$3 \times 10^{-\sim}$
4	− 0.51104J 786445	2×10^{-9}	2×10^{-9}
5	− 0.511041788036	0	0

Table 2.2 *Results from the Newton method*

Thus, the values for x_k become closer and closer to the same value. This means that we have found the approximate solution. In fact, the solution for this equation is $x_1 = -0.51104$ and its obtained after only three iterations.

2.7.2 The Lagrangian Method

Many applications of mathematical modeling involve the optimization of an objective function subject to certain constraining conditions, or more simply, constraints. These constraints represent restrictions that can influence the degree of which an objective function is optimized. Constraints may reflect such restrictions as limited resources (e.g, Labor, Materials, or Capital), In this section we will examine a method for solving certain nonlinear constrained optimization problems [18].

The Lagrange Multiplier Method (Equality Constraint):

Consider the constrained optimization problem.

Maximize (or Minimize)

$y=f(x1,X2)$

Subject tog $(x_1, x2) = k$ (2.1)

In this equation, f is the objective function and g $(x_1,x_2) = k$ is an equality constraint. One way of solving this type of problem is to combine the information in equation 1 into the composite function

$l.lX_i, x2, A) = J(x_1, x2) - A[g(xI, x2)-kl$ (2.2)

This composite function is called the lagrangian function, and the variable ;l (lambda) is referred to as the lagrange multiplier. The lagrangian function is composed of the objective function and a linear multiple of the constraint equation.

However in the lagrangian function ;l can equal any value and the term

;l $[g(xi, x2) - k]$ will equal 0,

Provided that (x_1, x_2) are values which satisfy the constraint. Thus, the value of the newly formed lagrangian function L will equal the value of the original objective function f. The creation of the lagrangian function ingeniously transforms the original constrained problem into an unconstrained problem, which can be solved by procedures very similar to those discussed in the previous sections. That is, to solve the original problem in equation 2.1, partial derivatives of $L(x_1, x_2, \lambda)$ are found with respect to x_1, x_2 and are then set equal to zero.

Now, considering the following nonlinear example:

Maximize $u(x_1, x_2) = x_1^{1/3} \cdot x_2^{1/3}$
Subject to: $p_1x_1 + p_2x_2 \leq I$.

To solve this problem, we assume an interior solution, $x_i > 0$ for all i.
Then we use the partial derivate for x_1, x_2 and λ with respect to function L,
After that, we substitute this into the constraint and solve for x_1 and x_2.
We formulate the Lagrangian function L as:
$L = x_1^{1/3} \cdot x_2^{1/3} + \lambda(I - p_1x_1 + p_2x_2)$
To solve this problem, partial derivatives of $L(x_1, x_2, \lambda)$ are found with respect to x_1, x_2 and λ, and then are set equal to zero.
The partial derivates are:

$$x_1 \frac{dL}{dx_1} = x_1 \left[\frac{1}{3}x_2^{1/3} \Big/ 3x_1^{2/3} - \lambda p_1 \right] = 0$$

$$x_2 \frac{dL}{dx_2} = x_2 \left[\frac{1}{3}x_1^{1/3} \Big/ 3x_2^{2/3} - \lambda p_2 \right] = 0$$

$$\lambda \, dL/d\lambda = \lambda(I - p_1x_1 + p_1x_1) = 0$$

Using the partial derivates to express x_2 in terms of x_1, and substitute this into the constraint and solving for x_1, then solving for x_2. Using this method, we get:

$$\frac{1}{3}x_2^{1/3} \Big/ 3x_1^{2/3} - \lambda p_1 = \frac{1}{3}x_1^{1/3} \Big/ 3x_2^{2/3} - \lambda p_1$$

Which can be rearranged to get: $p_1x_1 = p_2x_2$

We use this is the constraint to get:

$$P_1X_1 = \frac{I}{2}$$

We solve for x_1 to get:

$$x_1 = \frac{I}{2p_1}$$

and then after a substitution,

$$x_2 = \frac{I}{2p_2}$$

Thus, we transformed an optimization problem subject to constraints, into another optimization problem with no constraints.

As another example, considering the nonlinear problem

Maximize $f(x_1, x_2) = 25 - x_1^2 - x_2^2$

Subject to $2x_1 + x_2 = 4$

The lagrange Multiplier method transforms this problem into the unconstrained form

$$L(x_1, x_2, \lambda) = 25 - x_1^2 - x_2^2 - \lambda(2x_1 + x_2 - 4)$$

First Partial derivatives are identified as:

$$L_{x_1} = -2x_1 - 2\lambda$$

$$L_{x_2} = -2x_2 - \lambda$$

$$L_{\lambda} = -2x_1 - x_2 + 4$$

Critical values are found by setting three partial derivatives equal to zero and solve Simultaneously.

$$-2x_1 - 2\lambda = 0 \qquad (2.3)$$

$$-2x_2 - \lambda = 0 \qquad (2.4)$$

$$-2x_1 - x_2 + 4 = 0 \qquad (2.5)$$

Multiplying both sides of Equation (2.4) by- 2 gives $4x_2 + 2A = 0$

Adding this to Equation (2.3)

$$4x_I \quad + 2J = 0$$

$$-2x_J - 2A = 0$$

$$-2x_J + 4x_2 = 0 \quad (2.6)$$

Solving for x_I in Equation (2.6) yields

$$4x_2 = 2x_J$$

$$2x_2 = x_J \quad (2.7)$$

If this value for $x_;$ is substituted into Equation (2.5)

$$-2(2x_2) - x_2 + 4 = 0$$

$$-5x_2 = -4$$

$$x_2 = Ys = 0.8$$

If this value is substituted into Equation (2.7), $x_1=1.6$, also, substituting $x_I = 0.8$ into Equation (2.4) yields $A = -1.6$. Thus, $x_1=1.6$, $x_2 = 0.8$ and $A = -1.6$ are critical values on the lagrangian function. These values for x_I ,$x2$ represent the only candidate points for relative maximum (or minimum).

In the lagrangian method we can solve any nonlinearly objective function with constraints by transforming it into an objective function with no constraints. This has many disadvantages when the problem we are trying to optimize is not converging using normal methods.

2.8 Summary

In this chapter, we explained optimization and its methods as a methodology and as applied in fields of industry and networks, linear and integer programming were the focus of this chapter, where we introduced the graphical method and how it can be used to optimization problems of two or which can be reduced to two variables. Then we introduced the simplex method, and demonstrated the superiority of this method compared to the graphical method. Then we talked about duality, pointing on its advantages in reforming the optimization problem and solving a once unsolvable problem. Also integer programming was discussed and we explained why it's hard to solve using hand mathefnaticsand the need for special purpose methods like branch and bound.

Then we introduced non-linear programming and how it differs from linear and integer programming. In this section we introduced the Newton's method and how it can solve any mathematical problem using an iterative approach. Finally we introduced the lagrangian method or penalty function, and how it can be used to transform a constrained optimization problem into an unconstrained optimization problem.

3. NETWORK OPTIMIZATION ALGORITHMS

3.1 Overview

Various kinds of network optimization problems appear in many fields of work, including Telecommunication systems, commodity transportation, railroad and highway traffic planning, electrical power distribution, and much more. The fundamental question in network optimization is how to efficiently transport some entity (data packets, electrical power, vehicles, etc.) from one point to another in a network, given a number of limiting constraints, such as the capacity of the communication links of the network.

General optimization problems can be approached in many different ways, e.g. using linear programming, operations research theory, discrete simulation, and using algorithmic approaches from the computer science field.

This chapter introduces some of the fundamental concepts, algorithms and applications of network optimization theory, using an algorithms perspective. The focus is on computer networks, but the theories and algorithms discussed are applicable also in other domains.

3.2 Types Of Network Optimization

Many of the most important network optimization problems can be related to the following general problems:

- Shortest path problems: Given a connected network of nodes, How can we get from one point to another in a network using the shortest (or cheapest) path?
- Maximum flow problems: Given a connected network of nodes, How can we achieve as high flows as possible between two points in a network, given some link capacity restrictions?
- Minimum cost flow problems: Given a cost per unit flow on each link in a network, how can we assign flows to the links in the most cost effective way?

A large number of related problems can be derived from the above-mentioned general problems, including assignment problems, transpoıtation problems, circulation problems, convex cost flow problems, multi commodity flow problems, minimum spanning tree problems, and matching problems. It can be shown that virtually all network flow problems can be transformed into one another. Hence, a solution to one of the problems is a solution to all others, using a suitable transform.

Specifically, the shortest path problem and the maximum flow problem can easily be stated as special cases of the minimum cost flow problem (MCFP). Therefore, the MCFP is considered the most fundamental network flow problem, and the design of algorithms for network flow problems are mainly targeting the MCFP.

3.3 Complexity Of An Algorithm.

The efficiency of network optimization algorithms is usually measured using big O, big U, and big el notation [38]:

• **Big** O - **notation**: An algorithm is said to execute in O(f(n)) time if for some numbers c and n_0 the time taken by the algorithm is at most cf(n) for all n = n_0. This is theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n, which is usually the number of items. Informally, saying some equation f(n) = O(g(n)) means it is less than some constant multiple of g(n). The notation is read, "f of n is big oh of g of n". the formal definition would be: f(n) = O(g(n)) means there are positive constants c and k, such that 0::;:; f(n)::;:; cg(n) for all n 2: k. The values of c and k must be fixed for the function f and must not depend on n.

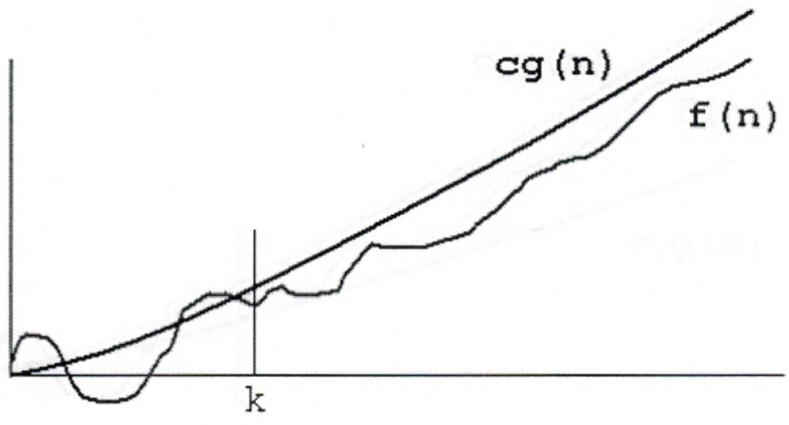


Figure 3.1 *Big Omega notation*

As an example, $n^2 + 3n + 4$ is $O(n^2)$, since $n^2 + 3n + 4 < 2n^2$ for all $n > 10$. Strictly speaking, $3n + 4$ is $O(n^2)$, too, but big-O notation is often misused to mean equal to rather than less than. The notion of "equal to" is expressed by O(n).

• **Big Q - notation:** An algorithm is said to execute in . Q (f(n)) time if for some numbers k and n_0 the time taken by the algorithm on some problem instance is at least kf(n) for all $n = n_0$. This is a theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n, which is usually the number of items. Informally, saying some equation f(n) = Q (g(n)) means it is more than some constant multiple of g(n). f(n) = Q (g(n)) means there are positive constants c and k, such that 0 s cg(n) ≤ f(n) for all n ≥ k. The values of c and k must be fixed for the function f and must not depend on n.

• **Big e - notation:** An algorithm is said to be e (f(n)) if the algorithm is both O(f(n)) and .(f(n)). Thus, O is an upper bound on algorithm complexity, Q is a lower bound, and 8 is both an upper and a lower bound. f(n) = e (g(n)) means there are positive constants c_1, c_2, and k, such that 0 ~ c_1g(n) ≤ f(n) ≤ c_2g(n) for all n ~ k. The values of c_1, c_2, and k must be fixed for the function f and must not depend on n.

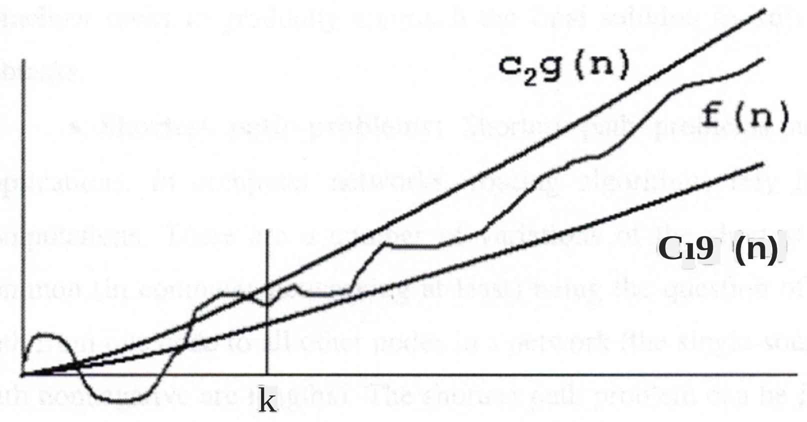


Figure 3.2 *Big Theta notation*

3.4 Algorithms Used In Network Optimization

The goal when designing network optimization algorithms is to arrive at an algorithm that solves the optimization problem in an efficient way, i.e. in polynomial time. Several approaches exists [39]:

• **Geometric improvement techniques:** An algorithm runs in polynomial time if at every iteration it makes an improvement in the objective function value that is proportional to the difference between the objective function values of the current (intermediary) solution and the optimal solution. That is, if the algorithm makes a significant contribution to the solution for each iteration, it will be efficient.

• **Dynamic programming:** The dynamic programming strategy decomposes the problem into stages and uses a recursive relationship to go from one stage to another.

• **Scaling:** The scaling approach solves a sequence of simpler approximate versions of a given problem determined by scaling the problem data (for instance using bit scaling by increasing the precision one bit) in such a way that the approximations gradually approach the solution of the final problem. A common trait of most algorithm design techniques for network optimization is an iterative divide-and-Conquer approach that somehow seeks to gradually approach the final solution by solving smaller intermediary subtasks.

• **Shortest path problems:** Shortest path problems arise frequently in many applications. In computer networks, routing algorithms rely heavily on shortest path computations. There are a number of variations of the shortest path problem, the most common (in computer networking at least) being the question of how to find the shortest path from one node to all other nodes in a network (the single-source shortest path problem with nonnegative arc lengths). The shortest path problem can be formulated as a minimum cost flow problem by modifying b(i) in the MCFP equation so that $b(s) = n - 1$, for the source node s, and $b(i) = -1$ for all other nodes in the network. Shortest path algorithms can be classified into two classes: label-setting algorithms and label-correcting algorithms. Both progress iteratively assigning labels to nodes. The label of a node represents the upper bound on the shortest path from the source to the node.

Label-setting algorithms designate one label as permanent (i.e. optimal) for each iteration of the algorithm, whereas label-correcting algorithms assign temporal labels that become permanent at the last step of the algorithm.

Label-setting algorithms are only applicable for a cyclic networks with arbitrary arc length and for networks with nonnegative arc lengths. Label-correcting algorithms work for all types of networks. Since computer networks generally have nonnegative link costs, label-setting algorithms are typically preferred, since the complexity of label-correcting algorithms is higher. (Label-correcting algorithms are NP-complete.)

• **Maximum flow:** The maximum flow problem is concerned with finding the maximum flow between a source node and a Sink node, without exceeding the arc capacities of the network. The shortest path problem and the maximum flow problem are complementary and capture different aspects of the minimum cost flow problem: The shortest path problem involves arc costs but not arc capacities, whereas the maximum flow problem involves arc capacities but not arc costs.

• **Minimum cost flow algorithms:** The minimum cost flow problem, Is a generalization of the shortest Path and maximum flow problems. Many of the algorithms developed for the minimum cost flow problem Combine techniques from both shortest path and maximum flow algorithms. Since the MCFP considers both arc capacities and arc costs it is harder to solve than the shortest path and maximum flow problems.

3.4.1 Shortest Path Algorithms

Another special class of transshipment models is the shortest path problem, where the objective is to find the shortest path of arcs from one node to another node in the network. As an example, the nodes may represent cities, the arcs may represent roads between some city pairs, and the arc costs can equal the the distance (in miles) between the nodes. We wish to find the shortest route from city A to city B[42]. This is the so-called 1-to-I shortest path problem. A related problem is the l-to-all problem of finding the shortest path from one node to all other nodes in the network. This can be accomplished by placing a demand of 1 at all destination nodes, and a supply equal to the total demand at the single origin node.

Such models are found in power distribution networks, vehicle routing, investment analysis, and aircraft-route planning applications. PC-based trip-planning software already solves shortest path problems to identify the most direct travel routes. Under development are traffic-information-broadcast systems to support on-board displays of maps and suggested courses of travel for trucks and automobiles. So it is reasonable to expect that in the near future, new cars will be solving shortest-path problems.
In this section we will discuss the shortest path problem and its algorithmic solutions

Consider the example discussed in the figure 3.3 where a motorist wishes to find the shortest possible route from location A to location B. Given a road map, on which the distance between each pair of adjacent intersections is marked, how can we determine this shortest route?

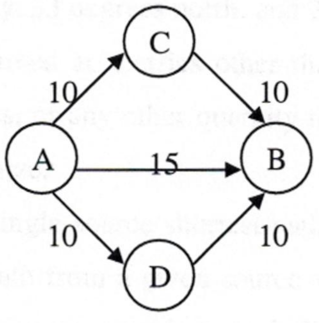


Figure 3.3 *The motorist example and shortest path problems*

One possible way is to enumerate all the routes from location A to the location B, add up the distances on each route, and select the shortest. It is easy to see, however, that even if routes that contain cycles are disallowed, there are more than one possibility, most of which are simply not worth considering. For example, a route from location A to location C to location B is obviously a poor choice, because location C will increase the distance of the way.

In a shortest-paths problem, we are given a weighted, directed graph $G = (V,E)$, with weight function $w : E \rightarrow R$ mapping edges to real-valued weights. The weight of path $p = (v_0, v_1, \ldots, v_k)$ is the sum of the weights of its constituent edges:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) .$$

We define the shortest-path weight from u to v by

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \leadsto v\} & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise.} \end{cases}$$

A shortest path from vertex u to vertex v is then defined as any path p with weight $w(p) = \delta(u, v)$. In the previous example, we can model the road map as a graph: vertices represent intersections, edges represent road segments between intersections, and edge weights represent road distances. Our goal is to find a shortest path from a given intersection in A location (say, 33 degrees north. and 9 degrees west.) to a given intersection in location B (say, 33 degrees north. and 29 degrees west).

Edge weights can be interpreted as metrics other than distances. They are often used to represent time, cost, penalties, or any other quantity that accumulates linearly along a path and that one wishes to minimize.

The focus is on the single-source shortest-paths problem: given a graph $G = (V,E)$, we want to find a shortest path from a given source vertex $s \in V$ to every vertex $v \in V$. The algorithm for the single-source problem, including the following variants, can solve many other problems.

• Single-destination shortest-paths problem: Find a shortest path to a given destination vertex t from every vertex v. By reversing the direction of each edge in the graph,~e can reduce this problem to a single-source problem.

• Single-pair shortest-path problem: Find a shortest path from u to v for given vertices u and v. If we solve the single-source problem with source vertex u, we solve this problem also. Moreover, no algorithms for this problem are known that run asymptotically faster than the best single-source algorithms in the worst case.

• All-pairs shortest-paths problem: Find a shortest path from u to v for every pair of vertices u and v. Running a single-source algorithm once from each vertex can solve this problem; but it can usually be solved faster, and its structure is another topic in its own right.

3.4.2 Negative-Weight Edges

In some instances of the single-source shortest-paths problem, there may be edges whose weights are negative. If the graph $G = (V,E)$ contains no negative-weight cycles reachable from the sources, then for all $v \in V$, the shortest-path weight $\ddot{o}(s,v)$ remains well defined, even if it has a negative value. If there is a negative-weight cycle reachable from s, however, shortest-path weights are not well defined. No path from s to a vertex on the cycle can be a shortest path, a lesser-weight path can always be found that follows the proposed "shortest" path and then traverses the negative-weight cycle. If there is a negative-weight cycle on some path from *s* to *v*, we define *ö (s, v)* = - oo .

Some shortest path algorithms, such as dijkstra' s algorithm, assume that all edge weights in the input graph are nonnegative, as in the road map example. Others, such as Bellman-Ford algorithm, allow negative weight edges in the input graph and produce a correct answer as long as no negative weight cycles are reachable from the source.

3.4.3 Shortest Paths And Relaxation

The single-source shortest-path algorithms are all based on a technique known as relaxation. To understand single-source shortest-paths algorithms, it is helpful to understand the techniques that they use and the properties of shortest paths that they exploit. The main technique used by shortest path algorithms is relaxation, a method that repeatedly decreases an upper bound on the actual shortest-path weight of each vertex until the upper bound equals the shortest-path weight [43]. In this section, we shall see how relaxation works and formally prove several properties it maintains.

Relaxation works for each vertex *v* ∈ *V*, we maintain an attribute *d[v]*, which is an upper bound on the weight of a shortest path from source *s* to *v*. We call *d[v]*, a shortest-path estimate. We initialize the shortest-path estimates and predecessors by the following procedure.

```
INITIALIZE-SINGLE-SOURCE( G, s)
1. for each vertex v E  V[G]
2.    do d[v] ~ oo
3.       JZ' [v] ~ NIL
4. d[s] ~ 0
```

After initialization, JZ' [v] = NIL for all *v* E *V*, *d[v]* = 0 for *v* = *s*, and *d[v]* = oo for v e V-{s}.

The process of relaxing an edge (u, v) consists of testing whether we can improve the shortest path to *v* found so far by going through *u* and, if so, updating *d[v]* and π[v]. A relaxation step may decrease the value of the shortest-path estimate *d[v]* and update *v's* predecessor field π[v]. The following code performs a relaxation step on edge (u, *v*).

```
RELAX(U, V, W)
1. if d[v] > d[u] + w(u,v)
2.     then d[v] ← d[u] + w(u,v)
3.          π[v] ← u
```

Figure 3.4. shows two examples of relaxing an edge, one in which a shortest-path estimate decreases (a), and one in which no estimate changes (b).

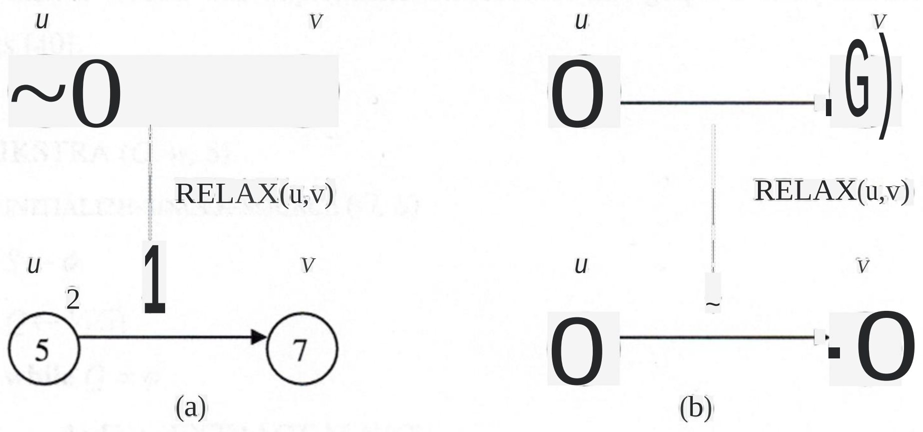


Figure 3.4 *Relaxation and shortest path estimates*

In Figure 3.4 Relaxation of an edge (*u*, v). The shortest-path estimate of each vertex is shown within the vertex. (a) Because *d[v]* > *d[u]* + *w(u,* v) prior to relaxation, the value of *d[v]* decreases, (b) Here, *d[v]* ≤ *d[u]* + *w(u,v)* before the relaxation step, so *d[v]* is unchanged by relaxation.

3.5. Dijkstra's Algorithm

Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted directed graph G = *(V,E)* for the case in which all edge weights are nonnegative. In this section therefore we assume that *w(u, v)* ~ 0 for each edge *(u, v)* E *E*.

