EXPERIMENTAL MODELS FOR NETWORK MESH TOPOLOGIES WITH DESIGNS THAT ENHANCE SURVIVABILITY

J. M. Serumaga-Zake

EXPERIMENTAL MODELS FOR NETWORK MESH TOPOLOGIES WITH DESIGNS THAT ENHANCE SURVIVABILITY

J. M. Serumaga-Zake, B.Sc., Hons. B.Sc.

Dissertation submitted in partial fulfilment of the requirements for the degree Magister Scientiae in Computer Science at the North-West University (Potchefstroom Campus)

Supervisor: Prof. J. M. Hattingh

2006

Potchefstroom

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Amongst the many persons who have supported me in my studies, a special word of thanks to:

Professor J. M. Hattingh for his support and guidance;

My parents for all their motivation, care, support and love;

My brother Steven and my sister Gladys for all their love and support;

To my Heavenly father all honour and gratitude.

ABSTRACT

Network design problems involving survivability usually include trade-off of the potential for lost revenues and customer goodwill against the extra costs required to increase the network survivability. It also involves selection of nodes and edges from lists of potential sets to accomplish certain desirable properties. In many applications it is imperative to have built-in reliability or survivability of the network. Delays of traffic are undesirable since it affects quality of service (QoS) to clients of the network.

In this dissertation we consider the construction of an optimization system for network design with survivability properties that may help in the planning of mesh topologies while maintaining a certain degree of survivability of the network. This is done by providing for at least two diverse paths between certain "special" nodes to provide protection against any single edge or node failure. This part is modelled by using mixed integer programming techniques. A software product called CPLEX then solves these models and various facilities are built into the decision support system to allow the decision maker to experiment with some topological and flow requirement changes.

Keywords: network design, node connectivity, survivability, mathematical programming.

SAMEVATTING

Netwerk-ontwerp probleme waar oorlewing van die netwerk belangrik is, sluit gewoonlik 'n koste-voordeel balans, tussen die moontlikheid van verlies aan inkomste en kliente welwillendheid een die eenkant, en die ekstra koste wat nodig is vir netwerk verbetering van robuustheid aan die ander kant.

Dit behels ook die seleksie van nodes en verbindings uit potensiële lyste om sekere verlangde eienskappe te bereik. In die meeste toepassings is dit van kardinale belang om ingeboude betroubaarheid of robuustheid van die netwerk te verseker. Vertraging in verkeer is onwenslik, siende dat die kwaliteit van diens, aan die kliënte van die netwerk, geaffekteer word.

In hierdie verhandeling beskou ons die konstruksie van 'n ge-optimeerde stelsel vir netwerk-ontwerp, met eienskappe van robuustheid wat van nut mag wees in die beplanning van netwerk topologieë. Dit word gedoen deur ten minste 2 verskillende paaie tussen sekere spesiale nodes van beskerming te voorsien teen enige enkele rand wat faal of enige node wat nie funksioneer nie.

Sleutelwoorde: netwerk ontwerp, node koppelbaarheid, oorleefbaarheid, wiskundige programmering.

1

INTRODUCTION

Networks surround us in our daily lives. Communication networks permit us to communicate in many different ways. Transportation networks provide us with the ability to travel from one destination to another while distribution networks are used in the distribution of goods all over the world.

It becomes apparent that in every network, one wants to move a certain commodity from one point to another. To provide a good service to the users of a network and to make optimal use of the underlying network the moving of the commodities must be done as effectively as possible. Apart from that, we often have to establish a (new) network or change an existing network and the problem becomes even more complex, since decisions have to be made concerning the choice of nodes and edges (links or arcs) between certain nodes in such a way that it permits the flow requirements.

1.1 Background and literature on network design

The problem of designing a survivable network mesh topology is relevant for this research because telecommunication network planning has become in the last decade an important area for developing and applying optimization models. Telephone companies have initiated extensive modelling and planning efforts to expand and upgrade their transmission facilities; hence, the great need for survivable network topologies. The design of survivable networks has become a major objective in the telecommunication industry.

The design of a two-node connected and a two-edge connected network problem is a fundamental problem in network mesh topology design. This problem arises in the design of communication networks that are resilient to single-link failures and is an important special case in the design of survivable networks [37], [38], and [45]. In such networks, there are at least two edge-disjoint paths between each pair of nodes. So if a link fails, it is always possible to reroute the traffic between two terminals along the second path. This problem is a particular case of the two-connected networks with bounded meshes problem studied by Fortz, Labbe and Maffioli [22].

Before discussing the design of survivable mesh networks in telecommunications, we briefly give the background of the major types of networks and their components to give a full perspective to the design issues and the techniques presented for their resolution. We often use the basic description using the terminology found in Kershenbaum [32].

1.1.1 Types of Networks

We describe some of the most common types of networks and the common functions they perform. In reality, networks do no often fall into single types since it is the nature of networks to combine applications and architectures to form hybrids in various ways. Although this research discusses specific optimization problems in detail, the most important network design decisions are those relating to choosing network architecture. A major goal of this research is to provide the reader with the ability to solve individual design problems quickly and reliably, to make possible a proper comparison between different architectures. If one required weeks to analyze a single architecture, it would not be realistic to compare many architectures with one another. However, if one could approximate the analysis of architecture in an hour with the aid of network design tool, the evaluation of dozens of alternative architectures and combinations of different architectures becomes quite feasible.

Many new types of networks have emerged over the past twenty years, and more will undoubtedly emerge in the future. It is not the intent of this research to catalogue the current "new" network architectures. However, it is important for the designer to understand the technology being

considered in a network including its costs and limitations. The basic principles of network design and the basic algorithms used do not change significantly as the technology develops although specific tradeoffs may change if, for example, one type of network component becomes much less expensive. Thus, we try as much as possible to describe the network design problems in generic terms, in terms of basic principles and device characteristics, rather than in terms of specific devices that become outdated very quickly.

1.1.2 Network components

A detailed discussion of communication hardware is beyond the scope of this research and the interested reader may wish to examine Schwartz [43], Stallings [46], and Tanenbaum [48] for a more complete treatment of this area. Nevertheless for completeness, we present an overview of the major components of a network.

1.1.2.1 Facilities

We refer to the communication facilities that interconnect locations on the network as **communication channels, links/edges, lines,** or **facilities**. These include telephone lines, coaxial cables, microwave links, satellite channels, and optical fibres. Each of these **physical media** has many interconnecting characteristics in terms of its ability to provide communication. We are primarily concerned with:

Cost: This includes the monthly lease cost, installation cost, cost per unit traffic, and maintenance costs. Costs vary depending on whether the facility is owned or leased. Usually, cost is modelled simply by a monthly cost figure, or, if cost is usage sensitive, by a cost per unit of usage (e.g., cost per hour or cost per bit).

Both methodologies used and the topologies selected in the design of a network are strongly influenced by the way cost varies, and it is important for the designer to take this into account. Thus, if cost varies roughly linearly with distance we choose a very different topology than if cost

is relatively invariant with distance. Similarly, if there is a strong economy of scale with respect to capacity, that is, if cost goes up much more slowly than capacity, it has a profound effect on the topology selected.

There is also a significant difference between fixed monthly costs and usage sensitive costs. Often, a network incurs both types of cost and one can be traded off against the other. The typical situation is that fixed costs are used for large volumes of traffic (where a fixed cost can be justified by spreading it over a high volume of traffic) and then let the remaining traffic flow over usage sensitive facilities.

Capacity: This is the amount of traffic the channel can carry. If the traffic is data, the units of capacity are usually bits per second (**bps**). Sometimes we speak of characters per second or messages per second. If traffic is voice traffic, we may refer to capacity in terms of the number of simultaneous telephone calls that can be carried. If the voice traffic is digitized (i.e., turned into a bit stream) it is treated as data. In this case, one voice conversation is converted into V bps, where V is typically between 4,000 and 64,000 bps.

Sometimes a private voice network is constructed with facilities leased from a public carrier. In this case, the links of the private network become requirements for the public network. From the voice network point of view, capacity is measured in calls. From the public network point of view, capacity maybe measured in terms of bits per second. A device (which may be part of either the public network or the private network) converts from one to the other.

Usually, we talk about the total capacity of a channel. In most real systems, however, part of this capacity is unavailable for communication. It is occupied by overhead of one sort or another. In that case, we may lower the total capacity and speak of **usable capacity**.

A channel may be full duplex, permitting simultaneous communication in both directions, half duplex, permitting communication in one direction at a time, or simplex, permitting communication in only one direction. It is essential that we distinguish among these cases in modelling a network.

The usable capacity of a channel is a function of the technology used in implementing the network. It is important to understand the technology well enough to make a realistic model of capacity, and to be able to determine the relationship between capacity and load. Having done so, however, we usually simply associate an effective capacity with each type of link and the use this number during the design process. The large number of alternatives that need to be considered during the design process generally precludes detailed capacity analysis during topology selection. Often, there are a number of iterations between network design and capacity analysis, refining the model and taking specifics of the data at hand into consideration.

Reliability: It is customary to define this as the fraction of time the channel is working. (This is also sometimes referred to as **availability**.) This takes into account the mean time between failures (**MTBF**) and the mean time to repair (**MTTR**). Thus, the reliability of a channel is

$$R = 1 - \frac{\text{MTTR}}{\text{MTBF}}$$

1.1.2.2 Devices

There are many types of devices used to construct a network often referred to generically as **nodes**. Sometimes nodes are distinguished by their functions. Types of nodes include:

Terminals: These are simple devices, usually serving a single user, sources and destinations of low volume traffic. They usually include a keyboard and CRT and can also include disk drives and a printer. A personal computer, workstation, or a telephone may serve as a terminal.

Hosts: It is a large computer serving many users providing computing capability or access to a database. It's a source and /or destination of a major amount of traffic. A large workstation might be a host. These are also sometimes referred to as servers.

Multiplexors and concentrators: These are devices, which join the traffic on low speed lines into a single stream, which can use a higher speed line. Some of these are transparent in that they work entirely at the electrical level using only hardware. Others use software and provide additional

functions. A simple multiplexor has an output channel whose speed is roughly equal to the sum of the speeds of the input channels. Likewise, concentrators usually buffer the inputs, and are able to have an output channel of a considerably lower speed that the sum of the input channels.

Local switches: These devices allow attached facilities and devices to communicate directly with one another. Most such devices also perform other functions noted in the following discussion. There are many different types of switches. **Circuit switches** establish a fulltime connection between the input and output ports, dedicating capacity to an individual session. **Packet switches** break messages into small parts (called packets) and interleave packets from different messages. They include buffers to hold packets for a short time, thereby introducing delay, but also accommodating more bursty traffic (i.e., traffic where the peak rate is much higher than the average rate). Recently, various types of **fast packet switches**, including **frame relays** and **cell relays** have been introduced. These devices have some of the characteristics of both circuit and packet switches, in that they introduce some delay and some loss but to a lesser extent than do pure circuit or packet switches. The principle difference between frame relay switches and older conventional packet switches is that the former carry out fewer functions but are capable of handling much more data. Such tradeoffs among speed, cost, and functionality are an important part of the network design process, and give rise to the need for algorithms capable of considering many different design alternatives.

Tandem switches: Devices, which interconnect nodes (attached devices described above) but also provide a path for traffic originating at other switches. These devices also perform many other functions, most notably routing described under routers.

Gateways: Tandem switches are devices, which interconnect networks with one another. These devices are generally able to handle multiple protocols (network control standards) and also can convert one protocol to another, allowing communication among different types of networks. Recently, **multiprotocol routers** have found wide use interconnecting wide area networks, implementing a variety of protocols.

Originally, many of these devices were predominantly, or entirely, communications hardware. Now, virtually all of them contain microprocessors and to some extent, are programmable. The boundaries between these devices are also often blurred; for example, a terminal may also do some concentration.
Again, such devices have many interesting characteristics, but our focus is on the following:

Cost: Similar to facilities; nodes have purchase costs, monthly costs, maintenance costs, and costs per unit capacity. Also, design cost of nodes that permit certain link attachments by having specialized slots (discrete varieties).

Capacity: The capacity of a node is usually more difficult to determine than that of an edge/link. It is dependent upon the speed of the processor, the length of the programs running, the amount of memory, and the nature of the traffic passing through it. Nevertheless, we usually need to associate a capacity with each node in terms of the bits/sec of traffic, messages/sec, calls/sec., number of links, and aggregate speed of all attached links. Constraints on the type, amount, and mix of traffic that can pass through a node of a given type are formed on the basis of such capacity models.

Availability: As with links, the fraction of time the device is working.

Compatibility: It deals with types of traffic and links, which the nodes can handle. Some devices cannot handle certain types of traffic (e.g., analog voice). Some cannot handle links above a given speed. Also, some types of devices cannot interconnect with other devices or participate in some network architectures because the software they are using (the protocol they support) is not compatible.

1.1.3 Network Functions

The simplest way of providing for communication requirements is to provide for each commodity separately, but this is usually not cost-effective. Networks are constructed to share resources and to improve cost-effectiveness. The most obvious example of sharing is many low speed or part time

communication being combined to justify fulltime, high-speed channels. Users benefit from the higher speed and the economy of scale of the high capacity facilities.

Less obvious, but also important, are capabilities justified within the network that could not be justified for individual requirements. These capabilities are often collectively referred to as "intelligence" within the network and they include: the ability to monitor itself (informing the user of congestion or failures), find paths for traffic through the network, modify these paths based on the state of the network, and maintain accounting records. Many of these capabilities are discussed below. Closely related to this is the fact that specialized staff for maintenance and to interface with outside vendors can be supported by the network, and these can become a resource for individual communications users.

There are many functions that may not be within the functional scope of the networks we design. We briefly consider some of these. Depending upon the applications the network is to handle, many of these functions may be essential parts of the services being offered or, alternatively, may just complicate its implementation, increasing cost and decreasing performance. In most cases the issue is how much of each function need be provided. Note that some of these capabilities are interdependent; that is, the network must have some of them in order to properly do others.

Switching: The ability to interconnect the channels attached to each network node and to move traffic from each incoming channel to the appropriate outgoing channel when the requirement neither originates nor terminates at the node.

Routing: It's the ability to select a path for each requirement. This capability varies widely. In some networks, only a single path is available for each requirement (fixed routing). The network may choose this path or the user may specify it. In other cases, several paths are predefined and the network chooses the best one (alternative routing). This choice may be static based on a predefined probability, or a dynamic based on the current state of the network.

Flow control: It's the ability to reject traffic, or slow the rate of entering traffic, in order to reduce network congestion.

Security: The ability to prevent unauthorized access to the network and the data it carries. This may include passwords, data encryption, and even physical security (limiting access to equipment connected to the network).

Failure monitoring: It deals with the ability to keep track of which components are working. As with traffic monitoring, this is useful on a short-term basis to route around failures and on a long-term basis, it is useful in planning network modifications (including replacement of unreliable components).

Traffic monitoring: The ability to keep track of traffic levels, possibly by type of usage. This can be useful both on short term and long-term bases. On a short-term basis, it can be used to support dynamic routing and flow control. Over a long term, it can be used in network design to identify parts of the network where capacity may be productively increased or decreased.

Internetworking: Performing the functions needed to communicate with and across other networks. This includes providing routes for traffic crossing through, into, and out of the network, and allocating resources such as buffers and link capacity to traffic originating in other networks.

Network management: This includes a broad range of functions related to the management of the network. Some of these functions are mentioned in previous categories. Others include maintaining lists of users and addresses of devices, fault isolation, and keeping track of scheduled changes to the network.

The ability to provide these functions has a profound effect on the type of network built. In particular, the nodes of the network become more complex and more expensive as their functionality increases. Likewise, most of these functions imply overhead communication between nodes, which requires additional link capacity. Thus, the decision to create a high functionality network is an important design decision, which has a major impact on the architecture, and hence the overall design of the network.

1.1.4 Network architecture

In the following chapters, the focus is on specific problems in topological optimization. Before detailed decisions about node and link placement are made, however, the more general issue of the overall "shape" of the network should be discussed. Decisions at this level are partially art and partially science. The goal is to bring as much science as possible into play.

Will the network be a single, homogeneous mesh comprised of a single type of node and a single type of link, or will it be a hybrid of different types of equipment? It might be a hierarchical network with one type of link riding on another, or applications at the highest level, that share common facilities. There may be uncertainty about the proper network architecture and several alternatives may need to be explored. Before significant effort is expended on exploring any detailed design, however, we need to get at least an approximate picture of which architectures are viable alternatives. Then, the most promising ones can be explored in more detail. We give a brief view of some of the network types below.

Centralized data networks: The simplest data networks are centralized, allowing terminals to access a single, central source of data. For example, an insurance company might maintain a centralized database with policy, claim, and employee information. Terminals in each office around the country could access this data and update the database. Some of this access might be real time, with a user waiting for the response while other applications are off-line, with bulk data being printed for later use.

Distributed data networks: This class of network includes data networks with data from many sources and destinations. As such, they may be thought of as extensions of centralized networks. This class also includes computer networks, where the "centers" themselves may communicate. Unlike in centralized networks, it may now be possible to keep more of the data closer to its original source and move data only on request. For example, a credit card network may be designed with regional centers where credit information is kept and distributed.

Voice Networks: These networks interconnect telephones. They can be analog, digital, or hybrids of both. Most voice networks are mesh networks, supporting point-to-point communication among

all pairs of nodes. Thus, they share many of the characteristics of distributed data networks from the point of view of network design. It is also possible to design a centralized voice network with all communication, taking place through a single central switch. Small voice networks, especially those with limited geographic scope, are often centralized.

Integrated Networks: There are many forms of integration. The simplest is where essentially separate networks share transmission facilities. Each network has its own nodes, which carry out all necessary functions. But the capacity of high-speed links is statically divided among the networks, thus realizing a greater economy of scale than the networks could individually. More complex integration involves sharing capacity across networks; that is, if one network is quiet, another network can make us of the spare capacity.

Local area networks (LANs): These are networks, which tie together users within a building or campus. They typically utilize high-speed links that consist of coaxial cable or optical fiber. Such networks can be stand-alone or can be part of the local access portion of larger networks like wide area networks (WANs). LANs can connect terminals within a building and these LANs can in turn be connected by routers and bridges to form a network over a campus. Campuses can be tied together into metropolitan area networks (MANs) and then into wide area networks (WANs). The boundaries between these types of networks are blurred and their respective technologies overlap.

Another issue relating to the overall architecture of the network is whether to decompose the network into subnetworks for the sake of design and operations. It's possible, for example, to begin by clustering the nodes into regions, locate gateways within each region, and then design high-speed links (backbone) connecting the gateways. All communication between regions is via this backbone. Local access networks are designed within each region. This is often how wide area networks, which include LANs, are designed.

Alternatively, the network might be designed as a single mesh. If LANs, which include routers, are considered, it may be in the organization's best interest to trade off node and link capacity within the LAN against capacity in the wide area network. Thus, the wide area network may connect parts

of the LAN (or different LANs on a campus), or the LAN may offload switches in the wide area network.

If the network is considered as a single mesh, it becomes more difficult to design and to operate but might function more efficiently because of the additional option available to share capacity among all nodes. This is an important design trade-off. Unless substantial gains can be made, it is usually better to consider the designs at different levels, separately.

1.1.5 Node placement and sizing

A fundamental problem in the topological optimization of a network is the selection of the network node sites. This actually encompasses several problems, which must be dealt with separately. The first is the choice of which sources and destinations of traffic will be 'on-net,' that is, part of the network at all. It may be that some of these do not have enough traffic to justify a dedicated connection to the network, and may, instead, access the network via common carrier facilities (e.g., dialup lines).

A second problem is the decision where to place **multiplexors**, **concentrators** and **switches**. These terms are often used to describe different types of devices. As mentioned before, they may even be equipment, like routers, which are usually associated with LANs. In theory, it is possible to place nodes anywhere. In practice, the placement of nodes is usually limited to a finite set of candidate sites. Typically, these are a subset of the sites, which are sources, and destinations of traffic. The set of candidate sites is usually an input to the network design process while the actual sites selected are an output.

Closely related to the problem of selecting where to place nodes is the problem of deciding exactly what type of devices to place at each location. This includes such high level questions as whether a specific site should be a switching site of just a multiplexor site. It also includes more detailed questions on equipment configuration.

1.1.6 Link/edge topology and sizing

Link topology and sizing involves selecting the specific links interconnecting nodes, and is one of the major focuses of this research. At the highest level, this is where the architecture of the network is derived. Thus, a hierarchy that includes a backbone as well as local access networks may be defined. It is possible to permit the backbone to be a mesh while the local access networks are constrained to be trees. At a lower level, having at least tentatively decided on architecture, a selection is made from among specific topologies, including and excluding specific links. The overall problem is often partitioned into subproblems for the different parts of the network.

As with node selection, the lowest level problem in link selection is the determination of the specific number and type of links. Here, unlike in node selection, it is easier to build accurate, detailed models of the links; and hence, it is possible to deal with the problem in greater detail. Thus, procedures are described for deciding on the exact number and type of links in a network.

1.1.7 Traffic Requirements

The traffic requirements are usually the most voluminous part of the database that must be collected by the user. As mentioned before, it is dependent upon the choice of location set. There are many other dimensions to this data as well since it is also a function of time of day, and application. There may be multiple views based on projections of different lengths.

Generally, traffic requirements start with raw data and may be as detailed as records of individual sessions (telephone calls or terminal sessions). These include records of packets, number of characters, type of session, application, route, and possibly other information as well. Alternatively, there might be only projections in terms of average number of sessions per terminal per day. This data is then processed into a more manageable form.

1.1.8 Link Costs (Tariffs)

Tariffs, the published rates for communication services filed by the common carries are, next to traffic information, the most voluminous part of the database. This section of the database contains the costs of all possible links in the network. Conceptually, for leased lines, this is a three dimensional matrix indexed by pairs of locations and link types. Usually, this information is extracted from a tariff database that is kept on-line in support of network design tools.

Costs include both monthly charges and one-time charges, such as installation costs. The latter can be converted to monthly costs by amortizing them over a reasonable period of time. Similarly, in the case where private communication facilities are constructed (e.g., a private microwave line) this cost is converted to a monthly cost by amortization over the expected lifetime of the facility, usually several years.

Some communication costs are usage sensitive, such as direct dial telephone costs or packet switched common carrier costs. There are also tariffs covering these. In this case we maintain cost per minute or cost per bit, as appropriate. Tariff costs are location sensitive. Usually it is sufficient to know the area code and exchange of a location. All necessary tariff information can be looked up on this basis. A location's LATA (Local Access and Transport Area) is also needed for some intrastate tariffs. In some cases, the distance from the telephone company central office is also needed; this is usually difficult to obtain.

1.1.9 Performance Objectives

The network design problem is usually thought of as one of minimizing cost while satisfying throughput requirements. In most of the algorithms presented in the following chapters, that is all that is explicitly considered. However, constraints on performance must also be satisfied. Thus, there is usually a limit on the tolerable delay in systems that queue or loss in systems that block. They're also maybe constraints on reliability.

Such constraints can be stated as input or thought of as additional costs, to be traded off against other objectives. In reality, there are rarely hard constraints on delay, loss, or reliability. If a network had to double the cost to reduce its delay by say 10%, few people would elect to reduce delay. Conversely, if delay could be reduced from 8 seconds to 4 seconds while increasing the cost by only 2%, few people would pass up the opportunity to do so. Ideally, the network design effort should yield as output the relationship between performance and cost, to allow the designer to make an informed decision about how much performance he or she is willing to pay for.

1.1.10 Network design Tools

We briefly discuss an overview of current approaches to network design and then describe the major components of automated tools for the design of networks.

1.1.10.1 Approaches to network design

It is important to distinguish between techniques used to design a network "from scratch" as opposed to incrementally. In reality, most real network designs are incremental beginning with an existing network. Unfortunately, most of what is known and can be proven relates to techniques that design an entire network from a given set of requirements.

1.1.10.1.1 Manual design

A surprisingly large number of networks are still designed by hand using the rules of thumb, or even no "rules" at all. The most attractive aspect of this approach is its flexibility: A database doesn't have to be assembled. All sorts of unusual constraints and objectives can be taken into account. The designer can be very responsive to changes in goals and requirements because there is no "setup" time. Also, incremental as well as total designs can be done.

This approach has several disadvantages, however. It is rarely quantitative. Often, the designer makes decisions subjectively and inconsistently, sometimes even unconsciously. Thus, it is difficult

to repeat a successful design when similar circumstances arise and, likewise, difficult to learn from previous mistakes.

The approach is also usually too labour intensive to allow for the proper consideration of all alternatives. Thus, designs tend to follow the designer's preconceived notions of what they should look like rather than what is actually best in each situation. Serious design mistakes can be done in this way. In particular, as the requirements evolve, there is little to motivate a corresponding change in network architecture. Automating the design process can overcome these problems. However, it is important to maintain the advantages of manual design as well. In particular, be sure to leave the designer "in the loop" and with the final word as to what the design should be. This is why good tools are interactive, allowing the user to retain control over the design process and still take into account information available to him but not to the tool. Indeed, this is why we call these software systems tools.

1.1.10.1.2 Heuristics

These are design principles incorporated into algorithms and thus are automatable. Like rules of thumb, they are good ideas embodying design experience. They differ from rules of thumb, however, in that they can be quantitative and repeatable. Thus, alternative heuristics can be compared by implementing them on the same problem, and then observing those which give rise to the best solution. It is therefore possible to refine heuristics, keeping what works well and discarding what does not. (This same process goes on in manual design, but to a much lesser extent.) With automated design, many more alternatives can be tried and objectively compared. Experience can be transferred from one designer to another by allowing them to share the automated system.

Some heuristics are specific to particular types of networks. The most useful heuristics are based on principles, which apply across many types of networks. One of the most widely used heuristics is the **greedy algorithm**. Confronted by a series of choices, the greedy algorithm chooses the best one it can at each stage. The greedy algorithm is a broadly applicable heuristic based on the simple observation that inexpensive networks tend to contain inexpensive links. This principle is true for

all types of networks. Therefore, expect this algorithm to yield good networks. Do not, however, expect it to yield the optimal solution. It is quite possible that the least expensive network does not contain some of the least expensive links. It happens that in the case of unconstrained network, the greedy algorithm does in fact yield the optimal solution. However, with added constraints, the greedy algorithm no longer yields the optimal solution.

There is a class of problems for which it can be proved that the greedy algorithm yields an optimal solution. There are other classes of problems for which the greedy algorithm does an acceptable job but cannot guarantee an optimal solution. Finally, there are problems where the greedy algorithm produces unacceptable results, possibly no feasible solution at all. One goal is to find ways of classifying these problems and then determining to which class a problem of interest belongs.