Dijkstra's algorithm maintains a set Sof vertices whose final shortest-path weights from the source s have already been determined. That is, for all vertices *u* E *V- S*, we have *d[v]* = ö (s, v).The algorithm repeatedly selects the vertex *u* E *V- S* with the minimum shortest-path estimate, inserts *u* into *S* and relaxes all edges leaving *u*. In the following implementation we maintain a priority queue *Q* that contains all the vertices in *V - S*, keyed by their *d* values. The implementation assumes that graph *G* is represented by adjacency lists [40].

```
DIJKSTRA (G, w, S)
1. INITIALIZE-SINGLE-SOURCE (G, S)
2. Sf- </J
3. Qf-V[G]
4. while Q :t </J
5.      do U f-EXTRACT-MIN(Q)
6.           Sf-SU{u}
7.           for each vertex vE Adj[u]
8.                do RELAX (u, v, w)
```

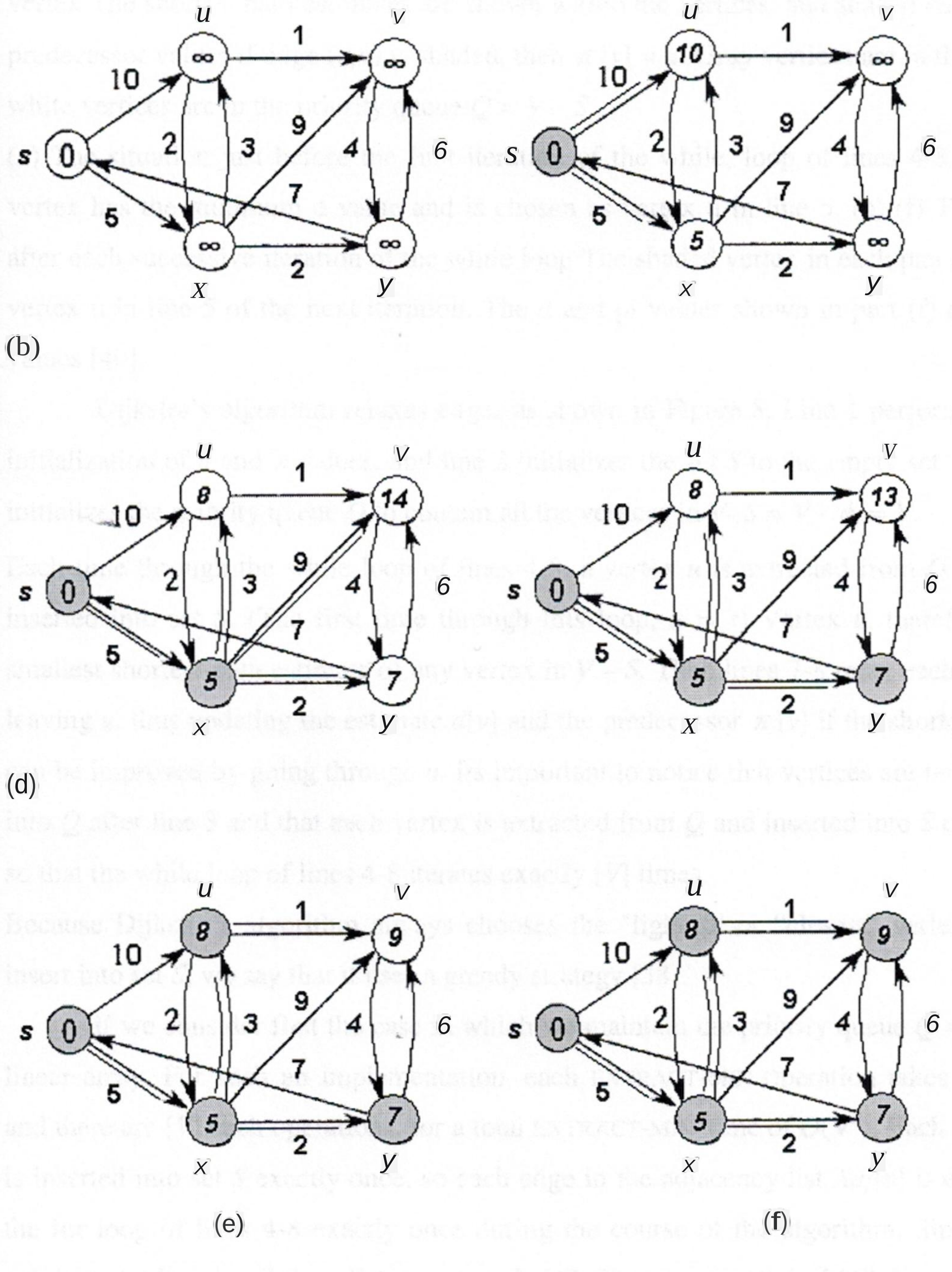


Figure 3.5. *The execution of Dijikstra 's algorithm*

In Figure 3.5 The execution of Dijkstra's algorithm. The source is the leftmost vertex The shortest path estimates are shown within the vertices, and shaded edges indicate predecessor values if edge (u, v) is shaded, then $\pi[v] = u$. Gray vertices are in the set S, and white vertices are in the priority queue $Q = V - S$.

(a) The situation just before the first iteration of the while, loop of lines 4-8 The shared vertex has the minimum d value and is chosen as vertex u in line 5. (b)-(f) The situation after each successive iteration of the while loop The shaded vertex in each pan is chosen as vertex u in line 5 of the next iteration. The d and pi values shown in part (f) are the final values [40].

Dijkstra's algorithm relaxes edges as shown in Figure 8. Line 1 performs the usual initialization of d and π values, and line 2 initializes the set S to the empty set. Line 3 then initializes the priority queue Q to contain all the vertices in $V - S = V - \emptyset = V$.

Each time through the while loop of lines 4-8, a vertex u is extracted from $Q = V - S$ and inserted into set S. (The first time through this loop, $u = s$) Vertex u, therefore has the smallest shortest-path estimate of any vertex in $V - S$. Then lines 7-8 relax each edge (u, v) leaving u, thus updating the estimate $d[v]$ and the predecessor $\pi[v]$ if the shortest path to v can be improved by going through u. Its important to notice that vertices are never inserted into Q after line 3 and that each vertex is extracted from Q and inserted into S exactly once so that the while loop of lines 4-8 iterates exactly [VJ times.

Because Dijkstra's algorithm always chooses the "lightest' or "closest" vertex in V-S to insert into set S, we say that it uses a greedy strategy [38].

If we consider first the case in which we maintain the priority queue $Q = V - S$ as a linear array. For such an implementation, each EXTRACT-MIN operation takes time O(V), and there are [VJ such operations, for a total EXTRACT-MIN time of $O(V^2)$. Each vertex $v \in V$ is inserted into set S exactly once, so each edge in the adjacency list $Adj[v]$ is examined in the for loop of lines 4-8 exactly once during the course of the algorithm. Since the total number of edges in all the adjacency lists is [E]. There are a total of [E]. Iterations of this for loop, with each iteration taking O(1) time. The running time of the entire algorithm is thus $O(V^2 + E) = O(V^2)$ [42].

3.6. Maximum Flow Algorithms

Just as we can model a road map as a directed graph in order to find the shortest path from one point to another, we can also interpret a directed graph as a "flow network" and use it to answer questions about material flows. It can be a material coursing through a system from a source, where the material is produced, to a sink, where it is consumed The source produces the material at some steady rate and the sink consumes the material at the same rate The "flow" of the material at any point in the system is intuitively the rate at which the material moves. Flow networks can be used to model liquids flowing through pipes, parts through assembly lines, current through electrical networks, information through communication networks, and so forth [39].

–Each directed edge in a flow network can be thought of as a conduit for the material. Each conduit has a stated capacity, given as a maximum rate at which the material can flow through the conduit, such as 200 gallons of liquid per hour through a pipe or 20 amperes of electrical current through a wire. Vertices are conduit junctions, and other than the source and sink, material flows through the vertices without collecting in them. In other words, the rate at which a material enters a vertex must equal the rate at which it leaves the vertex. We call this property "flow conservation," and it is the same as Kirchhoff's Current Law when the material is electrical current [40].

The maximum-flow problem is the simplest problem concerning flow networks It asks, What is the greatest rate at which material can be shipped from the source to the sink without violating any capacity constraints? As we shall see in this section, efficient algorithms can solve this problem. Moreover, the basic techniques used by these algorithms can be adapted to solve other network-flow problems.

3.6.1 Flow Networks

In this section, we give a graph-theoretic definition of flow networks, discuss their properties, and define the maximum-flow problem precisely. We also introduce some helpful notation.

A flow network G = *(V, E)* is a directed graph in which each edge *[u, v)* E *E* has a nonnegative capacity *c(u, v)* ~ 0. If *(u, v)*~ *E,* we assume that *c(u, v)* = 0. We distinguish two vertices in a flow network: a source *s* and a sink *t*. For convenience, we assume that every vertex lies on some path from the source to the sink. That is, for every vertex *v* E *V,* there is a path *s* ~ *v* ~ *t*. The graph is therefore connected, and |E| ~ |V| - 1. Figure 9 shows-an example of a flow network.

Let G = *(V,E)* be a flow network (with an implied capacity function c). Lets be the source of the network, and let *t* be the sink. A flow in G is a real-valued function *f:* *V* x *V*~ *R* that satisfies the following three properties:

• Capacity constraint: For all *u, v* E *V,* we require *f(u, v)* ~ *c(u, v).*

• Skew symmetry: For all *u, v* EV, we requiref(u, *v)* = - *f(v,u).*

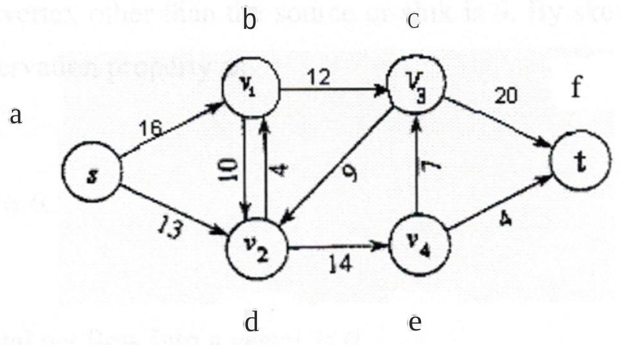
• Flow conservation: For all uE *V'* - *{s, t},* we require

$$\underset{veV}{L} \quad f\,(u, v) = 0.$$

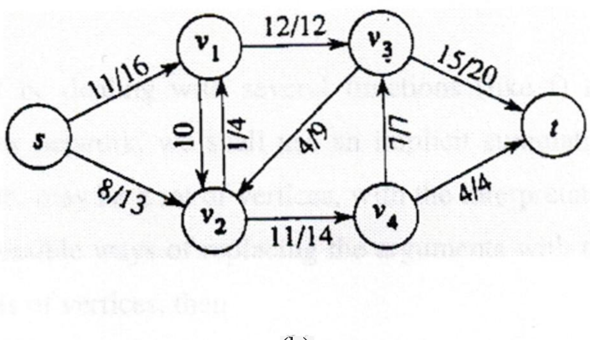
The quantity *f(u, v),* which can be positive or negative, is called the net flow from vertex *u* to vertex *v*. The value of a flow is defined as

$$r11 = \underset{vEV}{L} \quad f(s,v)$$

that is, the total net flow out of the source. In the maximum-flow problem, we are given a flow network G with source *s* and sink *t,* and we wish to find a flow of maximum value from *s* to *t.*



(a)



(b)

Figure 3.6 *Flow networks, Sources and Sinks demonstration*

In Figure 3.6 (a) A flow network G ≡ *(V, E)*. The a node is the sources, and the f node is the sink *t*. units are shipped through intermediate nodes, but only *c(u, v)* units per session can go from node *u* to node *v*. Each edge is labeled with its capacity, (b) A flow *fin* G with value [*f*]= 19. Only positive net flows are shown. If *j(u, v)* > *0,* edge *(u, v)* is labeled by *ff u, v)* / *c(u, v)*. (The slash notation is used merely to separate the flow and capacity; it does not indicate division.) .

If *f(u, v)* ~ *0,* edge *(u, v)* is labeled only by its capacity.

The net flow from a vertex to itself is 0, since for all $u \in V$, we have $f(u, u) = -f(u, u)$, which implies that $f(u, u) = 0$. The flow-conservation properly says that the total net flow out of a vertex other than the source or sink is 0. By skew symmetry, we can rewrite the flow-conservation property as

$$\sum_{u \in V} f(u, v) = 0.$$

That is, the-total net flow into a vertex is 0.

3.6.2 Working With Flows

We shall be dealing with several functions (like f) that take as arguments two vertices in a flow network, we shall use an implicit summation notation in which either argument, or both, may be a set of vertices, with the interpretation that the value denoted is the sum of all possible ways of replacing the arguments with their members. For example, if X and Y are sets of vertices, then

$$f(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y).$$

The flow-conservation constraint can be expressed as the condition that $f(u, v) = 0$ for all $u \in V - \{s, t\}$. Also, for convenience, we shall typically omit set braces when they would otherwise be used in the implicit summation notation [39], [38].

3.6.3 The Ford-Fulkerson Method

This section presents the Ford-Fulkerson method for solving the maximum-flow problem. We call it a "method" rather than an "algorithm" because it encompasses several implementations with differing running times The Ford-Fulkerson method depends on three important ideas that transcend the method and are relevant to many flow algorithms and problems. residual networks, augmenting paths, and cuts. These ideas are essential to the important max-flow min-cut theorem which characterizes the value of a maximum flow in terms of cuts of the flow network. We end this section by presenting one specific implementation of the Ford-Fulkerson method and analyzing its running time

The Ford-Fulkerson method is iterative. We statt with $f(u, v) = 0$ for all $u,v \in V$, giving an initial flow of value 0. At each iteration, we increase the flow value by finding an "augmenting path," which can be thought of simply as a path from the source s to the sink t along which we can push more flow, and then augmenting the flow along this path We repeat this process until no augmenting path can be found. The max-flow min-cut theorem will show that upon termination, this process yields a maximum flow [38].

```
FORD-FULKERSON-METHOD(G, s, t)
1. Initialize flow J to 0
2. While there exists an augmenting path p
3.     Do augment flow f along p
4. Return!
```

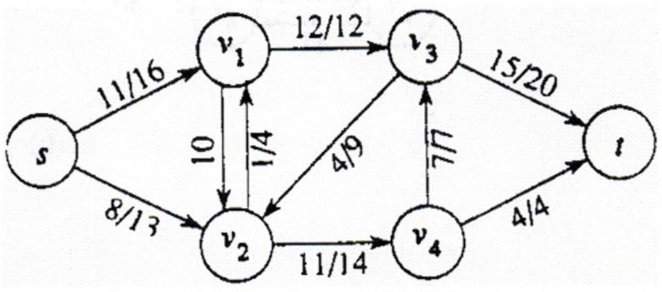
3.6.4 Residual Networks

Intuitively, given a flow network and a flow, the residual network consists of edges that can admit more net flow. More formally, suppose that we have a flow network $G = (V,E)$ with sources and sink t Letfbe a flow in G, and consider a pair of vertices $u, v \in V$. The amount of additional net flow we can push from u to v before exceeding the capacity $c(u,v)$ is the residual capacity of (u,v), given by

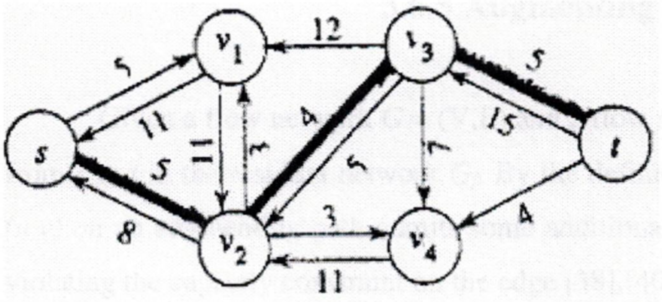
$c_f(u,v) = c(u,v)-f(u,v)$.

For example, if $c(u,v)= 16$ and $f(u,v) = 11$, then we can ship $c_f(u,v) = 5$ more units of flow before we exceed the capacity constraint on edge (u,v) When the net flow $f(u,v)$ is negative, the residual capacity $c_f(u, v)$ is greater than the capacity $c(u,v)$

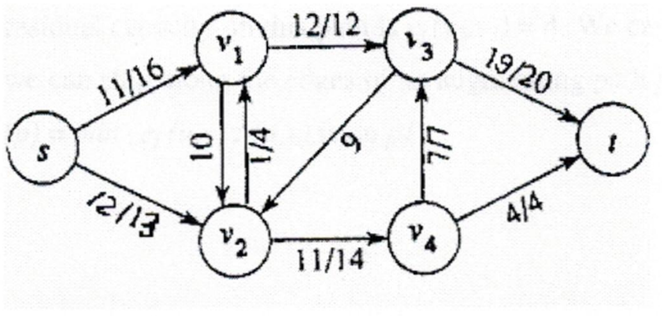
In Figure 3.7 (a) The flow network *G* and flow f of Figure 2.7.(b) The residual network G_f with augmenting path *p* shaded its residual capacity is $C_f(P) = c(v2, v3) = 4$ (c) The flow in *G* that results from augmenting along path *p* by its residual capacity 4. (d) The residual network induced by the flow in (c)



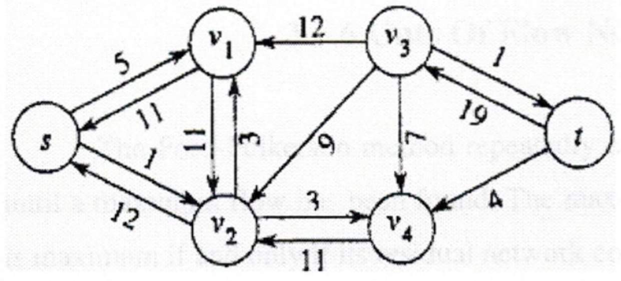
(a)



(b)



(c)



(d)

Figure 3.7. *Residual network demonstration*

3.6.5 Augmenting Paths

Given a flow network $G = (V,E)$ and a flow f, an augmenting path p is a simple path from s to t in the residual network G_f. By the definition of the residual network, each edge (u, v) on an augmenting path admits some additional positive net flow from u to v without violating the capacity constraint on the edge [38],[40].

The shaded path in Figure 2.7(b) is an augmenting path Treating the residual network G_f in the figure as a flow network, we can ship up to 4 units of additional net flow through each edge of this path without violating a capacity constraint, since the smallest residual capacity on this path is $c_f(v_2, v_3) = 4$. We call the maximum amount of net flow that we can ship along the edges of an augmenting path p the residual capacity of p, given by $c_f(p) = \min \{c_f(u,v) : (u,v) \text{ is on } p\}$.

3.5.6 Cuts Of Flow Networks

The Ford-Fulkerson method repeatedly augments the flow along augmenting paths until a maximum flow has been found. The max-flow min-cut theorem, tells us that a flow is maximum if and only if its residual network contains no augmenting path.

A cut (S, *T)* of flow network G = *(V, E)* is a partition of *V* into *S* and *T* = *V - S* such that SES and *ie T.* Figure 2.8 shows the cut *({S,v1,v2}, {v3,v4,t})* in the flow network of Figure 9.(b). The net flow across this cut is *f(v1,v2)* + *f(v2y3)* + *f(,v3, v4)* = 12+(-4)+11= 19, and its capacity is *c(v1,, v3) +c(v2, vs)* = 12+14 = 26. [40]

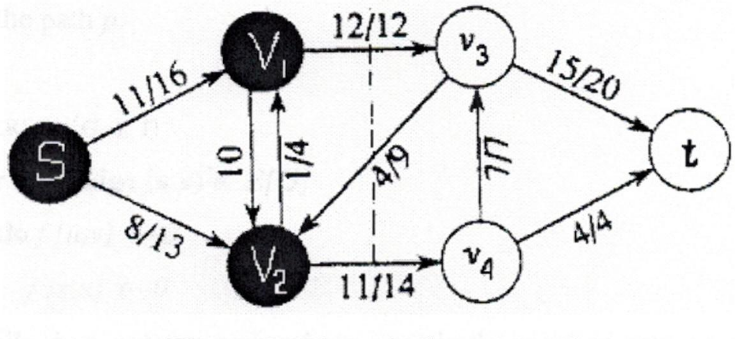


Figure 3.8 *A cut inflow networks*

Figure 3.8 A cut *(S, T)* in the flow network of Figure 9(b). where *S* = {S, *v1.*v2} and *T* = *[vs.vş.t}* The vertices in Sare black, and the vertices in *Tare* white. The net flow across *(S,T) isf(S,T)* = 19, and the capacity is *c(S,T)* = 26.

The Max-flow min-cut theorem:

If f is a flow in a flow network G = *(V,E)* with source *s* and sink *t,* then the following conditions are equivalent:

1. *f* is a maximum flow in G.
2. The residual network G1contains no augmenting paths.
3. If |= *c(S, T)* for some cut *(S, T)* of G.

3.6.7 The Basic Ford-Fulkerson Algorithm

In each iteration of the Ford-Fulkerson method, we find any augmenting path *p* and augment flow *f* along *p* by the residual capacity *cj(p)*. The following implementation of the method computes the maximum flow in a graph G = *(V,E)* by updating the net flow *f [u,v}* between each pair *u, v* of vertices that are connected by an edge. If *u* and *v* are not connected by an edge in either direction, we assume implicitly *thatf[u, v}* = *0* The code assumes that the capacity from *u* to vis provided by a constant-time function *c̦iu,v)*, with *c(u,v)* = *0* if *(u, v)* e *E*. The residual capacity *c1(u, v)* is computed in accordance with previous formulas. The expression *cj(p)* in the code is actually just a temporary variable that stores the residual capacity of the path *p*.

```
FORD-FULKERSON(G, s, t)
    1.for each edge (u, v) ∈ E[GJ
    2.  dof [u,v} f-0
    3.     f [v,u} f-O
    4.while there exists a path p from s tot in the residual network Gı
    5.     do ct(p) f- min {c1(u, v) : (u, v) is in p}
    6.           for each edge (u,v) in p
    7.                  dof [u,v} f-j [u,v} + cjip)
    8.                      f [v,u] f-j [u, v}
```

The FORD-FULKERSON algorithm simply expands on the FORD-FULKERSON-METHOD pseudocode given earlier. Figure 3.9 shows the result of each iteration in a sample run. Lines 1-3 initialize the flow *f* to 0. The while loop of lines 4-8 repeatedly finds an augmenting path p in *Gı* and augments flow *f* along *p* by the residual capacity *cfp)*. When no augmenting paths exist, the flow *f* is a maximum flow.

The running time of FORD-FULKERSON depends on how the augmenting path p in line 4 is determined. If n is chosen poorly, the algorithm might not even terminate, the value of the flow will increase with successive augmentations, but it need not even converge to the maximum flow value.

In Figure 3.9 The execution of the basic Ford-Fulkerson algorithm is shown, (a)-(d) Successive iterations of the while loop The next side of each part(a.1, b.1, c.l, d.l) shows the residual network *Gƒ* from line 4 with a shaded augmenting path *p* The first side of each part (a,b,c,d) shows the new flow f that results from adding j,- *to f* .The residual network in (a) is the input network G .(e) The residual network at the last while loop test It has no augmenting paths, and the flow *f* shown in (d) is therefore a maximum flow.

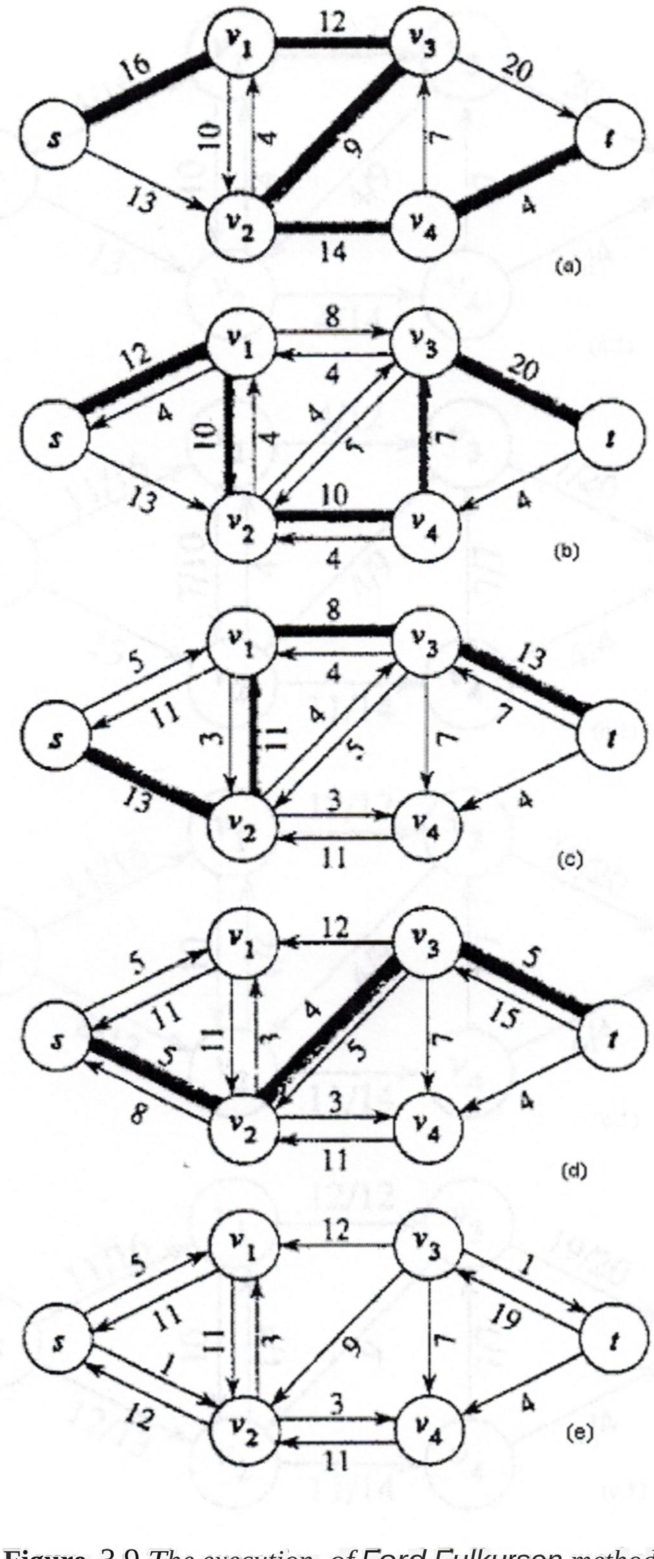


Figure 3.9 *The execution of Ford-Fulkurson method*

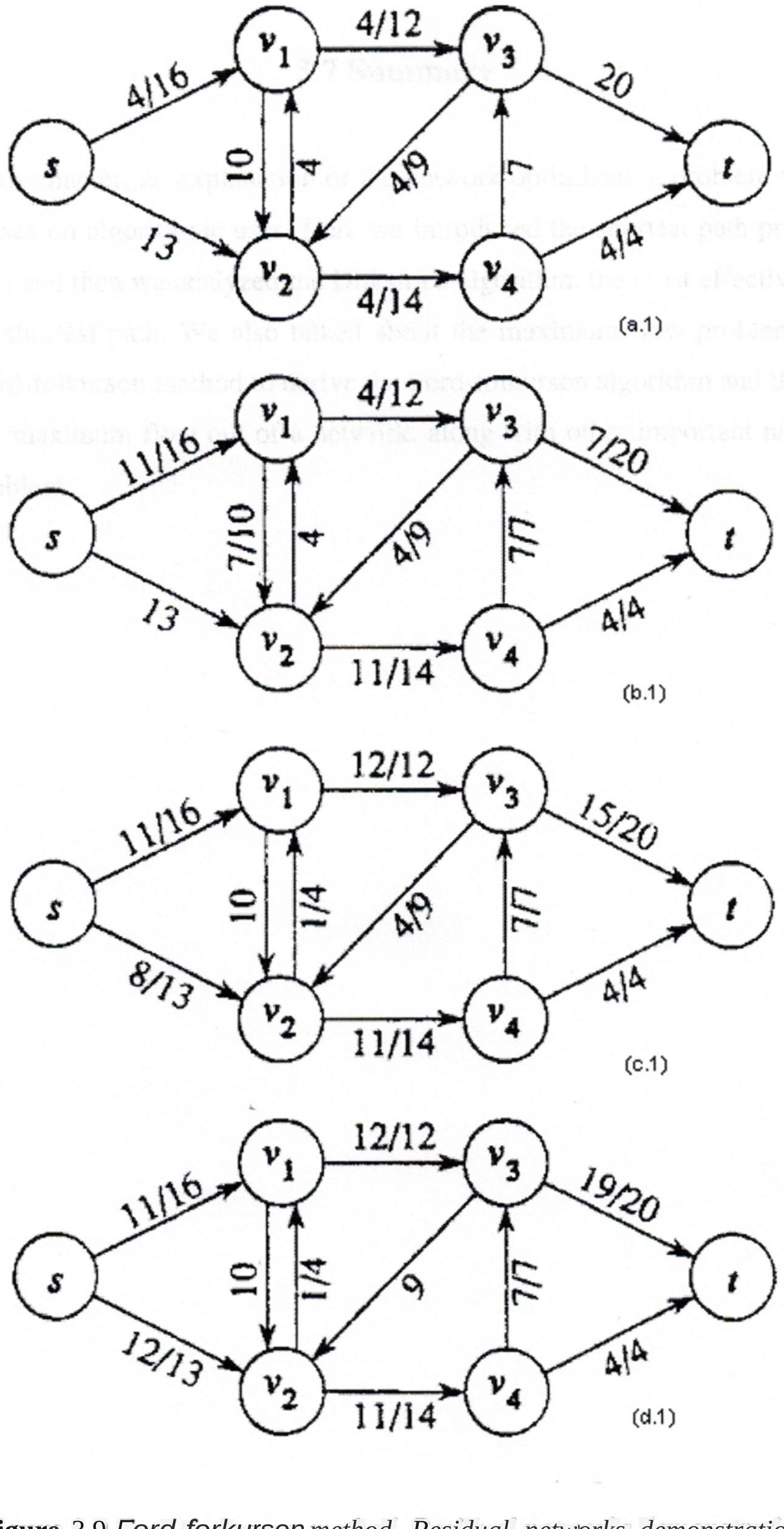


Figure 3.9 *Ford-forkurson method, Residual networks demonstration*

3.7 Summary

In this chapter, an explanation of the network optimization problem was discussed, with emphases on algorithmic uses. Here we introduced the shortest path problem with its various parts and then we analyzed the Dijkstra's algorithm, the most effective algorithm in finding the shortest path. We also talked about the maximum flow problem and how we used the Ford-folkurson method to derive the Ford-folkurson algorithm and then how to use it to get the maximum flow out of a network, along with other important notations in any network problem.

4. USING OPTIMIZATION TECHNIQUES TO IMPROVE PERFORMANCE ON IP NETWORKS

4.1 Overview

Computer networks exist to transfer information from one computer to another, or from source nodes to destination nodes. Accordingly, one of the most significant functions performed by the computer networks is the routing of traffic from source nodes to destination nodes. For this reason one of the most distinctive functions performed by network optimization is the control and optimization of the routing function, to steer traffic throughj he network in the most effective way.

Computer networks optimization or traffic engineering is defined as that aspect of Internet network engineering dealing with the issue of performance evaluation and performance optimization of operational computer networks. Traffic engineering encompasses the application of technology and scientific principles to the measurement, characterization, modeling, and control of computer traffic [44].

Enhancing the performance of an operational network, at both the traffic and resource levels, are major objectives of traffic engineering. This is accomplished by addressing traffic oriented performance requirements, while utilizing network resources economically and reliably. Traffic oriented performance measures include delay, delay variation, packet loss, and throughput.

4.2 Linear Integer Programming And Network Optimization

Linear programming is a powerful approach to any optimization problem with a linear objective function and linear constraints. In essence, linear programming solves an optimization problem by solving a system of linear equations. The most pervasive and powerful method for solving linear programming problems is the simplex method.

Since the minimum cost flow problem can be stated as a linear programming problem, the simplex method can be used to solve shortest path, maximum flow, and minimum cost flow problems. However, because of the special structure of network flow problems, the general simplex method without modifications to exploit the underlying network structure of the problem is not a competitive approach [47].

Many of the theoretical results of network flow theory have their counterparts in linear programming theory. For instance, the max-flow min-cut theorem is a special case of the strong duality theorem, which states that any linear minimization problem can be formulated as an equivalent maximization problem and vice versa.

The minimum cost flow problem can be then, stated as a linear program as follows:

Minimize ex

Subject to

$Nx = b, \ 0 \leq x \leq u.$

Here, N is an m x n matrix called the node-arc incidence matrix and the linear equation system Nx = b is The mass balance equations, representing the inflow and outflow of each node in the network.

Network optimization is concerned with how to efficiently transport some entity from one point to another in a network, given a number of limiting constraints, such as the capacities and costs of the communication links of the network. We have seen that the minimum cost flow problem is a generalization of the shortest path problem and the maximum flow problem, that considers both link capacities and link costs. Since the objective function and the capacity and cost constraints are usually represented by linear relationships, network optimization can be seen as a special case of general discrete linear optimization, and can be solved by linear programming. Due to the strong duality theorem of linear programming, any minimization problem can be converted into an equivalent maximization problem [10], [21].

Integer programming is another powerful approach used in computer networks, given a certain network it is not logical to state the amount of data transmission to be 345.5

bytes. Similar research concluded the same results, [6], [7], [8]. It is better to state wither the link is operational or not.

4.3 Quality Of Service And Traffic Engineering

An important proposed optimized routing capability is the quality of service (QoS) support. Two mechanisms provide a range of QoS to packets passing through a router or a tag switch:

• Classification of packets into different classes.

• Handling of packets via appropriate QoS characteristics (such as bandwidth and loss).

–Optimized routing provides an easy way to mark packets as belonging to a particular class after they have been classified the first time. Initial classification uses information carried in the network layer or higher-layer headers. A label corresponding to the resultant class then would be applied to the packet. Labeled packets could be handled efficiently, by label switched routers (LSRs) in their path without needing to be reclassified.

The actual packet scheduling and queuing is largely orthogonal: The key point here is that optimized routing enables simple logic to be used to find the state that identifies how the packet should be scheduled. The exact use of optimized routing for QoS purposes depends a great deal on how QoS is deployed (24],[48]. The most common optimized routing protocol using in large computer networks as the Internet is MPLS.

One of the fundamental properties of destination-based routing is that the only information from a packet that is used to forward the packet is the destination address. Although this property enables highly scalable routing, it also limits the capability to influence the actual paths taken by packets. This limits the capability to evenly distribute traffic among multiple links, taking the load off highly utilized links and shifting it toward less-utilized links (48].

4.4 Congestion On Computer Networks.

Packets contend for the use of network resources as they are conveyed through the network. A network resource is considered to be congested if the arrival rate of packets, exceed the output capacity of the resource over an interval of time. Congestion may result in some of the arrival packets being delayed or even dropped.

Congestion increases transit delays, delay variation, packet loss, and reduces the predictability of network services. Clearly, congestion is a highly undesirable phenomenon. Combating congestion at a reasonable cost is a major objective of traffic engineering [44].

Efficient sharing of network resources by multiple traffic streams is a basic economic premise for packet switched networks in general and for the Internet in particular. A fundamental challenge in network operation, especially in a large scale public IP network, is to increase the efficiency of resource utilization while minimizing the possibility of congestion.

In practice, the delivery requirements of a specific set of packets may be specified explicitly or implicitly. Two of the most important traffic delivery requirements are capacity constraints and QoS constraints.

Capacity constraints can be expressed statistically as peak rates, mean rates, burst sizes, or as some deterministic notion of effective bandwidth.

QoS requirements can be expressed in terms of :

I .Integrity constraints such as packet loss

2. In terms of temporal constraints such as timing restrictions for the delivery of each packet (delay) and timing restrictions for the delivery of consecutive packets belonging to the same traffic stream (delay variation).

Congestion is therefore one of the most significant problems in an operational IP context. A network element is said to be congested if it experiences sustained overload over an interval of time. Congestion almost always results in degradation of service quality to end-users [44]. Congestion control schemes can include demand side policies and supply side policies. Demand side policies may restrict access to congested resources and/or dynamically regulate the demand to alleviate the overload situation. Supply side policies may expand or augment network capacity to better accommodate offered traffic. Supply side policies may also re-allocate network resources by redistributing traffic over the infrastructure. Traffic redistribution and resource re-allocation serve to increase the 'effective capacity' seen by the demand.

4.5 Congestion Management policies In Computer Networks.

Congestion management policies can be categorized based upon the following criteria:

1. Response time scale, which can be characterized as long, medium, or short.
2. Reactive versus preventive, which relates to congestion control and congestion avoidance.
3. Supply side versus demand side congestion management schemes.

These aspects are discussed in the following paragraphs.

4.5. 1 Congestion Management based on Response Time Scales

• Long (weeks to months): Capacity planning works over a relatively long time scale to expand network capacity based on estimates or forecasts of future traffic demand and traffic distribution. Since router and link provisioning take time and are generally expensive, these upgrades are typically carried out in the weeks-to-months or even years time scale.

• Medium (minutes to days): Several control policies fall within the medium time scale category. Examples include: Adjusting IGP and/or BGP parameters to route traffic away or towards certain segments of the network; Setting up and/or adjusting some explicitly routed label switched paths (ER-LSPs) in MPLS networks to route some traffic trunks away from possibly congested resources or towards possibly more favorable routes; or re-configuring the logical topology of the network to make it correlate more closely with the spatial traffic distribution using for example some underlying path-oriented technology such as MPLS LSPs, ATM PVCs, or optical channel trails.

Many of these adaptive medium time scale response schemes rely on a measurement system that monitors changes in traffic distribution, traffic shifts, and network resource utilization and subsequently provides feedback to the online and/or offline traffic engineering mechanisms and tools which employ this feedback information to trigger certain control actions to occur within the network. The traffic engineering mechanisms and tools can be implemented in a distributed fashion or in a centralized fashion, and may have a hierarchical structure or a flat structure. The comparative merits of distributed and centralized control structures for networks are well known. A centralized scheme may have global visibility into the network state and may produce potentially more optimal solutions. However, centralized schemes are prone to single points of failure and may not scale as well as distributed schemes [44], [24].

-.Moreover, the information utilized by a centralized scheme may be stale and may not reflect the actual state of the network. This is a choice that network administrators must make based on their specific needs.

• Short (Pico seconds to minutes): This category includes packet level processing functions and events on the order of several round trip times. It includes router mechanisms such as passive and active buffer management. These mechanisms are used to control congestion and/or signal congestion to end systems so that they can adaptively regulate the rate at which traffic is injected into the network. One of the most popular active queue management schemes, especially for TCP traffic, is Random Early Detection (RED), which supports congestion avoidance by controlling the average queue size. During congestion (but before the queue is fille?), the RED scheme chooses arriving packets to "mark" according to a probabilistic algorithm, which takes into account the average queue size. For a router that does not utilize explicit congestion notification (ECN), the marked packets can simply be dropped to signal the inception of congestion to end systems. On the other hand, if the router supports ECN, then it can set the ECN field in the packet header. Several variations of RED have been proposed to support different drop precedence levels in multi-class environments.

There is general consensus that RED provides congestion avoidance performance, which is not worse than traditional Tail-Drop (TD) queue management (drop arriving packets only when the queue is full).

Importantly, however, RED reduces the possibility of global synchronization and improves fairness among different TCP sessions. However, RED by itself cannot prevent congestion and unfairness caused by sources unresponsive to RED, e.g., UDP traffic and some misbehaved greedy connections. Other schemes have been proposed to improve the performance and fairness in the presence of unresponsive traffic. Some of these schemes were proposed as theoretical frameworks and are typically not available in existing commercial products. Two such schemes are Longest Queue Drop (LQD) and Dynamic Soft Pll[titioning with Random Drop (RND) [23], [24].

4.5.2 Congestion Management: Reactive versus Preventive Schemes:

• Reactive: reactive (recovery) congestion management policies react to existing congestion problems to improve it. All the policies described in the long and medium time scales above can be categorized as being reactive especially if the policies are based on monitoring and identifying existing congestion problems, and on the initiation of relevant actions to ease a situation.

• Preventive: preventive (predictive/avoidance) policies take proactive action to prevent congestion based on estimates and predictions of future potential congestion problems. Some of the policies described in the long and medium time scales fall into this category. They do not necessarily respond immediately to existing congestion problems. Instead forecasts of traffic demand and workload distribution are considered and action may be taken to prevent potential congestion problems in the future. The schemes described in the short time scale (e.g., RED and its variations, ECN, LQD, and RND) are also used for congestion avoidance since dropping or marking packets before queues actually overflow would trigger corresponding TCP sources to slow down.

4.5.3 Congestion Management: Supply Side versus Demand Side Schemes.

• Supply side: supply side congestion management policies increase the effective capacity available to traffic in order to control or obviate congestion. This can be accomplished by augmenting capacity. Another way to accomplish this is to minimize congestion by having a relatively balanced distribution of traffic over the network. For example, capacity planning should aim to provide a physical topology and associated link bandwidths that match estimated traffic workload and traffic distribution based on forecasting (subject to budgetary and other constraints). However, if actual traffic distribution does not match the topology derived from capacity panning (due to forecasting errors or facility constraints for example), then the traffic can be mapped onto the existing topology using routing control mechanisms, using path oriented technologies (e.g., MPLS LSPs and optical channel trails) to modify the logical topology, or by using some other load redistribution mechanisms.

• Demand side: demand side congestion management policies control or regulate the offered traffic to alleviate congestion problems. For example, some of the short time scale mechanisms described earlier (such as RED and its variations, ECN, LQD, and RND) as well as policing and rate shaping mechanisms attempt to regulate the offered load in various ways. Tariffs may also be applied as a demand side instrument. To date, however, tariffs have not been used as a means of demand side congestion management within the Internet.

In summary, a variety of mechanisms can be used to address congestion problems in computer networks. These mechanisms may operate at multiple time-scales (25), [44], [24].

4.6 Optimization In Computer Networks.

Network performance optimization involves resolving network issues by transforming such issues into concepts that enable a solution, identification of a solution, and implementation of the solution. Network performance optimization can be corrective or perfective. In corrective optimization, the goal is to remedy a problem that has occurred or that is incipient. In perfective optimization, the goal is to improve network performance even when explicit problems do not exist and are not anticipated [45].

Network performance optimization is a continual process, as noted previously. Performance optimization iterations may consist of real-time optimization sub-processes and non-real-time network planning sub-processes. The difference between real-time optimization and network planning is primarily in the relative time- scale in which they operate and in the granularity of actions. One of the objectives of a real-time optimization sub-process is to control the mapping and distribution of traffic over the existing network infrastructure to avoid and/or relieve congestion, to assure satisfactory service delivery, and to optimize resource utilization. Real-time optimization is needed because random incidents such as fiber cuts or shifts in traffic demand will occur irrespective of how well a network is designed. These incidents can cause congestion and other problems to manifest in an operational network. Real-time optimization must solve such problems in small to medium time-scales ranging from microseconds to minutes or hours. Examples of real- time optimization include queue management, IGP/BGP metric tuning, and using technologies such as MPLS explicit LSPs to change the paths of some traffic trunks [44].

One of the functions of the network planning sub-process is to initiate actions to systematically evolve the architecture, technology, topology, and capacity of a network. When a problem exists in the network, real-time optimization should provide an immediate remedy. Because a prompt response is necessary, the real- time solution may not be the best possible solution. Network planning may subsequently be needed to refine the solution and improve the situation. Network planning is also required to expand the network to support traffic growth and changes in traffic distribution over time. As previously noted, a change in the topology and/or capacity of the network may be the outcome of network planning [45], [44].

Clearly, network planning and real-time performance optimization are mutually complementary activities. A well-planned and designed network makes real-time optimization easier, while a systematic approach to real-time network performance optimization allows network planning to focus on long term issues rather than tactical considerations. Systematic real-time network performance optimization also provides valuable inputs and insights toward network planning.

4.7 Constrained–Based Routing

Constraint-based routing refers to a class of routing systems that compute routes through a network subject to the satisfaction of a set of constraints and requirements. In the most general setting, constraint-based routing may also seek to optimize overall network performance while minimizing costs.

The constraints and requirements may be imposed by the network itself or by administrative policies. Constraints may include bandwidth; hop count, delay, and policy instruments such as resource class attributes. Constraints may also include domain specific attributes of certain network technologies and contexts, which impose restrictions on the solution space of the routing function. Path oriented technologies such as MPLS have made constraint-based routing feasible and attractive in public IP networks [44].

MPLS is an advanced forwarding scheme, which also includes extensions to conventional IP control plane protocols. MPLS extends the Internet routing model and enhances packet forwarding and path control. MPLS is a very powerful technology for Internet traffic engineering because it supports explicit LSPs which allow constraint-based routing to be implemented efficiently in IP networks [24], [44], (45].

Routing control is a significant aspect of Internet traffic engineering. Routing impacts many of the key performance measures associated with networks, such as throughput, delay, and utilization. Generally, it is very difficult to provide good service quality in a wide area network without effective routing control. A desirable routing system is one that takes traffic characteristics and network constraints into account during route selection while maintaining stability.

Traditional shortest path first (SPF) interior gateway protocols are based on shortest path algorithms and have limited control capabilities for traffic engineering. These limitations include[49], (50], [51]:

1. The well known issues with pure SPF protocols, which do not take network constraints and traffic characteristics into account during route selection. For example, since IGPs always use the shortest paths (based on administratively assigned link metrics) to forward traffic, load sharing cannot be accomplished among paths of different costs. Using shortest paths to forward traffic conserves network resources, but may cause the following problems:

.:If traffic from a source to a destination exceeds the capacity of a link along the shortest path, the link (hence the shortest path) becomes congested while a longer path between these two nodes may be under-utilized;

-The shortest paths from different sources can overlap at some links. If the total traffic from the sources exceeds the capacity of any of these links, congestion will occur. Problems can also occur because traffic demand changes over time but network topology and routing configuration cannot be changed as rapidly. This causes the network topology and routing configuration to become sub-optimal over time, which may result in persistent congestion problems.

2. The Equal-Cost Multi-Path (ECMP) capability of SPF IGPs supports sharing of traffic among equal cost paths between two nodes. However, ECMP attempts to divide the traffic as equally as possible among the equal cost shortest paths. Generally, ECMP does not support configurable load sharing ratios among equal cost paths. The result is that one of the paths may carry significantly more traffic than other paths because it may also carry traffic from other sources. This situation can result in congestion along the path that carries more traffic.

3. Modifying IGP metrics to control traffic routing tends to have network-wide effect. Consequently, undesirable and unanticipated traffic shifts can be triggered as a result. Because of these limitations, new capabilities are needed to enhance the routing

function in IP networks. Some of these capabilities have been described elsewhere and are summarized below [50].

Constraint-based routing is desirable to evolve the routing architecture of IP networks, especially public IP backbones with complex topologies.

Constraint-based routing computes routes to fulfill requirements subject to constraints. Constraints may include bandwidth; hop count, delay, and administrative policy instruments such as resource class attributes.

This makes it possible to select routes that satisfy a given set of requirements subject to network and administrative policy constraints. Routes computed through constraint-based routing are not necessarily the shortest paths. Constraint-based routing works best with path oriented technologies that support explicit routing, such as MPLS.

Constraint-based routing can also be used as a way to redistribute traffic onto the infrastructure (even for best effort traffic). For example, if the bandwidth requirements for path selection and reserved bandwidth attributes of network links are appropriately defined and configured, then congestion problems caused by uneven traffic distribution may be avoided or reduced. In this way, the performance and efficiency of the network can be improved.

A number of enhancements are needed to conventional link state IGPs, such as OSPF and IS-IS, to allow them to distribute additional state information required for constraint-based routing. Essentially, these enhancements require the propagation of additional information in link state advertisements. Specifically, in addition to normal link-state information, an enhanced IGP is required to propagate topology state information needed for constraint-based routing. Some of the additional topology state information include link attributes such as reservable bandwidth and link resource class attribute (an administratively specified property of the link).

4.8 Topological Optimization of Network using Integer Programming

Conventional routing such as OSPF is the most common routing protocol inside an Autonomous System (AS), such as an Internet Service Provider (ISP) network. Each link is assigned a cost, which is often set proportional to the inverse link capacity. Traffic is routed via the shortest path between the source and the destination, or "load balanced" among equal cost shortest paths where they exist. Therefore, the only way that the network operator can change the routing is by changing the link weights. The usual way that network operators try to reduce network congestion is by increasing the link cost of the congested link(s). However doing this may lead to the shifting of a large amount of traffic from that particular link to other less congested ones, overloading the other links, often in an unexpected way. There is a need for a tool for modeling OSPF routing and for determining the link weights before implementing changes to a real network [5], [45].

Several approaches have been presented [45], [8], [7], [6], [5], [46]. The approach that will be implemented in this thesis uses a integer programming module based on link costs and link reliability. The objective is to minimize the usage of links and at the same time maintain a highly reliable n~twork.

In the design of communication networks, one of the fundamental considerations is the reliability and the availability of the communication paths between all terminals. Together, these form the network system reliability. The other important aspect is the layout of the paths to minimize cost while meeting the reliability demands.

An important part of network design is to find the best way to layout the components (nodes and arcs) to minimize cost while meeting performance requirements such as transmission delay, throughput or reliability. This design stage is called 'Network Topological Optimization'. In a topological network design problem, the main concern is to design networks, which will operate effectively and without interruption in the presence of component failure. Reliability is concerned with the ability of a network to carry out desired network operations.

The mathematical approach used in this thesis is based on integer programming where a binary coding structure for representing candidate solutions is used. If we have a network that has five nodes and 10 possible links, they can be represented as:

[1 1 0 1 0 1 0 1 0 1]

[x_{12} x_{13} x_{14} x_{15} x_{23} x_{24} x_{25} x_{34} x_{35} x_{45}]

Where x_{ij} represents a link connecting two nodes I and j. if x_{ij} is equal to 1, there is a connection between these nodes. If x_{ij} is equal to 0, then there is no connection.

UsingJ.tiis approach, the following characteristics will be used:

1. Arc probabilities between [0, 1] which determine the existence of an arc between nodes, are selected.

2. The system reliability value of each connected network is estimated using a random generation function on a scale of 10. {1 is 10% and 10 is 100%}.

3. The cost values for each connected link is simulated using network metrics generation functions.

The aim is to determine the best combination of links such that, the network will meet the required reliability measure.

The mathematical module is based on the minimum cost flow problem that can be stated as a integer program as follows:

$$\textit{Minimize } ex$$

$$\textit{Subject to: } Nx = b, \textit{ Where } x = (0,1\}.$$

Here, N is an m x n matrix called the node-arc incidence matrix and the integer equation system Nx =bis the mass balance equations, representing the inflow and outflow of each node in the network.

Network planning is concerned with the design of sufficiently reliable networks at reasonable cost, to deliver high capacity and speed [8].

A network is modeled by a probabilistic undirected graph G=(N, L, p), in which N represents the set of nodes, L is a given set of possible links, and p is reliability of each link. It is assumed one bi-directional link between each pair of nodes.

The optimization problem may be stated as:

$$\textit{Minimize } Z = \overset{I}{\underset{i=1}{L}} \; \overset{J}{\underset{J=l}{L}} \; Ci\} \times i\}$$

$$\textit{Subject to: } R(x)'?\!:\; R_0$$

"Where X_{iJ} is a decision variable (0,1), C_{iJ} is the cost of the link *(i,j)*, $R(x)$ is the network reliability and R_0 is the minimum reliability requirements.

Its possible that sometimes the link reliability is so low that additional constrains are required, if the network link reliability is below 30% or 3 on a scale of 10. Additional constraints may be needed to decrease or limit that link.

To solve this problem, the following assumptions are made:

• The N nodes are perfectly reliable. A problem with a node may be simulated by the failure of its incident links.

• The cost C_{iJ} and the reliability R_{iJ} of each link *(i,j)*, are already known or to be known.

• The links have two states: either operational (x_{iJ}= 1) or failed (x_{iJ}= 0).

• The links failure are independent.

• No repair is considered.

• Two connectivity is required.

Below is the network diagram to be solved:

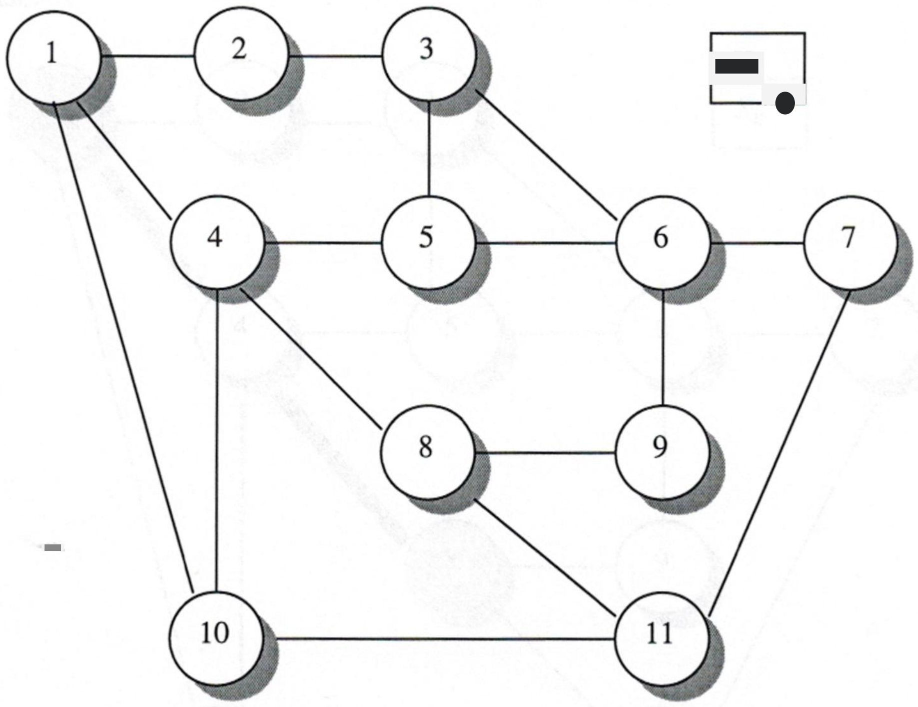


Figure 4.1 *The network topology considered for optimization*

While in conventional routing there is no relation to network reliability, or link bandwidth. There is only a cost constraint and the routing protocol will route packets based on the lowest cost. Cost in networks is a linear function based on the load [5], capacity [6], or a combination of the two [7].