1.1.10.1.3 Formal optimization techniques

Except for the smallest problems, it is not possible to enumerate all possible solutions and then choose the best one. The set of all possible solutions to a problem is referred to as the **solution space**. The notion of best is expressed in formal optimization techniques by the **objective function**. The objective function associates a value with the design variables. Thus, for example, each link I can be included or excluded from a design. There is a cost associated with each link and to minimize cost, use the sum of the costs of the links chosen as the objective function.

While cost is the most commonly used objective function, it is not the only one. We might want to maximize reliability. In this case, associate reliability with each possible node and link in the network, and then compute the reliability of candidate networks comprised of specific nodes and links. The best value of the objective function is referred to as the **optimum**.

If the optimum is to be found for problems of realistic size, we must rely on properties of the problem, which allow us to avoid looking at most of the possible solutions. One such situation was mentioned before, the greedy algorithm, which examines only a very limited number of alternatives and guarantees an optimal solution for a special class of problems.

There are also algorithms, which always produce optimal solutions. The problem with these, however, is that they only apply to a limited class of problems; for problems outside this, they do not work at all. One such algorithm is the simplex method Dantzig [17]. This algorithm only works for the class of problems called **linear programming problems**, which is the main focus of this research. Linear programs are problems where both the constraints and the objectives are weighted sums of the variables. In this case, the solution space can be searched in a very orderly way, avoiding most of the possible solutions and reaching the optimal solution in a reasonable amount of time. Often, a problem can be phrased as a linear program with the additional constraint that the variables must be integers. This situation arises frequently in network design; that is, when the decision is whether or not to include a link in the solution, and some of these situations are seen in later chapters.

1.1.10.2 Structure of a network design tool

A complete network design tool contains modules that allow the user to use each of the above techniques as appropriate. Also, it relieves the user of the burden of collecting and processing the inputs required by the tool to the extent that this is possible. It presents its output interactively in a tabular as well as graphical form.

Finally, it should allow the user to interact with it in all phases of the design process, both before and after the algorithms are run. Thus, the user gets the first word, specifying allowable candidate nodes and links, and also last word, editing the tool's design decisions. The tool should also include analysis routines to check the validity of the user's design decisions. More information about this is given in later chapters.

1.1.11 Mesh Network topology optimization

Now we turn to the problem of determining the topology of a mesh network, which is the main topic of this research. Thus the objective is to determine which locations to choose and directly connect. This decision cannot, however, be made in a vacuum. It is closely related to decisions on

what speed links to use and how to route traffic through the network. In particular, the requirement is sufficient capacity to carry the load routed onto each link.

The general mesh topology optimization is complex; it involves the selection of nodes and links, the assignment of capacities to these links, and routing of requirements on these links. Ideally, all these capacities are jointly optimized, leading to a minimum cost network, which meets given objectives on delay and throughput. In practice, these problems are often solved sequentially since the full problem may have too high complexity for exact solutions.

We consider the problem of designing a mesh network, which satisfies a pre-specified survivability criterion with minimum cost. The survivability criterion demands that there be at least a minimum number of node disjoint paths between certain pairs of nodes. This design problem appears to be at least as difficult as the travelling salesman problem, and present techniques cannot provide a computationally feasible exact solution. Therefore heuristic approaches are considered, Clarke [12].

An important consideration in the design of a communication or transportation network is the degree to which connectivity between given pairs of nodes is vulnerable to the failure or destruction of other parts of the network. In later chapters we shall try to determine in a sense a minimum enemy effort required to disrupt a strategic communication network.

1.2 Problem description and research motivation

In designing mesh network with survivability, the topology optimization problem is clearly difficult and even sub-problems like routing and capacity assignment have been shown to be difficult, Kershenbaum [32]. Integer programming techniques have been used to produce optimal solutions or tight bounds on optimal solutions only for problems of a few dozen nodes, Gavish [24]. For problems of practical size, virtually all approaches have been heuristics.

Survivability is a particularly important issue for mesh topology networks. Any potential loss of network services due to edge (link) or node failures can result in customer frustration and lost

revenues. Work on methods for designing survivable communication networks by Martin Grotschel *et al.* [27] concludes that "two-connected" topologies provide a high level of survivability in a cost effective manner, and that it is feasible to compute minimum cost networks that satisfy survivability for small problems. For large problems one often has to be satisfied with approximately optimal solutions.

The problem under investigation in this research was to design a model to solve small (sub) problems of survivable network mesh topology using exact integer linear programming methods. We tried to investigate how far we can go in solving problems with exact integer linear programming given the resources we have. The system developed would help network operators or administrators to zoom into the network and get hold of a part of network that might be giving problems for a reasonable number of node size. Then he/she can easily use the system with few nodes to solve the problem.

1.3 Research aims and objectives

In this research study we considered the construction of an optimization system that may help in the planning of mesh topologies while maintaining a certain degree of survivability of the network. Thus the objective was to determine which locations to directly connect. This decision cannot, however, be made in vacuum. It is closely related to decisions on what speed edges (links) to use and how to route traffic through the network. In particular, the requirement is sufficient capacity to carry the load routed onto each edge (link).

We wanted to determine a network topology, that apart from accommodating the flow requirements also provides for at least two independent paths between certain "special" offices (nodes), thus providing for protection against any single edge or single node failure for traffic between these offices (nodes).

1.4 Basic hypothesis/central theoretical statement

The aim of this study was to create a system that will integrate survivability into the network mesh topology design model, as an aid for solving some of the problems in network services due to edge (link) or node failures and also for congestion relief.

1.5 Methodology

The following are the methods used in this research study:

- A study of the literature in the field of network optimization;
- Investigation of different network design models;
- Development of a system which can assist in survivable network topology design;
- Empirical experimentation to test the applicability of the system to certain selected instances;
- Exploration of the capacity of the system for variable problem size.

1.5.1 Empirical investigation

In this research study, we used exact integer programming methods to solve smaller (sub) problems and we created a decision support system that can be used as an aid in analyzing strategies for topological decisions and routing plans under various conditions and traffic demands. Some parts of the system are based on research that was done by D. de Villiers and J.M. Hattingh [18]. The system also allows for the investigation of some survivability issues.

The survivable network design problem was modelled with mathematical modelling techniques as a mixed integer linear programming (MILP) problem, and the MILP model was then solved by means of a professional software product of ILOG called CPLEX.

We developed the software in Qt, a multiplatform C++ application development framework. The source runs natively on four different platforms (Windows, Unix/Linux, Mac OS X, and embedded Linux). The software can be ported to multiple platforms with a simple recompile. The system uses

a plain text MILP problem that is transferred to a machine capable of solving mixed integer linear programming problems. It then gives the solution as output using the user interface.

1.6 Chapters and content

In Chapter 2, the network terminologies and graph concepts relevant to this dissertation are given.

Chapter 3 deals with network representations and network mathematical models.

Chapter 4 gives solutions to some of network designing problems.

Chapter 5 describes the system considerations used for creating a survivable network design tool.

Chapter 6 deals with the system design description.

Chapter 7 gives illustrations of the system functionality.

Chapter 8 shows the experimental results obtained with the system developed.

In Chapter 9, a conclusion is presented and possible future work and system enhancements are considered.

2

TERMINOLOGY AND GRAPH CONCEPTS

In this section we give some basic definitions from graph theory and some basic notation of networks that will be used in the following chapters. The notation used by Ahuja et al. [1] will be adapted.

2.1 Network as graphs

The basic terminology is introduced for describing networks, graphs, and their properties. Although graph theory is a well-established discipline, there are, unfortunately, several different accepted terms for many of its basic concepts. The terminology given here is just one of a number of possible accepted sets of terms and used consistently throughout the remainder of this dissertation.

A **graph**, G, is defined by its vertex set, V, and its edge set, A. We often write $G = (V, A)$. The vertices are more commonly called **nodes** and these represent locations (e.g., sources of traffic or sites of communications equipment). The **edges** (also called links or arcs) facilitate communication transmissions between nodes.

A graph is called a **network** if the edges and nodes associated with it have properties (e.g., length, capacity, type, etc.). Networks are used to model problems in communications, and the specific properties of the nodes and edges relate to the specific problem at hand. See Kershenbaum [32].

In this dissertation we use the terms "graph" and "network" synonymously. It is possible to have more than one edge between the same pair of nodes. For example, it can correspond to multiple communication channels between two switches. Such edges are referred to as **parallel edges**.

A graph with parallel edges is called a **multigraph**. It is possible for a graph to have an edge between a node and itself. These edges are called **self-loops**. Figure 2.1 illustrates a graph with parallel edges and self loops. A graph without parallel edges or self-loops is called a simple graph. Simple graphs are easier to represent and manipulate.

In a communications network a channel is said to be **full duplex** if it can be used in both directions at the same time. It is said to be **half duplex** if it can be used in only one direction at a time. Occasionally, as with satellite networks, a channel can have capacity in only one direction. Clearly, if network channels have capacity in only one direction, they are modelled as directed edges. Half duplex edges and full duplex edges can be modelled as edges with two capacities (two separate properties), or as a pair of oppositely directed edges between the same pair of nodes. This is often an important decision, which can have an effect on both the simplicity and effectiveness of the algorithm used to solve the problem. It is also possible, of course, to convert from one form to another if necessary, but it's best to try to avoid this as it complicates implementation and increases runtime and storage requirements.

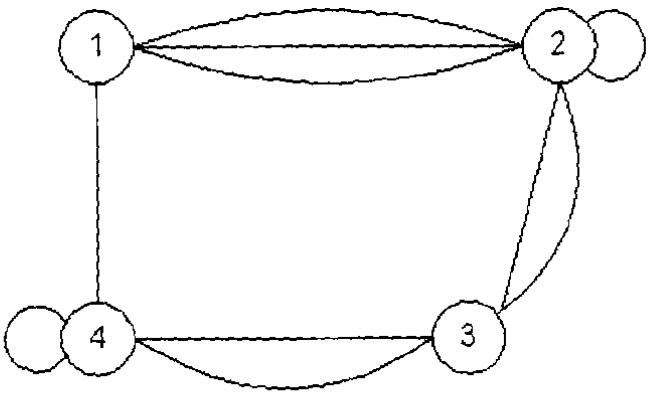


Figure 2.1 Graph with parallel edges and self-loops

We can define a network as $G = (N, A)$ consisting of a finite set N of n nodes (vertices or points) $N = \{1, 2,..., n\}$ and a set $A = \{(i, j), (k, l), \ldots, (s, t)\}$ of edges (branches, arcs or links) defining joined distinct nodes in N. The number of edges is denoted by m. We begin by defining directed and undirected graphs.

2.1.1 Directed graphs and networks

A directed network $G = (N, A)$ consists of a set N of nodes and a set A of edges whose elements are ordered pairs of distinct nodes. A directed edge behaves like a one-way street and permits flow only from a node i to a node j for instance. A directed network is a directed graph whose nodes and /or edges have associated numerical values like costs, capacities, and /or supplies and demand. An example of a directed network, $N = \{1, 2, 3, 4, 5, 6, 7\}$ and $A = \{(1,2), (1,3), (2,3), (2,4), (3,6), (4,5), (4,7), (5,2), (5,3), (5,7), (6,7)\}$ is shown in Figure 2.2.

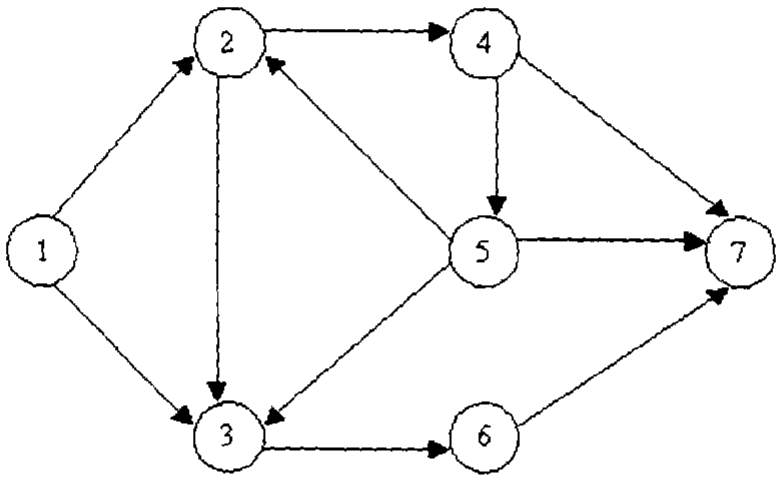


Figure 2.2 Graphical representation of a directed graph

2.1.2 Undirected graphs and networks

The undirected network is defined in the same manner as the directed network except that edges are unordered pairs of distinct nodes. Figure 2.3 shows an example of an undirected network $N = \{1, 2, 3, 4, 5, 6\}$ and a set A of an undirected edges as illustrated. In an undirected network we can refer to an edge joining the node pair i and j as either (i, j) or (j, i). An undirected edge (i, j) can be regarded as a two-way street with flow permitted in both directions: either from node i to j or from node j to i. It is possible for networks to have both directed and undirected edges. We can replace an undirected edge by two directed edges.

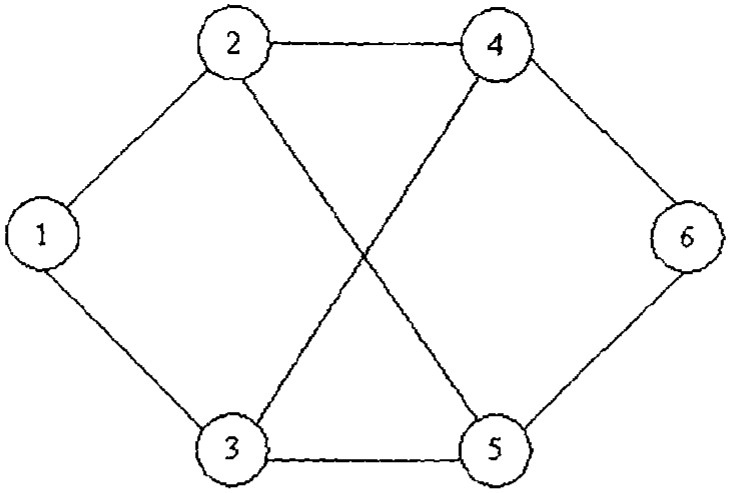


Figure 2.3 Graphical representation of an undirected graph

2.1.3 Subgraph

A graph $G' = (N', A')$ is a *subgraph* of $G = (N, A)$ if $N' \subseteq N$ and $A' \subseteq A$. Which means all the nodes in N' belong to the node set N of G and all the edges in A' belong to the edge set A of G.

2.1.4 Walk, path and cycle

A **walk** in a directed graph $G = (N, A)$ is a subgraph of G consisting of a sequence of nodes and edges $i_1 - a_1 - i_2 - a_2 - \ldots - i_{p-1} - a_{p-1} - i_p$ satisfying the property that for all $1 \leq k \leq p - 1$, either $a_k = (i_k, i_{k+1}) \in A$ or $a_k = (i_{k+1}, i_k) \in A$. Alternatively, we shall sometimes refer to a walk as a set of (sequence of) edges (or of nodes) without any explicit mention of the nodes (without explicit mention of edges). We illustrate this definition using the graph shown in Figure 2.2. Figure 2.4(a) and (b) illustrates two walks in this graph: 1-2-3-4 and 1-2-4-5-2-3. A *directed walk* is an "oriented" version of walk in the sense that for any two consecutive nodes i_k and i_{k+1} on the walk, $(i_k, i_{k+1}) \in A$. The walk shown in Figure 2.4(a) is not directed; the walk shown in Figure 2.4(b) is directed. See Kershenbaum [32].

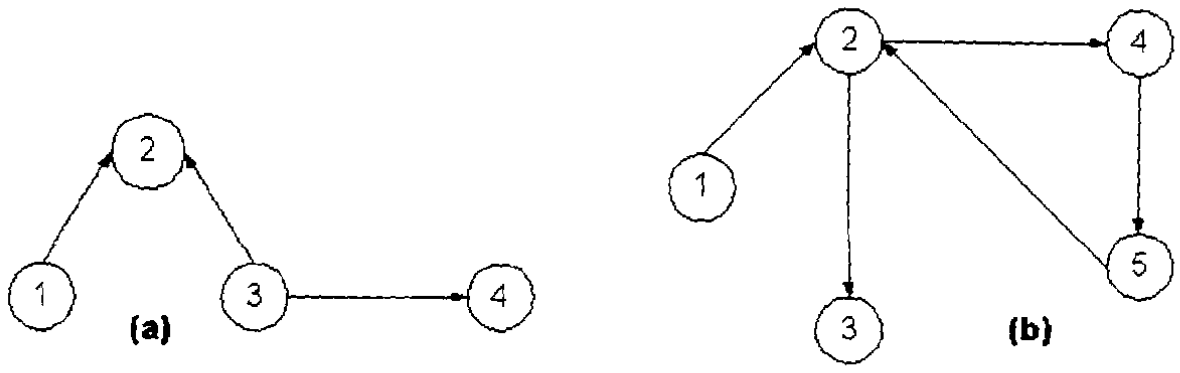


Figure 2.4 Examples of walks

A **path** (from node i_0 to i_p) is a sequence of edges $P = \{(i_0, i_1), (i_1, i_2), \ldots, (i_{p-1}, i_p)\}$ in which the initial node of each edge is the same as the terminal node of the preceding edge in the sequence and $i_0, \ldots, i_p$ are all distinct nodes. Thus each edge in the path is directed "toward" i_p and "away from" i_0. A **chain** is a similar structure to a path except that not all edges are necessarily directed toward node i_p. Figure 2.5a illustrates a path, and Figure 2.5b presents a chain. A **circuit** is a path from some node i_0 to i_p plus the edge (i_p, i_0). Thus a circuit is a *closed* path. Similarly, a **cycle** is a closed chain. Figures 2.5c and 2.5d depict circuits and cycles. Every path is a chain but not vice versa. Every circuit is a cycle but not conversely.

These definitions refer to what is known as *simple* paths, chains, circuits or cycles. We will assume that these definitions apply unless otherwise specified. A *nonsimple* path, for example, can permit nodes in the set $i_0, i_1, \ldots, i_p$ to repeat and hence may include circuits. For example, the set of edges {(1, 2), (2, 3), (3, 4), (4, 2), (2, 3), (3, 5), (5, 6), (6, 7), (7, 5), (5, 8)} describes a nonsimple path from node 1 to 8 with the sequence of nodes $i_0, i_1, \ldots, i_p$ visited being $i_0 = 1, 2, 3, 4, 2, 3, 5, 6, 7, 5,$ and $8 = i_p$. A *complete* graph is one where each node is connected by an edge to every other node. See Bazaraa *et al.* [3].

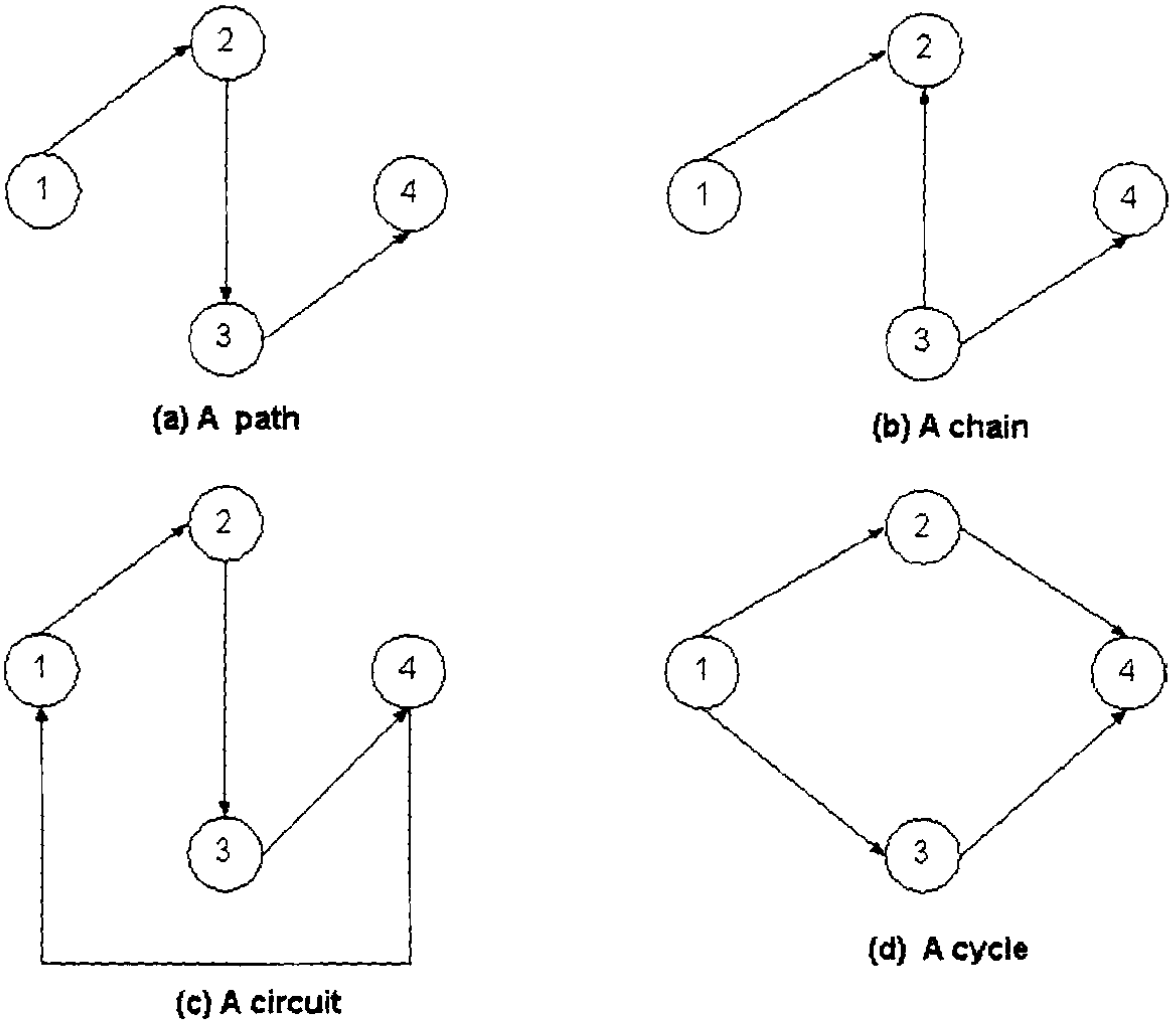


Figure 2.5 Examples of a path, chain, circuit and cycle

An **s, t-path** in a network is a sequence of edges which begins at some node, *s*, and ends at some node, *t*. A *directed path* is a directed walk without any repetition of nodes. In other words, a directed path has no backward edges.

2.1.5 Cut definition

A **cut** is a partition of the node set N into two parts, S and $\bar{S} = N - S$. Each cut defines a set of edges consisting of those edges that have one endpoint in S and another endpoint in $\bar{S}$. Therefore,

we refer to this set of edges as a cut and represent it by the notation $[S, \bar{S}]$. We illustrate a cut in Figure 2.6 with $S = \{1, 2, 3\}$ and $\bar{S} = \{4, 5, 6, 7\}$. The set of edges in this cut are {(2, 4), (5, 2), (5, 3), (3, 6)}.

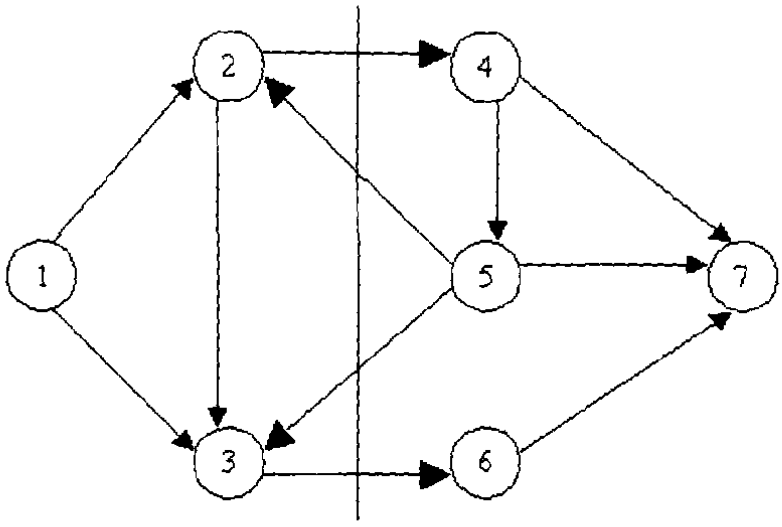


Figure 2.6 Graphical representation of a cut

2.1.6 *s-t* Cut

An ***s-t* cut** is defined with respect to two distinguished nodes s and t, and is a cut $[S, \bar{S}]$ satisfying the property that $s \in S$ and $t \in \bar{S}$. For instance, if $s = 1$ and $t = 6$, the cut depicted in Figure 2.6 is an s-t cut; but if $s = 1$ and $t = 3$, this cut is not an *s-t* cut.

2.1.7 Connectivity

A graph is **connected** if there is at least one path between every pair of nodes otherwise it is disconnected. The sets of nodes with paths to one another are **connected components**, or more simply, **components**. The edges between these nodes are also part of the components. A connected graph has a single component. Notice that if there is a path from i to j there is also a path from j to i. Also, if there is a path from i to j and a path from j to k there is also a path from i to k. By definition, there is always a path from i to i. Thus, components form equivalence classes over the

set of nodes in a graph. Each node is a member of exactly one component. Also, each edge is a member of exactly one component. Therefore, the component structure of a graph can be described as a partition on its node set or edge set.

A directed graph with a directed path *from* every node *to* every other node is called **strongly connected**. Connectivity in directed graphs is not symmetric. There may be a directed path from i to j without there being one from j to i. A set of nodes with directed paths from any one node to any other is called a **strongly connected component**. Note that, again a node is part of exactly one strongly connected component but that an arc can be part of at *most* one. Specifically, some arcs may not be part of any strongly connected component.

Example 2.1 Consider the directed graph in Fig. 2.7. The strongly connected components are defined by the nodal partition {1, 2, 3, 4}, {5, 6, 7}, {8}, {9}, {10}.

The edges (1, 8), (4, 9), (9, 10) and (10, 7) are not part of any strongly connected component. Considered as an undirected graph (i.e., treating the edges as undirected edges), the graph has a single component, and is a connected graph.

2.1.8 Tree

A **tree** is a connected graph with no cycles. A **spanning tree**, defined with respect to some underlying graph G is a tree that includes every node of the graph, that is, it is a spanning, connected subgraph with no cycles. Such a graph is referred to more simply as a tree. If the graph is not necessarily connected, it is referred to as a **forest**. We generally speak of trees in undirected graphs.

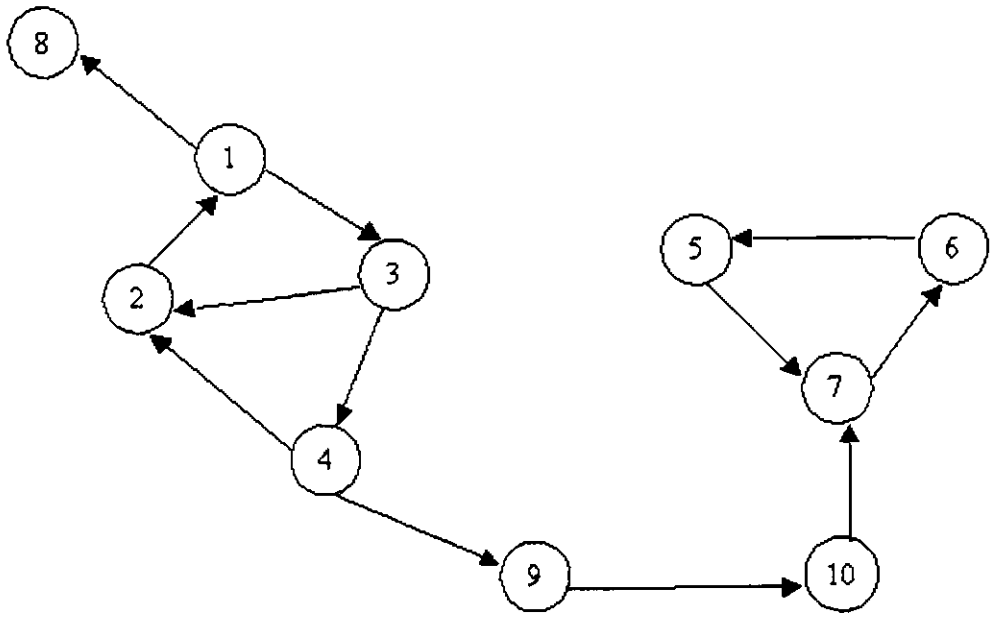


Figure 2.7 Directed graph with connected components

A tree T is a spanning tree of G if T is a spanning subgraph of G. Every spanning tree of a connected N-node graph G has $(N\text{-}1)$ edges. Spanning trees have many interesting properties, which make them useful in designing communication networks. If the objective were to simply design a connected network of minimum cost, a tree would be the optimal solution. Closely related to this is the fact that there is exactly one path between every pair of nodes in a tree. This makes routing a trivial problem in trees and greatly simplifies the communications equipment involved.

Any forest with k components contains exactly $N\text{-}k$ edges. This can be seen by noting that a graph with N nodes and no edges has N components, and each edge added connects two previously unconnected components thereby reducing the number of components by one.

A set of edges whose removal disconnects a graph (or, more generally, increases the number of its components) is called a **disconnecting set**. A disconnecting set, which partitions the set of nodes into two sets, X and Y, is called a **cutset** or sometimes a **XY-cutset**. The main concern is often the minimal cutsets (i.e., cutsets which are not subsets of other cutsets). In a tree, any edge is a minimal cutset. A minimal set of nodes whose removal partitions the remaining nodes into two connected sets is called a **cut**. Again, the interest is usually with minimal cuts.

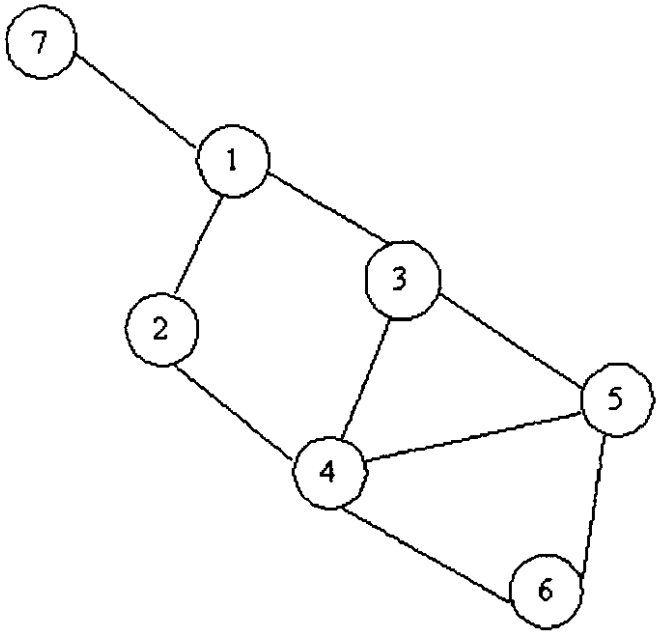


Figure 2.8 Cutsets, cuts, trees

Example 2.2 Figure 2.8 shows an undirected graph. The sets of edges

$$\{(1, 3), (2, 4)\}$$

and

$$\{(3, 5), (4, 5), (5, 6)\}$$

are examples of minimal cutsets. The latter set is an example of the fact that the set of all edges incident on any node is a cutset separating that node from all the others. The set of nodes {3, 4} is a cut. The node 1 by itself is also a cut. A single node whose removal disconnects the graph is called an articulation point.

The set of edges

$$\{(1, 2), (1, 3), (1, 7), (3, 4), (3, 5), (5, 6)\}$$

is a tree. Any subset of this set, including the entire set and the empty set, is a forest.

2.1.9 Nodes with supply and or demand

With each node i in G we can associate a number $b(i)$ that is the available supply of an item (if $b(i) > 0$) or the required demand for the item (if $b(i) < 0$). Nodes with $b(i) > 0$ are called source or supply nodes, while nodes with $b(i) < 0$ are called destination or sink nodes. If $b(i) = 0$, then none of the items are available at node i and none is required. In this case node i is sometimes called an intermediate or transshipment node.

According to Hu [30], a network can be considered as a pipeline system with the edges representing pipelines, the source being the inlet of the water, the sink being the outlet of the water, and all other nodes just being junctions between pipelines.

2.1.10 Flow (x_{ij}^k)

With each edge (i, j) in network G we associate an amount of flow denoted by x_{ij}^k for commodity k (≥ 1) if more than one commodity flows across the network. We assumed that ($x_{ij}^k \geq 0$).

2.1.11 Flow cost (c_{ij}^k)

The flow (or shipping) cost c_{ij}^k denotes the unit price of shipping a unit of commodity k on edge (i, j) from node i to node j. We assume that the flow cost varies linearly with the amount of flow. *Cost* includes the monthly lease cost, installation cost, cost per unit traffic, and maintenance costs. Costs vary depending on whether the facility is owned or leased. Usually, according to Kershenbaum [32], cost is modelled simply by a monthly cost figure, or, if cost is usage sensitive, by a cost per unit of usage (e.g., cost per hour or cost per bit).

Both the methodology used and the topologies selected in the design of a network are strongly influenced by the way cost varies, and it's important for the designer to take this into account. Thus, if cost varies roughly linearly with distance we may choose a very different topology than if cost is relatively invariant with respect to distance.

2.1.12 Capacity (u_{ij}^k) and lower bound (l_{ij}^k) of flow on an edge

With each edge $(i, j) \in A$ we associate a capacity u_{ij}^k that denotes the maximum flow on the edge (i, j) and a lower bound l_{ij}^k that denotes the minimum amount flow permitted on the edge (i, j) for commodity k. *Capacity* is the amount of traffic the channel or edge can carry. If the traffic is data, the units of capacity are usually bits per second (bps). Sometimes we speak of characters per second or messages per second. If the traffic is voice traffic, we may refer to capacity in terms of the number of simultaneous telephone calls that can be carried. If the voice traffic is digitized (i.e., turned into a bitstream) it is treated as data.

2.1.13 Tails and heads

A directed edge (i, j) has two endpoints i and j. We refer to a node i as the **tail** of edge (i, j) and node j as its **head**. We say that edge (i, j) emanates from node i and terminates at node j. An edge (i, j) is incident to nodes i and j. The edge (i, j) is an outgoing edge of node i and an incoming edge of node j. Whenever we have an edge $(i, j) \in A$, we say that node j is adjacent to node i.

2.1.14 Degrees of a node

The indegree of a node is the number of incoming edge of that node and its outdegree is the number of its outgoing edges. The degree of a node is the sum of its indegree and outdegree.

2.1.15 Adjacency list

The **edge adjacency** list $A(i)$ of a node i is the set of edges emanating from that node, that is, $A(i) = \{(i, j) \in A : j \in N\}$. The **node adjacency** list $A(i)$ is the set of nodes adjacent to that node; in this case, $A(i) = \{j \in N : (i, j) \in A\}$. Often, we shall omit the terms "edge" and "node" and simply refer to the adjacency list. In all cases it will be clear from the context whether we mean edge

adjacency list or node adjacency list. We assume that edges in the adjacency list A(i) are arranged so that the head nodes of edges are in increasing order.

Notice that |A(i)| equals the outdegree of node i. Since the sum of all node outdegrees equal m, we immediately obtain the following property:

$$\sum_{i \in N} | A(i) | = m$$

2.2 Chapter summary

This chapter gave some background on network notations and definitions. In the following chapter methods for representing these network models in a computerized system will be illustrated. Sometimes it may be desirable to transform certain network problems. Some of the transformation methods will also be discussed. In order to solve these models we must be able to formulate them in mathematical form. More of these models and their mathematical representations will be presented in chapter 3.

3

NETWORK REPRESENTATION AND MATHEMATICAL MODELS

Many network problems can be represented in mathematical programming format. In this chapter we consider both situations where the problem can be described as a flow problem on a given network or a problem where both the topology and the flow of the network have to be determined.

3.1 Representing network problems

Linear programming is concerned with the maximization or minimization of a linear expression - called the objective function - subject to linear constraints, which may be in the form of linear equations or linear inequalities. This way of formulating network problems is particularly suitable for use with computers. In the case where topological decisions also have to be made, it is usual to formulate the problem as a Mixed Integer Linear Program (MILP).

In some cases of network planning the objective may be to design a topology for the network where only potential sites for nodes are available and potential edges may be introduced, often at the expense of a fixed design cost. In such cases the introduction of an edge between node i and node j can for example be represented by $x_{ij} = 1$, where $x_{ij} = 0$ represents the situation where the edge between i and j is not introduced. These network flow problems may occur in the design and analysis of communications systems, oil pipeline systems, scheduling problems, and a variety of other areas.

If there are non-linear functions in the problem formulation it is called a non-linear program. Such cases may occur for example if the cost is not linear in the flow along the edges. In the following section we discuss some of the network representations and network transformations.

3.2 Network representations

According to Ahuja *et al.* [1], the performance of a network algorithm depends not only on the algorithm, but also on the manner used to represent the network within a computer and the storage scheme used for maintaining and updating the intermediate results. By representing a network in a more clever way and by using improved data structures, the running time of an algorithm can often be improved.

In representing a network we typically need to store two types of information: (1) the network topology, that is, the network's node and edge structure, and (2) data such as costs, capacities, and supplies/demands associated with the network's nodes and edges.

3.2.1 Node-edge incidence matrix representation

The node-edge incidence matrix representation, or simply the incidence matrix representation, stores the network as an n x m matrix N, which contains one row for each node of the network and one column for each edge. For a network we can give a representation by using the concept of an incidence matrix where all elements are zero but a column corresponding to edge (i, j) has only two nonzero elements: It has a +1 in the row corresponding to node i and a -1 in the row corresponding to node j.

Thus only $2m$ of its nm entries are nonzero, all of the nonzero entries are +1 or -1, and each column has exactly one +1 and one -1. Furthermore, the number of +1's in the row equals the outdegree of the corresponding node and the number of -1's in the row equals the indegree of the node. We consider a network example from Ahuja *et al.* [1] in Figure 3.1 and give its Node-edge incidence matrix in Figure 3.2.

Because the node-edge incidence matrix N contains so few nonzero coefficients, the incidence matrix representation of a network is not very space efficient.

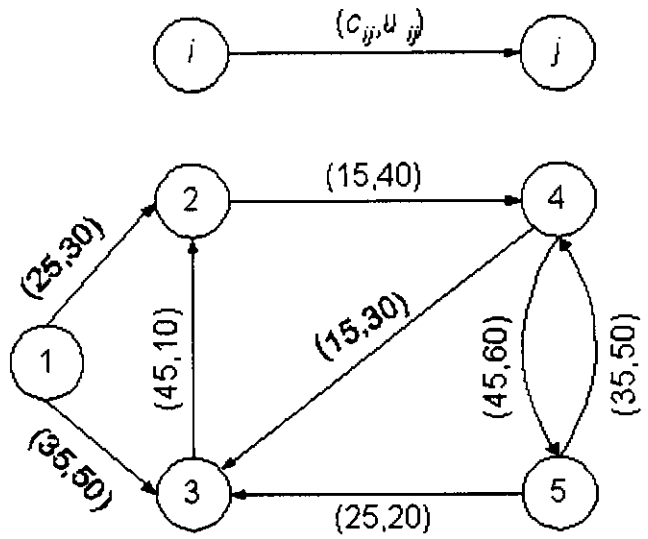


Figure 3.1 The Network example

		Edges							
		(1, 2)	**(1, 3)**	**(2, 4)**	**(3, 2)**	**(4, 3)**	**(4, 5)**	**(5, 3)**	**(5, 4)**
Nodes	1	1	1	0	0	0	0	0	0
	2	-1	0	1	-1	0	0	0	0
	3	0	-1	0	1	-1	0	-1	0
	4	0	0	-1	0	1	1	0	-1
	5	0	0	0	0	0	-1	1	1

Figure 3.2 Node-edge incidence matrix of the network sample displayed in Figure 3.1

3.2.2 Node-node adjacency matrix representation

The node-node adjacency matrix representation, also referred to as the adjacency matrix representation, stores the network as an n x n matrix $H = \{h_{ij}\}$. The matrix has a row and a column corresponding to every node, and its ijth entry h_{ij} equals 1 if $(i, j) \in A$, and equals 0 otherwise. If we wish to store edge costs and capacities as well as the network topology, we can store this information in two additional n x n matrixes C and U.

The adjacency matrix has n^2 elements, only m of which are nonzero. Consequently, this representation is space efficient only if the network is sufficiently dense; for sparse networks this representation also wastes considerable space. The simplicity of the adjacency matrix representation makes it easy to implement.

		1	2	3	4	5
	1	0	1	1	0	0
	2	0	0	0	1	0
Nodes	3	0	1	0	0	0
	4	0	0	1	0	1
	5	0	0	1	1	0

Figure 3.3 Node-node adjacency matrix of the network sample displayed in Figure 3.1

3.2.3 Adjacency list representation

Earlier we defined the **edge adjacency** list A(i) of a node i as the set of edges emanating from that node, that is, the set of edges $(i, j) \in$ A obtained as j ranges over the node of the network. Similarly we defined the **node adjacency** list A(i) of a node i as the set of nodes j for which $(i, j) \in$ A. The adjacency list representation stores the node adjacency list of each node as a singly linked list (a description of singly linked lists is described in Appendix A).

A linked list is a collection of cells each containing one or more fields. The node adjacency list for node i will be a linked list having |A(i)| cells and each cell will correspond to an edge $(i, j) \in$ A. The cell corresponding to the edge (i, j) will have as many fields as the amount of information we wish to store. One data field will also store node j. Each cell will contain one additional field, called the edge, which stores a pointer to the next cell in the adjacency list. If a cell happens to be the last cell in the adjacency list, by convention we set its edge value equal to zero. For the network example presented in Figure 3.1, we give the adjacency list representation in Figure 3.4.

Since we need to be able to store and access n linked lists, one for each node, we also need an array of pointers where each element points to the first cell in each linked list. We accomplish this objective by defining an n-dimensional array, *first*, whose element *first*(i) stores a pointer to the first cell in the adjacency list of node i. If the adjacency list of node i is empty, we set *first*(i) = 0. This array *first* in Figure 3.4 corresponds to the first column.

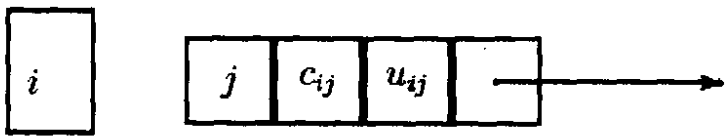
The major advantage of adjacency list representation is its ease of implementation in a programming language such as C that is able to manipulate linked lists efficiently.

3.2.4 Summary of attributes of network representations

The efficiency of the representations is summarized in Table 3.1. Other representations can be made like the *Forward Star Representation* and *Reverse Star Representation*. See Ahuja *et al.* [1]. These representations are not considered in this dissertation.

3.3 Network transformations

Network transformations are frequently required to simplify a network or to state it in a different form. In the following section some of the important network transformations will be described.



The array of pointers corresponding to the network examples displayed in Figure 3.1 called *first* is indicated below in the first column.

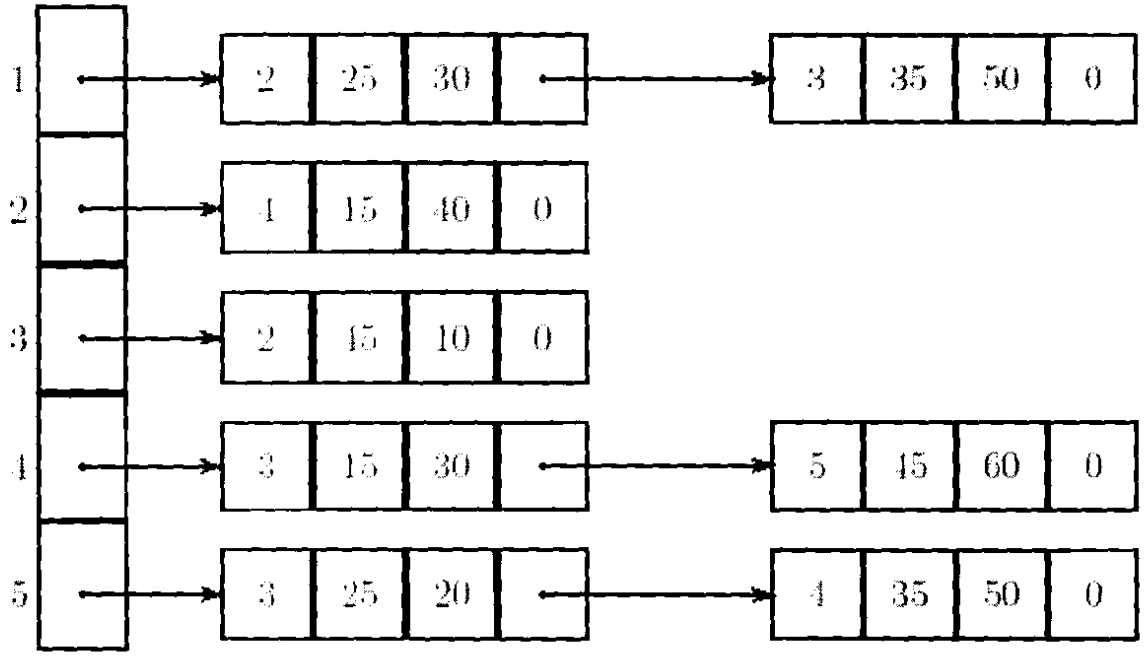


Figure 3.4 Adjacency list representation of the network example displayed in Figure 3.1

Network representation	Storage space	Features
Node-edge incidence matrix	nm	1. Space inefficient. 2. Too expensive to manipulate. 3. Important because it represents the constraint matrix of the minimum cost flow problem. 4. The storage space will have to be increased to store the information on flow costs, capacities etc.
Node-node adjacency matrix	kn^2 for some constant k	1. Appropriate for dense networks. 2. Easy to implement.
Adjacency list	$k_1n + k_2m$ for some constants k_1 and k_2	1. Space efficient. 2. Efficient to manipulate. 3. Appropriate for dense as well as sparse networks.

Table 3.1 Comparison of various network representations

3.3.1 Undirected edges to directed edges

Sometimes minimum cost flow problems contain undirected edges. An undirected edge (i, j) with cost $c_{ij} \geq 0$ and capacity u_{ij} permits flow from node i to node j and also from node j to node i; a unit of flow in either direction costs c_{ij}, and the total flow (i.e., from node i to node j plus from node j to node i) has an upper bound u_{ij}. If we have $c_{ij} \neq c_{ji}$, then it can also be accommodated in the model. That is, the undirected model has the constraint $x_{ij} + x_{ji} \leq u_{ij}$ and the term $c_{ij}x_{ij} + c_{ji}x_{ji}$ in the objective function. Since the cost $c_{ij} \geq 0$, in some optimal solution one of x_{ij} and x_{ji} will be zero. We refer to any such solution as non-overlapping.

We refer to the undirected edge (i, j) as $\{i, j\}$ and assume (with some loss of generality) that the flow in either direction on edge $\{i, j\}$ has a lower bound of value 0. Our transformation is not valid if the edge flow has a nonzero lower bound or the edge cost c_{ij} is negative. To transform the undirected case to the directed case, we replace each undirected edge $\{i, j\}$ by two edges (i, j) and (j, i), both with cost c_{ij} and capacity u_{ij}. See Ahuja *et al.* [1].

3.3.2 Removing nonzero lower bounds

If an edge (i, j) has nonzero lower bound l_{ij} on the flow edge x_{ij}, we can replace x_{ij} by $x'_{ij} + l_{ij}$ in the problem formulation. The flow bound constraints then become $l_{ij} \leq x'_{ij} + l_{ij} \leq u_{ij}$ or $0 \leq x' \leq (u_{ij} - l_{ij})$. Making this substitution in the mass balance constraints decreases $b(i)$ by l_{ij} units and increases $b(j)$ by l_{ij} units. This substitution changes the objective function value by a constant that we can record separately and then ignore when solving the problem. The following figure illustrates this transformation graphically.

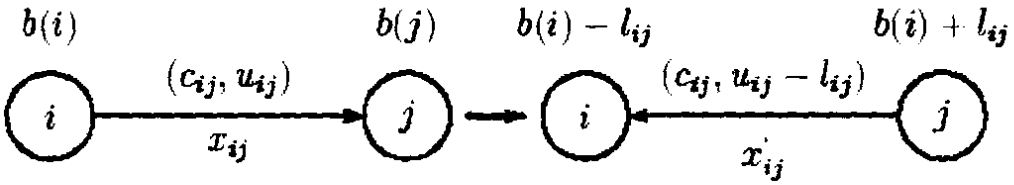


Figure 3.5 Graphical representation of removing nonzero lower bounds

3.3.3 Networks with several sources and destinations

According to Dolan and Aldous [19], in many networks arising in practice there are a large number of sources and destinations, corresponding to factories and markets in economic networks, telephony subscribers in telecommunication networks etc.

If a network has several sources S_1, S_2, ..., S_n and several destinations D_1, D_2, ...,D_n, we can transform it into a network with only one source node S and one destination node D by adjoining S to all existing sources S_1, S_2, ..., S_n and adjoining destination D to all existing destinations D_1, D_2, ..., D_n. Each added edge (S, S_i) is assigned a capacity larger than or equal to the sum of capacities of the edges out of S_i and each added edge (D_i, D) is assigned a capacity larger than or equal to the sum of the capacities of the edges entering D_i.

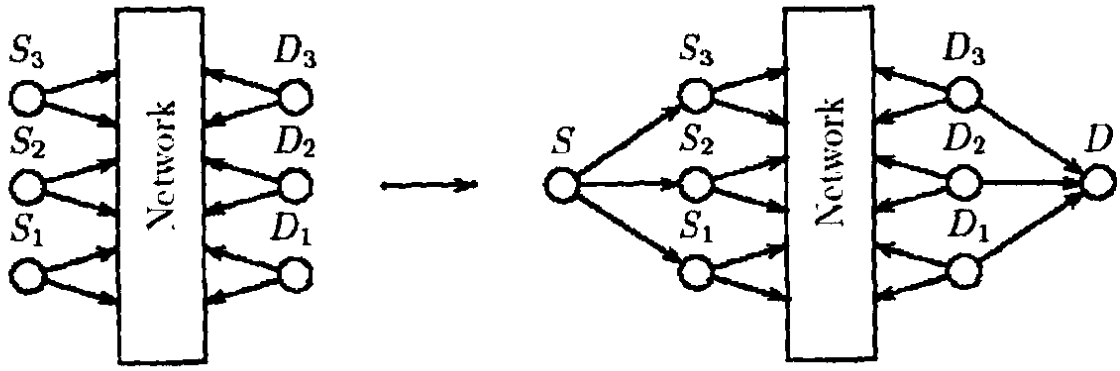


Figure 3.6 Graphical representation of transforming networks with multiple sources and destinations to a single source and single destination

3.3.4 Other network transformations

Other network transformations that can be performed on a network include *Edge Reversal*, *Removing of Edge Capacities* and *Node Splitting*. These transformations are not considered in this dissertation. More information on these transformations can be obtained from Ahuja *et al.* [1].

3.4 Some flow problems for network design

3.4.1 Tree knapsack problem

According to Van der Merwe [50], the Tree Knapsack Problem (TKP) can be seen as choosing a subtree of a tree. Given an undirected tree $T = (N, A)$ with n nodes rooted at node 0, where $V = 0, 1,..., n-1$ is the set of nodes labelled in either a breadth or depth first manner and E is the set of edges. Assume that the demand of a node is satisfied fully, or not at all, this being the indivisible demand assumption. Also assume the following:

d_i = **demand used by including node** i **in the subtree,**

c_i = **profit gained by including node** i **in the subtree,**

p_i = **the predecessor or parent of node** i**,**

H = **the total capacity of the knapsack.**

In the subtree a node can only be included if the parent of the node is also included in the subtree. This is the only additional restriction added to the model of the simple 0-1 *knapsack problem*. This can also be stated such that if a node i is to be included, all the nodes on the unique path between node i and the root node 0 must also be included. This will be referred to as the *contiguity assumption*. The tree knapsack problem includes the 0-1 knapsack problem as a special case and can be formulated as follows:

$$\textbf{Maximize} \quad \sum_{i=0}^{n-1} c_j x_j \tag{3.34}$$

$$\textbf{Subject to} \quad x_{p_i} \geq x_i, \qquad i = 1, 2, \ldots, n-1, \tag{3.35}$$

$$\sum_{j=0}^{n-1} d_j x_j \leq H, \tag{3.36}$$

$$x_j \in \{0, 1\} \qquad j = 0, 1, \ldots, n-1 \tag{3.37}$$

A node i has value $x_i = 1$ if the node is included in the subtree and $x_i = 0$ otherwise. The tree can be visualized in the usual manner. In Figure 3.2 the nodes of a sample tree are labelled in a breadth

first manner. To visualize the *contiguity assumption* note that node 12 cannot be included in a subtree without nodes 0, 2 and 6 being included as well.