In conventional routing a path is taken to be singular route from start to finish. And when two paths have the exact shortest value from start node to end node, one of those paths maybe chosen or the load is distributed across the two using a round robin fashion, opening a door for packet integrity and redundancy loss [23], [44].

Below, is a demonstration of conventional routing path selection, and optimized routing path selection. In conventional routing only one route is taken from node 1 to node 8.While in optimized routing more than one path is taken depending on links reliability.

This is how conventional routing path selection process from node 1 to node 8 are made:

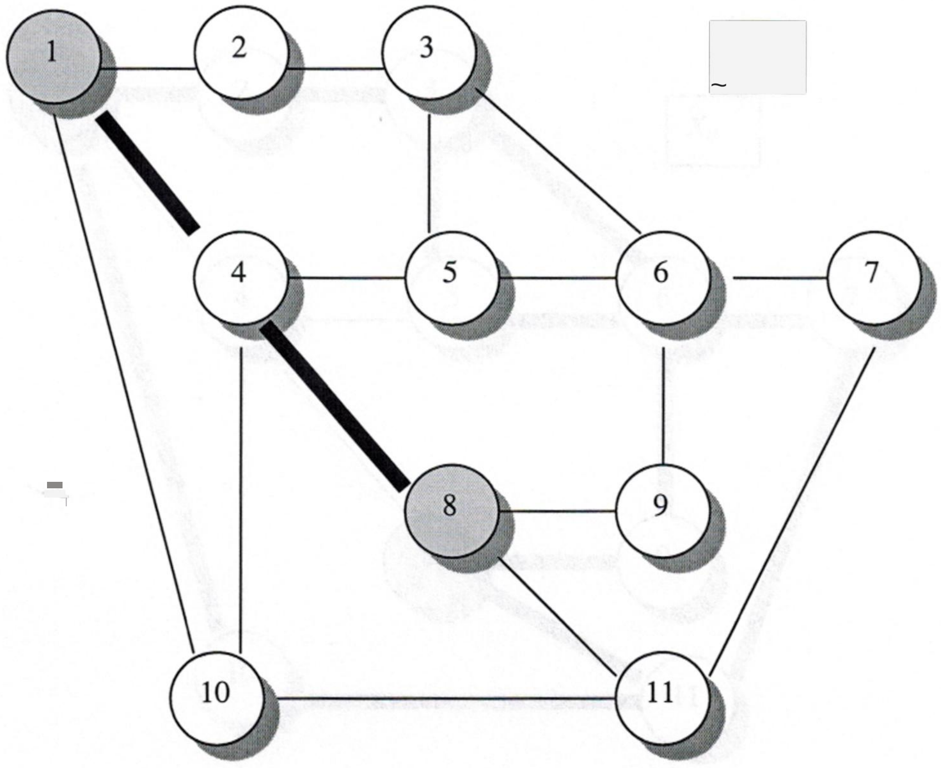


Figure 4.2 *Typical selection ofpaths based on conventional routing*

This can demonstrate the ease of using Dijksrtra's algorithm; there is only the cost metric that the routing decision is made. If the cost from node to nodel to 4 and from node 4 to node 8 contains the shortest costs in order to reach node 1 to node 8. Then the path 1-4-8 will be granted to route units from node 1 to node 8.

This is how optimized routing is selecting paths based on link reliability assuming that the shortest path 1-4-8 is under very low reliability:

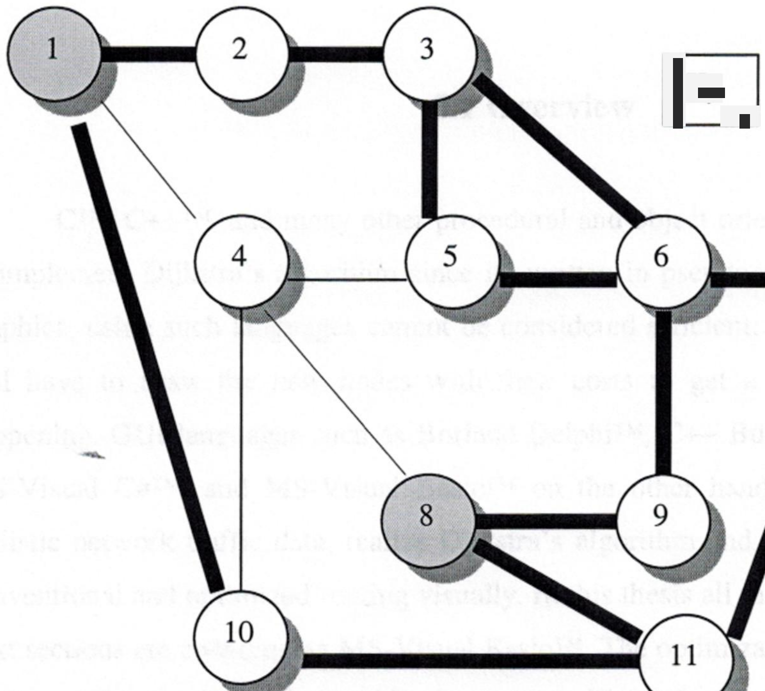


Figure 4.3 *Candidates paths achieved after optimization*

4.9 Summary

In this chapter, we have discussed optimization over IP networks or Traffic Engineering. We have explained Network optimization since the early days of computers and how routing was made on those days. Continuing to new generation routing as Constrained-Based Routing being used today in Internet backbones. We also formulated a module used to improve computer networks performance. Since this problem is NP-Complete. We decided to use integer programming to solve this objective function and thus, get the best combination of paths to maintain good network reliability, as it will be demonstrated in chapter 5.

5. IMPLEMENTATION OF DIJKSTRA'S ALGORITHM AND SIMULATION OF NETWORK TRAFFIC OPTIMIZATION PROBLEM.

5.1 Overview

C™, C++™, and many other procedural and object oriented languages can be used to implement Dijkstra's algorithm since its written in pseudocode. However in respect to graphics, using such languages cannot be considered efficient. Since after every result we will have to draw the new nodes with their costs to get a mental picture of what is happening. GUI languages such as Borland Delphi™, C++ Builder™, MS-Visual C++™, MS-Visual C#™, and MS-Visual Basic™ on the other hand, can be used to generate realistic network traffic data, realize Dijkstra's algorithm and finally display results from conventional and optimized routing visually. In this thesis all the programs presented in the next sections are coded using MS-Visual Basic™. The optimization package Win QSB was used to solve the network optimization model. Then charting was displayed using MS-Visual Basic.

This Program consists of the following sections:

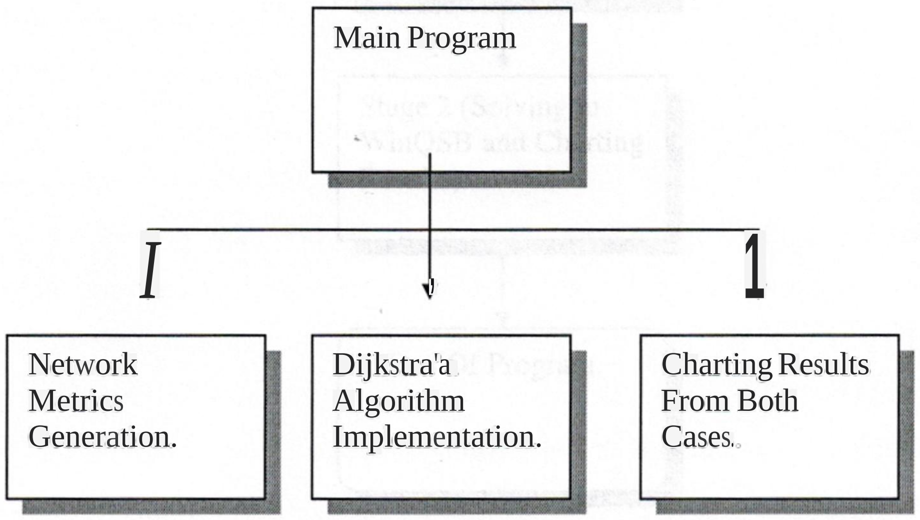

Main Program
Network Metrics Generation.
Dijkstra'a Algorithm Implementation.
Charting Results From Both Cases.

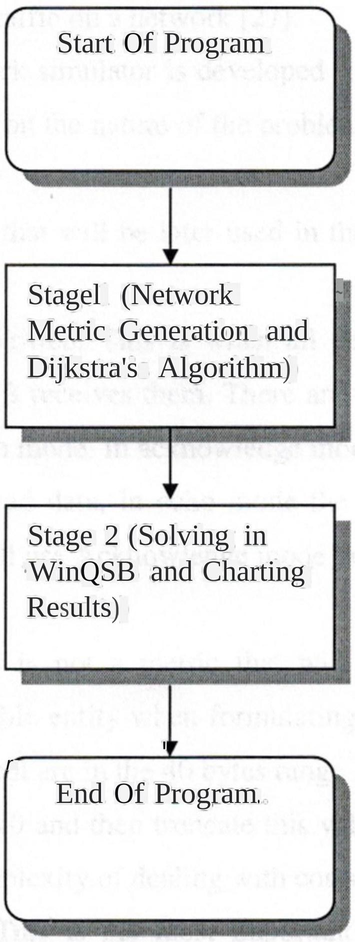
Figure 5.1 *Main braches of the simulation process*

5.2.1 Implementation Stages

In this Section, the flowcharts and logic structure of the solution is presented, First the solution consist of two main stages:

Stage 1: This stage is concerned with getting all the network metrics that will later be used in the optimization package. Based on functions taken from relevant research [6], [7], [5] and [8]. And then, apply Dijkstra's algorithm on the network.

Stage2: This stage is where we feed the metrics we received from the network metrics generation to the optimization package WinQSB. And then later after we receive the results we store it in a database to use that data to chart the results from the both cases, conventional routing and optimized or QoS routing.



Start Of Program
Stagel (Network Metric Generation and Dijkstra's Algorithm)
Stage 2 (Solving in WinQSB and Charting Results)
End Of Program.

Figure 5.2 *Overall Stages of simulation*

5.2.2 Network Metrics Generation

Network metric generation or network simulation is an important step when trying to improve network reliability. Since the data experimented upon must be randomly chosen and realistic to derive a solution as practical as possible. For this reason a number of network simulators exist.

1. Network Simulator™: This is the most will known network simulator, developed by the university of Berkeley. It runs on the Unix platform and can generate huge amount of network traffic [30].

2. Traffic™: This simulator is among the few network simulators based on the window platform, Traffic is a network traffic generator following a server/client model for generating high volumes of traffic on a network [27].

In this thesis a network simulator is developed for generating routing metrics such as cost and reliability. Based on the nature of the problem, this simulator is designed to run on single machine mode.

There are important metrics that will be later used in the mathematical module and those are:

-Bytes Read\Bytes Written: This is when all network simulation starts. Point A sends some bytes, and point B receives them. There are two main network routing modes, Acknowledge mode and Echo mode. In acknowledge mode the sent data are always less 5-10% of the original bytes read data, in echo mode the same data that is read is written without any loss [27]. We will use Acknowledge mode because we will later truncate some values to avoid complexity.

• Data packets: This is not a metric that will directly affect the mathematical module, but it forms a valuable entity when formulating the cost metric. 99% of network packets under the TCP protocol are in the 40 bytes range (25]. Thus, using the bytes written we can divide that value by 40 and then truncate this value to the nearest possible integer. This way will reduce the complexity of dealing with costs that have fractions.

-The cost function: This is the most important network metric in any network routing problem. This metric can be linear function based on the network current load [5], or can be based on the network capacity [5], or it can be based on both [6] Here, based on results from chapter 3, cost function will be based on load in data packets over 5.

• Reliability: This metric has a diverse way of measuring it (20]. But in general, its based on how fast and on how much data packets can be transferred on that link. Its noted that this metric is taken based on random numbers from 1.0 (100%) to 0 (0%). With other values in between [8].

Flow Chart Of Network Metric Generator:

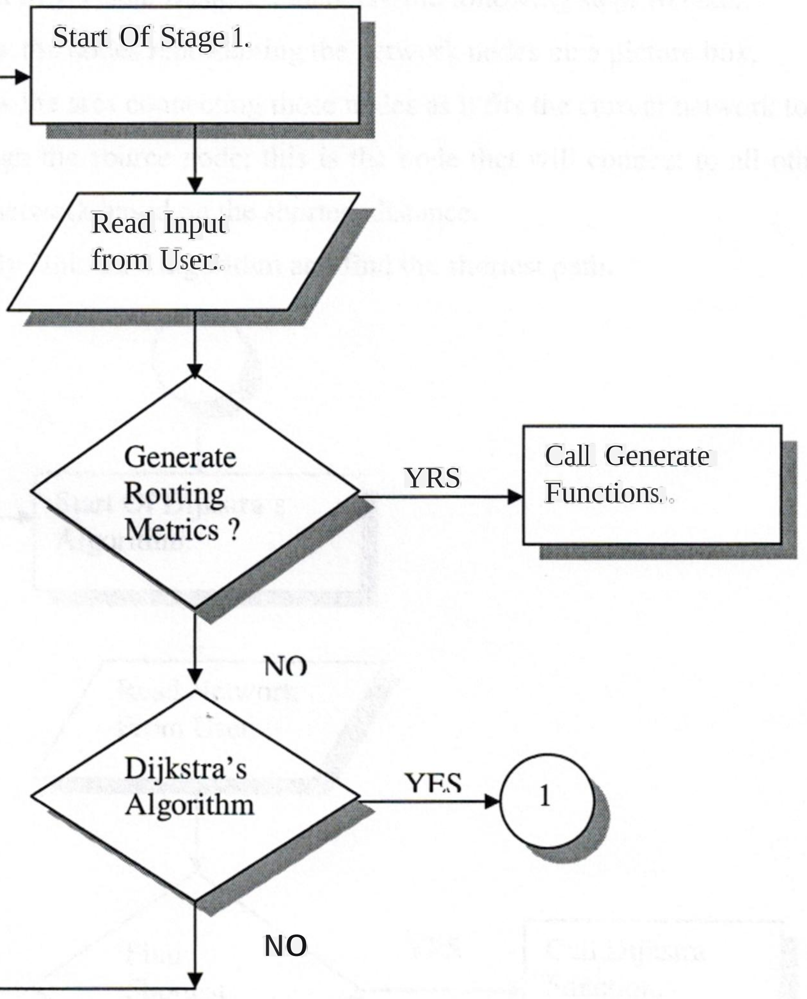


Figure 5.3 *Stage 1 main menu, flow of operations*

5.2.3 Dijkstr'a Algorithm

Dijkstra's algorithm is the main algorithm used in all network routing protocols [23],[24] including SPF, OSPF, RIP, IGP, BGP. As in this thesis, we are not concerned with the actual packet switching that happen in the OSI layers. Rather, we are working on determining the cost of all paths on a given network. And then, find the shortest path to connect point A to point B.

Dijkstra's algorithm is discussed in detail in chapter 2, and in order to implement this algorithm in MS-Visual Basic™, there are the following steps to take:

1. Draw the nodes representing the network nodes on a picture box.
2. Draw the arcs connecting those nodes as it fits the current network topology
3. Assign the source node; this is the node that will connect to all other nodes in the network based on the shortest distance.
4. Apply Dijkstra's algorithm and find the shortest path.

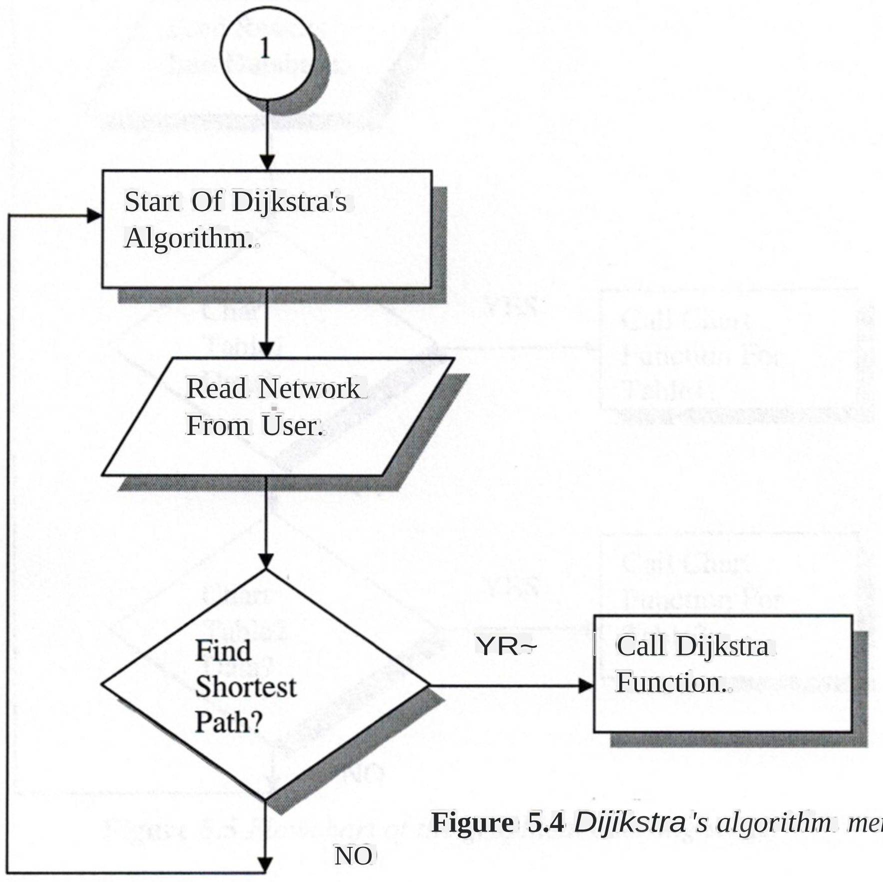


Figure 5.4 *Dijikstra's algorithm menu*

5.2.4 Charting Results

Charting is the final stage of the program, we reach this stage after we solve the module using optimization package Win QSB, Here we store the results we achieved from Dijkstra's algorithm implementation and from the optimization package in a database file, then we use this data in a MS-Visual Basic TM program to chart it.

Charting then consist of the following steps:

1. Feed the data manually from both conventional and optimized routing.
2. Use the MS-Chart control TM to chart this data in MS-Visual Basic TM

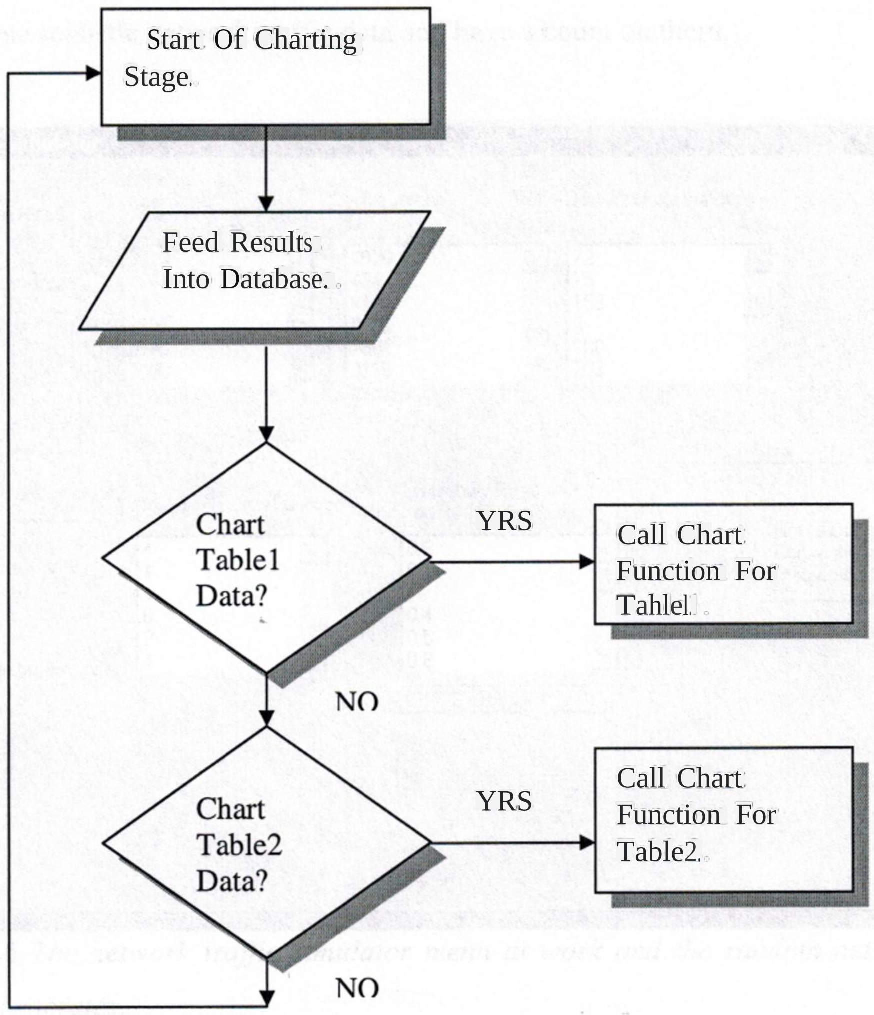

Start Of Charting Stage.
Feed Results Into Database.
Chart Table1 Data?
YRS
Call Chart Function For Tahlel.
NO
Chart Table2 Data?
YRS
Call Chart Function For Table2.
NO

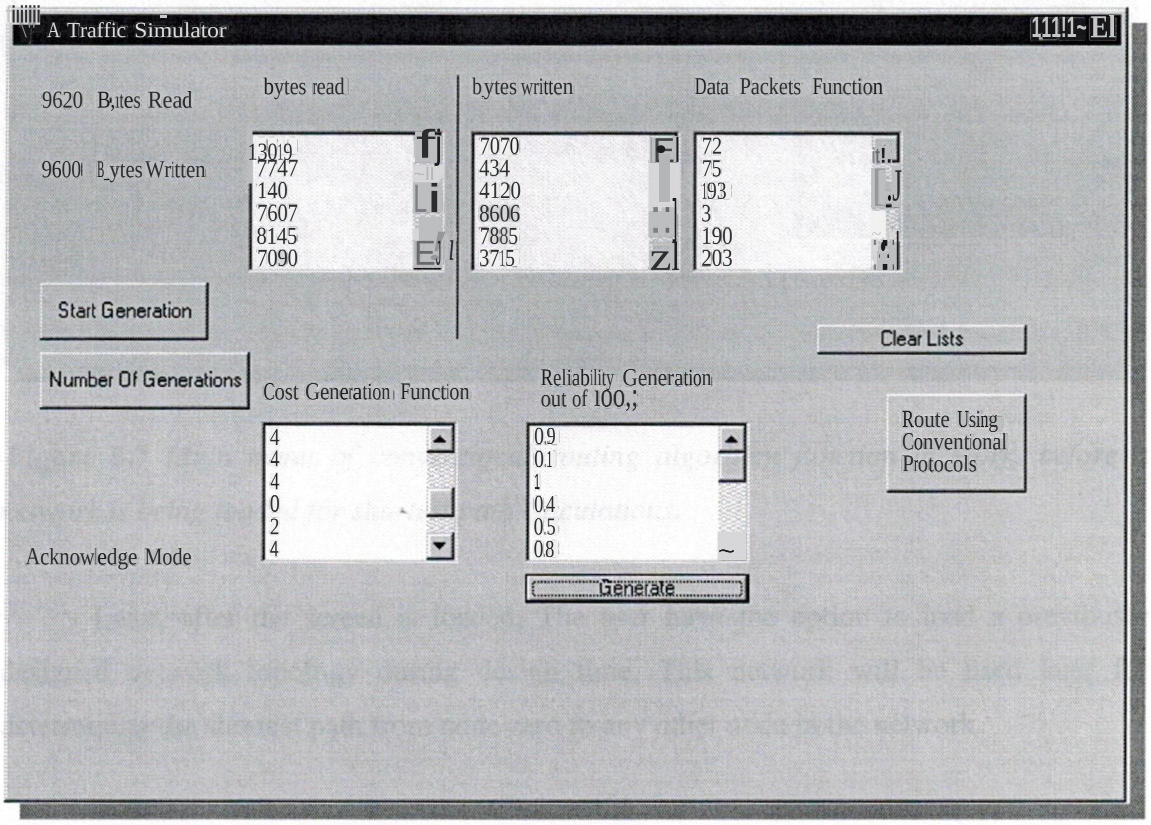
Figure 5.5 *Flowchart of the graphical charting stage.*

5.3 Using The Application

As mentioned before, the program is made up of two stages. The logic structure of the stages is discussed in section 5.1; this section will display the looks of the application.

5.3.1 Network Generator Menu

Here, the program starts with the network metrics generation menu; there is a button that will take the user to the Dijkstra's implementation menu. The program in this section will generate realistic network traffic data and have a count on them.



9620 Bytes Read

9600 Bytes Written

```
bytes read    bytes written    Data Packets Function
13019         7070             72
7747          434              75
140           4120             193
7607          8606             3
8145          7885             190
7090          3715             203
```

```
Cost Generation Function    Reliability Generation out of 100,;
4                           0.9
4                           0.1
4                           1
0                           0.4
2                           0.5
4                           0.8
```

Figure 5.6 *The network traffic simulator menu at work and the random network traffic packets it generates.*

5.3.2 Dijkstra's Implementation Menu

This is the how the screen will look when the user click on the 'Conventional Routing' Button. In essence Dijkstra's algorithm is what all conventional-routing protocols use.