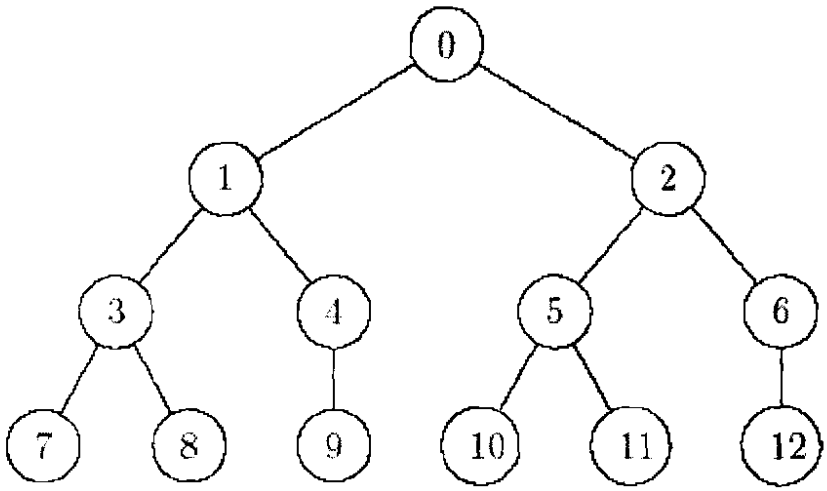


Figure 3.7 Sample tree as illustrative example of a TKP instance

3.4.2 Uncapacitated network design problem

Let $G = (N, A)$ be a directed network with f_{ij} the design (construction) cost and c_{ij} the per-unit flow cost of edge (i, j). The problem is to find the design that minimizes the total network cost. Although we consider the case here of multiple commodities (K in number) we will for simplicity considerations first assume that each commodity k has a single source node s^k and a single destination node d^k. Once an edge is introduced into the network we assume that we will have sufficient capacity to route all of the flow of all commodities on this edge (uncapacitated assumption).

Let x^k denote the vector of flows of commodity k on the network. Rather than letting x_{ij}^k model the total flow of commodity k on edge (i, j), however, we let x_{ij}^k denote the fraction of the required flow of commodity k to be routed from the source s^k to destination d^k that flows on edge (i, j).

Let c^k denote the cost vector for commodity k, which we scale to reflect the way that we have defined x_{ij}^k [i.e., c_{ij}^k is the unit cost for each commodity k on edge (i, j) times the flow requirement

of that commodity.] Also let y_{ij} be a zero-one vector indicating whether or not we select edge (i, j) as part of the network design. We also define f to be the vector of fixed costs (design costs) of designing the edges (i, j).

According to Ahuja *et al.* [1], the uncapacitated network design problem can then be formulated as follows:

$$\textbf{Minimize} \quad \sum_{1 \le k \le K} c^k x^k + fy \tag{3.38}$$

$$\textbf{Subject to} \quad \sum_{\{j:(i,j)\in A\}} x_{ij}^k - \sum_{\{j:(j,i)\in A\}} x_{ji}^k$$

$$= \begin{cases} 1 & \text{if } i = s^k \\ -1 & \text{if } i = d^k \quad \forall i \in N, k = 1, 2, \ldots, K \\ 0 & \text{otherwise} \end{cases} \tag{3.39}$$

$$x_{ij}^k \le y_{ij} \qquad \forall (i,j) \in A, \quad k = 1, 2, \ldots, K \tag{3.40}$$

$$x_{ij}^k \ge 0 \qquad \forall (i,j) \in A \text{ and all } k = 1, 2, \ldots, K \tag{3.41}$$

$$y_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A \tag{3.42}$$

3.4.3 Capacitated network design problem

The capacitated problem is defined in the same manner as the uncapacitated network design problem. The only difference is that for the upper bounds the total flow of commodities must be smaller than or equal to the total capacity of the specific edge (i, j), if that edge is included in the topology (see constraint 3.45). For the lower bounds we must similarly ensure that the flow of each commodity on an edge (i, j) is larger than the lower bound for that edge, if the edge is part of the design (see equation 3.46). The mathematical formulation for this type of problem (as given by De Villiers [14]) is given below.

$$\textbf{Minimize} \quad \sum_{1 \le k \le K} c^k x^k + fy \tag{3.43}$$

$$\textbf{Subject to} \quad \sum_{\{j:(i,j)\in A\}} x_{ij}^k - \sum_{\{j:(j,i)\in A\}} x_{ji}^k$$

$$= \begin{cases} 1 & \text{if } i = s^k \\ -1 & \text{if } i = d^k \quad \forall i \in N, k = 1, 2, \ldots, K \\ 0 & \text{otherwise} \end{cases} \tag{3.44}$$

$$\sum_{1 \le k \le K} \frac{1}{u_{ij}^k} x_{ij}^k \le y_{ij} \qquad \forall (i,j) \in A, \quad k = 1, 2, \ldots, K \tag{3.45}$$

$$l_{ij}^k y_{ij} \le x_{ij}^k \qquad \forall (i,j) \in A, \quad k = 1, 2, \ldots, K \tag{3.46}$$

$$x_{ij}^k \ge 0 \qquad \forall (i,j) \in A \text{ and all } k = 1, 2, \ldots, K \tag{3.47}$$

$$y_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A \tag{3.48}$$

Where

c^k = cost vector for commodity k on links (i, j)

x^k = flow vector for commodity k on links (i, j)

f = vector of design costs of links

y = vector that indicates the existence of links (i, j)

A = set of all potential links (edges / arcs)

s^k = index of the source node of commodity k

d^k = index of the sink node of commodity k

K = number of commodities

u_{ij}^k = upper flow bound on a link (i, j) for commodity k

l_{ij}^k = lower flow bound on a link (i, j) for commodity k

3.5 Reliability of networks

The network design problem is challenging, both as an engineering and as a mathematical one, even when all of the sites behave in the mode for which the network is designed and all links provide the level of service expected. In real life, however, through human error, design faults, operational faults, environmental factors, or random wear-out, sites and links do not always function correctly. The network planner must therefore address *reliability issues.*

Satisfactory techniques to anticipate and accommodate component failures are, in many ways, less well understood than design and performance in a "failure-free" setting. Our effort here is to give some discussion of reliability and some references to the interested reader. One of the hardest tasks faced by a network designer is anticipating the sources of failures and reduced performance. Different types of failures require different responses. Guarding against intentional damage requires an assessment of the portions of the network that are sensitive, and the likelihood of damage within that region; the extent of damage possible, together with the estimate of how likely that damage is, may dictate a response ranging from the redesign of the network to avoid the sensitive area totally, or the reinforcement of communication links in that portion, to simply accepting the (hopefully low) probability of extensive damage.

Network reliability concerns the capability of the underlying network to provide connections to support required network functionality. In small networks with relatively unreliable components, disconnection is a major concern. This is one reason why there has been an almost total focus in the literature on measures of network connectivity, often under simplifying assumptions about failure causes and probabilities. A second reason according to Colbourn [14] is that, even with many simplifying assumptions, the problems that remain are challenging and not yet solved in a satisfactory way.

According to Soriano *et al.* [45], a network is said to be *survivable* if traffic interrupted by the failure of some of its elements can be rerouted via spare or excess capacity specifically placed in the network for that purpose. In addition, if that rerouting process can be automated and the

network can reconfigure itself to cope with failures without human intervention, then the network will be called *self-healing*. Of course, designing a network to survive any type of failure even those involving several elements at the same time would result in extremely costly designs. However, since is it generally considered that failures affecting more than one element at a time are extremely improbable; operators will instead define a restricted set of realistic failure scenarios for which the network will need to be survivable. For our research we did not consider reliability, but we considered survivability of the networks, which is the discussion of the next section.

3.6 Models for survivable network design

There are two concepts dealing with network failure, reliability and survivability. Reliability is the probability that a network functions according to a specification. Survivability is the ability of a network to perform according to a specification after it has been damaged. In most cases, this means that the network is still functional after the failure of certain network components.

Under certain assumptions, survivability implies 100% reliability. There is also a less discussed term called vulnerability, which is concerned with the difficulty of destroying the network. There are two survey papers, Boesch [7], and Christofides and Whitelock [11] that can give the reader an excellent introduction to the basics of reliability and survivability. It should be noted at the outset that our work has focused on survivability and not reliability.

Survivability is extremely important in modern telecommunication networks. Currently fiber-optic technology is being rapidly deployed to cover larger regions and carry more information. Losing end-to-end customer service for example in a fiber-optic network even for a minute or two could lead to millions of dollars of lost revenue for commercial providers of telecommunication services like AT&T. Thus designing a network topology that provides protection against link failure is vitally important. The important practical and theoretical problems of designing survivable communication networks are outlined in the report of Grotschel *et al.* [27].

A network is k-connected if each demand pair has at least k disjoint paths. A set of disjoint paths is one where there are no nodes or edges in common between any of the paths except the origin and

destination nodes. If two paths are edge-disjoint, they have no edge in common. If two paths are node-disjoint, they have no nodes in common. Being node-disjoint is stronger than being edge-disjoint. If two paths are node-disjoint, they are also edge disjoint. Two paths could not share an edge without sharing the vertices that terminate that edge.

A k-connected network, $k \geq 2$, has survivability properties. If there are k - 1 distinct failures, then at most k - 1 paths can be affected since no failure can affect more than one path. Therefore, with up to k - 1 failures we are guaranteed at least one unaffected path for each demand pair and all communication attempts will be successful. With today's technology the probability of a failure is small, but still potentially harmful.

In the survivable network design problem, we have a set of nodes and want to determine where to place links in a cost effective manner. The underlying graph for the network is not necessarily complete. The objective is to minimize all the costs involved in building the network. The problem we consider is constrained to have at least two disjoint paths for every demand pair. The problem of designing such a network at minimum cost is an NP-hard problem. This is why this dissertation concentrates on some problems of limited size in terms of the number of nodes and edges using exact methods as opposed to heuristics. See Clarke and Anandalingam [12].

Before we consider the model, we have to introduce certain concepts and notation for survivability issues. We follow the notation in Grotschel *et al.* [27] for these purposes. The problem of designing survivable mesh networks can be modelled as a minimum cost network design problem with certain low-connectivity constraints. We are given a *graph* $G = (V, A)$, where V is a set of *nodes* that represents offices that must be interconnected by a network, and A is a collection of *edges* that represent the possible pairs of nodes between which direct transmission links can be placed.

The *survivability conditions* require that the network satisfy certain edge and node connectivity requirements. In particular, a nonnegative integer r_s is associated with each node $s \in V$ that represents its *connectivity requirement*. We implicitly assume that $G = (V, A)$ is a graph and r a vector of node connectivity types with $r \in \{0, 1, 2\}^V$.

For each pair of distinct nodes $s,t \in V$, the network $N = (V, F)$ where $F \subseteq A$ to be designed has to contain at least $r(s, t) := \min\{r_s, r_t\}$ edge-disjoint (or node-disjoint) $[s, t]$ paths. We define the concept of a cut (induced by $W \subseteq V$) as $\delta(W) := \{ij \in A \mid i \in W, j \in V \setminus W\}$ or $\delta_G(W)$ to make it clear that the graph G is referred to.

We extend the connectivity requirement function r to functions operating on sets by setting:

$$r(W) := \max\{r_s \mid s \in W\} \text{ for all } W \subseteq V \text{ and}$$

$$con(W) := \max\{r(s, t) \mid s \in W, t \in V \setminus W\} \text{ for all } W \subseteq V \text{ and } \phi \neq W \subset G.$$

For any subset of links $F \subseteq A$, we define

$$Q(F) := \sum_{(i,j)\in F} q_{ij}. \text{ Where } q_{ij} = \begin{cases} 1 & \text{if } (i,j) \in F \\ 0 & \text{otherwise} \end{cases}$$

For our purposes we decided as a first step to concentrate on the network topology ignoring edge (arc or link) capacities and concentrating on survivability issues of the network.

An important practical case is where connectivity requirements satisfy $r_s \in \{0, 1, 2\}$ for all $s \in V$. See also Cardwell *et al.* [9] and [10], and Monma and Shallcross [37], where they address the problem of designing survivable fiber optic networks. We follow the notation of Grotschel *et al.* [27], and define 2ECON (and 2NCON) problems where edge-disjoint (respectively node-disjoint) paths are required.

Formally a 2ECON or 2NCON problem is given by (G, r) where we implicitly assume that $G = (V, A)$ is a graph and r a vector of node types with $r \in \{0, 1, 2\}^V$ as mentioned before. Grotschel *et al.* [27] then considered a model where the objective only considers fixed costs c_{ij} if edge ij is used in the design.

The model the can be formulated as:

$$Minimize \qquad \sum_{(i,j)\in A} c_{ij} q_{ij}$$

Subject to the constraints

$$Q(\delta(W)) \geq con(W) \qquad \forall\, W \subseteq V, \quad \phi \neq W \neq V; \qquad (1a)$$

$$Q(\delta_{G-z}(W)) \geq 1 \qquad \forall\, z \in V, \text{ and } \forall\, W \subseteq V \setminus z, \quad \phi \neq W \neq V \setminus z \text{ with } r(W) = 2 \text{ and } r(V \setminus (W \cup z)) = 2; \qquad (1b)$$

$$q_{ij} \in \{0,1\} \qquad \forall\, ij \in A. \qquad (1c)$$

Where q_{ij} denote variables that take on values 1 (*or* 0) for the case where edge (i, j) is used (not used) in the design. It follows from Menger's theorem that, for every feasible solution X of (1), the subgraph $N = (V, F)$ of G defines a network satisfying the two-connected node survivability requirements. Menger's theorem is a basic result about connectivity in finite undirected graphs. According to Wikipedia [51], Menger's theorem states that given a finite undirected graph G and two nonadjacent nodes i and j, the size of the minimum node cut for node i and node j (the minimum number of nodes whose removal disconnects node i and node j) is equal to the maximum number of pair-wise node-independent paths from node i to node j. It is indicated that the solution to this model ignoring $(1b)$ gives rise to a so-called 2ECON problem that protects independent paths for certain node pairs. Similarly they use the notation of 2NCON for problems where node independence of paths between certain high-connectivity nodes also needs to be protected. For 2NCON problems $(1b)$ must be included in the model. More will be said about this in a later chapter.

Illustrative Example

We give now an illustrative example of 2ECON and 2NCON problems based on the model formulation given above. Figure 3.6 gives a network design problem for five nodes.

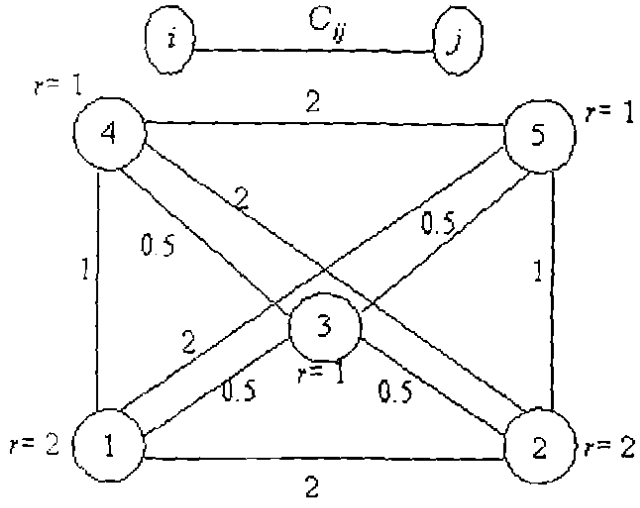


Figure 3.8 An example of a 5-node network

In this example we use the above edge costs as the cost for using a particular edge in the network. The r denotes the connectivity of the nodes. The cost figures and connectivity of the nodes were chosen to illustrate the 2ECON and 2NCON concepts.

To solve this network survivable problem we begin by constructing edge cut sets and node cut sets. Then we construct constraints from the generated cut sets, after which we construct the linear program problem, which we then solve with the CPLEX software. We give the entire procedure in appendix C. A part of the problem formulation is given below:

The Network considered is $V = \{1, 2, 3, 4, 5\}$

As a first task to construct constraints $(1a)$ in the model all sets W have to be identified. First, we note that the number of such W-sets are:

The number of combinations of set size 1 = 5

The number of combinations of set size 2 = 10

The number of combinations of set size 3 = 10

The number of combinations of set size 4 = 5

The total number of combinations to be considered = 30

Obviously these 30 combinations contain duplicates.

The Output of Set Combinations:

First, taking

$W1 = \{1\}$ $W2 = \{2\}$ $W3 = \{3\}$ $W4 = \{4\}$ $W5 = \{5\}$

We find the edge sets induced by W1 to W5 as:

$\delta(W1) = \{(1,2)\ (1,3)\ (1,4)\ (1,5)\}$ $\quad con(W1) = 2$

$\delta(W2) = \{(1,2)\ (2,3)\ (2,4)\ (2,5)\}$ $\quad con(W2) = 2$

$\delta(W3) = \{(1,3)\ (2,3)\ (3,4)\ (3,5)\}$ $\quad con(W3) = 1$

$\delta(W4) = \{(1,4)\ (2,4)\ (3,4)\ (4,5)\}$ $\quad con(W4) = 1$

$\delta(W5) = \{(1,5)\ (2,5)\ (3,5)\ (4,5)\}$ $\quad con(W5) = 1$

Similarly for 2-node combinations we have:

$W6 = \{1,2\}$ $W7 = \{1,3\}$ $W8 = \{1,4\}$ $W9 = \{1,5\}$ $W10 = \{2,3\}$ $W11 = \{2,4\}$

$W12 = \{2,5\}$ $W13 = \{3,4\}$ $W14 = \{3,5\}$ $W15 = \{4,5\}$

and

$\delta(W6) = \{(1,3)\ (2,3)\ (1,4)\ (2,4)\ (1,5)\ (2,5)\}$ $\quad con(W6) = 1$

$\delta(W7) = \{(1,2)\ (2,3)\ (1,4)\ (3,4)\ (1,5)\ (3,5)\}$ $\quad con(W7) = 2$

$\delta(W8) = \{(1,2)\ (2,4)\ (1,3)\ (3,4)\ (1,5)\ (4,5)\}$ $\quad con(W8) = 2$

$\delta(W9) = \{(1,2)\ (2,5)\ (1,3)\ (3,5)\ (1,4)\ (4,5)\}$ $\quad con(W9) = 2$

$\delta(W10) = \{(1,2)\ (1,3)\ (2,4)\ (3,4)\ (2,5)\ (3,5)\}$ $\quad con(W10) = 2$

$\delta(W11) = \{(1,2)\ (1,4)\ (2,3)\ (3,4)\ (2,5)\ (4,5)\}$ $\quad con(W11) = 2$

$\delta(W12) = \{(1,2)\ (1,5)\ (2,3)\ (3,5)\ (2,4)\ (4,5)\}$ $\quad con(W12) = 2$

$\delta(W13) = \{(1,3)\ (1,4)\ (2,3)\ (2,4)\ (3,5)\ (4,5)\}$ $\quad con(W13) = 1$

$\delta(W14) = \{(1,3)\ (1,5)\ (2,3)\ (2,5)\ (3,4)\ (4,5)\}$ $\quad con(W14) = 1$

$\delta(W15) = \{(1,4)\ (1,5)\ (2,4)\ (2,5)\ (3,4)\ (3,5)\}$ $\quad con(W15) = 1$

Since the 3-node combinations give the same edge sets as the 2-node combinations, it is not necessary to do them. The same is true for the 4-node combinations as that of the 1-node combinations.

We thus have 15 constraints in (1*a*). The first constraint is for example:

C1: y12 + y13 + y14 + y15 >= 2

The others C2 to C15 are given in the full statement of the problem in Appendix C.

Solving the resulting integer 2ECON linear program gives an optimal solution of 4 for the objective function. Figure 3.7 also gives the solution network of the 2ECON survivable problem. This network is 2ECON survivable but not 2NCON survivable since the two edge-independent paths between nodes 1 and 2 share node 3.

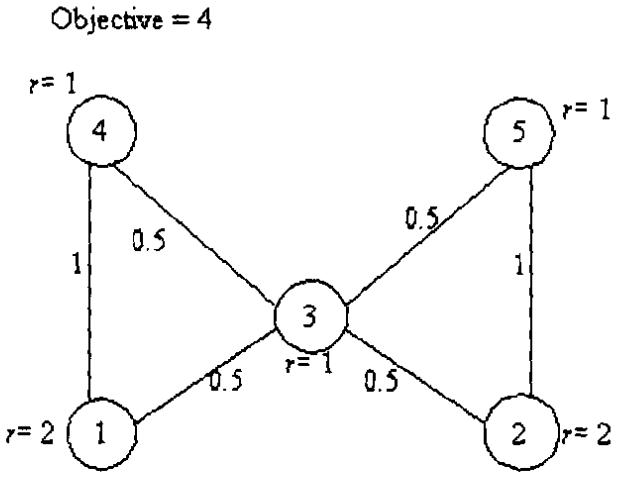


Figure 3.9 Network solution of 2ECON

To formulate the constraints (1*b*) in the model we note that the information for (1*a*) can be used selectively for each choice of z and W. If we take $z = 3$ for example and $W = \{1\}$. We find $r(W) = r(\{1\}) = 2$ and $r(V\backslash(W \cup z)) = r(\{2,4,5\}) = 2$ giving $\delta_{G-z}(W) = \delta_{\{1,2,4,5\}}(\{1\}) = \{(1,2), (1,4), (1,5)\}$ and the constraint y12 + y14 + y15 >= 1 (N3C6 in the Appendix C).

Solving a 2NCON linear program problem gives an objective solution of 4. Figure 3.8 gives the solution network of 2NCON survivable problem. In this network we have 2-node survivability of nodes 1 and 2 as required.

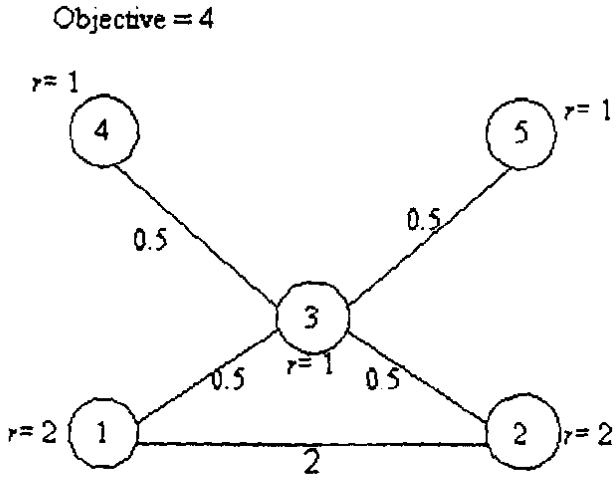


Figure 3.10 Network solution of 2NCON

General solution for 2ECON and 2NCON

Grotschel *et al.* [27] investigates various algorithmic ideas to solve 2ECON and 2NCON problems of the type and size encountered by Bell Communications Research who provided some test data. In this dissertation we decided to experiment with solutions to the exact MILP problem to ascertain the limitations that a designer can expect from this approach in terms of problem size which we loosely define as the number of nodes and potential edges in the network. This approach is partly motivated by the improvements in computer and software technology over the last few decades.

The idea is then that a network designer could "zoom in" on a portion of a network design problem and try to alleviate the survivability and other problems by using these exact models. This will obviously lead to suboptimal results in the sense that the complete network design problem is not solved but in practical situations this approach may give satisfactory results.

3.7 Chapter summary

Many different types of network problems exist. It is clear that these models can become increasingly difficult to solve as the problem size increases. Over the last few decades, algorithms have been created to solve some of these problems. In the following chapter some of these algorithms will be outlined.

4

SOLUTION STRATEGIES FOR SOME NETWORK PROBLEMS

In this chapter we give some solution strategies to some of network problems. We begin by considering integer linear program problems (ILP) and mixed integer linear program problems (MILP).

4.1 Solving ILP and MILP problems

4.1.1 Notation and definitions

In the zero-one integer linear programming problem (0-1 ILP) the objective function and the constraints are all linear and all variables are constrained to be zero or one. In some situations they are constrained to non-negative integers.

Definition 4.1.1 An ILP in which all variables are required to be integers is called a *pure (or all) integer linear programming* problem and can be represented by:

$$\textbf{Minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{4.1}$$

$$\textbf{Subject to} \quad \sum_{j=1}^{n} \mathbf{a}_j x_j \leq \mathbf{b} \tag{4.2}$$

$$x_j \geq 0 \text{ and integer} \qquad \forall j \in \{1, 2 \ldots, n\} \tag{4.3}$$

Where c_j are objective function coefficients and, $\mathbf{a_j}$ and $\mathbf{b}$ are given vectors.

Definition 4.1.2 An ILP in which only some of the variables are required to be integers is called *a mixed integer programming* problem and can be represented by:

$$\text{Minimize} \quad \sum_{j=1}^{n_1} c_j x_j + \sum_{j=1}^{n_2} d_j y_j \tag{4.4}$$

$$\text{Subject to} \quad \sum_{j=1}^{n_1} a_j x_j + \sum_{j=1}^{n_2} g_j y_j \leq b \tag{4.5}$$

$$x_j \geq 0 \quad \forall j \in \{1, 2, ..., n_1\} \tag{4.6}$$

$$y_j \geq 0 \text{ and integer} \quad \forall j \in \{1, 2, ..., n_2\} \tag{4.7}$$

Where $\mathbf{a_j}$, $\mathbf{g_j}$ and $\mathbf{b}$ are given vectors and, c_j and d_j are objective function coefficients.

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{4.8}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} a_j x_j \leq b \tag{4.9}$$

$$x_i \in \{0, 1\} \qquad \forall i \in \{1, 2 \ldots, n\} \tag{4.10}$$

where $\mathbf{a_j}$ and $\mathbf{b}$ are given vectors.

Definition 4.1.3 An ILP problem in which all variables must equal 0 or 1 is called a 0-1 ILP.

And c_j are objective function coefficients.

Definition 4.1.4 The LP obtained by relaxing all integer constraints on variables is called the LP relaxation of the ILP. (In the case of 0-1 ILP problems we retain the constraints $0 \leq x_j \leq 1$.)

4.1.2 The solution of MILP's

4.1.2.1 General considerations

For solving LP's we have general purpose and computationally effective algorithms (e.g. the Simplex Method). These methods are able to solve large LP problems. For solving IP's or MILP's no similar general purpose and computationally effective algorithms for very large problems exist.

In general IP's are a lot harder to solve than LP's. In the last few decades, significant progress has been made to solve larger instances of such problems. See for example Crowder *et al.* [16].

Two types of algorithms may be used in solving IP's. We can either use an algorithm that produces (and proves) an optimal solution or a heuristic algorithm that 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.

4.1.2.2 Exact algorithms to solve MILP's

A great variety of algorithms to solve IP optimization problems have risen during the last decades. Among the best-known exact algorithms for solving IPs or MILPs are the following methods:

4.1.2.2.1 Cutting plane algorithms for IP's/MILP's are derived from the Simplex algorithm. See Gomory [25]. After computing the continuous optimum by LP-relaxation of the integrality constraints new constraints are added to the MILP step by step. With the help of these additional inequalities (cuts) non-integer values are systematically forced to take on integer values.

This method is called the cutting plane algorithm for integer programs and was developed by Ralph Gomory in 1960. Salkin *et al.* [42] give a good description of these methods. The cutting plane method never eliminates any feasible integer solutions from the feasible region and all cuts added to the region are retained as the search process progresses, so that the feasible region shrinks continually until a solution is obtained or the problem can be proved to be infeasible.

Many researchers in this area have empirically tested this method and usually either performs very well (gets an integer solution fast) or takes an inordinate time for convergence. This has detracted from the popularity of the method. The last few decades have seen the rise in popularity of the so-called branch and bound method (a structured enumerative approach) described in subparagraph 3 below.

4.1.2.2.2 Complete Enumeration. Unlike LP's where variables in general can take on continuous values $x_{ij} \geq 0$, in IP's each variable can only take on discrete values. Hence the obvious solution approach is simply to try to enumerate all these possibilities-calculating the value of the objective function at each one and choosing the (feasible) one with the optimal value.

Suppose we have 100 integer variables each with 2 possible integer values then there are 2^{100} possibilities. It becomes clear that for large problems this approach for solving IP's is computationally infeasible. What makes solving the problem easy when it is small is precisely what makes it become hard very quickly as the problem size increases.

IP nowadays is often considered to be part of "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.

4.1.2.2.3 Efficient implicit enumerative algorithms enumerate only a fraction of feasible solutions to the IP. Salkin *et al.* [42] state that Land and Doig first suggested this method in the early 1960's. These enumerative algorithms employ pruning criteria so that not all feasible solutions have to be tested for finding the optimal solution. The widely used **Branch-and-Bound** algorithm with LP-relaxation is the most important representative of enumerative algorithms. A description of this algorithm is presented in Appendix B. This is perhaps the most well-known method (or method of choice) for solving MILP's and most of the available software codes employ some version of this algorithm. See also section 4.1.2.2.4 below.

Note that in certain forms of this method, such as complete enumeration, powers of two are also involved as we progress down the (binary) tree. However, because we do not enumerate all possible integer solutions, the computational efficiency is improved.

4.1.2.2.4 Hybrid algorithms. These algorithms combine certain strategies to find solutions faster. A well-known algorithm is the **Branch-and-Cut** algorithm that combines the structure of branch-and-bound with cutting plane strategies. See for example Padberg and Rinaldi [39] and [40], they

described a branch-and-cut algorithm for solving large-scale instances of the symmetric travelling salesman problem (STSP) to optimality.

Similarly, Terblanche [49] used a branch-and-cut solution approach for solving the multi-hour survivable network design problem entailing finding the most cost efficient network design given two or more demand matrices that represent network traffic for different (busy-) hours.

4.1.2.2.5 General comments on solution strategies.

Good computer packages (solvers) exist for finding optimal solutions to IP's/MIP's via LP-based tree search. The solver package used for problem solving in this dissertation was done in CPLEX version 8.1, on which more information will be given in following chapters.

Many of the computational advances in IP optimal solution methods (e.g. constraint aggregation, coefficient reduction, problem reduction, automatic generation of valid inequalities) are included in these packages.

Often the key to making successful use of such packages for any particular problem is to put effort into a good formulation of the problem in terms of the variables and constraints. By this we mean that for any particular IP there may be a number of valid formulations. Deciding which formulation to adopt in a solution algorithm is often the combination of an art form that combines experience and trial and error.

In the following paragraphs some algorithms are described that can be viewed as special purpose algorithms for certain defined (famous) problems.

4.2 Lagrangian relaxation technique

Lagrangian relaxation is a general solution strategy for solving mathematical programs that permits decomposition of problems to exploit their special structure. As such, this solution approach is perfectly tailored for solving many models with embedded network structure. Since it is often possible to decompose models in several ways and apply Lagrangian relaxation to each

decomposition, Lagrangian relaxation is a very flexible solution approach. Because of its flexibility, Lagrangian relaxation is more of a general problem solving strategy and solution framework than any single solution techniquc, Ahuja *et al.* [1].

Lagrangian relaxation technique is used to obtain lower bounds (for minimization problems) on the objective function values of (discrete) optimization problems. By relaxing the integrality constraints in the integer programming formulation of a discrete optimization problem, thereby creating a linear programming relaxation, we obtain an alternative method for generating a lower bound. The lower bound obtained by the Lagrangian relaxation technique is at least as sharp as that obtained by using a linear programming relaxation. As a result, and because the Lagrangian relaxation bound is often easier to obtain than the linear programming relaxation bound, Lagrangian relaxation has become a very useful lower bounding technique in practice.

Lagrangian relaxation is a flexiblc solution strategy that permits modellers to exploit the underlying structure in any optimization problem by relaxing (i.e., removing) complicating constraints. This approach permits us to "pull apart" models by removing constraints and instead place them in the objective function with associated Lagrange multipliers.

The starting point for the application and theory of Lagrangian relaxation (as applied to a model specified as a minimization problem) is a key bounding principle stating that for any value of the Lagrangian multiplier, the optimal value of the relaxed problem, called the Lagrangian subproblem, is always a lower bound on the objective function value of the problem. To obtain the best lower bound, the Lagrangian multiplier is chosen so that the optimal value of the Lagrangian subproblem is as large as possible.

We now give an example of the Lagrangian relaxation as applied to a network design from Ahuja *et al.* [1]. Suppose that we have the flexibility of designing a network as well as determining its optimal flow (routing). That is, we have a directed network $G = (N, A)$ and can introduce an edge or not into the design of the network: If we use (introduce) an edge (i, j), we incur a design (construction) cost f_{ij}. Our problem is to find the design that minimizes the total systems cost – that is, the sum of the design cost and the routing cost. This type of model arises in many application

contexts, for example, the design of telecommunications or computer networks, load planning in the trucking industry (i.e., the design of a routing plan for trucks), and the design of production schedules.

Many alternative modelling assumptions arise in practice. We consider one version of the problem, the *uncapacitated network design problem*. In this model we need to route multiple commodities on the network; each commodity k has a single source node s^k and a single destination node d^k. Once we introduce an edge (i, j) into the network, we have sufficient capacity to route all of the flow by all commodities on this edge.

To formulate this problem as an optimization model, let x^k denote the vector of flows of commodity k on the network. Rather than letting x^k_{ij} model the total flow of commodity k on an edge (i, j), however, we let x^k_{ij} denote the fraction on the required flow commodity k to be routed from the source s^k to the destination d^k that flows on an edge (i, j). Let c^k denote the cost vector for commodity k, which we scale to reflect the way that we have defined x^k_{ij} [i.e., c^k_{ij} is the per-unit cost for commodity k on an edge (i, j) times the flow requirement of that commodity]. Also, let y_{ij} be a zero-one vector indicating whether or not we select an edge (i, j) as part of the network design.

Using this notation, we can formulate the network design problem as follows:

$$\text{Minimize} \sum_{1 \le k \le K} c^k x^k + fy \tag{4.2a}$$

Subject to

$$\sum_{\{j:(i,j)\in A\}} x^k_{ij} - \sum_{\{j:(j,i)\in A\}} x^k_{ji} = \begin{cases} 1 \text{ if } i = s^k \\ -1 \text{ if } i = d^k \\ 0 \text{ otherwise} \end{cases} \quad \text{for all } i \in N, k = 1, 2, \ldots, K, \tag{4.2b}$$

$$x^k_{ij} \le y_{ij} \quad \text{for all } (i,j) \in A, k = 1, 2, \ldots, K, \tag{4.2c}$$

$$x^k_{ij} \ge 0 \quad \text{for all } (i,j) \in A \text{ and all } k = 1, 2, \ldots, K, \tag{4.2d}$$

$$y_{ij} \in \{0, 1\} \quad \text{for all } (i,j) \in A. \tag{4.2e}$$

In this formulation, the "forcing constraints" (4.2c) state that if we do not select edge (i, j) as part of the design, we cannot flow any fraction of commodity k's demand on this edge, and if we do select edge (i, j) as part of the design, we can flow as much of the demand of commodity k as we like on this edge.

Note that if we remove the forcing constraints from this model, the resulting model in the flow variables x^k decomposes into a set of independent shortest path problems, one for each commodity k. Consequently, the model is another attractive candidate for the application of Lagrangian relaxation. To see why this type of solution approach might be attractive, consider a typically sized problem with, say, 50 nodes and 500 candidate edges. Suppose that we have a separate commodity for each pair of nodes (as is typical in communication settings in which each node is sending messages to every other node). Then we have 50(49) = 2450 commodities. Since each commodity can flow on each edge, the model has 2450(500) = 1,225,000 flow variables, and since (1) each flow variable defines a forcing constraint, and (2) each commodity has a flow balance constraint at each node, the model has 1,225,000 + 2450(50) = 1,347,500 constraints. In addition, it has 500 zero-one variables. So even as a linear program, this model really extends the capabilities of current state of the art software systems. By decomposing the problem, however, for each choice of the vector of Lagrange multipliers, we will solve 2450 small shortest path problems.

4.3 Chapter summary

Some network problems that are special cases of linear programs have been discussed and methods of solving these so-called mixed integer linear programs have been discussed. Some networks where the optimal flow over the existing edges has to be determined and networks where both the topology and the flow have to be determined from many potential configurations have been presented.

In telecommunication planning many other aspects also needs to be considered before solutions can be obtained that have practical use. There is, for instance, more than one way in which a network design problem may be approached. In the following chapter we will look at some of the system consideration for network design in telecommunication.

5

SYSTEM CONSIDERATIONS FOR NETWORK DESIGN

A network design tool is a decision support system that can ideally assist the network designer in selecting the optimal or near optimal configuration from the many potential configurations in the design of new network meshes topologies. It can also assist in the optimization or redesign of existing networks to relieve congestion, for example.

Such a system may also provide the network designer with the ability to test the resilience of the network by disabling nodes and edges in the optimal design to simulate possible node or edge failures.

5.1 Structure of a network design tool

According to Kershenbaum [31], a network design tool should relieve the network designer of the burden of collecting and processing the inputs required by the tool to the extent that this is possible. It should also contain modules that allow the network designer to apply different algorithms and techniques to the design. Such a tool should present its output interactively in tabular as well as graphical form. Finally, it should allow the network designer to interact with it in all phases of the design process, both before and after the algorithms are executed. Thus, the user gets the first word, specifying allowable candidate nodes and edges, and also the last word, editing the tool's design decisions.

5.2 Components of a network design tool

5.2.1 Front end

The front end is responsible for both the input and output interfaces to the user. This is the part that makes the tool interactive. It presents the user with menus for selecting functions, it recognizes commands, it displays the network and associated information, it generates reports detailing designs and the results of analysis, and it allows the user to manipulate parameters. It further allows the user to edit the network, adding and deleting nodes and edges, and to change their status (enable or disable a potential edge or node). A good network display capability allows the user to zoom and pan, displaying geographic segments of the network. It also allows the user to move nodes to clarify the display. Finally, it permits the user to obtain hard copies of the displays, via a printer or plotter.

5.2.2 Data input and output

The user can either construct a potential network with the tools provided by the front end, or the data can be obtained from other sources. To work on realistic problems, the tool must be able to deal with bulk data needed to support the design process. The key to making this work is to rely on existing databases rather than having to input the data manually. This not only requires less effort; it is also more reliable. It is therefore wise to make use of text files as input to the network design tool, so that data can easily be converted to the appropriate format.

The network design tool may also include a network generator, which allows the user to parametrically generate random networks. This is useful for testing network algorithms, learning to use the tool, and in studying the relationship between network designs and network characteristics. The user should also be able to make hard copies of designs and be able to store a design for later retrieval.

5.2.3 Model formulation

The objective of a network design tool should be clearly formulated before the development of such a decision support system is employed. One has to ask what the tool will be used for, and what the objectives of the system are. Will it be to minimize the design cost, maximize the flow, and maximize the reliability or to do resilience testing? Under what types of constraints must these objectives be met?
Many other assumptions for the system could be made, like:

- The type of formulation that will be used in the model. In an exact flow-dimensioning problem, this can either be an edge-based or a path-based formulation.
- Will there be a constant demand for transmission of a commodity between a source and a destination?
- Must the model be able to support stochastic demands and queuing delays?
- May it be accepted that there will be an optimal distribution of messages over available network capacity? Such perfect algorithms rarely exist but often remains an ideal.
- In what mode must routing of messages be done? Is it possible to split messages or not?
- Will the network support directed or undirected edges?
- What level of sophistication must be supported in the hardware design, e.g. specific "slots" for different edge types for node equipment etc?
- Must multicommodity flow be accommodated?
- Must every node be considered to be both a source and destination for commodities?

In practice a design cost is associated with the development of a node or an edge in a network configuration. It is therefore desirable that we can assign a function for design cost to the potential nodes and edges included in the network configuration mechanism in the network design tool. We furthermore may want to be able to assign a flow cost for the commodities on every edge in our potential network.

Except for helping the network designer in obtaining an optimal solution from the many potential configurations, it is also desirable that this optimal configuration is tested to see how it would react to possible node or edge failures, and to consider alternative configurations that can be used as a

backup. To conduct such a test, the network design tool should provide the network designer with the ability to exclude (disable) certain nodes or edges from the potential configuration.

Other desirable alteration functions would include alteration of flow bounds on certain potential edges or to increase the flow requirements of commodities to study the effect on the optimal solution and the solution cost.

5.2.4 Algorithms

Network design tools make use of algorithms in obtaining the optimal solution. Such algorithms may make use of other existing optimization engines rather than personal code from the developer. Decisions on the approach to be used may include the following:

- **Exact algorithms**, where the solution found is optimal and where no better solution satisfying the constraints exists.

- **Heuristics**. This refers to techniques designed to solve a problem that searches for solutions that are probably correct. It does so by often solving a simpler problem or by relaxation of the problem that contains or intersects with the solution of the more complex problem.

  A heuristic often does not guarantee an exact solution to the problem, but often solves it well enough for most uses. It usually does so more quickly than an exact algorithm.

- **Multicriteria objectives**. In many situations the network designer may want to find a balance between many criteria, like cost, resilience, manageability, etc. and this may have to be considered when searching for acceptable solutions.

- **Survivability requirements** (connectivity). In the design of many communication networks, it is important to have built-in survivability. It provides for at least two independent paths between certain "special" offices (nodes) to provide protection against any single edge or single node failure between them.

5.3 Chapter summary

The concept of a network design tool and some of the desired properties of such a decision support system were outlined in this chapter. Furthermore the components, which would be preferable in such a tool, were discussed. In the design of a network design tool many questions regarding model formulation within the system arise. Some of these questions were discussed in this chapter.

At the outset we have to acknowledge that it is quite ambitious to design a large network where the design tool incorporates many of the desirable functions. It can be taken for granted that one often has to make compromises and simplifications in order to arrive at a workable system. The network design tool must of necessity find a reasonable balance between the complexities in real life situations and scientific soundness.

In the following chapter the design parameters for a decision support system developed for this dissertation will be discussed.

6

GENERAL SYSTEM DESCRIPTION

In this chapter we discuss the general system description in terms of the hardware and software platforms used in the development of the survivable network designer decision support system that was developed for this dissertation. The objective of this system is to aid the network designer in the planning of network mesh topologies and edge capacities in order to avoid costly designs and congestion (or to give advice on congestion relief in existing networks).

6.1 Software and hardware

The network design tool employs a Graphical User Interface (GUI), as displayed in Figure 6.1. It is given in the form of menu and submenu modules that allow the user to traverse the functions as required. Figure 6.1 is only given to indicate the typical interface for the user. The full description of the modules can be found in Appendix D.

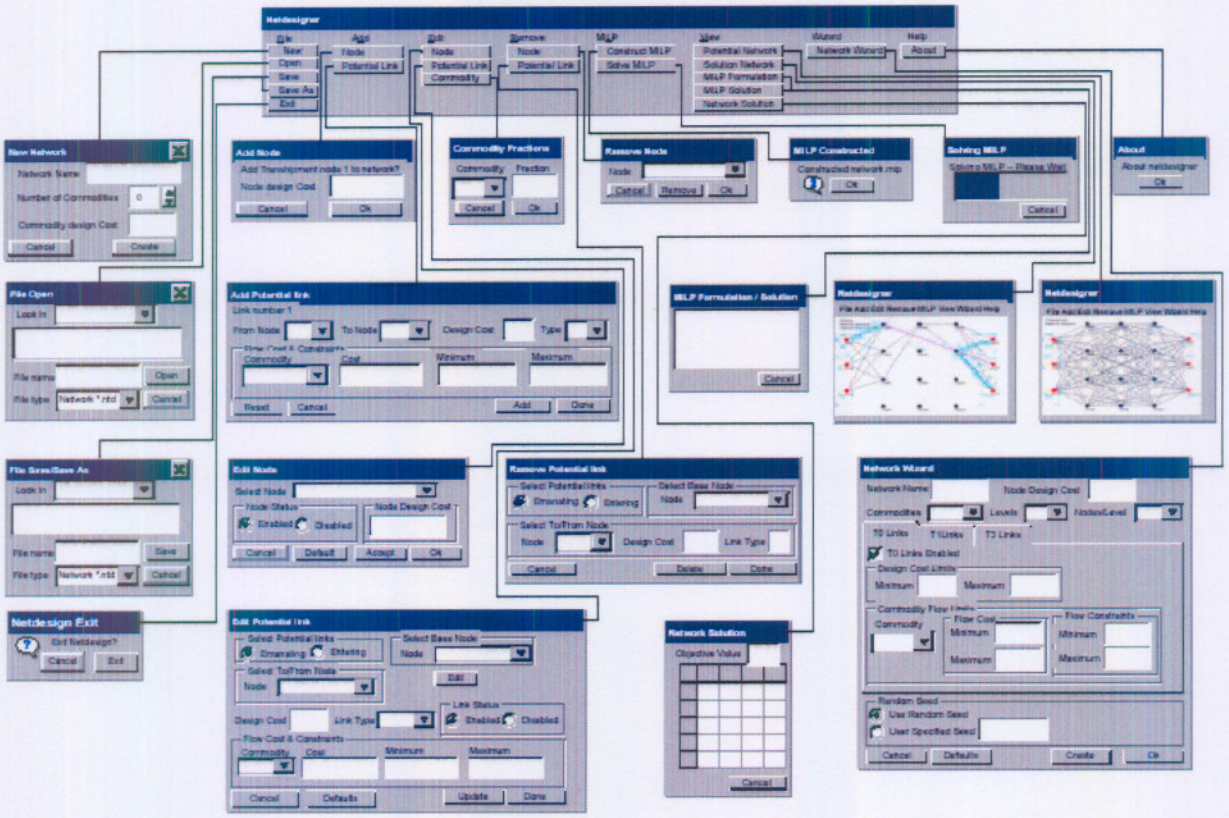


Figure 6.1 Screen layouts of survivable netdesigner

6.1.1 Software development aspects

The software was developed in Qt a multiplatform C++ application development framework. One source runs natively on four different platforms (Windows, Unix/ Linux, Mac OS X, Embedded Linux). The software can be ported to multiple platforms with a simple recompile. The system output is a plain text Mixed Integer Linear Programming (MILP) problem that can be transferred to a machine capable of solving mixed integer problems [3].

6.1.2 Optimization engine used

The mixed integer programming problems that are created are solved by CPLEX. The version of the software used in this dissertation was CPLEX 9.1. CPLEX has been developed by the ILOG Company. ILOG CPLEX delivers high-performance, robust, flexible optimizers for solving linear,

mixed integer and quadratic programming problems in mission-critical resource allocation applications. More information on the product can be obtained from [31].

Van der Merwe [50] states that there are certain benefits involved in using standardized software like CPLEX instead of custom-built software implementing heuristics and other approaches. These advantages can be summarized as follows:

- **Robustness**. For a software package to be useful in solving mathematical programming problems, it needs to be robust and thus able to recover from difficulties encountered during the solution process. This ensures that abnormal program termination is minimized and that the software is tested for stability, both in execution and also in numerical stability.

- **Support**. The vendors of commercial packages offer support to users of their software. This ensures that if problems arise that the users cannot solve on their own, help is available. It also ensures rapid development of software utilizing a solver developed by reputable solvers.

- **Ongoing improvements**. Vendors of commercial packages strive towards improving their software. This means that development of software may improve the stability or efficiency of the package offered by vendors. This is more desirable than code that may not be continually updated. It is also considered important for software vendors to keep up with current trends in the optimization field.

- **Tested**. Applications supplied by vendors have generally been thoroughly tested. This means that the results obtained from these applications can be trusted and compared to results obtained by other users. Custom developed software solutions may not always produce correct answers.

Apart from these advantages there is also another major advantage in using standard software with improved modelling instead of custom applications with "in-house" code development.
The main advantage is that modelling works with the standard software and aims to improve the efficiency of standard software. When standard software improves, the advantages gained by the

modelling may also be ported to the new software without any difficulty and this can enhance performance even more.

6.1.3 Hardware used

In this section the hardware platform that was used for the system development as well as the software installed on the platform is discussed.

The first phase of the development, which includes the layout of the GUI, file handling procedures and development of data structures was done on an Intel desktop PC on a Linux platform. The second phase, which includes the interface with the optimization software, was developed on an IBM Deskside Server pSeries 630 with AIX version 5.2. All the empirical work was also done on this server.
The hardware specifications will not be discussed in detail. Table 6.1 displays a summary of the hardware employed in the system.

Processor	2 x 2 – Way 1.0 GHz Power 4 PROC Cards
Hard drive	1 x 36.4 GB 10K RPM U3 SCSI DDA 1 x 73.4 GB 10K RPM U3 SCSI DDA
Ram	2 x 1024MB DIMMS, 208pin, 8ns DDR
CD-Rom	1 x 4.7GB SCSI DVD-RAM DRIVE
AIX	Version 5.2

Table 6.1 Summary of the hardware configuration in the IBM Pserver

6.2 System requirements

It is recommended that the system be used on an AIX 5.3 platform with a processor capable of at least 1GHz calculation speed. The platform must be able to support CPLEX version 9.1, which is used by the system during the optimization process. Furthermore the platform must be able to

provide high-resolution graphical data. The system files do not require more than 3MB of storage space.

The size of the problem to be solved by the system is linearly dependent on the processor speed and amount of memory available on the system. For the empirical work done in this dissertation the hardware as summarized in table 6.1 was sufficient to produce optimal results in reasonable times.

6.3 Properties of the survivable network design tool

The main objective of this decision support system is to minimize the total design cost of the network configuration under the constraint that the resulting design must allow the flow requirements of the network.

The basic point of departure that was used in the system design was to concentrate on rather small network synthesis problems and to try to solve these with exact methods instead of considering large complex networks that are computationally infeasible in real time. This small network can in practical situations consist of a part of a larger network where the traffic analysis indicates that congestion becomes problematic. Thus one may be able to zoom in on a part of a larger network and solve this sub-problem with exact methods. Although this gives rise to sub-optimization if the total network is considered, it is our view point here that this is a practical way to approach the problem.

The decision support system created for this dissertation was designed making use of the guidelines provided in the previous chapter. It therefore consists of a graphical user interface (GUI) and makes use of text for both data input and data storage.

For the model used in the survivable network design tool the following assumptions were made:

- The model makes use of arc-based as well as path-based mathematical models.
- There is a given demand matrix indicating the required transmissions (of a commodity) between demand pairs (source – destination).
- No queuing delays are allowed except in cases where breakdowns occur in edges or nodes.

- It was accepted that there is an optimal distribution of messages over available network capacity. This assumes an algorithm that controls packet switching optimally.
- The network supports undirected edges.
- The model can support three different potential edge types each with different capacities (T0, T1, and T2) but our experimentation consider only data instances with two types (T0 and T1).
- Only single commodity flow networks are considered.
- If a T0 or T1 or both edges/links exist between nodes *i* and *j*, we consider them connected when two independent edge-connected paths or two independent node-connected paths are required to provide a measure of survivability.

6.4 Exact algorithm for survivable network design tool

6.4.1 Capacitated model considered for empirical work and system (Survivable Netdesigner) development

For the model we considered for the survivable network design tool, we combined the model for survivable network given in section 3.6 with the capacitated model in section 3.4.3 to produce the model shown below. In this model the edges consist of certain facility "lines" (e.g. *T0*, *T1* and *T2*). These can be introduced at a cost (design cost) to enable the network to handle increased flow more economically.

In addition it may also be desirable to assign a design cost associated with the development of a node in the network. To accommodate the three different edge alternatives and node design costs the capacitated network model must be altered. The altered model is given below. It is a representation of the model incorporated in the network design tool developed for this dissertation.

Minimize

$$\sum_{1 \le k \le K} (c_0^k x_0^k + c_1^k x_1^k + c_2^k x_2^k) + f_0 y_0 + f_1 y_1 + f_2 y_2 + \sum_{i \in N} g_i z_i$$

subject to

$$\sum_{p=0}^{2} \sum_{\{j:(i,j) \in A\}} x_{pij}^k - \sum_{p=0}^{2} \sum_{\{j:(j,i) \in A\}} x_{pji}^k$$

$$= \begin{cases} 1 & \text{if } i = s^k \\ -1 & \text{if } i = d^k \\ 0 & \text{Otherwise} \end{cases} \qquad \forall i \in N, k = 1,2,.... K$$

$$\sum_{1 \le k \le K} \frac{1}{u_{pij}^k} x_{pij}^k \le y_{pij} \qquad \forall p = 0,1,2., \quad (i,j) \in A, k = 1,2,...., K;$$

$$l_{pij}^k y_{pij} \le x_{pij}^k \qquad \forall p = 0,1,2., \quad (i,j) \in A, k = 1,2,...., K;$$

$$\sum_{p=0}^{2} \sum_{\{j:(i,j) \in A\}} y_{pij} \le E.z_i \qquad \forall i \in N, [or \ y_{pij} \le z_i \ \forall p = 0,1,2 \ and \ \forall (i,j) \in A];$$

$$\sum_{p=0}^{2} y_{pij} \ge q_{ij} \qquad \forall (i,j) \in A;$$

$$Q(\delta(W)) \ge con(W) \qquad \forall W \subseteq V, \ \phi \ne W \ne V; \qquad (1a)$$

$$Q(\delta_{a-z}(W)) \ge 1 \qquad \forall z \in V, and \ \forall W \subseteq V \setminus z, \ \phi \ne W \ne V \setminus z \ with \ r(W) = 2 \ and \ r(V \setminus (W \cup z)) = 2; \qquad (1b)$$

$$x_{pij}^k \ge 0 \qquad \forall p = 0,1,2, \ (i,j) \in A, \ k = 1,2,... K;$$

$$y_{pij} \in \{0,1\} \qquad \forall p = 0,1,2., \ (i,j) \in A;$$

$$z_i \in \{0,1\} \qquad \forall i \in N;$$

$$q_{ij} \in \{0,1\} \qquad \forall (i,j) \in A.$$

Where

c_p^k = cost vector for all ***Tp*** links and for commodity ***k***

x_p^k = flow vector for all ***Tp*** links and for commodity ***k***

f_p = vector of design costs of ***Tp*** link for all links

y_p = vector that indicates the existence of a link ***Tp*** for all links

z_i = an indication of the existence of node ***i***

g_i = design cost of node ***i***

x_{pij}^k = flow for commodity ***k*** on a ***Tp*** link between ***i*** and ***j***

A = set of all potential links (edges / arcs)

s^k = index of the source node for commodity ***k***

d^k = index of the sink node for commodity ***k***

K = number of commodities

p = 0, 1, 2 denotes the discrete link types ***T0***, ***T1***, ***T2***

N = set of potential nodes

E = maximum outdegree of all nodes

u_{pij}^k = upper bound of flow along the edge (i, j) of type p for commodity k

l_{pij}^k = lower bound of flow along the edge (i, j) of type p for commodity k

Q = function defined in chapter 3 section 3.6

q_{ij} = variable indicating existence of an edge (of any type) between ***i*** and ***j***

6.4.2 Revised capacitated model considered for empirical work with demand matrices

In this model we extend the model to allow for demands between nodes i and j in the form of given demands between d_{ij} that must be catered for by the network design. The notation used in the model below thus makes use of variables (path-based) f_{ijkl} that designates the flow fraction of the demand d_{ij} that traverses the edges from k to l in that direction.