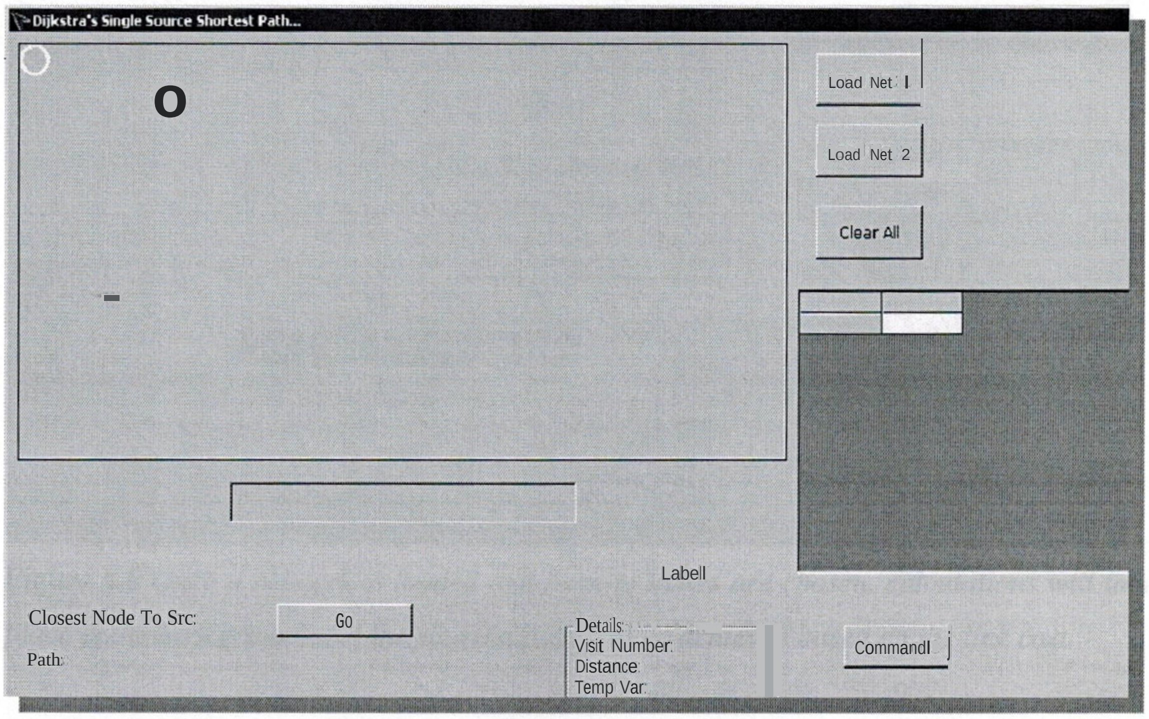
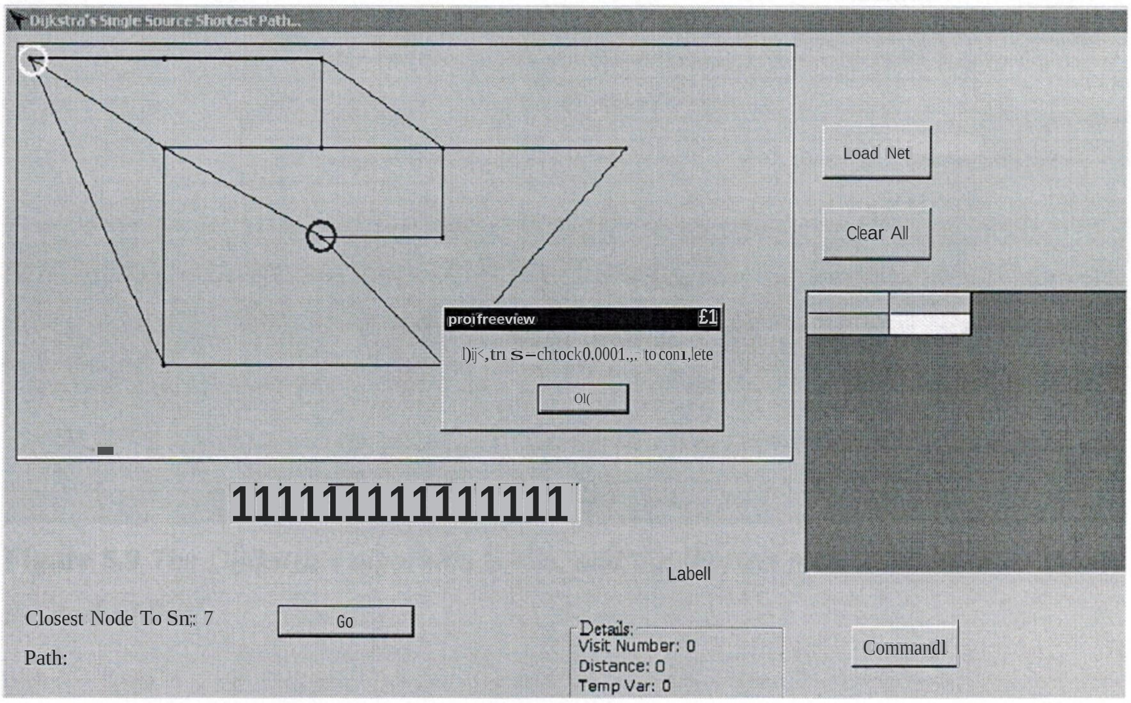


Figure 5.7 *Main menu of conventional routing algorithm function at work, before a network is being loaded for shortest path calculations.*

Later, after the screen is loaded, The user have the option to load a previously designed network topology during design time, This network will be used later for determining the shortest path from node zero to any other node in the network.

This is what will happen when the user load a network topology, using the load net button. This is a network structure based on nodes and arcs. And with connections that have the value of $\sqrt{(x_1-x_2)^2-(y_2-y_1)^2}$. This is however the better way to handle link costs compared to the one used in [22].



Dijkstra's took 0.0001 ms to complete

Closest Node To Sn: 7

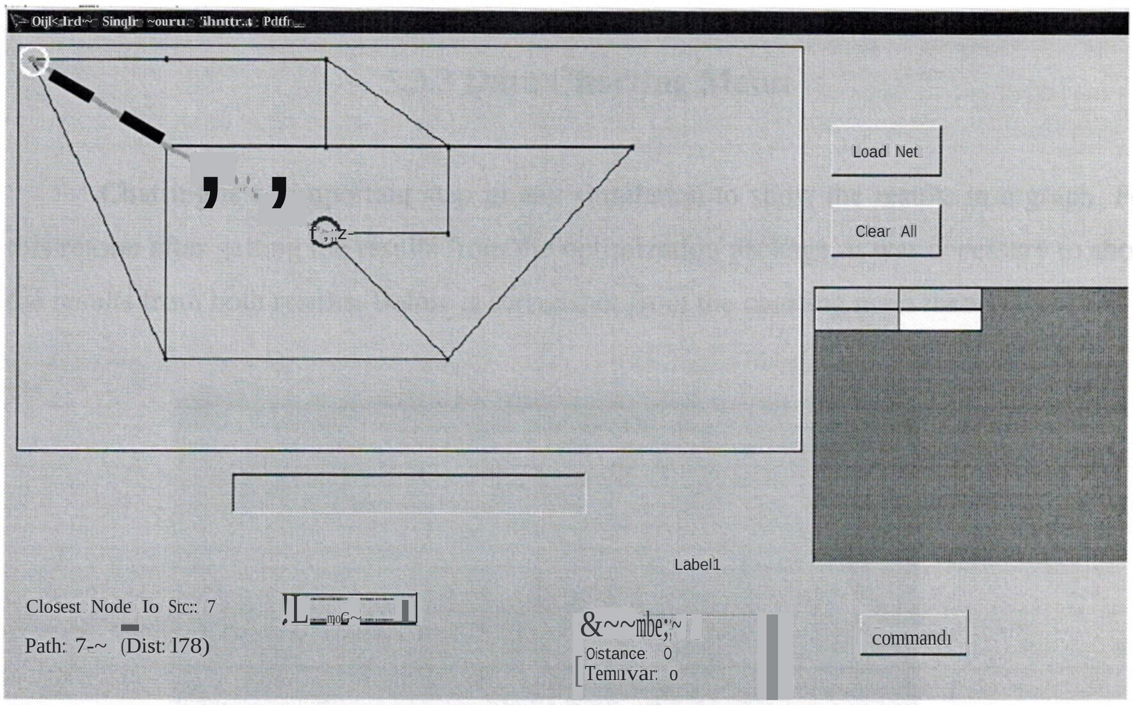
Path:

Details:
Visit Number: 0
Distance: 0
Temp Var: 0

Figure 5.8 *Once a network is loaded and from-to nodes are chosen, calculations will take place at the background and the shortest path will be achieved based on the link cost.*

After pressing the 'Go' button, Dijkstra's algorithm will start executing in the background. This algorithm is of O(n2) complexity and is of very fast running using the computer. It took only 0.0001 ms to find the shortest path from node zero (starting point) until node 6.

Here, a line will be drawn on the network map. Showing the shortest path from start node to destination node. As mentioned before all nodes distances are measured using the circle equation.



Closest Node Io Src:: 7

Path: 7-~ (Dist: 178)

Distance 0

Temııvar: o

Figure 5.9 *The Dijikstra's algorithm is run, and the shortest path is achieved as shown in the dashed line.*

This way of implementing Dijkstra's algorithm can allow having hundreds and even thousands of nodes, and the algorithm can find the shortest path in a quick and efficient way.

It is also possible to take the values of all the arcs and display them to the user, so that the user can be assured that the algorithm is executing properly. Data representing the distance array, are fetched from the distance calculation function and then are dumped into a flex grid showing all the values of the distances.

The network that will be used for routing can have different distances or costs from the start node to all other nodes. Dijkstra's algorithm will find the shortest path from this starting point to the destination node.

5.3.3 Data Charting Menu

Charting is an important step in any simulation to show the results in a graph. For this reason after getting the results from the optimization package, it was necessary to show the results from both results. Below is screenshot from the charting main menu.

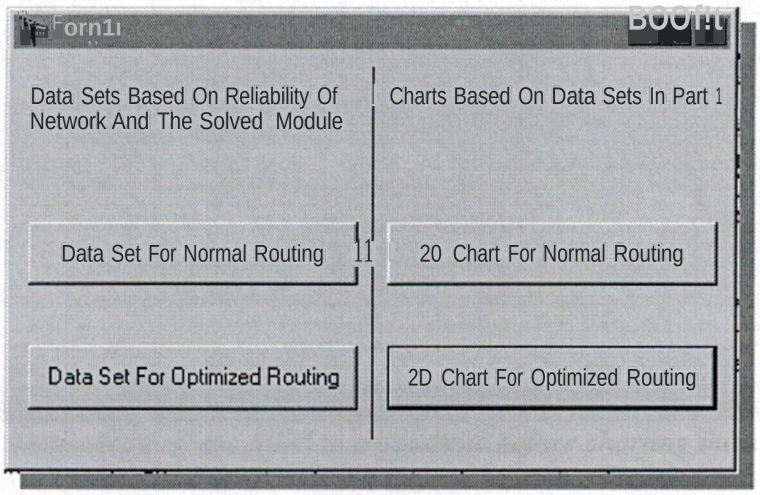

Forn1
Data Sets Based On Reliability Of Network And The Solved Module
Charts Based On Data Sets In Part 1
Data Set For Normal Routing
20 Chart For Normal Routing
Data Set For Optimized Routing
2D Chart For Optimized Routing

Figure 5.10 *Main submenu of graphical charting stage.*

This is the database view that will hold all the values from the conventional routing solution 'Tablel' and the optimized routing 'Table2' solution. The x-axis represents the number of runs that was made. Having different routing metrics, and random reliability values in each iteration. The y-axis is the reliability value average achieved from both cases.

Depending on routing metrics, the values may converge if the reliability is high on the conventional routing. For optimized routing the user can alter the reliability requirements based on different situations. If from point A to Point B data being routed are based on video conferencing data, live audio streaming or other sensitive data, that require high network reliability, the mathematical module will rearrange itself to find the best combination of network routes that are high candidates for routing.

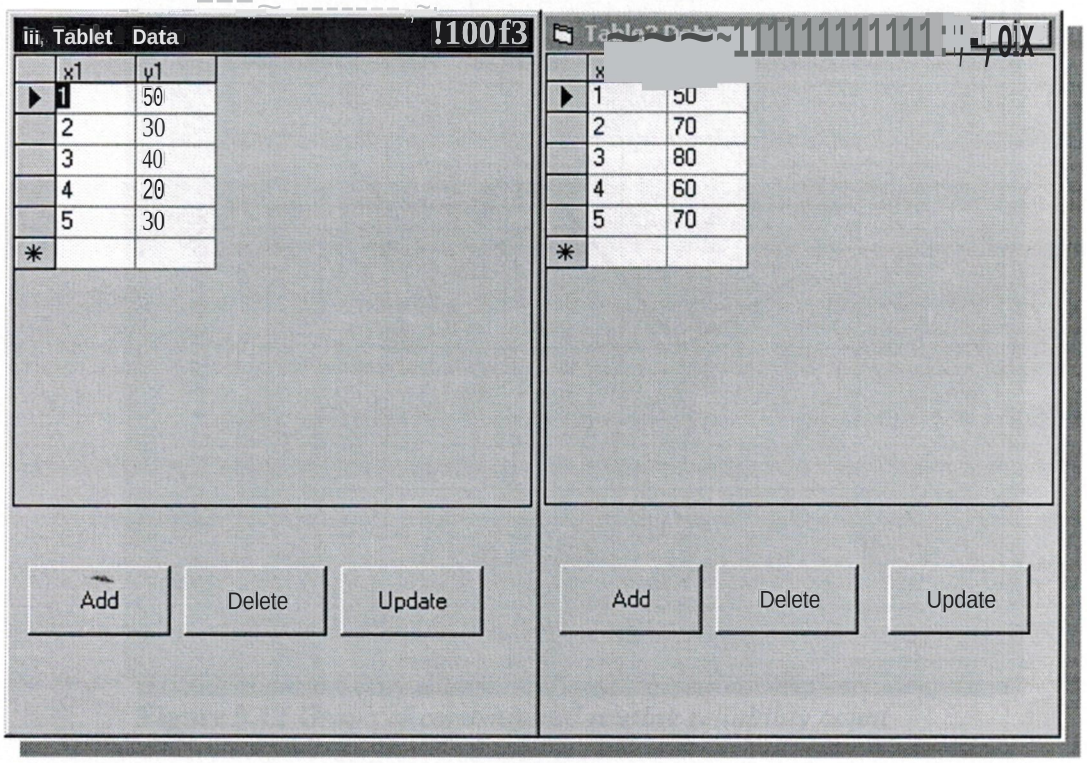


Table1 Data

	x1	y1
▶	1	50
	2	30
	3	40
	4	20
	5	30
*		

Table2 Data

	x	[illegible]
▶	1	50
	2	70
	3	80
	4	60
	5	70
*		

Figure 5.11 *Data obtained are filled to a database before charting them in a graph.*

This is the data of the conventional routing being represented by lines chart. The charting ability of MS-Visual Basic TM allows numeric data taken from a database to be displayed on the screen.

In real networks, the value of the cost can have numerous measures [34], It can be based on hop count, distance, load, capacity, reliability or a linear function derived from all or some of those metrics [7].

In Figures 5.12 and 5.13, the results taken from the both cases are charted using MS-Visual Basic™. Here, it's shown clearly that the optimized routing solution in Figure 5.13 is maintaining a network reliability count higher than the ones obtained using conventional routing shown in Figure 5.12. The solution is still in favor of the optimized method after 5 iterations. This concludes that optimized routing is a better method for maintaining the network reliability.

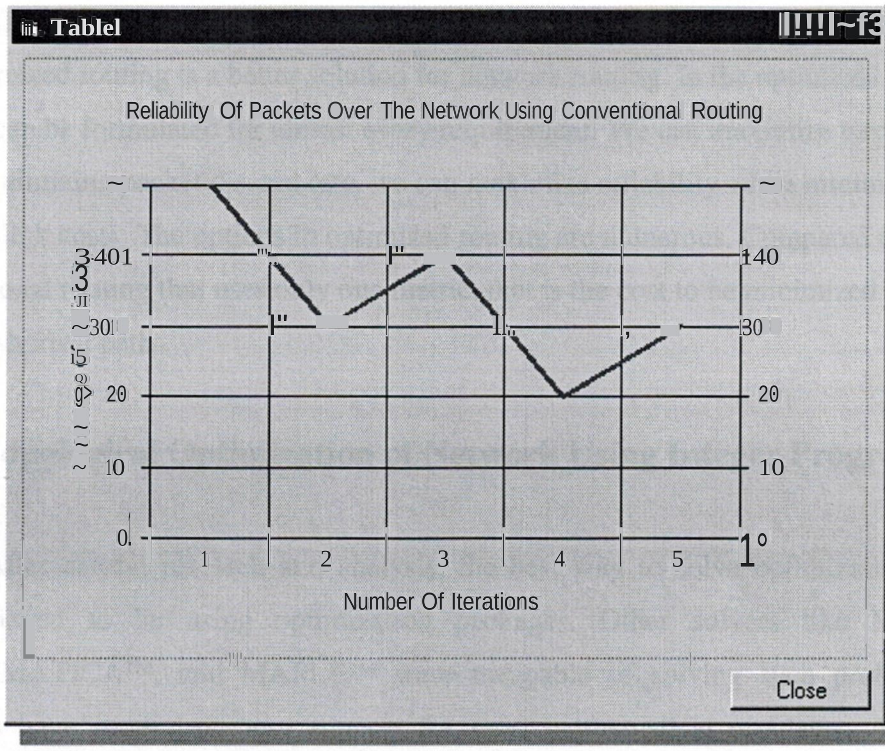


Figure 5.12 *Graph of conventional routing reliability count*

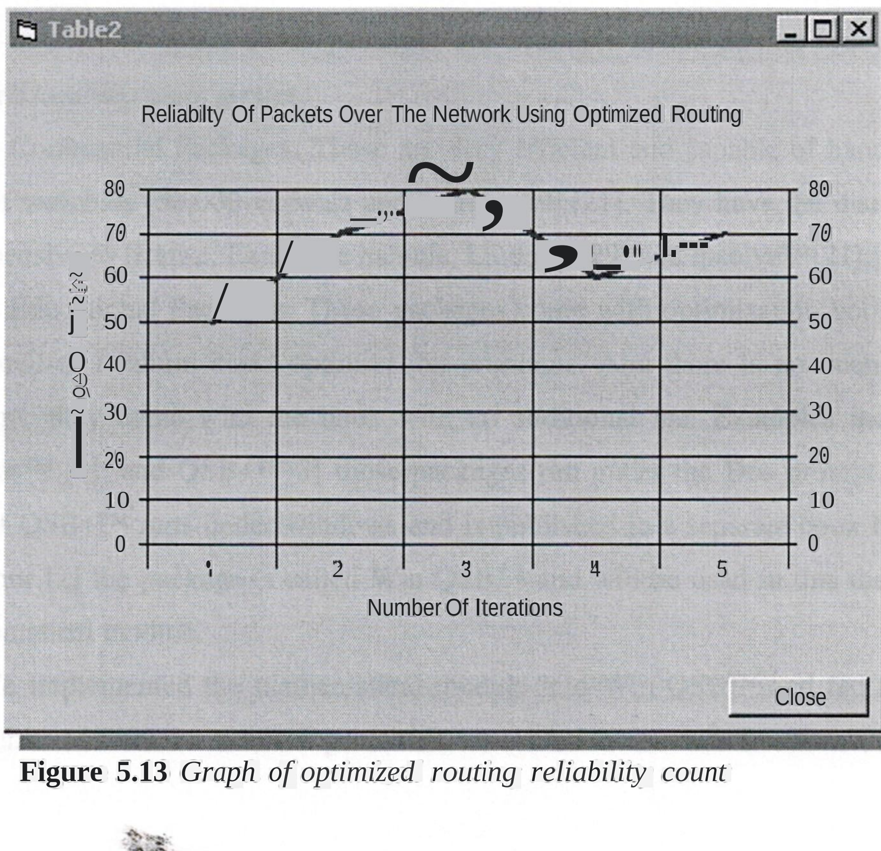


Figure 5.13 *Graph of optimized routing reliability count*

This is the data of the optimized routing being represented by lines chart. Its evident that optimized routing is a better solution for network routing. In the optimized routing a module can be formulated for almost every requirement. We can maximize throughput while minimizing packet discard rate, we can maximize reliability while minimizing network link costs. The options in optimized routing are numerous. Compared to conventional routing that uses only one metric, that is the cost to be minimized in order to find the shortest path

5.4 Topological Optimization of Network Using Integer Programming

After careful research and analysis, the best way to solve optimization problems was believed to be using optimization packages. Other solvers like MATLAB™, MATHEMATICA™, and MAPLE™ were incapable of solving such problems; those packages have general-purpose support for most mathematical problems. And was not efficient when solving the integer programming module discussed in this thesis [3 1], [32],[33].

A number of optimization packages are available in the market today, which are categorized into two main groups:

1. Commercial Packages: Those are very efficient and capable of handling a huge amount of variables (50,000 variable and more). [29],[21]. They have the disadvantage of being expensive to license. Examples include, Lindo™ [29] and lpsolve™[21],[10]

2. Educational Packages: Those packages come with optimization books, they can handle small to medium sized optimization problems. And there is no licensing fee for using them; they come with the book with no additional fee. Examples include Quick Quant Plus™ [4], and QSB+™[3] those packages run under the Dos prompt. The newer version of QSB+ ™ runs under windows and is published in a separate book but from the same author [2] the package is called Win QSB™ and will be used in this thesis to solve the mathematical module.

We implemented the mathematical module into Win QSB™ and fed the network metrics achieved from the network generator, we solved the module five times, in each time optimized routing equals or is slightly less than the required network reliability requirement.

Conventional routing has no means to adjust it self based on the current network reliability requirements. The most useful network protocols can provide administrators with network metrics that they have to manually alter the network link costs to maintain network stability and reliability [34].

First, all the possible paths from all nodes are labeled using the x_{ij} labeling convention.

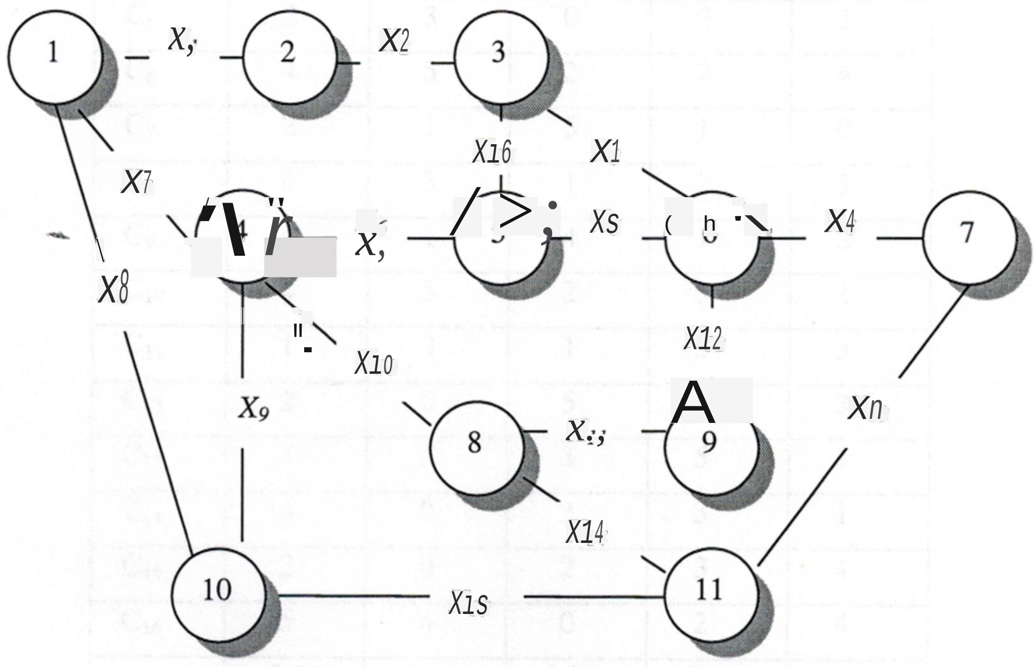


Figure 5.14 *The network topology considered for optimization.*

Second, all the variables are defined to feed them later to the optimizer. All the variables are shown in the table below.

C_{ij}	Cost Function.
λ_{ij}	Load on the network or effective bandwidth.
R_{ij}	Reliability of the network links.
X_{ij}	Unknown path combinations.

The cost is achieved using the linear function Cost = load in packets / 5

Costs	Run1	Run2	Run3	Run4	Run5
C_1	4	3	3	1	4
C_2	3	5	1	5	0
C_3	3	5	1	0	5
C_4	1	1	3	2	2
C_5	2	3	0	2	3
C_6	4	5	2	2	4
C_7	2	1	5	1	0
C_8	4	3	1	2	3
C_9	4	1	4	1	2
C_{10}	4	5	2	3	1
C_{11}	1	3	1	3	3
C_{12}	2	0	5	1	3
C_{13}	4	3	3	5	1
C_{14}	4	0	3	3	1
C_{15}	2	0	2	3	4
C_{16}	5	4	0	2	4
Average	3.06	2.62	2.25	2.25	2.5

Table 5.1 *The cost value achieved in 5 random runs for all costs*

The reliability is generated based on a probability of I.0 (100%) to 0 (0%). The numbers here are random packet size numbers.