We also decided to adhere to the x_{pij} variables that represent the total flow along the link type p over the edge (i, j). This refined notation thus allows us to trace the way in which the demand d_{ij} is satisfied in the solution.

We also decided to formulate this model for a single commodity to keep the complexity of the model and the experiments within manageable bounds. This means that the index k is suppressed here.

Minimize

$$c_0 x_0 + c_1 x_1 + f_0 y_0 + f_1 y_1 + \sum_{i \in N} g_i z_i$$

subject to

$$\sum_{p=0}^{1} \sum_{\{j:(i,j)\in A\}} x_{pij} - \sum_{p=0}^{1} \sum_{\{j:(j,i)\in A\}} x_{pji}$$

$$= \begin{cases} 1 & \text{if } i = src \\ -1 & \text{if } i = dst \\ 0 & \text{Otherwise} \end{cases} \qquad \forall i \in N$$

$$\frac{1}{u_{pij}} x_{pij} \le y_{pij} \qquad \forall p = 0,1., (i,j) \in A;$$

$$l_{pij}\, y_{pij} \le x_{pij} \qquad \forall p = 0,1., (i,j) \in A;$$

$$\sum_{p=0}^{1} x_{plk} = \sum_{j} d_{ij} \qquad \forall (i,j) \in A, \forall (l,k) \in A, l = src,$$

$$k \ne dst, i \ne src, j \ne dst;$$

$$\sum_{p=0}^{1} x_{plk} = \sum_{i} d_{ij} \qquad \forall (i,j) \in A, \forall (l,k) \in A, l \ne src,$$

$$k = dst, i \ne src, j \ne dst;$$

$$f_{ijhk} = 0 \quad for \ i = j \ or \ h = k \quad \forall (i,j) \in A, i \ne src, j \ne dst;$$

$$\sum_{l} f_{ijil} = d_{ij} \qquad \forall (i,j) \in A, i \ne j, i \ne src, j \ne dst;$$

$$\sum_{k} f_{ijkj} = d_{ij} \qquad \forall (i,j) \in A, i \ne j, i \ne src, j \ne dst;$$

$$\sum_{k} f_{ijvk} - \sum_{k} f_{ijkv} = 0 \quad \forall V \in N \setminus \{i,j\}, \forall (i,j) \in A, i \ne j,$$

$$i \ne src, j \ne dst;$$

$$\sum_{p} x_{pij} = \sum_{k} \sum_{l} f_{klij} \qquad \forall (i,j) \in A, i \neq j, i \neq src, j \neq dst;$$

$$y_{pij} \leq z_i \qquad \forall p = 0,1, and \quad (i,j) \in A;$$

$$y_{pij} = y_{pji} \qquad \forall p = 0,1, and \quad (i,j) \in A, i \neq src, j \neq dst;$$

$$\sum_{p=0}^{1} y_{pij} \geq q_{ij} \qquad \forall (i,j) \in A, i \neq src, j \neq dst;$$

$$Q(\delta(W)) \geq con(W) \qquad \forall W \subseteq V, \quad \phi \neq W \neq V; \qquad (1a)$$

$$Q(\delta_{c-z}(W)) \geq 1 \qquad \forall z \in V, and \quad \forall W \subseteq V \setminus z, \quad \phi \neq W \neq V \setminus z \quad with \quad r(W) = 2 \quad and \quad r(V \setminus (W \cup z)) = 2; \qquad (1b)$$

$$q_{ij} = q_{ji} \qquad \forall (i,j) \in A, i \neq src, j \neq dst;$$

$$x_{pij} \geq 0 \qquad \forall p = 0,1,2, and \quad (i,j) \in A;$$

$$y_{pij} \in \{0,1\} \qquad \forall p = 0,1,2, and \quad (i,j) \in A;$$

$$z_i \in \{0,1\} \qquad \forall i \in N;$$

$$q_{ij} \in \{0,1\} \qquad \forall (i,j) \in A.$$

Where

d_{ij} = denotes the variable indicating the demand flow requirement over edge (i, j)

f_{ijkl} = denotes the variable of the flow for demand pair (i, j) routed over edge (k, l)

src = denotes the source node

dst = denotes the sink/destination node

This model was used in the empirical work with demand matrices. The results using the model are discussed in chapter 8.

6.5 Required properties of the system

It is desirable that the decision support system must have the functionality to open existing potential network configurations or to construct new potential network configurations. It would also be desirable to store these network configurations for later modifications and further future experimentation.

The decision support system should be able to model this potential network configuration as a Mixed Integer Linear Programming (MILP) by making use of the capacitated model discussed in the previous section. When this MILP problem is solved, the results should be interpreted and presented to the user in a user-friendly way.

Furthermore such a system should give the network designer the ability to add potential edges or nodes to an existing configuration to study the effect on the configuration. Once the potential network configuration has been made and solved, it is often desirable to be able to alter (edit) one or more nodes or the edge design costs, edge flow cost of one or more commodities or to change the flow requirements of one or more commodities.

6.6 Chapter summary

In this chapter the software and hardware platforms of the decision support system developed for this dissertation have been discussed. We also discussed the properties of the survivable network design tool. The capacitated model used in obtaining the optimal solution has been also discussed. In the next chapter we give the illustration of the system functionality of the survivable network designer.

7

DESCRIPTION OF SYSTEM FUNCTIONALITY

This chapter describes the system functionality of the survivable network designer decision support system that was developed for this dissertation.

7.1 Survivable netdesigner system specifications

Figure 7.1 gives a graphical representation of the flow chart of the properties and functions of the survivable network design system.

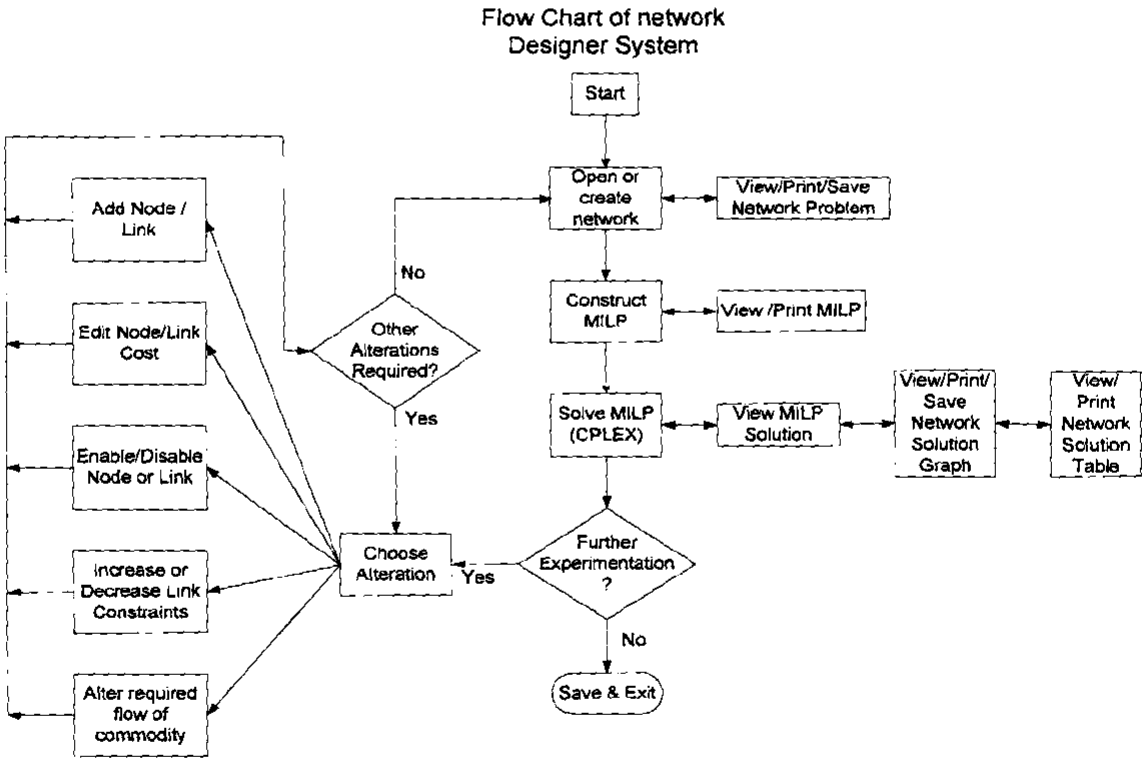


Figure 7.1 Flow chart of survivable netdesigner

In the exposition below we describe some of the alteration functions indicated in the flow chart on the left-hand side as well as some of the other activities shown.

7.1.1 Open or create a network

An existing network can be opened and altered or a new network can be constructed. In the construction phase the number of commodities that will be present in the network as well as the construction (design) cost of the commodities must be specified. You have the option of choosing to include a 2ECON or 2NCON survivability property to the network. In this version the system can only accommodate 2 nodes with higher connectivity than 1.

7.1.2 Editing nodes

Any node may be edited or disabled (excluded from the potential configuration if one wishes to study the effect of an edge failure).

7.1.3 Edit capacities

The flow requirement of a specific commodity may be altered. It may either be increased or decreased to study the effect on the resulting network and associated (minimum) cost.

7.1.4 Construct MILP

The current network design problem is modelled with mathematical modelling techniques as an optimization problem (MILP), and the MILP may be viewed, printed and solved as indicated below.

7.1.5 Solve MILP

The MILP is solved by means of a software product of ILOG viz CPLEX which is reputed to be a state of the art optimization software. The solution to the problem may be viewed in text or as a graphical representation. In the graphical representation it is possible to view only a certain link

type or all link types included in the solution. It is also possible to save and print graphical representations. A table showing the links to be included in the design as well as the link utilization is also available.

7.1.6 Save

The network layout can be saved to a file for later use.

7.2 Chapter summary

In this chapter the features, properties and functionalities of the system have also been described. In the following chapter the experimental results of the survivable network design system will be illustrated by means of examples.

8

EXPERIMENTAL RESULTS

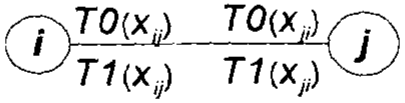
In this chapter we give some examples to illustrate the functionality of the survivable network design tool. In paragraphs 8.1 to 8.2 we consider an example where the system has to determine topologies and flow in a given network design situation. We also indicate the solutions when some measure of survivability is considered (2ECON and 2NCON instances).

In paragraph 8.3 we show the design when a certain node is disabled. In paragraph 8.4 we consider the increase in design cost with commodity volume increases.

8.1 Examples

In the following illustration we show an undirected network in which we want to ship a single commodity using only two edge types namely *T0* and *T1*. Each *T0* line can carry 4 units of messages and each *T1* line can carry 16 units of messages. As indicated in chapter 3, we again assume that one unit of flow has to be accommodated with the necessary adjustments to the data.

The following legend describes the notation in the graphical network representation:



Where: $T0(X_{ij})$ = denote the flow units X_{ij} on a *T0* link from i to j

$T1(X_{ij})$ = denote the flow units X_{ij} on a *T1* link from i to j

$T0(X_{ji})$ = denote the flow units X_{ji} on a *T0* link from j to i

$T1(X_{ji})$ = denote the flow units X_{ji} on a *T1* link from j to i

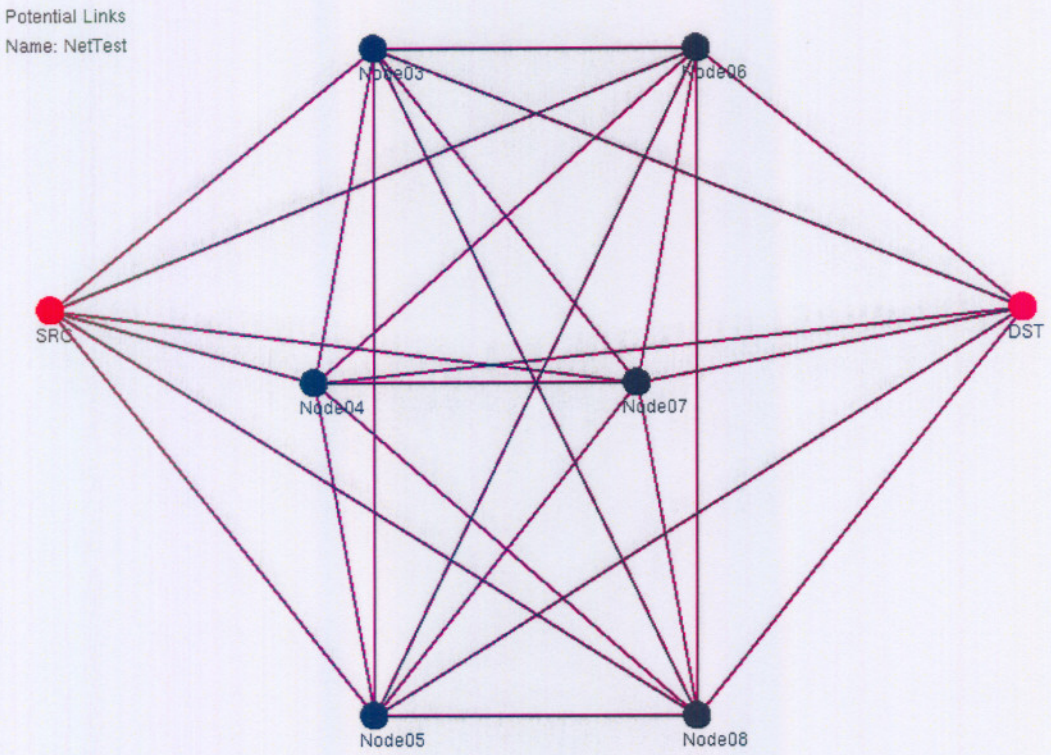


Figure 8.1 Graphical representation of potential network configuration

Figure 8.1 above gives a graphical representation of this potential network layout with a single commodity. The symbol *SRC* denotes the source node for the commodity and *DST* denotes the sink node for the commodity.

We apply a demand matrix to the potential network as shown below.

Nodes (*i*, *j*)	3	4	5	6	7	8		Totals
3	0	4	0	0	0	5	=	9
4	2	0	0	1	0	0	=	3
5	0	2	0	2	1	1	=	8
6	1	4	3	0	0	3	=	11
7	0	0	0	0	0	0	=	0
8	1	2	0	2	3	0	=	9

Totals	4	15	3	5	4	9		40

Table 8.1 The demand matrix of the potential network configuration

8.2 Optimal solution

In the network design system the potential network configuration in Figure 8.1 is modelled as a MILP as indicated in chapter 4 and solved by CPLEX. The optimal configuration that will satisfy the flow requirements of the network is displayed in Figure 8.2. The optimal cost is 1017.5735. The optimal solution and edge utilization is summarized in Table 8.2. The detailed results for these non-survivable model as well as those with the 2ECON and 2NCON requirements with flow fractions traversing the edges are given in Appendix C.

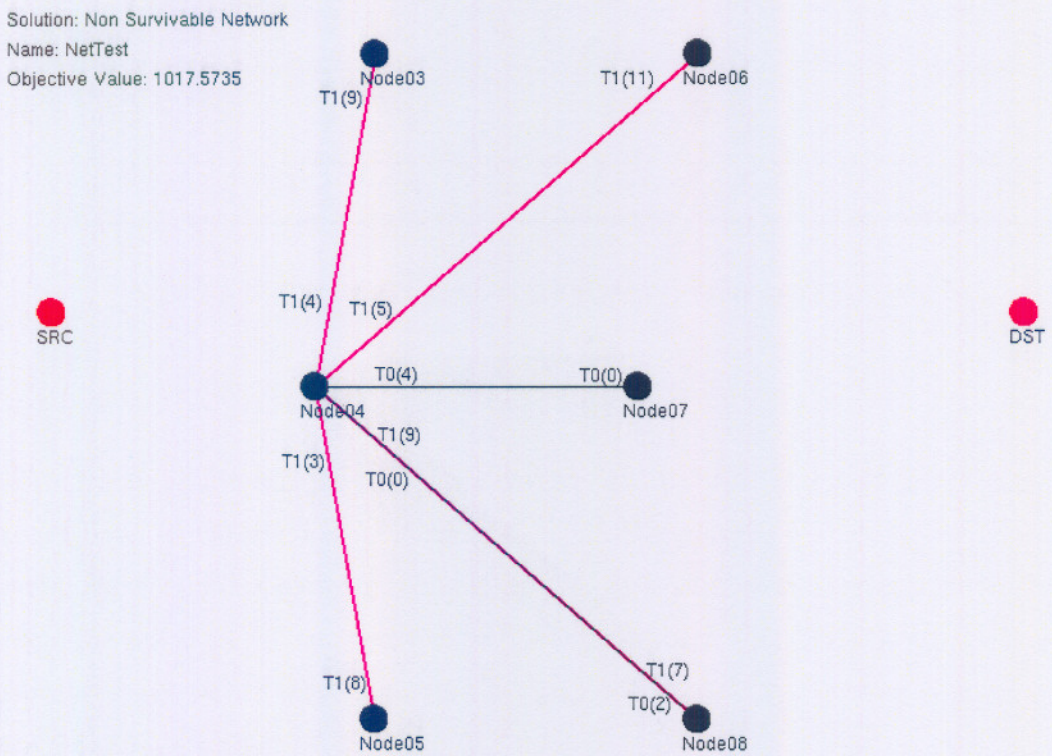


Figure 8.2 Optimal network solution for a non-survivable network

Link	From	To	Flow	Max Flow	Utilization	Flow Cost	Design Cost	Link Type
001	SRC	Node03	9	16	56.25%	0.69	23.69	T1

002	SRC	Node04	3	16	18.75%	1.47	21.93	T1
003	SRC	Node05	8	16	50.00%	0.61	20.57	T1
004	SRC	Node06	11	16	68.75%	3.14	15.17	T1
005	SRC	Node08	9	16	56.25%	3.83	20.32	T1
006	Node03	DST	4	16	25.00%	3.03	22.97	T1
007	Node03	Node04	9	16	56.25%	3.66	11.05	T1
008	Node04	Node03	4	16	25.00%	0.48	14.31	T1
009	Node04	DST	15	16	93.75%	1.12	18.97	T1
010	Node04	Node05	3	16	18.75%	0.88	20.07	T1
011	Node05	Node04	8	16	50.00%	2.86	14.08	T1
012	Node04	Node06	5	16	31.25%	3.44	18.88	T1
013	Node06	Node04	11	16	68.75%	3.91	22.37	T1
014	Node04	Node07	4	4	100.00%	7.46	10.93	T0
015	Node08	Node04	2	4	50.00%	4.63	11.85	T0
016	Node04	Node08	9	16	56.25%	0.61	11.91	T1
017	Node08	Node04	7	16	43.75%	3.78	21.14	T1
018	Node05	DST	3	16	18.75%	1.52	18.60	T1
019	Node06	DST	5	16	31.25%	1.02	18.26	T1
020	Node07	DST	4	16	25.00%	2.76	16.37	T1
021	Node08	DST	9	16	56.25%	2.10	23.80	T1

Table 8.2 Optimal solution results for Figure 8.2

Note that the model only allows *T1*lines for flow that involves the *SRC* and *DST* nodes.

A constructed data-generating tool called "the network designer's wizard" randomly generated the data used in these examples. This tool is briefly described in Appendix D.

➢ 2ECON survivability designs

We now consider designs with measures of survivability. We start with the 2ECON problem and assuming that nodes 3 and 8 have 2 connectivity requirements, we find that the optimal solution of the 2ECON network giving an objective value of 1118.5095 is represented in the following figure:

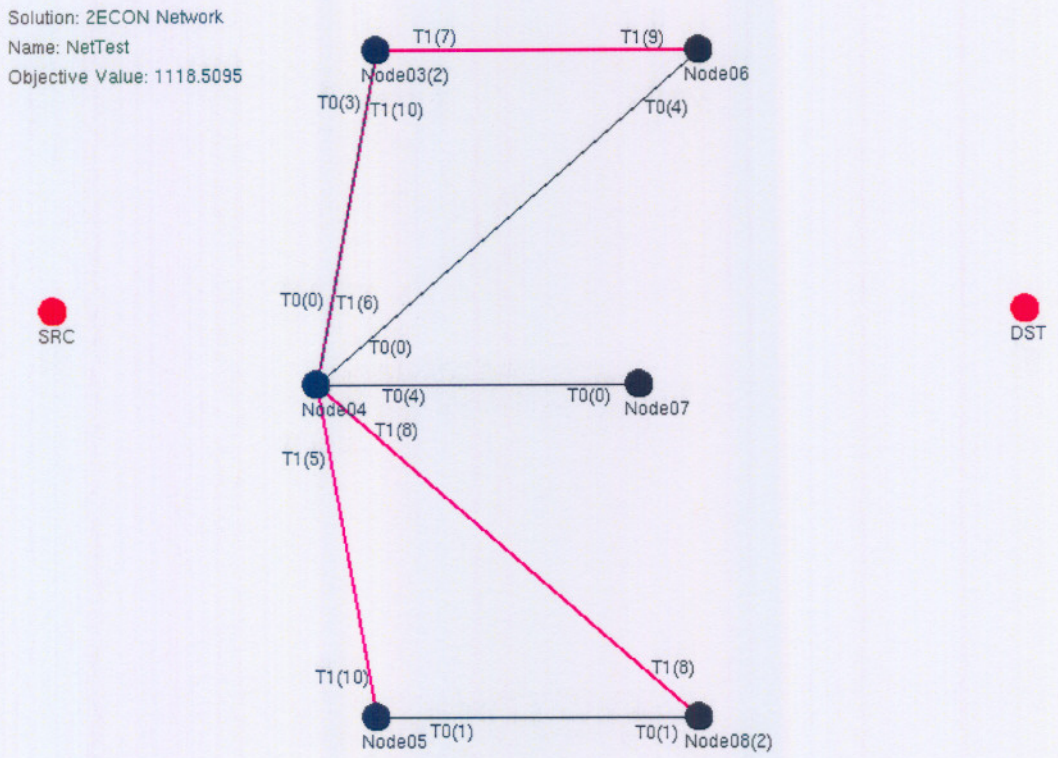


Figure 8.3 Optimal network solution for a 2ECON

Note that two independent paths exist between 3 and 8 as required. The optimal cost has an increased value of 1118.5095 (from 1017.5735 for the non-survivable situation).

Link	From	To	Flow	Max Flow	Utilization	Flow Cost	Design Cost	Link Type
001	SRC	Node03	9	16	56.25%	0.69	23.69	T1
002	SRC	Node04	3	16	18.75%	1.47	21.93	T1
003	SRC	Node05	8	16	50.00%	0.61	20.57	T1
004	SRC	Node06	11	16	68.75%	3.14	15.17	T1
005	SRC	Node08	9	16	56.25%	3.83	20.32	T1
006	Node03	DST	4	16	25.00%	3.03	22.97	T1
007	Node03	Node04	3	4	75.00%	7.65	5.65	T0
008	Node03	Node04	10	16	62.50%	3.66	11.05	T1
009	Node04	Node03	6	16	37.50%	0.48	14.31	T1
010	Node03	Node06	7	16	43.75%	2.79	13.06	T1
011	Node06	Node03	9	16	56.25%	1.60	17.43	T1
012	Node04	DST	15	16	93.75%	1.12	18.97	T1
013	Node04	Node05	5	16	31.25%	0.88	20.07	T1

014	Node05	Node04	10	16	62.50%	2.86	14.08	T1
015	Node06	Node04	4	4	100.00%	7.45	8.75	T0
016	Node04	Node07	4	4	100.00%	7.46	10.93	T0
017	Node04	Node08	8	16	50.00%	0.61	11.91	T1
018	Node08	Node04	8	16	50.00%	3.78	21.14	T1
019	Node05	DST	3	16	18.75%	1.52	18.60	T1
020	Node05	Node08	1	4	25.00%	6.26	9.53	T0
021	Node08	Node05	1	4	25.00%	6.34	10.26	T0
022	Node06	DST	5	16	31.25%	1.02	18.26	T1
023	Node07	DST	4	16	25.00%	2.76	16.37	T1
024	Node08	DST	9	16	56.25%	2.10	23.80	T1

Table 8.3 Optimal solution results for Figure 8.3

➢ 2NCON survivability designs

Considering the same problem with 2NCON connectivity requirement, we find the optimal objective value as 1245.54825 and the resulting network in the figure 8.4 below:

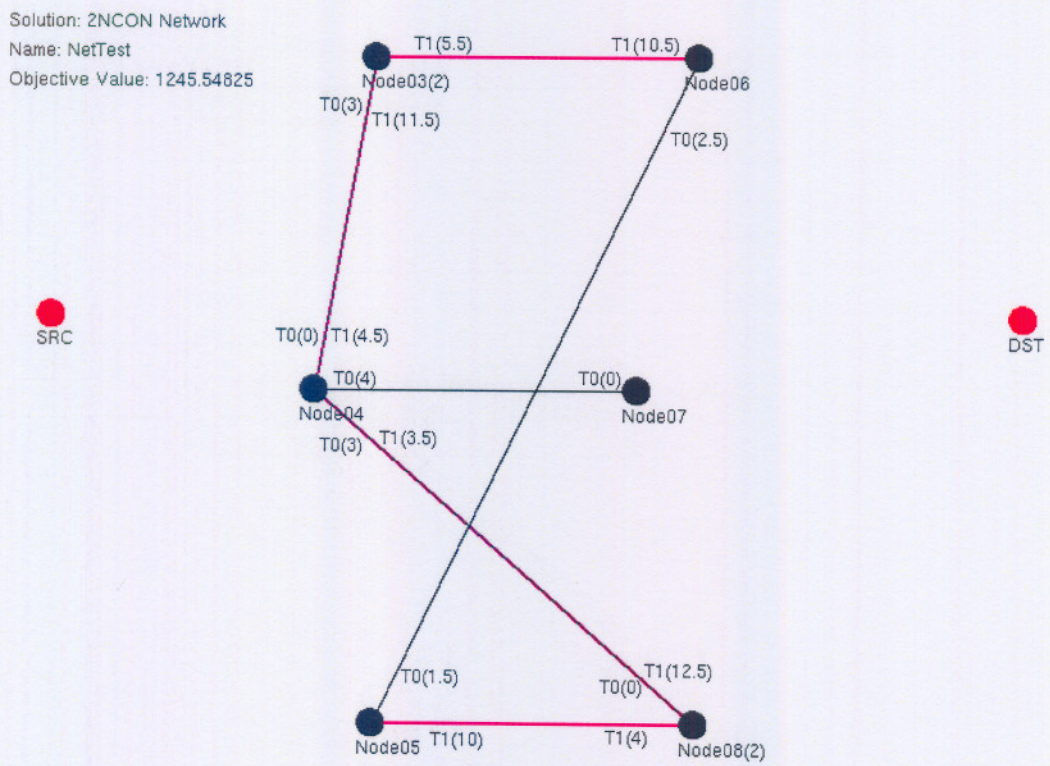


Figure 8.4 Optimal network solution for a 2NCON

Note that at a higher cost of 1245.54825, this topology now gives node-independent paths between 3 and 8.