Reliability	Run I	Run2	Run3	Run4	Run5
Rl	0.9	0.3	0.6	O.I	0.6
Rl	0.1	0	0.7	0.8	I
R3	1.0	0.3	0.9	0.5	0.9
~	0.4	0.4	0.8	0.8	0.2
Rs	0.5	0.3	0	0.6	0.7
R6	0.8	0.9	0.5	0.8	I
R1	O.I	1.0	0.9	0	0.2
Rs	0.6	0.4	0.4	0.2	0.5
R9	0.5	0.3	0.7	0.1	0.1
R10	0.3	0.2	0.5	0.1	I
R11	0.6	0.2	0.5	0.3	0.7
R12	0.6	0.6	0.5	O.I	0
R13	0.3	0.4	0.4	0	0.6
R14	0.3	0.4	0.4	0.5	0.1
RIS	0.8	0.7	0.3	0.7	O.I
R16	0.8	0.3	0.1	0.5	0.8
Average	0.537	0.418	0.512	0.381	0.531

Table 5.2 *Average reliability for 5 random runs for all paths.*

Paths	Cost	Reliab.
X_1	4	0.9
X_2	3	0.1
X_3	3	1.0
X_4	1	0.4
X_5	2	0.5
X_6	4	0.8
X_7	2	0.1
X_8	4	0.6
X_9	4	0.5
X_{10}	4	0.3
X_{11}	1	0.6
X_{12}	2	0.6
X_{13}	4	0.3
X_{14}	4	0.3
X_{15}	2	0.8
X_{16}	5	0.8
Average	3.06	0.537

Table 5.3 *Summary of costs and reliabilities.*

Run	Conventional Routing		Optimized Routing	
	Avr. Cost	Reliability	Avr. Cost	Reliability
1	3.06	0.7	3.06	0.7
2	2.62	0.5	2.62	0.8
3	2.25	0.4	2.25	0.7
4	2.25	0.3	2.25	0.8
5	2.50	0.6	2.50	0.8

Table 5.4 *Results achieved for conventional and optimized approaches consequently*

Run 1:

In this case, the reliability was quite high for link x_5 and link x_{15}. However, in some cases the links reliability can be very low, resulting in a degrading performance for the whole network.

	Overall Reliability
Conventional	0.50
Optimized	0.76

Table 5.5 *Reliability overall results from conventional and optimized approaches*

Using conventional routing, the shortest path from node1 to node11, based on the shortest cost is determined to be x_5 -x_{15}. Having with an average reliability of 0.6 + 0.8 = 1.4 / 2 = 0.7 or 70%.

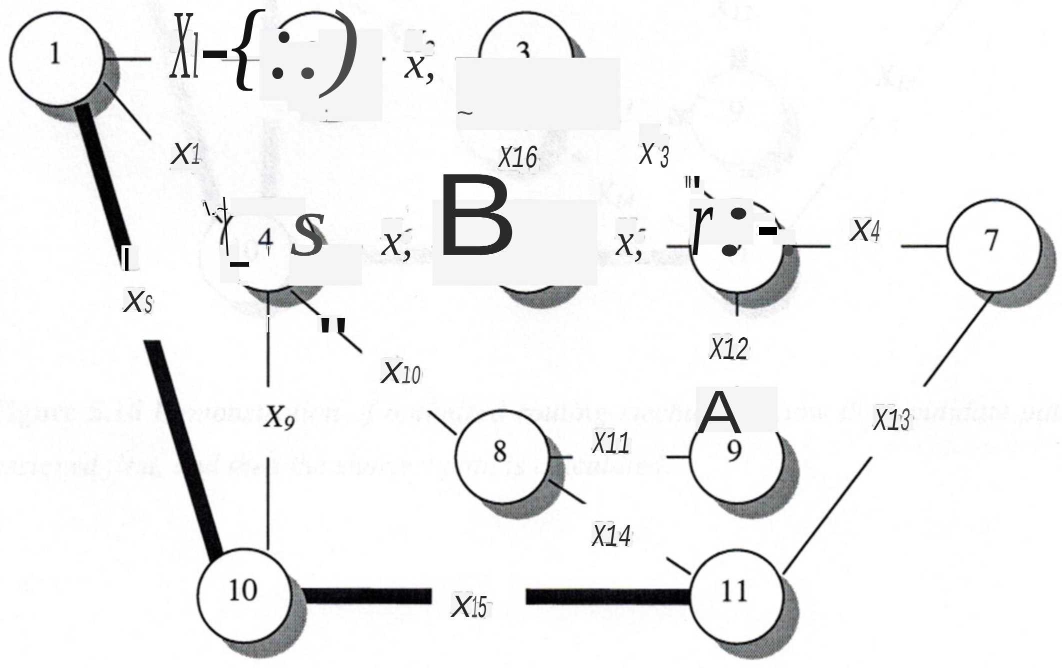


Figure 5.15 *A demonstration of path finding mechanism in conventional routing, assuming* x_{10}, x_{15} *are having the lowest cost to connect node 1 to node 11.*

Here, the candidate links, using optimized routing are shown in the figure below, in this particular case, optimized routing cannot exceed 70% due to the inclusion of x_8 and x_{15}.

If the shortest path from nodel to nodell, is to be $x_7 - x_{10} - x_{14}$ then based on optimized routing solution that path is not to be taken, because x_{10} and x_{14} are out of the candidate paths. They are under very low reliability. The paths that are not considered as candidate paths are x_2, x_{10}, x_{13} and x_{14} as shown below.

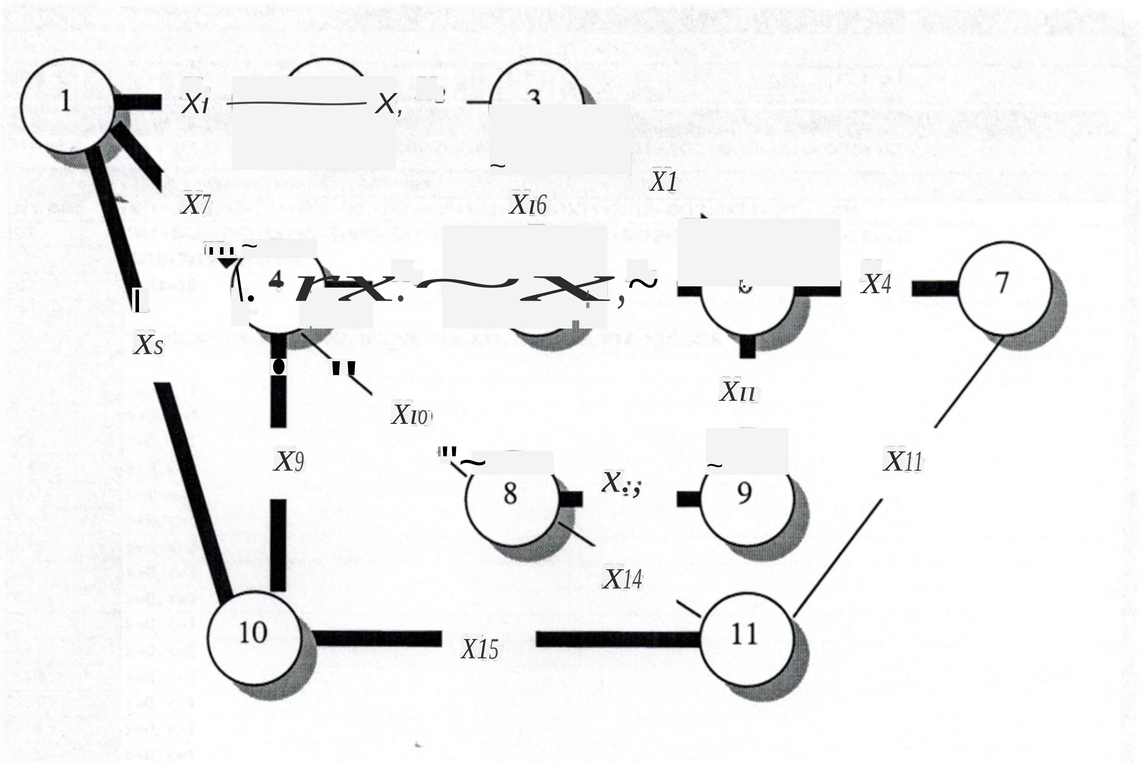
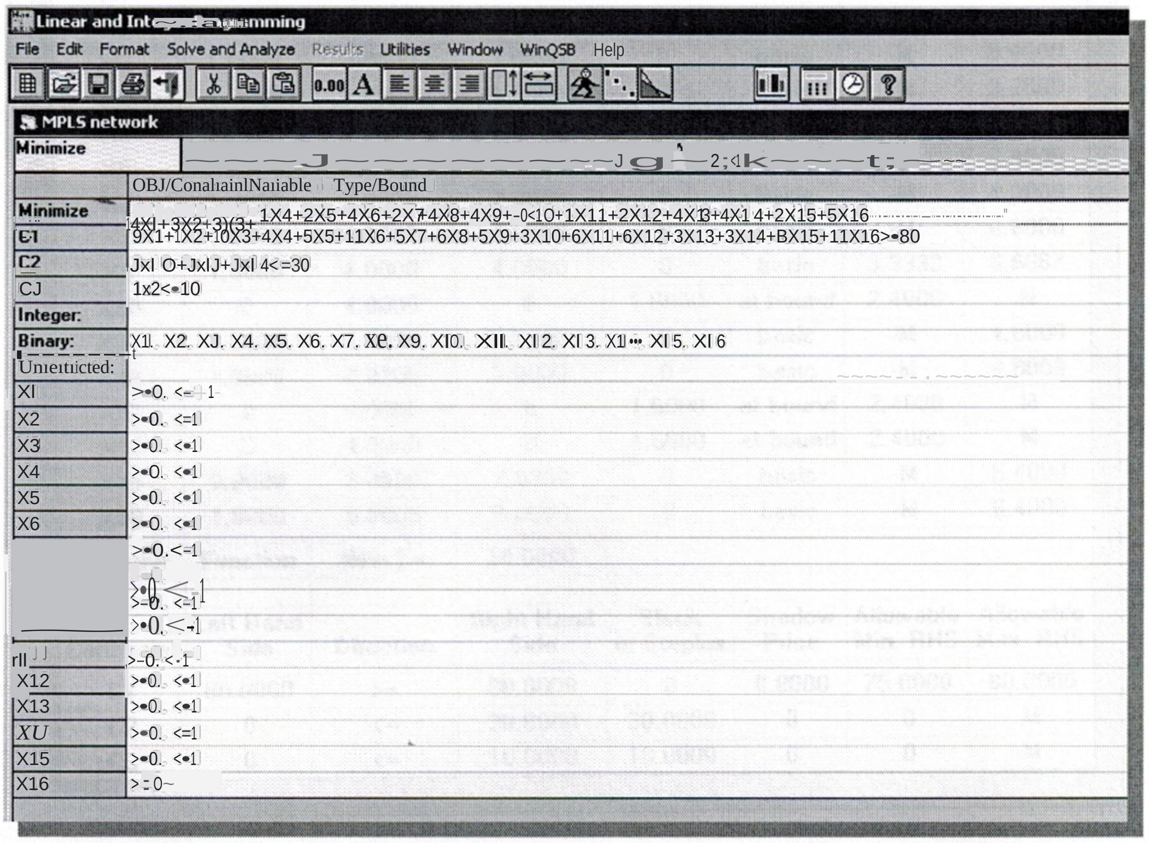


Figure 5.16 *Demonstration of optimized routing mechanism, how the candidate paths are assigned first, and then the shortest path is calculated.*

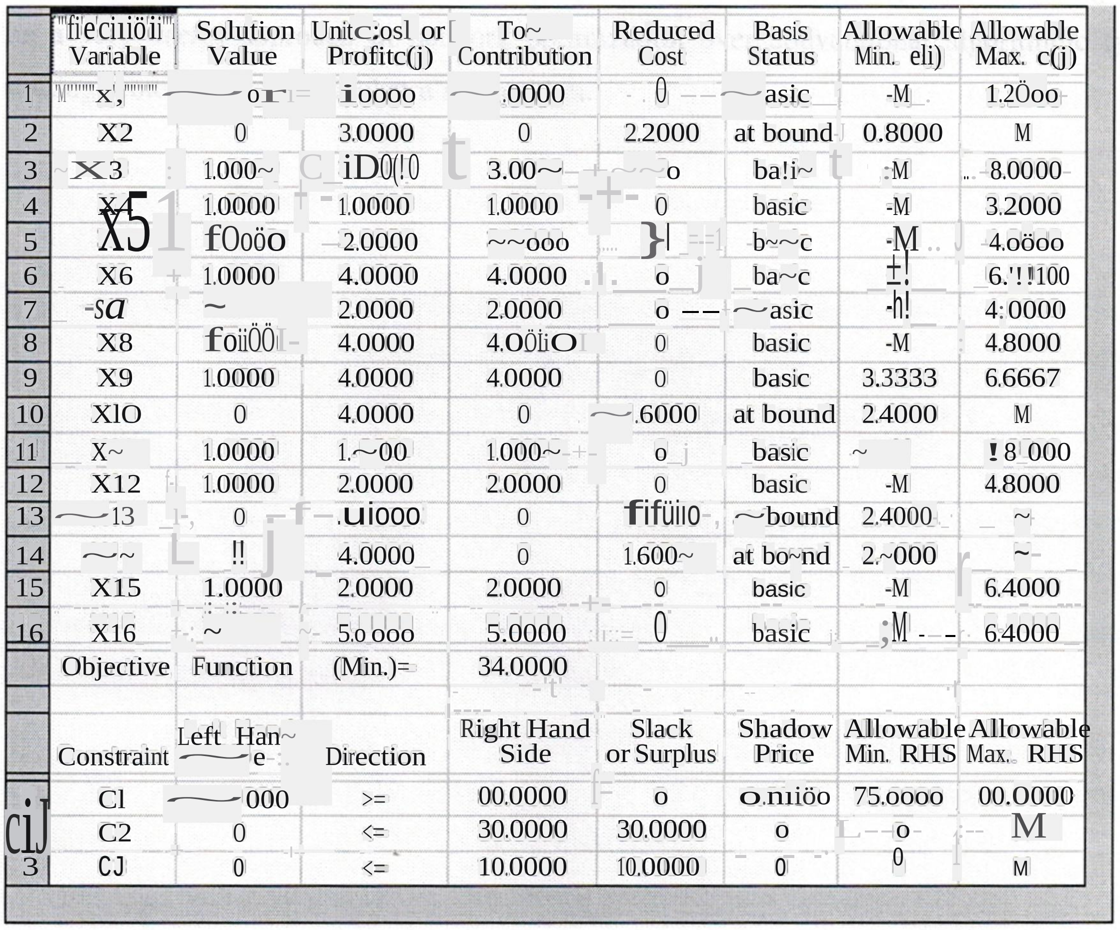
Below is a view of the Win QSB™ solver, this is where the module is written into the solver, and up to 16 limiting constraints can be added into this solver. It will then work its way, using branch and bound approach discussed in chapter 2 and it finally display the results based on the reliability requirements. Sometimes the solver converges to the solution on other times it can only be as close as possible.


Linear and Integer Programming
File Edit Format Solve and Analyze Results Utilities Window WinQSB Help
MPLS network

	OBJ/Constraint/Variable Type/Bound
Minimize	4X1+3X2+3X3+1X4+2X5+4X6+2X7+4X8+4X9+-0<10+1X11+2X12+4X13+4X14+2X15+5X16
C1	9X1+1X2+10X3+4X4+5X5+11X6+5X7+6X8+5X9+3X10+6X11+6X12+3X13+3X14+BX15+11X16>=80
C2	Jxl O+JxlJ+Jxl 4<=30
CJ	1x2<=10
Integer:	
Binary:	X1, X2, XJ, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16
Unrestricted:	
X1	>=0, <=1
X2	>=0, <=1
X3	>=0, <=1
X4	>=0, <=1
X5	>=0, <=1
X6	>=0, <=1
	>=0,<=1
	>=0,<=1
	>=0, <=1
	>=0,<=1
rll	>=0, <=1
X12	>=0, <=1
X13	>=0, <=1
XU	>=0, <=1
X15	>=0, <=1
X16	>=0~

Figure 5.17 *The mathematical model is fed into the solver, objective function and constraints.*

In figure 4.17, the output after running Win QSB™. The solution value 1 means that this path will be used or its operational. If the solution value is zero it means that the path is not used or not operational. In this module the solution of 80% reliability is achieved in this network. Only 4 paths out of 16 will not be operational. The rest of the paths are operational or candidates to route data from any two nodes in the network.



	"fiecii iöii" Variable	Solution Value	Unitc:osl or Profitc(j)	To~ Contribution	Reduced Cost	Basis Status	Allowable Min. eli)	Allowable Max. c(j)
1	x,	or	ioooo	.0000	0	asic	-M	1.2Öoo
2	X2	0	3.0000	0	2.2000	at bound	0.8000	M
3	X3	1.000~	iD0(!0	3.00~	o	ba!i~	:M	8.0000
4	X4	1.0000	1.0000	1.0000	0	basic	-M	3.2000
5	X5	fOoöo	2.0000	~~ooo	}|	b~~c	-M	4.oöoo
6	X6	1.0000	4.0000	4.0000	o	ba~c	±!	6.'!!100
7	sa	~	2.0000	2.0000	o	~asic	-h!	4:0000
8	X8	foiiÖÖ	4.0000	4.OÖliO	0	basic	-M	4.8000
9	X9	1.0000	4.0000	4.0000	0	basic	3.3333	6.6667
10	XlO	0	4.0000	0	.6000	at bound	2.4000	M
11	X~	1.0000	1.~00	1.000~	o	basic	~	!8 000
12	X12	1.0000	2.0000	2.0000	0	basic	-M	4.8000
13	13	0	.uiooo	0	fifüll0	~bound	2.4000	~
14	~~	!!	4.0000	0	1.600~	at bo~nd	2.~000	~
15	X15	1.0000	2.0000	2.0000	0	basic	-M	6.4000
16	X16	~	5.o ooo	5.0000	0	basic	;M	6.4000
	Objective	Function	(Min.)=	34.0000				
	Constraint	Left Han~e	Direction	Right Hand Side	Slack or Surplus	Shadow Price	Allowable Min. RHS	Allowable Max. RHS
1	Cl	000	>=	00.0000	o	o.nuöo	75.oooo	00.0000
2	C2	0	<=	30.0000	30.0000	o	o	M
3	CJ	0	<=	10.0000	10.0000	0	0	M

Figure 5.18 *results from the solver, indicating wither the path is a candidate path or not.*

5.5 Summary

In this chapter, the model discussed in chapter three is solved mathematically; the Dijkstra's algorithm is implemented. Generations ofrouting metrics streams are used in the solver. The results in a graph from are displayed for both cases, showing the results of conventional and optimized routing. This leads to the conclusion that optimized routing can be a very useful approach in network optimization over conventional algorithmic type routing protocols that use Dijkstra's algorithm.

CONCLUSION

The Internet and computer networks have quickly evolved into a very critical communications infrastructure, supporting significant economic, educational, and social activities. The delivery of network communications services has become very competitive and end-users are demanding very high quality service from their service providers. Consequently, the performance of large-scale IP networks, especially public Internet backbones, has become an important problem. Network performance requirements are multi-dimensional, complex, and sometimes contradictory making the traffic-engineering problem very challenging.

-In this thesis, the congestion problem of IP networks is addressed. Using a mathematical optimization approach, an implementation and simulation of the mathematical model took place to use in IP networks based on the fact that all links cannot be equally under the same load, or be of the same capacity and have different link reliability measures. The model solves this problem, using integer programming to effectively share the load between all links and at the same time maintains the reliability demands constraints. This method is already under 'request for comment' by the Internet engineering task force to be standardized for new generation routing protocols such as MPLS™.

The algorithms used in routing protocols such as OSPF, and other conventional routing protocols are discussed. With a conclusion that conventional protocols, uses Dijkstra'a algorithm to determine the shortest path. An implementation of this algorithm was programmed using MS-Visual Basic ™. Due to the nature of this problem the optimization package Win QSB™ was used to solve the mathematical module. A network generator was also developed to give random and realistic data packets values to use as network metrics. Finally the results were charted and graphically displayed from conventional routing and from optimized routing, showing the results of both approaches.

REFERENCES

[1] Mathias Johnson (2001) Network optimization: An overview, Alkit Communications. Available from the World Wide Web: http://www.research.att.com/-mthorik/PAPERS/ papers.html.

[2] Yih-Long Chang (2001) Win QSB Quantitative systems for business plus (version 1.0), Wiley publishing.

[3] Yih-Long Chang and Robert S. Sullivan (1998). QSB+, Quantitative systems for business plus (version 2.1). Prentice-Hall publishing.

[4] Dr. Alpin Clark (1998). Quick Quant Plus. Published by Prentice-Hall.

[5] Huan Pham, Bill Lavery. (2002). An Improved Method for Determining Link Weights for Optimising OSPF Routing, School of Information Technology, James Cook University, Qld 4811, Australia. Available from the World Wide Web: http://www.research.att.com/-mthorup/P ¯'APERS/papers.html.

[6] Roberto Alexandre Dias, Eduardo Camponogara (2001). Using Lagrangian relaxation to improve performance on IP Networks Over MPLS. Available from the World Wide Web: http://www.research.att.com/-mthorup/P ¯'APERS/papers.html.

[7] Alex Chpenst and Tommy Curran (1999) Optimization of QoS traffic with a number of constraints in DiffServ/MPLS networks, Dublin City University, Teltec. Available from the World Wide Web: http://www.research.att.com/-mthorup/PAPERS/ papers.html.

[8] Berna Dengiz, Fulya Altiparmak, Alice E. Smith. (2003) A Genetic Algorithm Approach To Optical Topological Design Of All Terminal Networks, Available from the World Wide Web: http://www.research.att.com/-mthorup/P 'APERS/papers.html.

[9] Optimization Online Digest (2003), available from the World Wide Web at: http://www.optimization-online.org/ARCHIVE_DIGEST/2003-12.html

[10] John W. Chinneck (2004) Practical Optimization: A Gentle Introduction, Systems and Computer Engineering, Carleton University, Ottawa, Ontario KIS 5B6, Canada.
Available at the World Wide Web http://www.sce.carleton.ca/faculty/chinneck/po.html

[11] Pr.Qf. Paul A. Jensen (2003) A course in optimization, available from the World Wide Web at http://www.me. utexas .edu/-jensen/ORMM/omie/ design/unit/process /index .html

[12] George Leitman (1999) Optimization Techniques, With Applications to Aerospace systems, Edited by, Academic Press

[13] Mathematics department of the University of British Columbia (2001), available from the web http://www.ugrad.math.ubc.ca/coursedoc/math _100/notes/index.html#apps

[14] Harvey J. Greenberg (2002). Dieticians and linear programming, A Short Course in Linear Programming available at http://www.cudenver.edu/-hgreenbe/myadr.html

[15] R. S. Jackson (2000). Routing in modern networks. Cisco Network Academy, Cisco Press.

[16] D,Z,Du and P,M, Pardalos (1998) Handbook of Combinatorial Optimization (Vol. 2), Kluwer Academic Publishers.

[17] Introduction to Scientific Programming, available from the World Wide Web, at http://www.cs.utah.edu/-zachary/IntroSciProg.html

[18] Optimization Technology Center of Northwestern University and Argonne National Laboratory available from the World Wide Web at http://www.ece.nwu.edu/OTC/

[19] Linear programming frequently asked question available from the World Wide Web at http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html#Ql

[20] The Internet Traffic Report available from the World Wide Web at: "http://www.intemetreport.com"

[21] Integer programming frequently asked question available from the World Wide Web at http ://w~w-unix.mcs .anl. gov/otc/Guide/faq/inte ger-programming-fag .html#Q 19

[22] Rod Stephens. (2000). Visual Basic and Algorithms, Que Publishing.

[23] B. Fortz and M. Thorup, (2003) "Optimizing OSPF/IS-IS Weights in a Changing World", Available from the World Wide Web at: http://www.research.att.com/ -rnthorup/P APERS/papers.html.

[24] Connected: An Internet Encyclopedia, OSPF-2 Protocol Overview. Available from the World Wide Web at "http://www.freesoft.org/CIE/index.htm."

[25] Prof. Kirk Mitchell (2004). Networking upside down, Published by Pearson education.

[26] Francesco Balena (1999). Programming in visual basic 6, Microsoft press.

[27] Robert Sandilands (2000). Traffic™, A client, server network simulator. Available from the World Wide Web at: www.netsoft.com/traffic.html

[28] Michele Trick (2001) Optimization notes, available from the World Wide Web at http ://mat. gsia.cmu .edu

[29] Lindo™. (2002). A large scale optimization package available from the World Wide Web at http://www.lindo.com

[30] Network Simulator™. Available from the World Wide Web at: http://www.mesh-cs.berkeley.edu/ns

[31) Darren Redfern, Colin Cambell. (2000). The MATLAB 5 Handbook, Springer Publishings.

[32) Mathmatica™, the official web site, available from the World Wide Web at: http://www.mathmatica.com

[33) Maple™, the official web site, available from the World Wide Web at: http://www.maple.com

[34] Mitch Tulloch (2000), Microsoft Encyclopedia Of Networking, Microsoft Press.

[35) S. Kluwer (2002) OPTIMIZATION and ENGINEERING, Academic Publishers

[36] Richard S. Barr. (2000) shortest path and optimzation, available from the World Wide Web at http://engr.srnu.edu/-barr/ip/ch0/nodel4.html#SECTION0004030000000 0000000

[37] The international Neos™ optimization Server, available from the World Wide Web at: http ://www-fp .mes .anl.gov /otc/Guide/CaseStudies/sirnplex/definitionhtml

[38) The national institute of standards and technology, algorithms. Available from the World Wide Web at: http://www.nist.gov/dads/HTMUalgorithm.html

[39] R. K. Ahuja, T. L. Magnati, J. B Orlin, (1993) "Network flows: Theory, algorithms, and applications," Prentice Hall Press.

[40) T. H. Cormen ; C. E. Leiserson ; R. L. Rivest. (1990) Introduction to Algorithms

The MIT Press, Cambridge, Mass.

[41] Survivable Network Topologies Homepage available from the World Wide Web at: http://www.sie.arizona.edu/ faculty/cole/snt.html#Description of Research

[42] I-Lin Wang (2002) Implementing the Premultiplier Method for Minimum-Cost Flow Problem available from the world wide web at http://ilin.iim.ncku.edu.tw/ilin/ contents.html#contents

[43] Disjkstra's Algorithm explanations and uses in industry. Available from the World Wide Web at: http://www-b2.is.tokushima-u.ac.jp/-ikeda/suuri/dijkstra /Dijkstra.shtml.en

[44]D.Awduche Movaz, Chiu Celion, ElwalidI Widjaja, X. Xiao (2002). Overview and Principles of Internet Traffic Engineering. NetworksA Press. Lucent Technologies , Red back Networks

[45] Cisco Design Guide for OSPF, available from the World Wide Web at: http://www.cisco.com/univercd/cc/td/ doc/cisintwk/ito_doc/routing.htm#xtocidl

[46] The Multi Protocol Label Switching (MPLS™) protocol overview. Available from the world wide web at: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ mpls_tsw .htm#xtocid 1

[47] Network Working Group, Request for Comments: 3272. Available from the World Wide Web at: http://www.faqs.org/rfcs/rfcsearch.htrnl

[48] Rosen, E., Viswanathan, A. and R. Callon (2001)[RFC-3031], "Multiprotocol Label Switching Architecture", available from the World Wide Web at: http://asg.web.cmu.edu/ rfc/rfc3031 .html

[49] X. Xiao, A. Hannan, B. Bailey, L. Ni, (May 2000). "Traffic Engineering with MPLS in the Internet", IEEE Network magazine, IEEE publishing

[50] Using the OSPF Protocol. Available from the World Wide Web at: http://www.nx.networks.com/Docs/software_documentation/openroute_2l/protl/ospfJ.htm #2953

[51] Cisco Internet design guide. (2004). Available from the World Wide Web at: http://www.cisco.com/uni vercd/ cc/td/ doc/ cisintw k/idg4/index .htm

[52] Benjamin Baran, Fabian Laufer. (2001). Topological optimization ofreliable networks using A-teams. National computer center, National university of Asuncion, University campus of San Lorenzo - Paraguay. Available from the World Wide Web at: www.utah.edu/papers.shtml

[53] Samuel Pierre (2003) Inferring New Design Rules by Machine Learning: A Case Study of Topological Optimization. IEEE symposiums CD, IEEE publishing.

[54] Tom Schouwenaars, Bart De Moor, Eric Feron, Jonathan HowEsat-Sista, (2003). MIXED INTEGER PROGRAMMING FOR MULTI-VEHICLE PATH PLANNING. Katholieke Universiteit Leuven, Belgium, Laboratory for Information and Decision Systems, Space Systems Laboratory, Massachusetts Institute of Technology, Cambridge Press, MA 02139

APPENDIX A

Charting Stage:

```
Private Const MARGIN_SIZE = 60
Private Const SHAPE_COMMAND = "SHAPE {select xl,yl from Table l} AS
ChildCommand COMPUTE ChildCommand, AVG(ChildCommand.[yl]) AS [yl] BY
[xl]"
Private Const CONNECT_STRING = "PROVIDER=MSDataShape;Data
Source=C:\DB I.mdb;Data Provider=Microsoft.J et.OLEDB.3.51"
Private Const FIELD_X = "xl"
Private Const FIELD_Y = "yl "
Private Const FIELD_Z = ""
Private Const VBERR_INVALID_PROCEDURE_CALL = 5
Private Const MARKERS_VISIBLE = 0
Private Const BRACKET_LEFT = "["
Private Const BRACKET_RIGHT = "]"
Private Const SPACE_CHAR =""

Private Sub chtReport_OLEStartDrag(Data As MSChart20Lib.DataObject, AllowedEffects
As Long)

End Sub

Private Sub cmdClose_Click()
   Unload Me
End Sub

Private Sub DisplayError(oError As ErrObject)
   MsgBox oError.Description, vbExclamation, App.Title
End Sub

Private Sub Form_Load()
   Dim conShape As ADODB.Connection
   Dim recShape As ADODB.Recordset
   On Error GoTo Form_Load_Error
Set conShape = New ADODB.Connection
   conShape.Open CONNECT _STRING
Set recShape = New ADODB.Recordset
   recShape.Open SHAPE_COMMAND, conShape
ShowRecordslnChart recShape, FIELD_X, FIELD_Y, FIELD_Z
ShowMarkers MARKERS_VISIBLE
   Exit Sub
Form_Load_Error:
  DisplayError Err
  Exit Sub
End Sub

Private Sub Form_Resize()
  Dim sngButtonTop As Single
  Dim sngScale Width As Single
  Dim sngScaleHeight As Single

  On Error GoTo Form_Resize_Error
  With Me
    sngScale Width = .Scale Width
    sngScaleHeight = .ScaleHeight
    With .cmdClose
      sngButtonTop = sngScaleHeight - (.Height+ MARGIN_SIZE)
      .Move sngScaleWidth - (.Width+ MARGIN_SIZE), sngButtonTop
    End With
    .chtReport.Move MARGIN_SIZE, _
                MARGIN_SIZE, _
                sngScale Width - (2 * MARGIN_SIZE), _
                sngButtonTop - (2 * MARGIN_SIZE)
  End With
  Exit Sub
Form_Resize_Error:
  'An error will occur if the user sizes
  'the form so small that negative heights
  'or widths are calculated
  Resume Next
End Sub
Private Function IsKeylnCollection(cCol As Collection, sKey As String) As Boolean
  Dim v As Variant
  On Error Resume Next
  v = cCol.Item(sKey)
 IsKeylnCollection =(Err.Number<> VBERR_INVALID_PROCEDURE_CALL)
  Err.Clear
End Function

Private Sub ShowMarkers(bShow As Boolean)
  Dim i As Long
  On Error GoTo ShowMarkers_Click_Error
  With chtReport.Plot
    For i = 1 To .SeriesCollection.Count
      .SeriesCollection(i).SeriesMarker.Show = bShow
    Next
  End With
  Exit Sub
```

```
Show Markers_Click_Error:
   DisplayError Err
   Exit Sub
End Sub

Private Sub ShowRecordslnChart(recParent As Recordset, _
                  sFldX As String, _
                  sFldY As String, _
                  sFldZ As String)

   Dim bUseZ As Boolean
   Dim cRows As Collection
   Dim cCols As Collection
   Dim !Col As Long
   Dim !Row As Long
   Dim lMaxCol As Long
   Dim lMaxRow As Long
   Dim s¥alue As String

   On Error GoTo ShowRecordslnChart_Error
   If Len(sFldZ) = 0 Then bUseZ = False Else bUseZ = True

   Set cRows = New Collection
   Set cCols = New Collection

   With Me.chtReport

      .Repaint = False
      With .DataGrid

         .DeleteRows 1, .RowCount
         .DeleteColwnns 1, .ColwnnCount
         .DeleteColumnLabels 1, .Columnl.abelf'ount
         .DeleteRow Labels 1, .Row Label Count

         .InsertColumnLabels 1, 1
         .InsertRow Labels 1, 1

         If Not bUseZ Then .InsertColumns 1, 1
         recParent.MoveFirst
         Do Until recParent.EOF

            sValue = FixNull(recParent.Fields(sFldX).Value, False)
            If Not IsKeyInCollection( cRows, sValue) Then
               !Max.Row = !Max.Row + 1
            lRow = lMaxRow

            cRows.Add lRow, sValue
            .InsertRows lRow, 1
            .RowLabel(lRow, 1) = sValue
          Else
            lRow = cRows.Item(sValue)
          End If


          If bUseZ Then
            sValue = FixNull(recParent.Fields(sFldZ).Value, False)
            If Not IsKeyInCollection(cCols, sValue) Then
              lMaxCol = lMaxCol + 1
              lCol = lMaxCol
              cCols.Add lCol, sValue
              .InsertColumns lCol, 1
              .ColumnLabel(lCol, 1) = sValue
            Else
              lCol = cCols.Item(sValue)
            End If
            .SetData lRow, lCol, FixNull(recParent.Fields.Item(sFldY).Value, True), 0
          Else
            .SetData lRow, 1, FixNull(recParent.Fields.Item(sFldY).Value, True), 0
          End If

          recParent.MoveNext
        Loop
      End With
      .Repaint = True
    End With
    Exit Sub
Show RecordsInChart_Error:
    Me.chtReport.Repaint = True
    DisplayError Err
    Exit Sub
End Sub
Private Function FixNull(vField As Variant, _
            bNumericRequired As Boolean) As Variant
    If IsNull(vField) Then
      If bNumericRequired Then
        FixNull = 0
      Else
        FixNull = vbNullString
      End If
    Else
    FixNull = vField
  End If
End Function
```

Dijkstra's Algorithm Stage:

```
Option Explicit

Private m_ArrayData() As String

Private Type NodePoint
  x As Long
  Y As Long
End Type

Private Const nNodes As Long= 11  '****************
Dim NodeList(0 To nNodes - 1) As NodePoint '****************

Private Type TreeNode
  CurrNode As Long
  NextNode(0 To 3) As Long
  Dist(0 To 3) As Double

  'DIJKSTRAs ALGORITHM ...

  VisitNumber As Long
  Distance As Double
  TmpVar As Double
End Type

Dim TreeNodeList(0 To nNodes - 1) As TreeNode

Dim nPathList As Long
Dim PATHLIST() As Long
Dim CurrDestNode As Long
Dim CurrSrcNode As Long

Private Declare Function GetTickCount Lib "kernel32" () As Long
Private Declare Function QueryPerformanceCounter Lib "kernel32" (lpPerformanceCount
As Currency) As Long
Private Declare Function QueryPerformanceFrequency Lib "kernel32" (lpFrequency As
Currency) As Long
```

```
Private Function DijkstraPathFinding(NodeSrc As Long, NodeDest As Long) As Boolean

  Dimi As Long
  Dim bRunning As Boolean
  Dim CurrentVisitNumber As Long
  Dim CurrNode As Long
  Dim LowestNodeFound As Long
  Dim LowestValFound As Double

  If NodeSrc = NodeDest Then

    nPathList = 2
    ReDim PATHLIST(2) As Long
    PATHLIST(1) = NodeSrc
    PATHLIST(2) = NodeDest
    DijkstraPathFinding = True
    Exirfunction
  End If

'///. Setup all the data
  For i = 0 To nNodes - 1
    TreeN odeList(i). VisitNumber = -1
    TreeNodeList(i).Distance = -1
    TreeNodeList(i).TmpVar = 99999
  Next i

  '//Set the first variable
  TreeNodeList(NodeSrc).VisitNumber = 1
    CurrentVisitNumber = 1
    CurrNode = NodeSrc
  TreeNodeList(NodeSrc).Distance = 0
  TreeNodeList(NodeSrc).TmpVar = 0

'//2. Start scanning

  Do While bRunning = False

        If      Not      (TreeNodeList(CurrNode).NextNode(0)      =      -1)      Then
TreeNodeList(TreeNodeList(CurrNode).NextNode(0)).TmpVar                          =
MIN(TreeNodeList(CurrNode).Dist(0)         +         TreeNodeList(CurrNode).Distance,
TreeNodeList(TreeNodeList(CurrNode).NextNode(0)).TmpVar)
        If      Not      (TreeNodeList(CurrNode).NextNode(l)      =      -1)      Then
TreeNodeList(TreeN ode List( CurrN ode) .N extN ode( 1)). Tmp Var                =
MIN(TreeNodeList(CurrNode).Dist(l)         +         TreeNodeList(CurrNode).Distance,
TreeNodeList(TreeNodeList(CurrNode ).NextNode(l) ).Tmp Var)
```

```
            If Not (TreeNodeList(CurrNode).NextNode(2) = -1) Then
TreeNodeList(TreeNodeList(CurrNode).NextNode(2)).TmpVar =
MIN(TreeNodeList(CurrNode).Dist(2) + TreeNodeList(CurrNode).Distance,
TreeNodeList(TreeNodeList(CurrNode).NextNode(2)).TmpVar)
            If Not (TreeNodeList(CurrNode).NextNode(3) = -1) Then
TreeNodeList(TreeNodeList(CurrNode).NextNode(3)).TmpVar =
MIN(TreeNodeList(CurrNode).Dist(3) + TreeNodeList(CurrNode).Distance,
TreeNodeList(TreeNodeList(CurrNode).NextNode(3)).TmpVar)


            LowestValFound = 100999
            For i = 0 To nNodes - 1
              If (TreeNodeList(i).TmpVar <= LowestValFound) And
(TreeNodeList(i).TmpVar >= 0) And (TreeNodeList(i).VisitNumber < 0) Then 'make sure
we ignore the -1's and visited nodes
                LowestValFound = TreeNodeList(i).TmpVar
                LowestNodeFound = i
              End If
            Next i



            CurrentVisitNumber = CurrentVisitNumber + 1
            TreeNodeList(LowestNodeFound).VisitNumber = CurrentVisitNumber
            TreeNodeList(LowestNodeFound).Distance =
TreeNodeList(LowestNodeFound).TmpVar
            CurrNode = LowestNodeFound

            If CurrNode= NodeDest Then
              bRunning = True
            Else
              bRunning = False
            End If
         Loop

         bRunning = False
         CurrNode = NodeDest
         Dim lngTimeTaken As Long
         lngTimeTaken = GetTickCount

         nPathList = 1
         ReDim PATHLIST(nPathList) As Long
         PATHLIST(1) = NodeDest
```

```
        Do While bRunning = False

            If CurrNode= NodeSrc Then
              bRunning = True
              GoTo SkipToEnd:
            ElseIf GetTickCount - lngTimeTaken > 1000 Then

              bRunning = True
              DijkstraPathFinding = False
              Exit Function
              GoTo SkipToEnd:
            End If

          If (TreeNodeList(CurrNode).NextNode(0) >= 0) Then
            If  (TreeNodeList(TreeNodeList(CurrNode).NextNode(0)).VisitNumber   >=   0)
Then
              If              TreeNodeList(CurrNode).Distance                          -
TreeNodeList(TreeNodeList(CurrNode).NextNode(0)).Distance                            =
TreeNodeList(CurrNode).Dist(0) Then

                  nPathList = nPathList + 1
                  ReDim Preserve PATHLIST(nPathList) As Long
                  PATHLIST(nPathList) = TreeNodeList(CurrNode ).NextNode(0)
                  CurrNode = TreeNodeList(CurrNode).NextNode(0)
                  GoTo SkipToEnd:
              End If
            End If
          End If

          If (TreeNodeList(CurrNode).NextNode(l) >= 0) Then
            If  (TreeNodeList(TreeNodeList(CurrNode).NextNode(l)).VisitNumber   >=   0)
Then
              If              TreeNodeList(CurrNode).Distance                          -
TreeNodeList(TreeNodeList(CurrNode).NextNode(l)).Distance                            =
TreeNodeList(CurrNode).Dist(l) Then

                  nPathList = nPathList + 1
                  ReDim Preserve PATHLIST(nPathList) As Long
                  PATHLIST(nPathList) = TreeN odeList( CurrN ode) NextN ode(l)
                  CurrNode = TreeNodeList(CurrNode).NextNode(l)
                  GoTo SkipToEnd:
              End If
            End If
          End If
```

```
            If (TreeNodeList(CurrNode).NextNode(2) >= 0) Then
               If  (TreeNodeList(TreeNodeList(CurrNode).NextNode(2)).VisitNumber   >=  0)
Then
                  If                          TreeNodeList(CurrNode).Distance                    -
TreeNodeList(TreeNodeList( CurrN ode).N extN ode(2)) .Distance                               =
TreeN odeList( CurrN ode) .Dist(2)  Then

                        nPathList  = nPathList + 1
                        ReDim Preserve PATHLIST(nPathList) As Long
                        PATHLIST(nPathList) = TreeNodeList(CurrNode ).NextNode(2)
                        CurrNode = TreeNodeList(CurrNode).NextNode(2)
                        GoTo SkipToEnd:
                  End If
               End If
            End If

            If (TreeNodeList(CurrNode).NextNode(3) >= 0) Then
               If  (TreeNodeList(TreeNodeList(CurrNode).NextNode(3)).VisitNumber   >=  0)
Then
                  If                          TreeNodeList( CurrN ode) .Distance                 -
TreeNodeList(TreeNodeList(CurrNode).NextNode(3)).Distance                                    =
TreeNodeList( CurrN ode) .Dist(3)  Then

                        nPathList = nPathList + 1
                        ReDim Preserve PA THLIST(nPathList) As Long
                        PATHLIST(nPathList)  = TreeN odeList( CurrN ode ).N extNode(3)
                        CurrNode = TreeNodeList(CurrNode).NextNode(3)
                        GoTo SkipToEnd:
                  End If
               End If
            End If

SkipToEnd:

        Loop


    Dim TmpArray() As Long
    ReDim TmpArray(nPathList) As Long
    For i = nPathList To 1 Step -1
        TmpArray(i) = PATHLIST(((nPathList - i) + 1))
    Next i
    For i = 1 To nPathList
        PATHLIST(i) = TmpArray(i)
    Next i
```

```
DijkstraPathFinding = True
End Function

Public Function MIN(A As Double, B As Double) As Double

   If A < B Then MIN = A
   If A> B Then MIN = B
   If A = B Then MIN= A
End Function

Private Sub DrawShortestPath()

   Dimi As Long
   picMain.DrawWidth = 4

   For i = 1 To (nPathList - 1)
      pkMain.Line (NodeList(TreeNodeList(PATHLIST(i)).CurrNode).x,
NodeList(TreeNodeList(PATHLIST(i)).CurrNode).Y)-
(NodeList(TreeNodeList(PATHLIST(i + 1)).CurrNode).x,
NodeList(TreeNodeList(PATHLIST(i + 1)).CurrNode).Y), RGB(0, 255, 0)
   Next i
End Sub


Private Function ScanForNearestNode(x As Long, Y As Long) As Long
   Dimi As Long
   Dim CurrClosestlndex As Long
   Dim CurrClosestDist As Long
   CurrClosestDist = 99999

For i = 0 To nNodes - 1
   If GetDist2D(NodeList(i).x, NodeList(i).Y, x, Y) < CurrClosestDist Then
      CurrClosestDist = GetDist2D(NodeList(i).x, NodeList(i).Y, x, Y)
      CurrClosestlndex = i
   End If


Next i

ScanForNearestNode = CurrClosestlndex
End Function

Private Function GetDist2D(x As Long, Y As Long, Xl As Long, Yl As Long) As Long
   GetDist2D = Sqr(((x - Xl) ^ 2) + ((Y - Yl ) ^ 2))
```

```
End Function

Private Sub RenderNodePoints()
Dim i As Long

picMain.DrawWidth = 4

For i = 0 To nNodes - 1
   picMain.PSet (NodeList(i).x, NodeList(i).Y), RGB(255, 0, 0) 'red nodes
Next i
End Sub

Private Sub RenderTreeNodes()
Dim i As Long

picMain.DrawWidth = 1

For i = 0 To nNodes - 1
   If Not (TreeNodeList(i).NextNode(0) = -1) Then picMain.Line
(NodeList(TreeNodeList(i).CurrNode).x, NodeList(TreeNodeList(i).CurrNode).Y)-
(NodeList(TreeNodeList(i).NextNode(0)).x, NodeList(TreeNodeList(i).NextNode(0)).Y),
RGB(0, 0, 255) 'blue paths
   If Not (TreeNodeList(i).NextNode(1) = -1) Then picMain.Line
(NodeList(TreeNodeList(i).CurrNode).x, NodeList(TreeNodeList(i).CurrNode).Y)-
(NodeList(TreeNodeList(i).NextNode(1)).x, NodeList(TreeNodeList(i).NextNode(1)).Y),
RGB(0, 0, 255)
   If Not (TreeNodeList(i).NextNode(2) = -1) Then picMain.Line
(NodeList(TreeNodeList(i).CurrNode).x, NodeList(TreeNodeList(i).CurrNode).Y)-
(NodeList(TreeNodeList(i).NextNode(2)).x, NodeList(TreeNodeList(i).NextNode(2)).Y),
RGB(0, 0, 255)
   If Not (TreeNodeList(i).NextNode(3) = -1) Then picMain.Line
(NodeList(TreeNodeList(i).CurrNode).x, NodeList(TreeNodeList(i).CurrNode).Y)-
(NodeList(TreeNodeList(i).NextNode(3)).x, NodeList(TreeNodeList(i).NextNode(3)).Y),
RGB(0, 0, 255)
Next i
End Sub


Private Sub cmdclearall_Click()
picMain.Cls
```

```
'Dim I As Long
'For I= 0 To nNodes - 1
'Next I


'picMain.Image  "c:\y.bmp"
End Sub

Private Sub cmdlaodnetwork_Click()
'Private Const nNodes As Long= 9
'Dim NodeList(0 To nNodes - 1) As NodePoint
'nNodes = 9
picMain. Cls
m
'RenderNodePoints
'RenderTreeN odes
End Sub

Private Sub cmdloadnetwork2_Click()
'Private Const nNodes As Long= 40
'Dim NodeList(O To nNodes - 1) As NodePoint

picMain.Cls
d
'RenderNodePoints
'RenderTreeNodes
End Sub

Private Sub cmdSearch_Click()
ProgressBarl.Value = 0
Dim seed As Integer


picMain.Cls
RenderNodePoints
RenderTreeNodes
Dim TimeTaken As Currency, TimeStarted As Currency, Freq As Currency
    Call QueryPerformanceCounter(TimeStarted)
      If DijkstraPathFinding(CurrSrcNode, CurrDestNode) = False Then GoTo BailOut
   Call QueryPerformanceCounter(TimeTaken)
    Call Query PerformanceFrequency(Freq)
    TimeTaken = (TimeTaken - TimeStarted) / Freq

For seed = 1 To 10
```

```
ProgressBarl.Value = ProgressBarl.Value + 10
Next seed
MsgBox "Dijkstra Search took " & TimeTaken & "ms to complete"
ProgressBarl.Value = 0

   Dim tmpstr As String
   Dimi As Long
   For i = 1 To nPathList
      tmpstr = tmpstr & PATHLIST(i) & "-"
   Next i
   lblPath.Caption = "Path: " & Left(tmpstr, Len(tmpstr) - 1) & " (Dist: " &
TreeNodeList(CurrDestNode).Distance & ")"
   Draw ShortestPath

Exit Sub

Bail Out:
   Msgfïox "A path could not be found within 1 second..."
End Sub


' Copy the array data into the grid.
Private Sub CopyArrayToGrid(ByRef m_ArrayData() As String, ByVal grd As
MSFlexGrid)
Dim rmin As Integer
Dim emin As Integer
Dim rmax As Integer
Dim cmax As Integer
Dim r As Integer
Dim c As Integer

   rmin = LBound(m_ArrayData, 1)
   rmax = UBound(m_ArrayData, 1)
   emin= LBound(m_ArrayData, 2)
   cmax = UBound(m_ArrayData, 2)
   grd.Rows = rmax - rmin + 1
   grd.Cols = cmax - emin+ 1
   grd.FixedCols = 0
   grd.FixedRows = 0

   For r = rmin To rmax
      For c = emin To cmax
         grd.TextMatrix(r - rmin, c - emin)= m_ArrayData(r, c)
      Nextc
   Next r
```

```
End Sub
' Make some data.
Private Sub MakeData()
Dim r As Integer
Dim c As Integer

   ReDim m_ArrayData(l To 10, 1 To 4)
   For r = LBound(m_ArrayData, 1) To UBound(m_ArrayData, 1)
      For c = LBound(m_ArrayData, 2) To UBound(m_ArrayData, 2)
         m_ArrayData(r, c) = "(" & Format$(r) & "," & Format$(c) & ")"
      Nextc
   Nextr
End Sub
Private Sub copyarray_Click()
   'Procedure name
   MakeData
   CopyArrayToGrid m_ArrayData, MSFlexGrid 1
End Sllb


Private Sub picMain_MouseDown(Button As Integer, Shift As Integer, x As Single, Y As
Single)
Dim ClosestNode As Long
ClosestNode = ScanForNearestNode(CLng(x), CLng(Y))

lblVisitNumber.Caption = "Visit Number:" & TreeNodeList(ClosestNode).VisitNumber
lblVisitDistance.Caption = "Distance: " & TreeNodeList(ClosestNode).Distance
lblTmpVar.Caption = "Temp Var: " & TreeNodeList(ClosestNode).TmpVar

If Button= 1 Then
   lblClosestSrc.Caption = "Closest Node To Src: " & ClosestNode
   CurrSrcNode = Closestblode
   shpSourceNode.Left = NodeList(ClosestNode).x - 12
   shpSourceNode.Top = NodeList(ClosestNode).Y - 12
Else If Button = 2 Then
   'lblClosestDest.Caption = "Closest Node To Dest: " & ClosestNode
   CurrDestNode = ClosestNode
   shpDestNode.Left = NodeList(ClosestNode).x - 12
   shpDestNode.Top = NodeList(ClosestNode).Y - 12
End If
End Sub

Private Sub d()
```

```
NodeList(0).x  = 10
NodeList(0).Y  = 10
NodeList(l).x  = 120
NodeList(l).Y  = 10
NodeList(2).x  = 250
NodeList(2).Y  = 10
NodeList(3).x  = 120
NodeList(3).Y  = 80
NodeList(4).x  = 250
NodeList(4).Y  = 80
NodeList(5).x  = 350
NodeList(5).Y  = 80
NodeList(6).x  = 500
NodeList(6).Y  = 80
NodeList(7).x  = 250
NodeList(7).Y  = 150
NodeList(8).x  = 350
Nodebist(8).Y  = 150
NodeList(9).x  = 120
NodeList(9).Y  = 250
NodeList(lO).x  = 350
NodeList(lO).Y  = 250
TreeNodeList(0).CurrNode  = 0
   TreeNodeList(0).NextNode(0)  = 1
   TreeNodeList(0).NextNode(l)  = 3
   TreeNodeList(0).NextNode(2)  = 9
   TreeNodeList(0).NextNode(3)  = -1
TreeNodeList(l).CurrNode  = 1
   TreeNodeList(l).NextNode(0)  = 0
   TreeNodeList(l).NextNode(l)  = 2
   TreeNodeList(l).NextNode(2)  = -1
   TreeNodeList(l).NextNode(3)  = -1
TreeNodeList(2).CurrNode  = 2
   TreeNodeList(2).NextNode(0)  = 1
   TreeNodeList(2).NextNode(l)  = 5
   TreeNodeList(2).NextNode(2)  = 4
   TreeNodeList(2).NextNode(3)  = -1
TreeNodeList(3).CurrNode  = 3
   TreeNodeList(3).NextNode(0)  = 0
   TreeNodeList(3).NextNode(l)  = 4
   TreeNodeList(3).NextNode(2)  = 7
   TreeNodeList(3).NextNode(3)  = 9
TreeNodeList(4).CurrNode  = 4
   TreeNodeList(4).NextNode(0)  = 2
   TreeNodeList(4).NextNode(l)  = 3
   TreeNodeList(4).NextNode(2)  = 5
   TreeNodeList(4).NextNode(3)  = -1
TreeNodeList(5).CurrNode  = 5
   TreeNodeList(5).NextNode(0)  = 4
   TreeNodeList(5).NextNode(l)  = 6
   TreeNodeList(5).NextNode(2)  = 2
   TreeNodeList(5).NextNode(3)  = 8
TreeNodeList(6).CurrNode  = 6
   TreeNodeList(6).NextNode(0)  = 5
   TreeNodeList(6).NextNode(l)  = 10
   TreeNodeList(6).NextNode(2)  = -1
   TreeNodeList(6).NextNode(3)  = -1
TreeNodeList(7).CurrNode  = 7
   TreeNodeList(7).NextNode(0)  = 3
   TreeNodeList(7).NextNode(l)  = 8
   TreeNodeList(7).NextNode(2)  = 10
   TreeNodeList(7).NextNode(3)  = -1
TreeNodeList(8).CurrNode  = 8
   Tre~odeList(8).NextNode(0)  = 5
   TreeNodeList(8).NextNode(l)  = 7
   TreeNodeList(8).NextNode(2)  = -1
   TreeNodeList(8).NextNode(3)  = -1
TreeNodeList(9).CurrNode  = 9
   TreeNodeList(9).NextNode(0)  = 0
   TreeNodeList(9).NextNode(l)  = 3
   TreeNodeList(9).NextNode(2)  = 10
   TreeNodeList(9).NextNode(3)  = -1
TreeNodeList(lO).CurrNode  = 10
   TreeNodeList(lO).NextNode(0)  = 7
   TreeNodeList(l0).NextNode(l)  = 9
   TreeNodeList(10).NextNode(2)  = 6
   TreeNodeList(10).NextNode(3)  = -1

'//Fill out the weight information ...
Dimi As Long
For i = 0 To nNodes - 1
     If Not (TreeNodeList(i).NextNode(0) = -1) Then TreeNodeList(i).Dist(0) =
GetDist2D(NodeList(TreeNodeList(i).CurrNode).x,
NodeList(TreeNodeList(i).CurrNode).Y,          NodeList(TreeNodeList(i).NextNode(0)).x,
NodeList(TreeNodeList(i).NextNode(O)).Y)
     If Not (TreeNodeList(i).NextNode(l) = -1) Then TreeNodeList(i).Dist(l) =
GetDist2D(NodeList(TreeNodeList(i).CurrNode).x,
NodeList(TreeNodeList(i).CurrNode).Y,          NodeList(TreeNodeList(i).NextNode(l)).x,
NodeList(TreeNodeList(i).NextNode(l)).Y)
     If Not (TreeNodeList(i).NextNode(2) = -1) Then TreeNodeList(i).Dist(2) =
GetDist2D(NodeList(TreeNodeList(i).CurrNode).x,
NodeList(TreeNodeList(i).CurrNode).Y,           NodeList(TreeNodeList(i).NextNode(2)).x,
NodeList(TreeNodeList(i).NextNode(2)).Y)
    If  Not  (TreeNodeList(i).NextNode(3)  =  -1)  Then  TreeNodeList(i).Dist(3)  =
GetDist2D(NodeList(TreeNodeList(i).CurrNode).x,
NodeList(TreeNodeList(i).CurrNode).Y,           NodeList(TreeNodeList(i).NextNode(3)).x,
NodeList(TreeN odeList( i) .NextN ode(3) ). Y)
Next i
picMain.Cls
RenderNodePoints
RenderTreeNodes
End Sub
```