Link	From	To	Flow	Max Flow	Utilization	Flow Cost	Design Cost	Link Type
001	SRC	Node03	9	16	56.25%	0.69	23.69	T1
002	SRC	Node04	3	16	18.75%	1.47	21.93	T1
003	SRC	Node05	8	16	50.00%	0.61	20.57	T1
004	SRC	Node06	11	16	68.75%	3.14	15.17	T1
005	SRC	Node08	9	16	56.25%	3.83	20.32	T1
006	Node03	DST	4	16	25.00%	3.03	22.97	T1
007	Node03	Node04	3	4	75.00%	7.65	5.65	T0
008	Node03	Node04	11.5	16	71.88%	3.66	11.05	T1
009	Node04	Node03	4.5	16	28.12%	0.48	14.31	T1
010	Node04	DST	15	16	93.75%	1.12	18.97	T1
011	Node05	Node06	1.5	4	37.50%	4.75	6.47	T0
012	Node06	Node05	2.5	4	62.50%	5.15	10.21	T0
013	Node03	Node06	5.5	16	34.38%	2.79	13.06	T1
014	Node06	Node03	10.5	16	65.62%	1.60	17.43	T1
015	Node04	Node07	4	4	100.00%	7.46	10.93	T0
016	Node04	Node08	3	4	75.00%	7.64	8.63	T0
017	Node04	Node08	3.5	16	21.88%	0.61	11.91	T1
018	Node08	Node04	12.5	16	78.12%	3.78	21.14	T1
019	Node05	Node08	10	16	62.50%	1.67	14.13	T1
020	Node08	Node05	4	16	25.00%	1.78	22.66	T1
021	Node05	DST	3	16	18.75%	1.52	18.60	T1
022	Node06	DST	5	16	31.25%	1.02	18.26	T1
023	Node07	DST	4	16	25.00%	2.76	16.37	T1
024	Node08	DST	9	16	56.25%	2.10	23.80	T1

Table 8.4 Optimal solution results for Figure 8.4

8.3 Disabling a node

It is also possible to disable nodes from the list of potential nodes. This enables the network designer to simulate possible node failures to study the effect on the optimal solution configuration and design cost. This also helps in the planning of possible backup solutions. Figure 8.5 shows the effect of disabling node 5 in Table 8.5 and the resulting solution in Figure 8.6 and Table 8.6.

The output of the demand matrix is given below when node 5 is disabled.

Nodes (*i*, *j*)	3	4	5	6	7	8		Totals
3	0	4	0	0	0	5	=	9
4	2	0	0	1	0	0	=	3
5	0	0	0	0	0	0	=	0
6	1	4	0	0	0	3	=	8
7	0	0	0	0	0	0	=	0
8	1	2	0	2	3	0	=	9
Totals	4	11	0	3	3	8		**29**

Table 8.5 The demand matrix of the potential network configuration with node 5 disabled

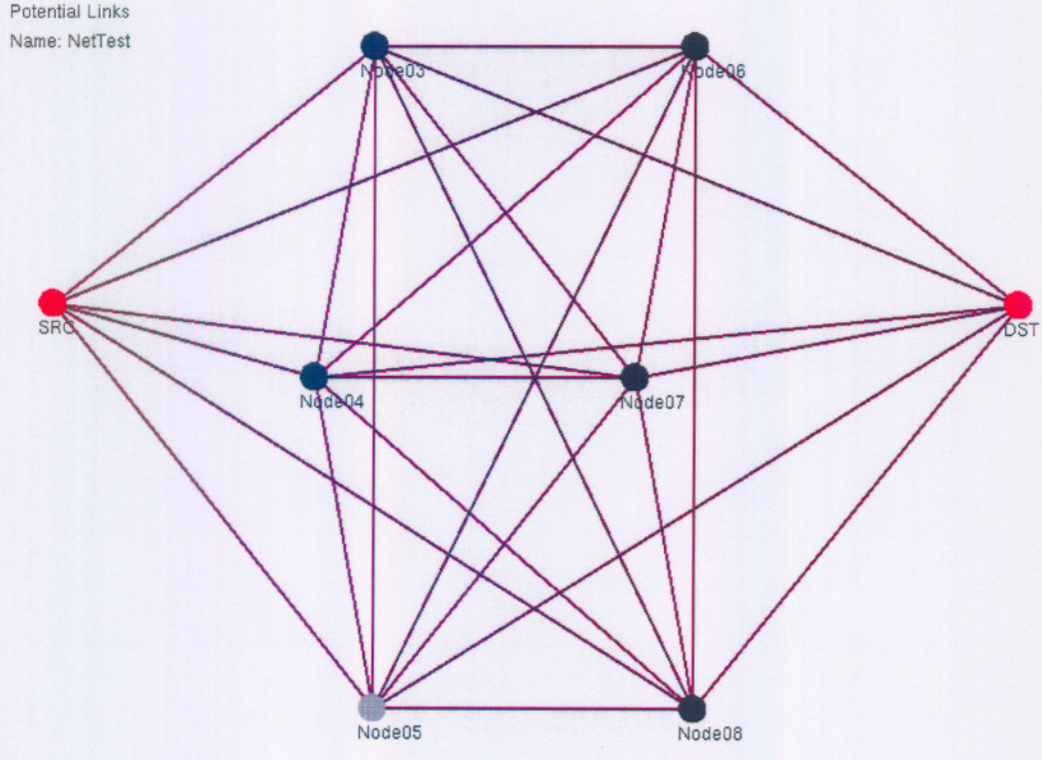


Figure 8.5 Graphical representation of potential network configuration with node 5 disabled.

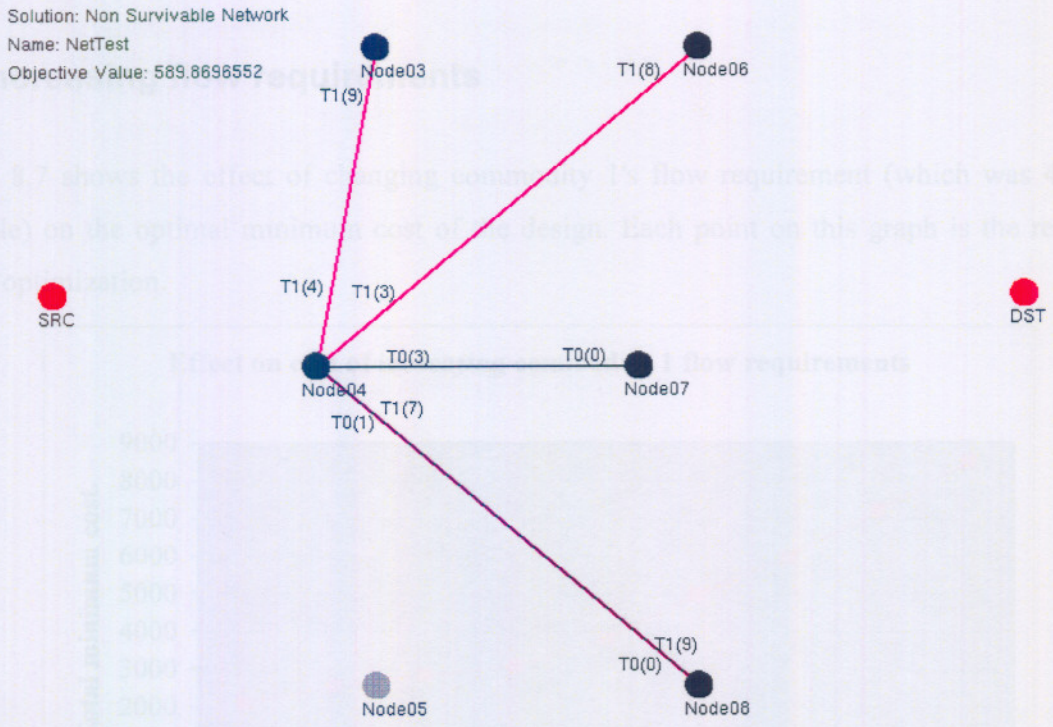


Figure 8.6 Optimal solution with transition node 5 disabled

Link	From	To	Flow	Max Flow	Utilization	Flow Cost	Design Cost	Link Type
001	SRC	Node03	9	16	56.25%	0.69	23.69	T1
002	SRC	Node04	3	16	18.75%	1.47	21.93	T1
003	SRC	Node06	8	16	50.00%	3.14	15.17	T1
004	SRC	Node08	9	16	56.25%	3.83	20.32	T1
005	Node03	DST	4	16	25.00%	3.03	22.97	T1
006	Node03	Node04	9	16	56.25%	3.66	11.05	T1
007	Node04	Node03	4	16	25.00%	0.48	14.31	T1
008	Node04	DST	11	16	68.75%	1.12	18.97	T1
009	Node04	Node06	3	16	18.75%	3.44	18.88	T1
010	Node06	Node04	8	16	50.00%	3.91	22.37	T1
011	Node04	Node07	3	4	75.00%	7.46	10.93	T0
012	Node04	Node08	1	4	25.00%	7.64	8.63	T0
013	Node04	Node08	7	16	43.75%	0.61	11.91	T1
014	Node08	Node04	9	16	56.25%	3.78	21.14	T1
015	Node06	DST	3	16	18.75%	1.02	18.26	T1
016	Node07	DST	3	16	18.75%	2.76	16.37	T1
017	Node08	DST	8	16	50.00%	2.10	23.80	T1

Table 8.6 Optimal solution results with transition node 5 disabled

8.4 Increasing flow requirements

Figure 8.7 shows the effect of changing commodity 1's flow requirement (which was 40 in the example) on the optimal minimum cost of the design. Each point on this graph is the result of a model optimization.

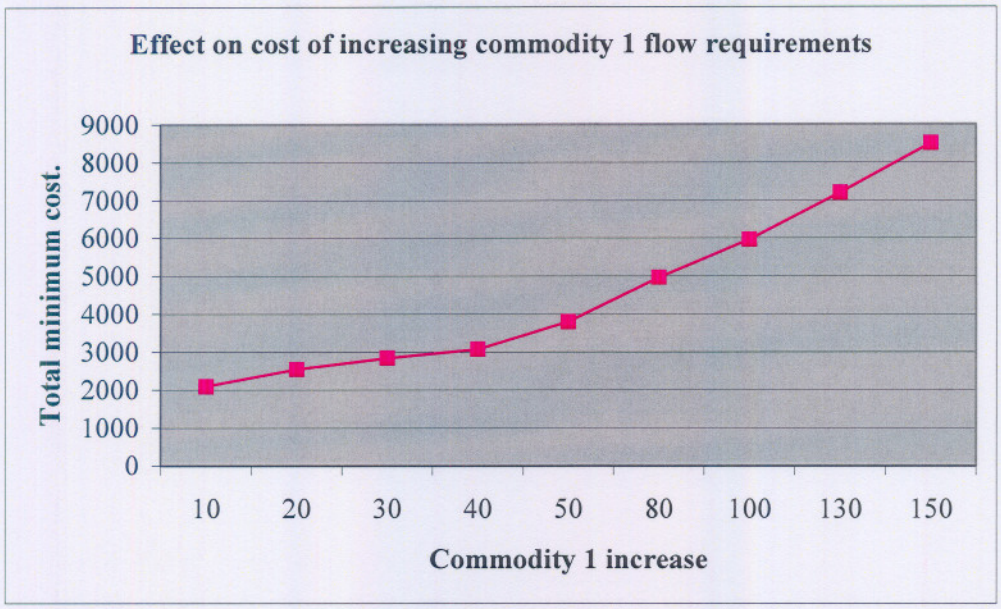


Figure 8.7 Effect on cost of increasing commodity 1

The slope of the graph may present the decision maker with topologies to get more capacity without large increases in cost. As an example, it should be noted that the cost increase from 30 units to 40 units for commodity 1 has a relatively low slope and may represent an opportunity for a decision maker.

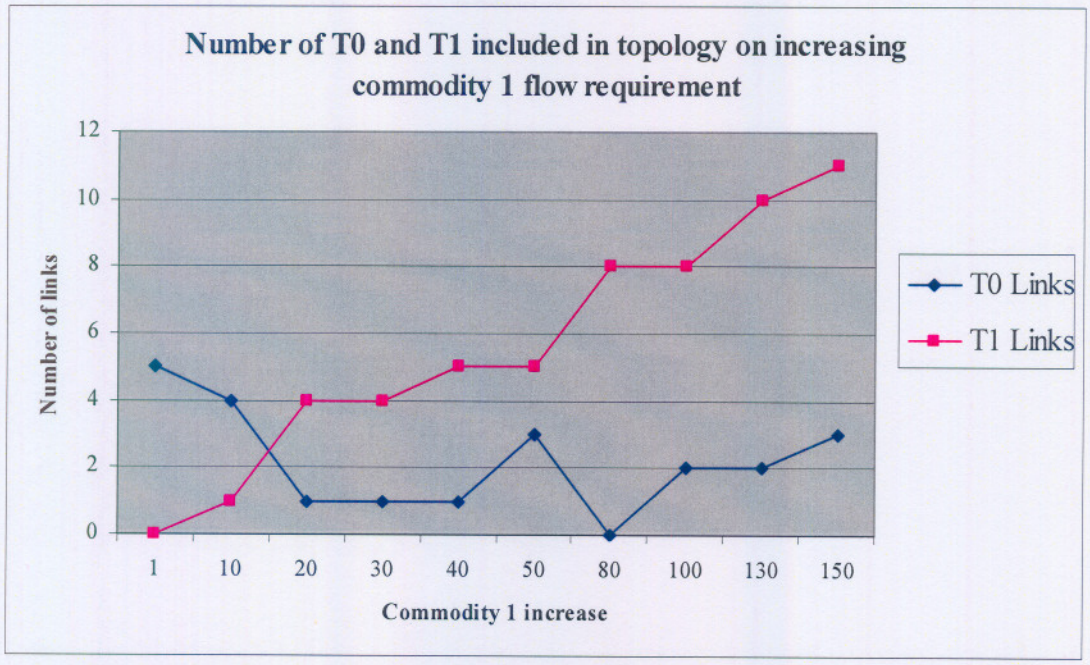


Figure 8.8 Number of T0 and T1 edges included in topology on increasing commodity1

Note that the more units for commodity 1 we send through the network, the less we require the use of T0 links.

8.5 System capacity

We obtained results for design instances of 8 nodes and 11 nodes. We failed to obtain results for 14 nodes, 17 nodes and 20 nodes with the time limit of 3hours. Instances with more than 11 nodes could not be solved on the available hardware and software platforms due to the complexity of the MILP generated.

The user has the option of selecting which two nodes he wants to assign the connectivity of 2 in the proposed design. We give graphical network configurations in Figures 8.9 to 8.13 and demand matrices in Tables 8.7 to 8.13.

The table below gives bounds on time (in seconds) for different instances.

Models	Non Survivable	2ECON	2NCON
8 nodes	< 10	< 15	< 20
11 nodes	< 26	< 40	< 60
14 nodes	> 180	....	....

The times reported were obtained on an IBM Deskside Server pSeries 630 with AIX version 5.2.

Demand matrix of **NetTest1** network with 11 nodes and 216 potential links is given below.

Nodes (*i*, *j*)	3	4	5	6	7	8	9	10	11		Totals
3	0	4	0	0	0	5	2	0	1	=	12
4	0	0	0	0	4	2	1	1	1	=	9
5	4	3	0	0	4	0	0	0	0	=	10
6	0	1	3	0	0	2	3	4	0	=	13
7	0	2	5	5	0	1	1	3	2	=	19
8	0	4	2	3	5	0	0	0	4	=	18
9	1	3	3	2	0	0	0	4	5	=	18
10	5	0	0	2	5	5	2	0	0	=	19
11	0	0	0	4	0	0	4	0	0	=	8
Totals	10	17	13	16	17	15	13	12	13	=	126

Table 8.7 The demand matrix of the potential network configuration of NetTest1

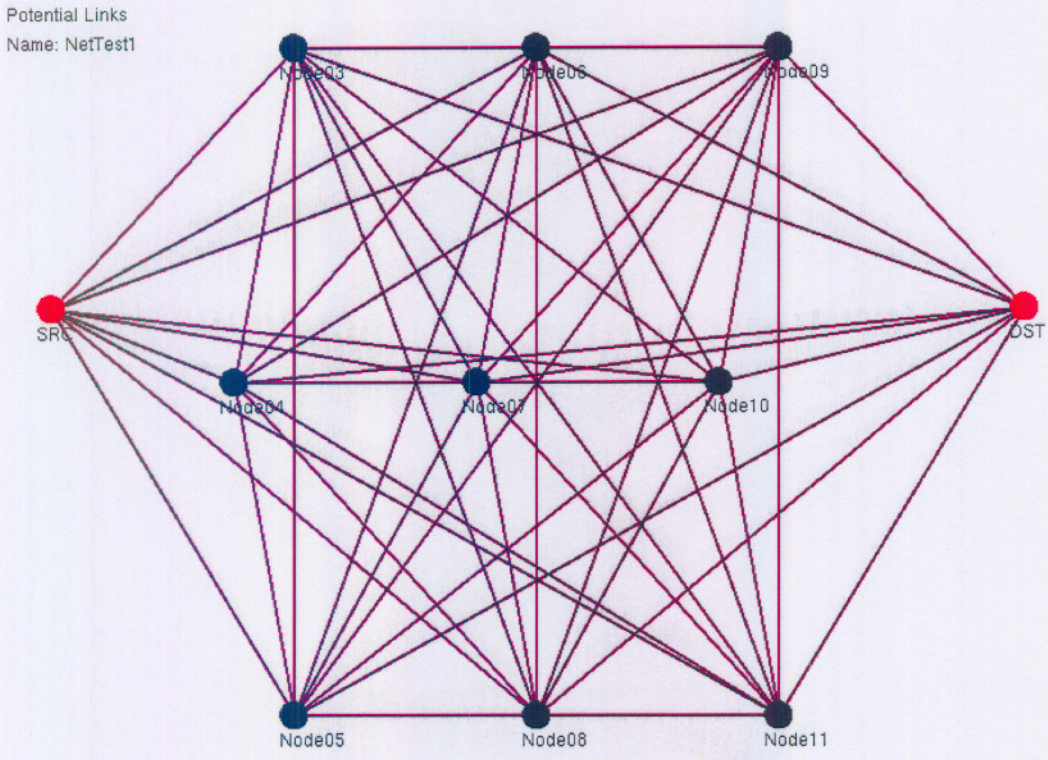


Figure 8.9 Graphical representation of potential network configuration of NetTest1

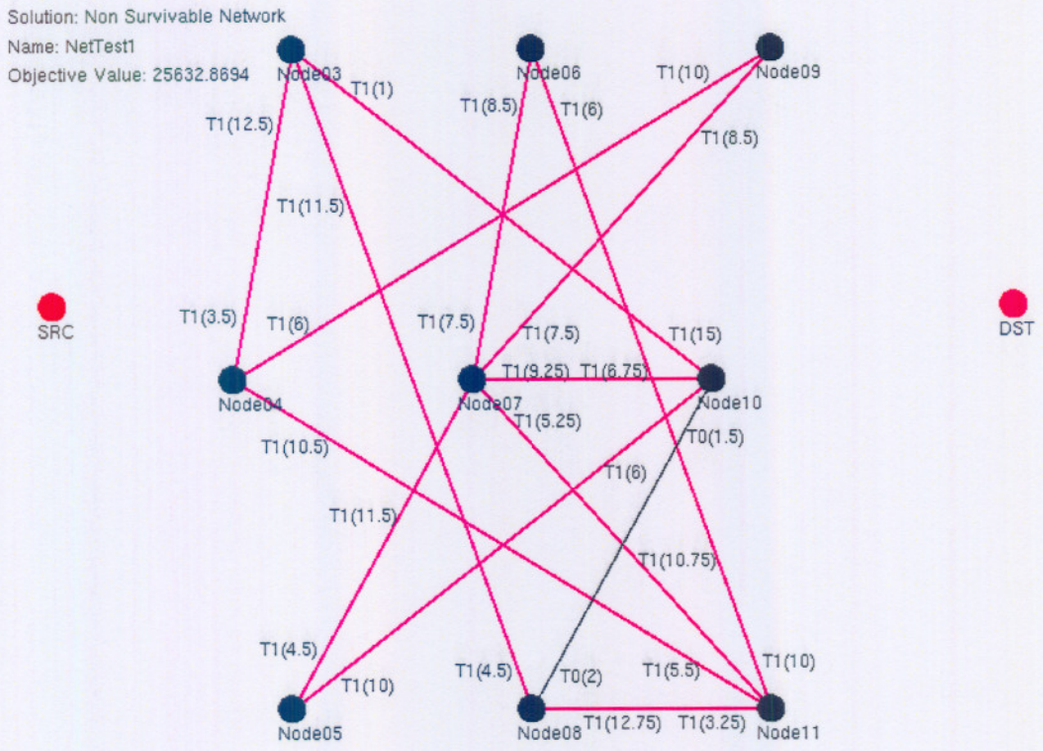


Figure 8.10 Optimal network solution for NetTest1

The solutions for the 2ECON and 2NCON problems gave identical topologies as Figure 8.10 but obviously took more time to optimize.

8.6 Evaluation of the system

All the solutions to the problems presented in this chapter were obtained in times ranging from 5 seconds to 60 seconds. It seems feasible to zoom into parts of a network that experiences congestion and solve the problems using survivable netdesigner system if the numbers of nodes considered do not exceed 11.

8.7 Chapter summary

In this chapter some examples of the network designer system were illustrated. A typical network problem has been solved and the solution was graphically displayed. The additional functions of the system have also been illustrated by examples where 2ECON and 2NCON designs have been shown. In the next chapter some future work and possible further system development will be discussed.

9

CONCLUSIONS AND FUTURE RESEARCH OPPORTUNITIES

In this chapter we give some concluding remarks and also give suggestions for future research opportunities. Although much work is done in this field of research, much work is still yet to be done since problems discussed in this dissertation are very difficult to solve.

9.1 Conclusion

We have found it feasible to solve network topologies for small problems of the types 2ECON and 2NCON using CPLEX, and have managed to integrate this into edge-capacitated models.

The results we obtained with the experimental system indicate that such a decision support system can be useful in mesh design networks and mostly to fiber optic communication networks with large amounts of traffic carried on each edge compared to traditional band-width-limited technologies in which network survivability may be of less importance. Some flow congestion and relief measures can also be investigated by using the system, which also has flow dimensioning properties.

9.2 Future work

The system developed has limitations and relaxing some of the constraints in the current version may contribute to making the system more applicable to large real life problems.

One such constraint is that large network problems cannot be solved with exact methods due to the limitation of the current software and computer technology. This aspect certainly would merit further investigation as more advanced computers and software technologies become available.

The system developed in this study has been developed with some simplifying assumptions that can be viewed as constraints imposed in order to get manageable complexity. Relaxing some of the constraints in the current version may contribute to make the system more applicable to real life problems. This, however, always leads to a trade off between realism and the feasibility of finding solutions.

A

DATA STRUCTURES

In this section we discuss some of the most popular ways of representing lists (i.e., ordered sets).

A.1 Arrays

An array is one of the simplest data structures used to store an ordered set. This representation uses an array of size n, called list. The i^{th} position (or index) of the array contains the i^{th} element of the set, which we denote by list[i]. The following figure gives a representation of an array of the ordered set {10, 5, 3, 6, 7}.

To keep track of the number of items in the array we define an integer variable *number* and set it equal to the number of items in the list. For the example shown in the Figure A.1 it is clear that number is equal to 5 and that the list does not contain data elements in positions larger than *number*.

To determine the k^{th} element of the list we simply access the data element list(k). To establish whether the list contains a given element α, we vary the index i from 1 to the last item in the list (*number*) and check whether list(i)= α. This operation requires time proportional to the number of elements in the set.

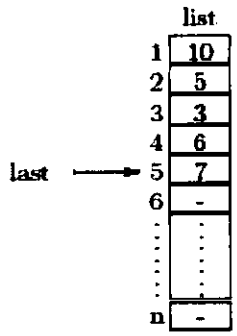


Figure A.1 Storing a set as an array

If we wish to insert an element at the end of the list, we increment *number* by one and store the new element as list(*last*). An array is not very well suited for performing some other set operations. For example suppose that we wish to delete the k^{th} element of the set and $k < number$. If we delete the k^{th} element from the list, we have to move each data item from position $k + 1$, $k + 2$, ..., *number* one backwards. In the worst case the number of moves can be equal to the *number* of elements in the list. A similar situation arises if we want to insert an element in the middle of the list.

A.2 Singly linked lists

Rather than store the elements of an ordered set sequentially as an array, a singly linked list may store elements in an arbitrary order. This, however, requires additional information that permits us to access the data in the order specified in the ordered set. A *cell* is the basic building block of linked lists. We can picture a cell as a box that is capable of holding several values, called *fields*. A singly linked list is a collection of cells that are linked together. Each cell in the singly linked list has two fields: a *data* field and an edge field. The *data* field holds the element of the list and the *edge* field stores a pointer to the location of the next element in the list. There are two popular methods for implementing linked lists: a pointer-based implementation and an array-based implementation.