Network generation Stage:

```
Private Sub Commandl_Click()
s
End Sub

Private Sub m()
Dim j As Integer
Dim i As Integer
Dim t As Integer
Dim flow As Integer
Dim packets As Integer
Dim cost As Integer
Dim limit As Long
Fori= 1 To 16
  t = FormatNumber(Rnd * 10000, 2)

flow= t - 20
packets = FormatNumber(flow / 40, 0)
cost = packets / 50


List l.Addltem t
List2.Addltem flow
List3 .Addltem packets
List4.Addltem cost

Label l.Caption= t
Label2.Caption = flow

Next i

End Sub
```

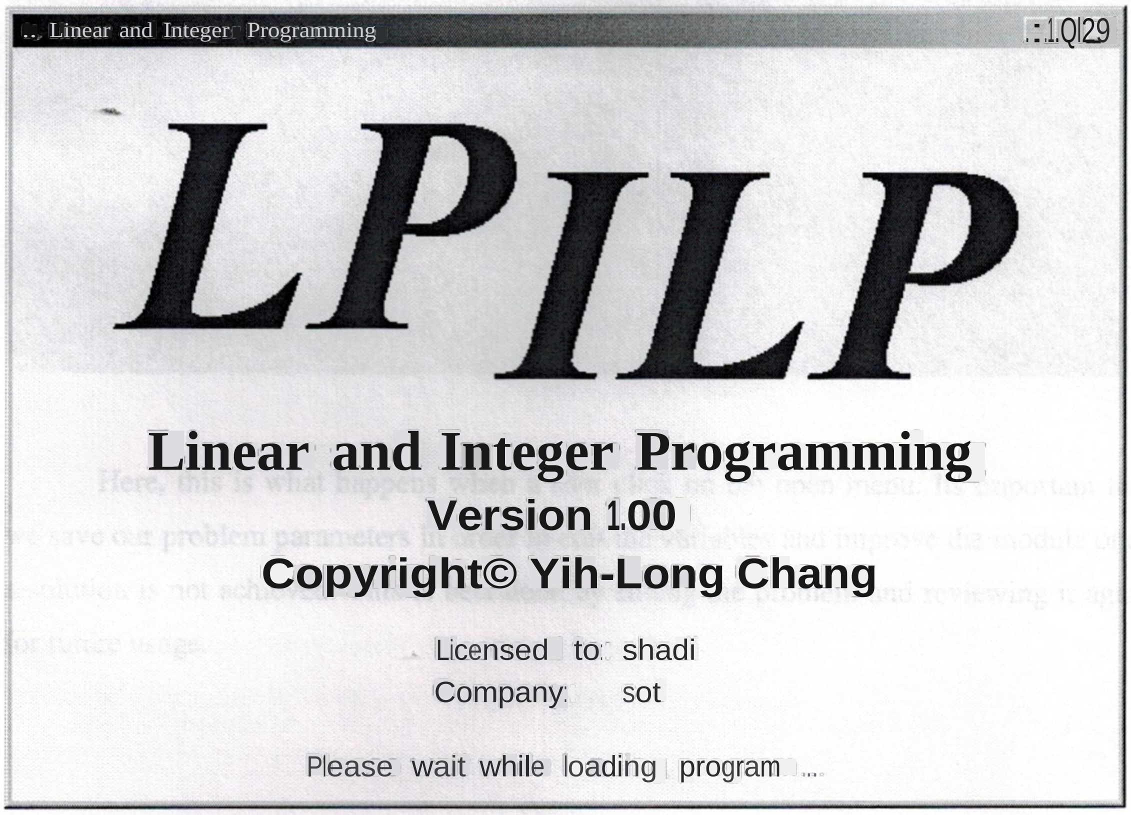
```
Private Sub Command2_Click()

Dim Total As Integer
Total = Listl .ListCount
 MsgBox "Total: " & Total

End Sub

Private Sub Command4_Click()

Listl .Clear
List2.Clear
List3.Clear
List4.Clear
List5.Cfi!ar

End Sub
Private Sub s()

Dim j As Integer
Dim x As Integer
For j = 1 To 16
x = FormatNumber(Rnd * 10, 2)
d = x/10
List5 .Addltem d
Nextj
End Sub

Private Sub Command5_Click()
frmMain.Show
End Sub

Private Sub Command6_Click()
m
End Sub

Private Sub Timerl_Timer()
'm
End Sub
```

APPENDIXB

Using Win QSB

Win QSB™ requires Windows 95/98/Me/2000/XP, 32 Megabytes of Memory and 20 Megabytes of Hard disk space.

As stated before Win QSB™ is well known educational optimization package used to solve small to medium sized optimization problems. Here we start the program from the start menu and point toward the Win QSB™ Linear and Integer Programming Section, the following screen will appear to the user.



Linear and Integer Programming

Version 1.00

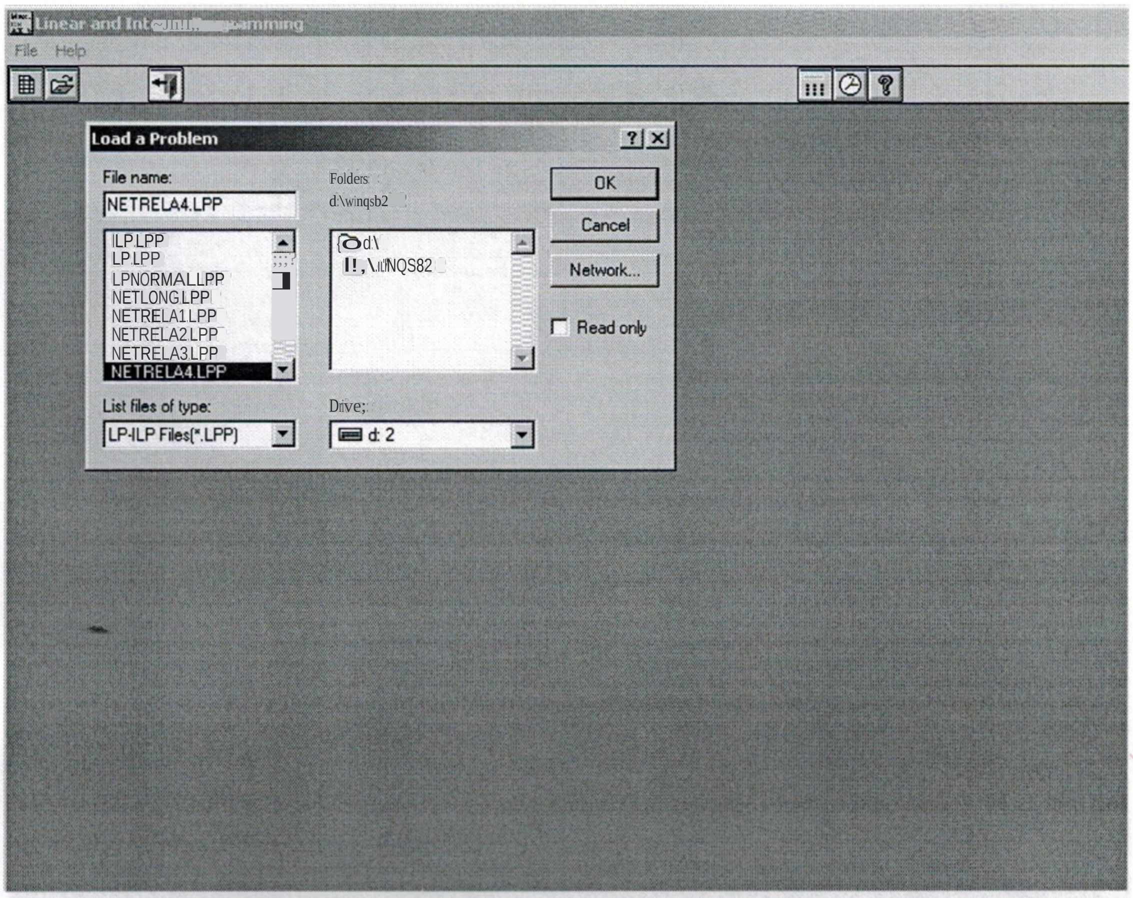
Copyright© Yih-Long Chang

Licensed to: shadi

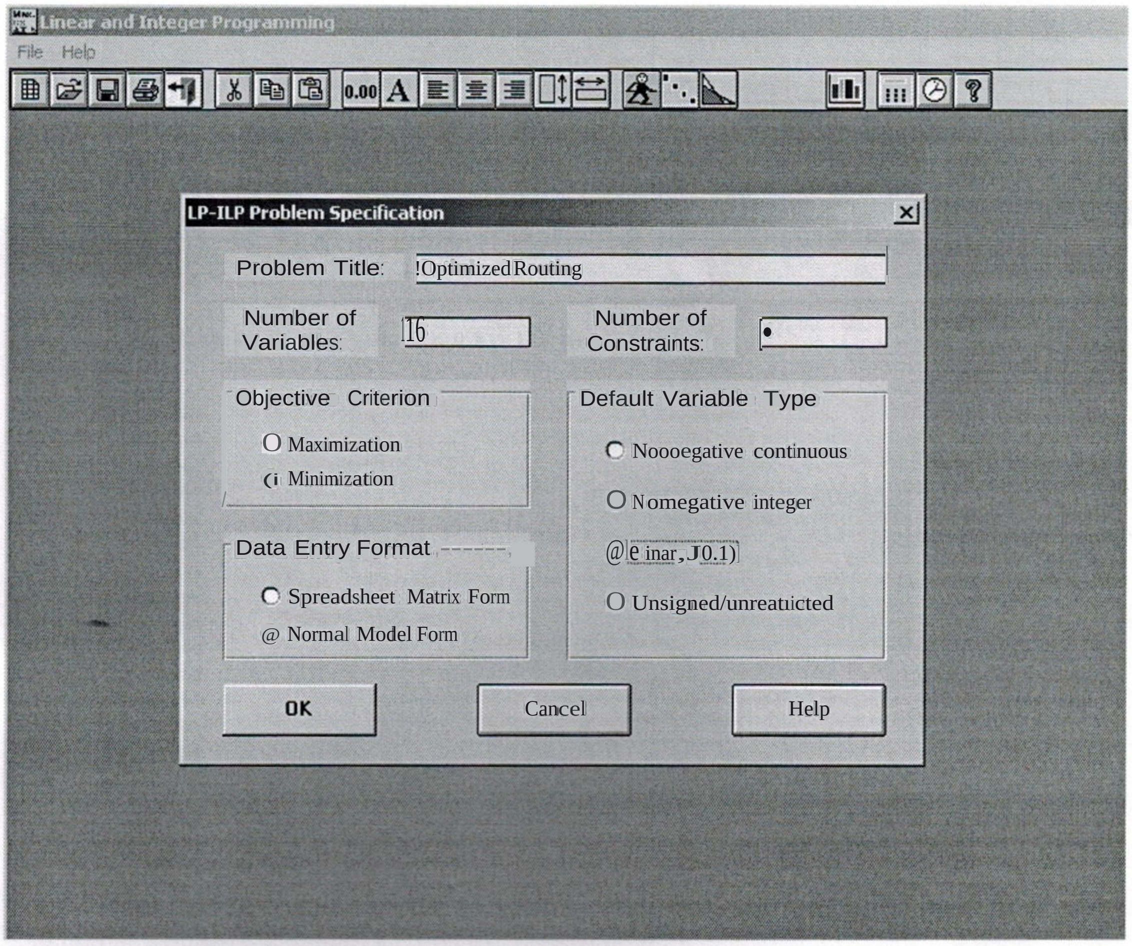
Company: sot

Please wait while loading program ....

Once this splash screen is gone, the user will be presented with a choice of loading a previously solved problem, or introducing a new problem to solve. For this, some information about the new problem must be entered so that the solver can find an optimal solution.



Here, this is what happens when a user click on the open menu. Its important that we save our problem parameters in order to edit the variables and improve the module once a solution is not achieved. This is best done by saving the problem and reviewing it again for future usage.



LP-ILP Problem Specification

Problem Title: Optimized Routing

Number of Variables: 16

Number of Constraints:

Objective Criterion

Maximization [✓ Minimization]

Data Entry Format

Spreadsheet Matrix Form [✓ Normal Model Form]

Default Variable Type

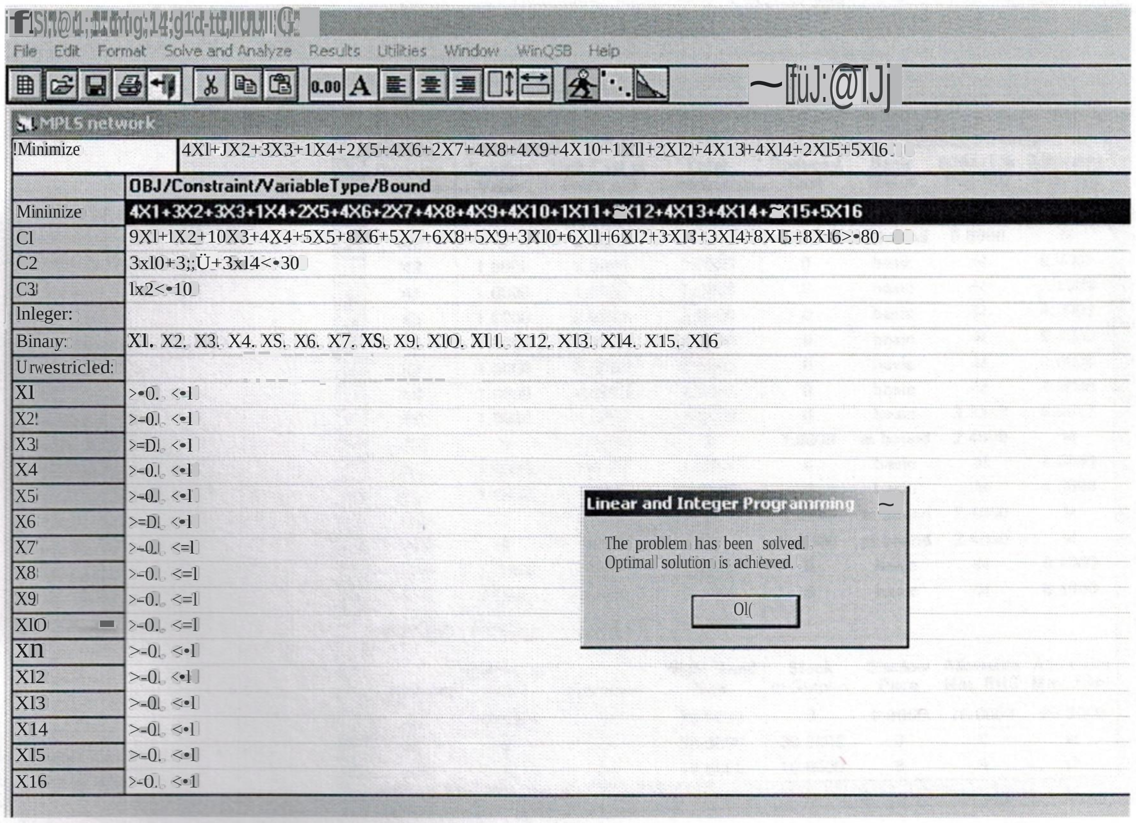
Nonnegative continuous

Nonnegative integer

[✓ Binary (0,1)]

Unsigned/unrestricted

This is where the user will fill the necessary information about the problem in question. We have set the following parameters as we have done for the original problem. Data entry format can be either in a matrix form or in normal editor mode. We have used the matrix form due to its ease of editions to do multiple runs.


File Edit Format Solve and Analyze Results Utilities Window WinQSB Help
MPLS network

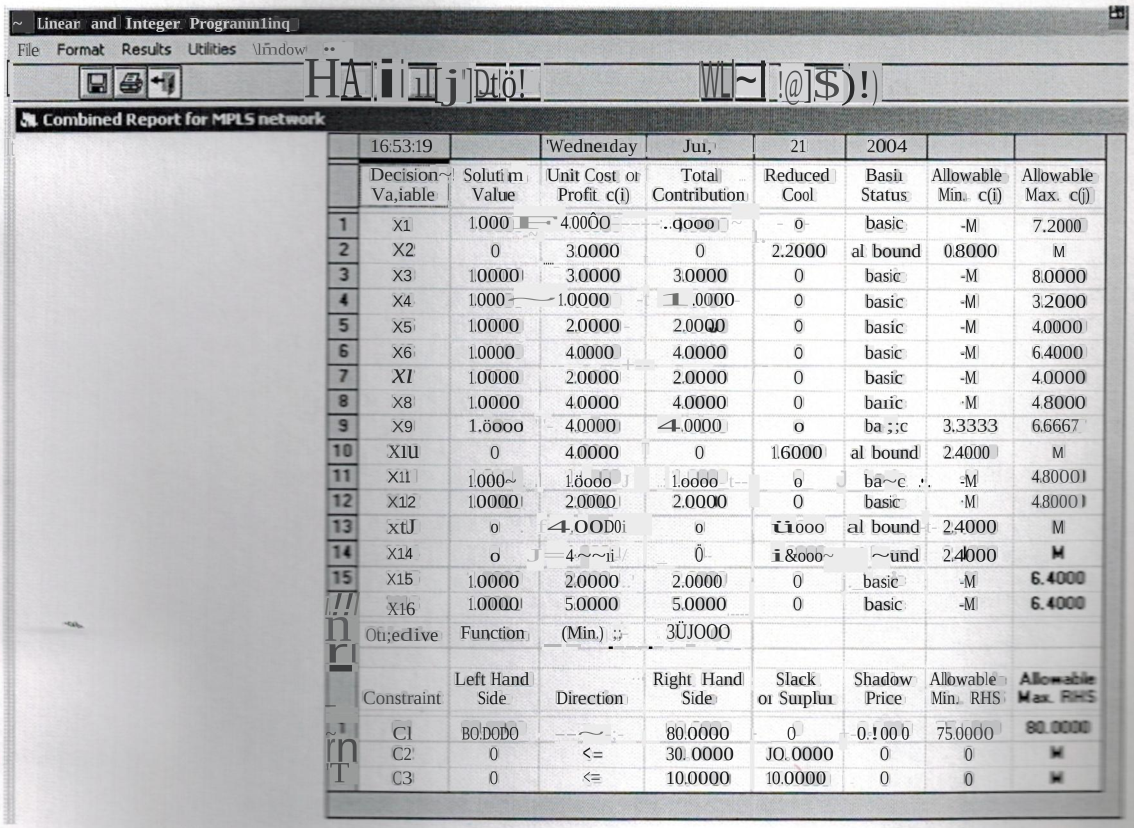
Minimize 4X1+3X2+3X3+1X4+2X5+4X6+2X7+4X8+4X9+4X10+1X11+2X12+4X13+4X14+2X15+5X16

	OBJ/Constraint/VariableType/Bound
Minimize	4X1+3X2+3X3+1X4+2X5+4X6+2X7+4X8+4X9+4X10+1X11+2X12+4X13+4X14+2X15+5X16
C1	9X1+1X2+10X3+4X4+5X5+8X6+5X7+6X8+5X9+3X10+6X11+6X12+3X13+3X14+8X15+8X16>=80
C2	3x10+3;;Ü+3x14<=30
C3	1x2<=10
Integer:	
Binary:	X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16
Unrestricted:	
X1	>=0, <=1
X2	>=0, <=1
X3	>=0, <=1
X4	>=0, <=1
X5	>=0, <=1
X6	>=0, <=1
X7	>=0, <=1
X8	>=0, <=1
X9	>=0, <=1
X10	>=0, <=1
X11	>=0, <=1
X12	>=0, <=1
X13	>=0, <=1
X14	>=0, <=1
X15	>=0, <=1
X16	>=0, <=1

Linear and Integer Programming

The problem has been solved.
Optimal solution is achieved.

Once the user inputs all the necessary conditions, the run button must be pressed in order to solve the problem. Here is the output just after pressing the button. The view shows that the problem has been solved and an optimal solution was achieved.



Linear and Integer Programming

File Format Results Utilities Window

Combined Report for MPLS network

	16:53:19		Wednesday	Jul,	21	2004		
	Decision Variable	Solution Value	Unit Cost or Profit c(i)	Total Contribution	Reduced Cost	Basis Status	Allowable Min. c(i)	Allowable Max. c(j)
1	X1	1.0000	4.0000	4.0000	0	basic	-M	7.2000
2	X2	0	3.0000	0	2.2000	at bound	0.8000	M
3	X3	1.0000	3.0000	3.0000	0	basic	-M	8.0000
4	X4	1.0000	1.0000	1.0000	0	basic	-M	3.2000
5	X5	1.0000	2.0000	2.0000	0	basic	-M	4.0000
6	X6	1.0000	4.0000	4.0000	0	basic	-M	6.4000
7	X7	1.0000	2.0000	2.0000	0	basic	-M	4.0000
8	X8	1.0000	4.0000	4.0000	0	basic	-M	4.8000
9	X9	1.0000	4.0000	4.0000	0	basic	3.3333	6.6667
10	X10	0	4.0000	0	1.6000	at bound	2.4000	M
11	X11	1.0000	1.0000	1.0000	0	basic	-M	4.8000
12	X12	1.0000	2.0000	2.0000	0	basic	-M	4.8000
13	X13	0	4.0000	0	1.6000	at bound	2.4000	M
14	X14	0	4.0000	0	1.6000	at bound	2.4000	M
15	X15	1.0000	2.0000	2.0000	0	basic	-M	6.4000
16	X16	1.0000	5.0000	5.0000	0	basic	-M	6.4000
	Objective	Function	(Min.) =	34.0000				
	Constraint	Left Hand Side	Direction	Right Hand Side	Slack or Surplus	Shadow Price	Allowable Min. RHS	Allowable Max. RHS
1	C1	80.0000	>=	80.0000	0	0.1000	75.0000	80.0000
2	C2	0	<=	30.0000	30.0000	0	0	M
3	C3	0	<=	10.0000	10.0000	0	0	M

This is the answer of the problem, note that the arrangements of ones and zeros for the variables, indicating wither they are functional or not functional. The problem parameters can also be saved for future runs and variables changes as shown in the figure.