In the array-based implementation we can store a singly linked list by defining two arrays of size n, data and edge. These two arrays define n cells, i indexed from 1 through n; the k^{th} cell consists of

the fields *data*(*k*) and *edge*(*k*). The data field of a cell contains an element of the set and the edge field contains the index of the cell containing the next element of the set. We also maintain a scalar called first that stores the index of the cell containing the first element of the set. If the set is empty, we set first equal to zero.

The following figure shows a geometric representation of an array form of a linked list for the set {3, 6, 8, 10}.

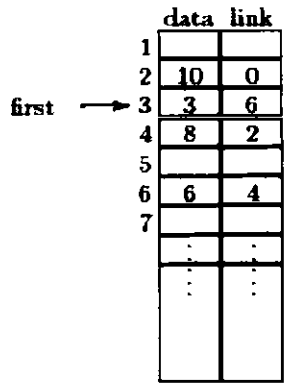


Figure A.2 Graphical representation of linked list stored in array form

In the pointer-based implementation the first item in the set is stored in a cell at a position in memory. We can also reference this position by a variable *first*. Thus *first* is a *pointer* to the first cell of the linked list. The edge field cell either contains the value *nil* or the *position* in memory (pointer to memory address) where the next item is stored. Thus if we start at position *first* we can pass through all cells exactly once.

Figure A.3 Graphical representation of a pointer-based linked list

It is fairly easy to manipulate singly linked lists. Suppose that we wish to scan all the elements of a set to determine membership of an element. We set a variable next = *first*. If next is either = 0 or equal to *nil* for the two representations respectively we know the list is empty and we can stop. If

not, we can check for the required item by checking data (next). If it is not equal to the required item we go to the next item by setting next = edge (next) and check again. This process is repeated until we find the required item or next equals either 0 or *nil*.

Insertion of new items in the beginning or middle of an existing singly linked list can also be done very easily without having to move items already in the list. The edge values are just altered to point to the new position.

Figure A.4 displays the insertion of a new item at the beginning of the list. The pointer of the new item is set to point to the position *first* is pointing to and *first* is then changed to point to the newly added item.

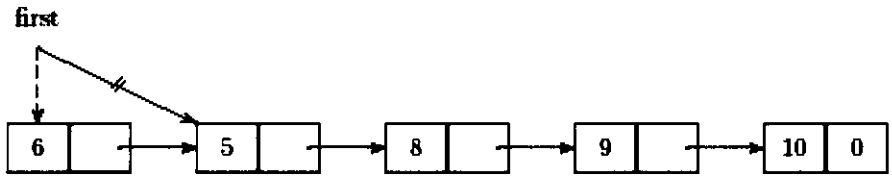


Figure A.4 Inserting an element at the beginning of a linked list

Figure A.5 is a graphical representation of the insertion of a new item in the middle of the singly linked list. The new item's pointer is set to point to the position of the item after the newly inserted item and the predecessor of the new item is set to point to the new item.

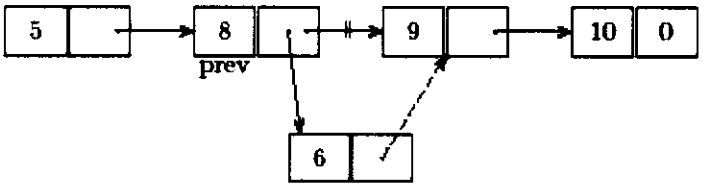


Figure A.5 Inserting an element in the middle of a linked list

Figure A.6 displays the situation where an item is deleted from the beginning of the list. *First* is set to point to the successor of the deleted item.

Figure A.7 displays the situation where an item is deleted from the middle of the linked list. The predecessor of the node to be deleted is set to point to the successor of the item to be deleted.

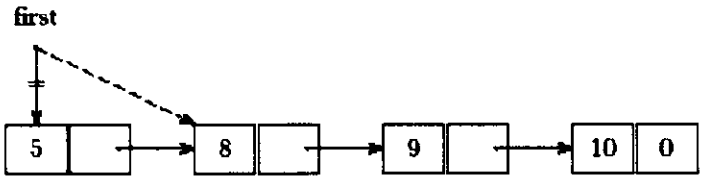


Figure A.6 Deleting an element at the beginning of a linked list

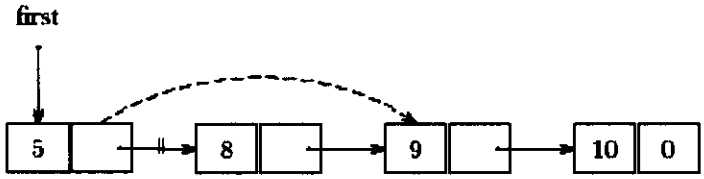


Figure A.7 Deleting an element in the middle of a linked list

B

BRANCH AND BOUND

B.1 The Branch and Bound procedure

The branch in Branch-and-Bound hints at the partitioning process where a "divide-and-conquer" type of approach is used to find and prove the optimality of a solution. Bounds are used during this process to avoid an exhaustive search in the solution space in the sense that the bounds enable one to employ a pruning process to eliminate certain areas of the search space.
Plane & McMillan [18] summarize the procedure as follows:

1. If the solution to a linear programming problem happens to yield an optimal solution with all integer restricted variables assuming integer values, this solution is the optimal solution to the corresponding mixed integer linear programming problem.

2. If we divide or partition the feasible region of a linear programming problem with a constraint of type: either $X3 \leq 4$ or $X3 \geq 5$ we have partitioned the feasible region of the linear programming problem.

We can incorporate this type of either/or constraint by solving two linear programming problems, one problem with each of the constraints added to the previous set of constraints. We will then choose as the solution to the either/or problem the solution with the more attractive value of the objective function. This procedure of adding constraints to form two new problems is called branching. We can call the problem at the end of each branch a node, or a descendent node from the previous nodes.

3. The objective function value at the solution to the linear programming problem at each node forms a bound for the value of the objective function in the original integer programming problem for that branch. We know we cannot find an integer solution for that branch with a higher (lower) value in a maximisation (minimization) problem. If we call a node without descendent nodes a terminal node, we have reached the solution to the original integer problem when we find an all-integer solution whose objective function value is at least as attractive as the bound on each terminal node.

A flow diagram for the branch and bound algorithm is shown in Figure B.1:

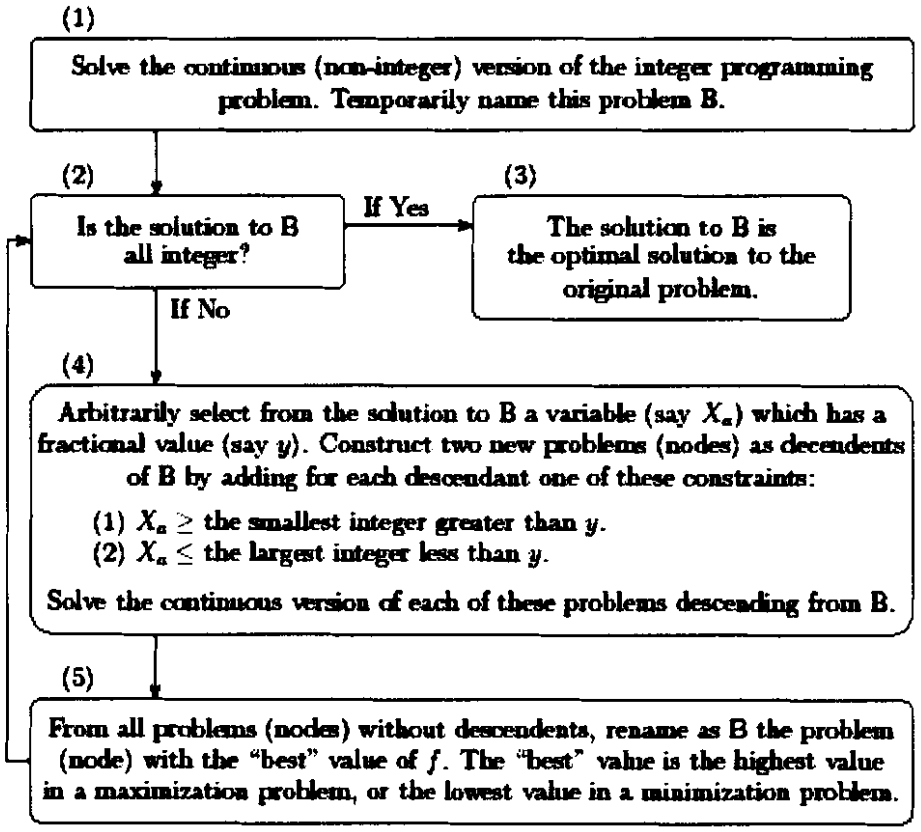


Figure B.1 Flow chart for branch and bound procedure

C

ILLUSTRATION OF 2ECON AND 2NCON

This section describes the procedure used to solve the example given in section 3.6 of chapter 3. We give an illustration of the C++ program used to generate the sets and the edge-cut and node-cut constraints. Next, we also give the linear programs for solving the 2ECON and 2NCON as generated by the C++ program. This feature is designed to make it easy and fast to obtain network instances.

C.1 GenerateNetwork.cpp program

This program is used to create **sets** combinations for us to be able to create edge-cut and node-cut constraints. It then saves the output to a file called SetsOutPut.txt, which then helps us to construct another C++ program called GenerateLP.cpp. GenerateLP.cpp is used to construct linear programming problems, which are then solved with cplex software to obtain optimal solutions for the survivable 2ECON and 2NCON network.

```
#include<iostream>
#include<fstream>
using namespace std;

int Comb(int num, int setn);
double Fact(int numb);

int main()
{
 int  set[5]={0},con[6]={0},sum=0,sum1=0,n=5,k,ct=0,count=0;

 ofstream myFile("c:/mycode/Research/SetsOutPut.txt");
      // Creates an ofstream object named myFile
```

```
if (! myFile) // Always test file open
  {
     cout << "Error opening output file" << endl;
     return -1;
  }
for (int i=1; i<=n; i++){
 if(i<=2)
  con[i]=2;
 else
  con[i]=1;
}
cout<<"The Network to consider, V = {";
myFile<<endl<<endl<<"The Network to consider, V = {";

for (int i=0; i<n; i++){
 set[i]=i+1;
 if(i==n-1){
  cout<<i+1<<"}";
  myFile<<i+1<<"}";
 }
 else{
  cout<<i+1<<",";
  myFile<<i+1<<",";
}
}
cout<<endl;
cout<<"Half of node set considered to avoid duplication is = " << n/2 <<endl<<endl;
myFile<<endl<<endl;
myFile<<"Half of node set considered to avoid duplication is = " << n/2 <<endl<<endl;
for (int t=0; t<n-1; t++){
 k = Comb(n,set[t]);
 if(t == n/2)
  sum1 = sum;
 sum = sum + k;
 cout<<"The number of combinations of set "<< set[t] << " = " << k;
 cout<<endl;
 myFile<<"The number of combinations of set "<< set[t] << " = " << k;
 myFile<<endl;
}
cout<<endl;
cout<<"The total sum of set combination is = "<< sum <<endl<<endl;
myFile<<endl;
myFile<<"The total sum of set combination is = "<< sum <<endl<<endl;

sum=0;
for (int t=0; t<n-1; t++)
```

```
switch (set[t]){
   case 1 : cout<<"The Output of Set Combinations:"<<endl<<endl;
            myFile<<"The Output of Set Combinations:"<<endl<<endl;
            for(int i=1; i<=n; i++){
                ct++;
                cout<<" W"<<ct<<" = {"<<i<<"}";
                myFile<<" W"<<ct<<" = {"<<i<<"}";
              }
              myFile<<endl<<endl;
              ct=0;
              for(int i=1; i<=n; i++){
               ct++;
               myFile<<" δ (W"<<ct<<") = { ";
               for(int v=1; v<=n; v++)
                  if(i!=v){
                     if(v>i)
                        myFile<<"{"<<i<<","<<v<<"} ";
                      else
                        myFile<<"{"<<v<<","<<i<<"} ";
                   }
                myFile<<"}"<<endl;
              }
              sum=sum+ct;
              cout<<endl<<"The count for set 1 is = "<<ct<<endl<<endl;
              myFile<<endl<<"The count for set 1 is = "<<ct<<endl<<endl;
              ct=0;
                   break;
    case 2 : for(int i=1; i<n; i++)
               for(int j=2; j<=n; j++){
                 if(count==6){
                    count=0;
                    cout<<endl;
                    myFile<<endl;
                   }
                  if(i<j){
                   ct++;
                   cout<<" W"<<ct+sum<<" = {"<<i<<","<<j<<"}";
                   myFile<<" W"<<ct+sum<<" = {"<<i<<","<<j<<"}";
                   count++;
                  }
                }
              myFile<<endl<<endl;
              ct=0;
              for(int i=1; i<n; i++)
                 for(int j=2; j<=n; j++)
                    if(i<j){
```

```
                ct++;
                myFile<<" δ (W"<<ct+sum<<") = { ";
                for(int v=1; v<=n; v++)
                 if((i!=v)&&(j!=v)){
                    if(v>i)
                      myFile<<"{"<<i<<","<<v<<"} ";
                    else
                     myFile<<"{"<<v<<","<<i<<"} ";
                    if(v>j)
                     myFile<<"{"<<j<<","<<v<<"} ";
                    else
                     myFile<<"{"<<v<<","<<j<<"} ";
                  }
                myFile<<"}"<<endl;
               }
          sum=sum+ct;
          count=0;
          cout<<endl<<"The count for set 2 is = "<<ct<<endl<<endl;
          myFile<<endl<<"The count for set 2 is = "<<ct<<endl<<endl;
          ct=0;
          break;
case 3 : for(int i=1; i<n-1; i++)
          for(int j=2; j<n; j++)
            for(int k=3; k<=n; k++){
              if(count==6){
               count=0;
               cout<<endl;
               myFile<<endl;
              }
             if((i<j)&&(j<k)){
              ct++;
              cout<<" W"<<ct+sum<<" = {"<<i<<","<<j<<","<<k<<"}";
              myFile<<" W"<<ct+sum<<" = {"<<i<<","<<j<<","<<k<<"}";
              count++;
             }
            }
          myFile<<endl<<endl;
          ct=0;
          for(int i=1; i<n-1; i++)
             for(int j=2; j<n; j++)
            for(int k=3; k<=n; k++)
                 if((i<j)&&(j<k)){
              ct++;
              myFile<<" δ (W"<<ct+sum<<") = { ";
              for(int v=1; v<=n; v++)
               if((i!=v)&&(j!=v)&&(k!=v)){
```

```
                    if(v>i)
                      myFile<<"{"<<i<<","<<v<<"} ";
                    else
                      myFile<<"{"<<v<<","<<i<<"} ";
                    if(v>j)
                      myFile<<"{"<<j<<","<<v<<"} ";
                    else
                      myFile<<"{"<<v<<","<<j<<"} ";
                    if(v>k)
                      myFile<<"{"<<k<<","<<v<<"} ";
                    else
                      myFile<<"{"<<v<<","<<k<<"} ";
                   }
                  myFile<<"}"<<endl;
                 }
            sum=sum+ct;
            count=0;
            cout<<endl<<"The count for set 3 is = "<<ct<<endl<<endl;
            myFile<<endl<<"The count for set 3 is = "<<ct<<endl<<endl;
            ct=0;
            break;
 case 4 : for(int i=1; i<n-2; i++)
           for(int j=2; j<n-1; j++)
            for(int k=3; k<n; k++)
             for(int r=4; r<=n; r++){
                   if(count==5){
               count=0;
               cout<<endl;
               myFile<<endl;
              }
             if((i<j)&&(j<k)&&(k<r)){
               ct++;
               cout<<" W"<<ct+sum<<" = {"<<i<<","<<j<<","<<k<<","<<r<<"}";
               myFile<<" W"<<ct+sum<<" = {"<<i<<","<<j<<","<<k<<","<<r<<"}";
               count++;
             }
            }
            myFile<<endl<<endl;
            ct=0;
            for(int i=1; i<n-2; i++)
                for(int j=2; j<n-1; j++)
               for(int k=3; k<n; k++)
                 for(int r=4; r<=n; r++)
                       if((i<j)&&(j<k)&&(k<r)){
                    ct++;
                    myFile<<" δ (W"<<ct+sum<<") = { ";
```

```
                        for(int v=1; v<=n; v++)
                          if((i!=v)&&(j!=v)&&(k!=v)&&(r!=v)){
                            if(v>i)
                              myFile<<"{"<<i<<","<<v<<"} ";
                            else
                              myFile<<"{"<<v<<","<<i<<"} ";
                            if(v>j)
                              myFile<<"{"<<j<<","<<v<<"} ";
                            else
                              myFile<<"{"<<v<<","<<j<<"} ";
                            if(v>k)
                              myFile<<"{"<<k<<","<<v<<"} ";
                            else
                              myFile<<"{"<<v<<","<<k<<"} ";
                            if(v>r)
                              myFile<<"{"<<r<<","<<v<<"} ";
                            else
                              myFile<<"{"<<v<<","<<r<<"} ";
                          }
                        myFile<<"}"<<endl;
                      }
                sum=sum+ct;
                count=0;
                cout<<endl<<"The count for set 4 is = "<<ct<<endl<<endl;
                myFile<<endl<<"The count for set 4 is = "<<ct<<endl<<endl;
                ct=0;
                break;
        default : cout<<"Sorry try again with something else!"<<endl;
    }
 cout<<"The number of sets used to avoid duplicates is = "<<sum1<<endl<<endl;
 myFile<<endl<<"The number of sets used to avoid duplicates is = "<<sum1<<endl<<endl;
 myFile<<"Node connectivity: "<<endl;
 for(int i=1; i<=n; i++){
  myFile<<"con["<<i<<"] = "<<con[i]<<" ";
  }
  myFile.close();
 }

int Comb(int num, int setn)
{
 return Fact(num) / (Fact(num-setn) * Fact(setn));
}

double Fact(int numb)
{
 if (numb <= 1)
```

```
  return 1;
return numb * Fact(numb-1);
}
```

C.2 SetsOutPut.txt file

```
The Network to consider, V = {1,2,3,4,5}
Half of node set considered to avoid duplication is = 2
The number of combinations of set 1 = 5
The number of combinations of set 2 = 10
The number of combinations of set 3 = 10
The number of combinations of set 4 = 5

The total sum of set combination is = 30
The Output of Set Combinations:
W1 = {1} W2 = {2} W3 = {3} W4 = {4} W5 = {5}
δ (W1) = { {1,2} {1,3} {1,4} {1,5} }
δ (W2) = { {1,2} {2,3} {2,4} {2,5} }
δ (W3) = { {1,3} {2,3} {3,4} {3,5} }
δ (W4) = { {1,4} {2,4} {3,4} {4,5} }
δ (W5) = { {1,5} {2,5} {3,5} {4,5} }
The count for set 1 is = 5

W6 = {1,2} W7 = {1,3} W8 = {1,4} W9 = {1,5} W10 = {2,3} W11 = {2,4}
W12 = {2,5} W13 = {3,4} W14 = {3,5} W15 = {4,5}
δ (W6) = { {1,3} {2,3} {1,4} {2,4} {1,5} {2,5} }
δ (W7) = { {1,2} {2,3} {1,4} {3,4} {1,5} {3,5} }
δ (W8) = { {1,2} {2,4} {1,3} {3,4} {1,5} {4,5} }
δ (W9) = { {1,2} {2,5} {1,3} {3,5} {1,4} {4,5} }
δ (W10) = { {1,2} {1,3} {2,4} {3,4} {2,5} {3,5} }
δ (W11) = { {1,2} {1,4} {2,3} {3,4} {2,5} {4,5} }
δ (W12) = { {1,2} {1,5} {2,3} {3,5} {2,4} {4,5} }
δ (W13) = { {1,3} {1,4} {2,3} {2,4} {3,5} {4,5} }
δ (W14) = { {1,3} {1,5} {2,3} {2,5} {3,4} {4,5} }
δ (W15) = { {1,4} {1,5} {2,4} {2,5} {3,4} {3,5} }
The count for set 2 is = 10

W16 = {1,2,3} W17 = {1,2,4} W18 = {1,2,5} W19 = {1,3,4} W20 = {1,3,5} W21 = {1,4,5}
W22 = {2,3,4} W23 = {2,3,5} W24 = {2,4,5} W25 = {3,4,5}
δ (W16) = { {1,4} {2,4} {3,4} {1,5} {2,5} {3,5} }
δ (W17) = { {1,3} {2,3} {3,4} {1,5} {2,5} {4,5} }
δ (W18) = { {1,3} {2,3} {3,5} {1,4} {2,4} {4,5} }
δ (W19) = { {1,2} {2,3} {2,4} {1,5} {3,5} {4,5} }
```

```
δ (W20) = { {1,2} {2,3} {2,5} {1,4} {3,4} {4,5} }
δ (W21) = { {1,2} {2,4} {2,5} {1,3} {3,4} {3,5} }
δ (W22) = { {1,2} {1,3} {1,4} {2,5} {3,5} {4,5} }
δ (W23) = { {1,2} {1,3} {1,5} {2,4} {3,4} {4,5} }
δ (W24) = { {1,2} {1,4} {1,5} {2,3} {3,4} {3,5} }
δ (W25) = { {1,3} {1,4} {1,5} {2,3} {2,4} {2,5} }
The count for set 3 is = 10

W26 = {1,2,3,4} W27 = {1,2,3,5} W28 = {1,2,4,5} W29 = {1,3,4,5} W30 = {2,3,4,5}
δ (W26) = { {1,5} {2,5} {3,5} {4,5} }
δ (W27) = { {1,4} {2,4} {3,4} {4,5} }
δ (W28) = { {1,3} {2,3} {3,4} {3,5} }
δ (W29) = { {1,2} {2,3} {2,4} {2,5} }
δ (W30) = { {1,2} {1,3} {1,4} {1,5} }
The count for set 4 is = 5
The number of sets used to avoid duplicates is = 15
Node connectivity:
con[1] = 2, con[2] = 2, con[3] = 1, con[4] = 1, and con[5] = 1
```

C.3 GenerateLP.cpp program

```
#include<iostream>
#include<fstream>
using namespace std;

int Comb(int num, int setn);
double Fact(int numb);
int main()
{
 int  set[15]={0}, con[16]={0}, sum=0, n=5, k, p=2, ct=0, first=0, count=0;


 //ofstream myFile("c:/mycode/Research/Network2Econ.lp");
 ofstream myFile("c:/mycode/Research/Network2Ncon.lp");
      // Creates an ofstream object named myFile
 if (! myFile) // Always test file open
   {
      cout << "Error opening output file" << endl;
```

```
      return -1;
   }
for (int i=1; i<=n; i++){
 if(i<=2)
   con[i]=2;
 else
  con[i]=1;
}
for (int i=0; i<n; i++){
 set[i]=i+1;
}
for (int t=0; t<n-1; t++){
 k = Comb(n,set[t]);
 if(t < n/2)
 sum = sum + k;
}
if (p == 1){
  myFile<<"\\\\ Problem Name : TwoEconNetwork"<<endl;
else
  myFile<<"\\\\ Problem Name : TwoNconNetwork"<<endl;
}
myFile<<"Minimize"<<endl;
myFile<<"Obj:";
 for(int i=1; i<n; i++)
  for(int j=2; j<=n; j++){
    if(count==6){
      count=0;
      myFile<<endl;
      myFile<<"    ";
     }
    if(i < j){
```

```
    if(first != 0)
      myFile<<" + Y"<<i<<j;
    else
      myFile<<" Y"<<i<<j;
    count++;
    if(first == 0)
      first=1;
   }
  }
 myFile<<endl<<"Subject To"<<endl;
count=0;
myFile<<endl<<"\\\\ Edge cut constraints."<<endl;
 if(p!=2){
 for (int t=0; t<n/2; t++){
  switch (set[t]){
    case 1 : cout<<"The Output of Set Combinations:"<<endl<<endl;
             for(int i=1; i<=n; i++){
                ct++;
                cout<<" W"<<ct<<" = {"<<i<<"}";
              }
             ct=0;
             sum=0;
             for(int i=1; i<=n; i++){
              ct++;
              first=0;
                    myFile<<" y(d(w"<<ct<<")): ";
              for(int v=1; v<=n; v++)
                 if(i!=v){
                    if(first != 0)
                       myFile<<" + ";
                    if(v>i)
```

```
                    myFile<<"Y"<<i<<v;
                  else
                    myFile<<"Y"<<v<<i;
                  if(first == 0)
                     first = 1;
                 }
                  myFile<<" >= "<<con[i]<<endl;
              }
            sum=sum+ct;
            cout<<endl<<"The count for set 1 is = "<<ct<<endl<<endl;
            ct=0;
            myFile<<endl;
            break;
case 2 : for(int i=1; i<n; i++)
          for(int j=2; j<=n; j++){
           if(count==6){
               count=0;
                cout<<endl;
                 }
                if(i<j){
                 ct++;
                 cout<<" W"<<ct+sum<<" = {"<<i<<", "<<j<<"}";
                 count++;
                }
               }
            ct=0;
            for(int i=1; i<n; i++)
              for(int j=2; j<=n; j++)
                if(i<j){
                    ct++;
                    first=0;
```

```
                    myFile<<" y(d(w"<<ct+sum<<")): ";
                    for(int v=1; v<=n; v++)
                     if((i!=v)&&(j!=v)){
                        if(first != 0)
                          myFile<<" + ";
                        if(v>i)
                          myFile<<"Y"<<i<<v;
                        else
                          myFile<<"Y"<<v<<i;
                        if(first == 0)
                          first = 1;
                        if(v>j)
                          myFile<<" + Y"<<j<<v;
                        else
                          myFile<<" + Y"<<v<<j;
                     }
                    if((i==1)&&(j!=2))||(i==2))
                      myFile<<" >= "<<con[i]<<endl;
                    else
                      myFile<<" >= 1"<<endl;
                   }
            sum=sum+ct;
            count=0;
            cout<<endl<<"The count for set 2 is = "<<ct<<endl<<endl;
            ct=0;
                 myFile<<endl;
            break;
      default : cout<<"Sorry try with a smaller number below 14 nodes!"<<endl;
      }
  }
}
```

```
else{
  for (int t=0; t<n/2; t++)
    switch (set[t]){
      case 1 : cout<<"The Output of Set Combinations:"<<endl<<endl;
            for(int i=1; i<=n; i++){
              ct++;
              cout<<" W"<<ct<<" = {"<<i<<"}";
            }
            ct=0;
            sum=0;
            for(int i=1; i<=n; i++){
             ct++;
             first=0;
             myFile<<" y(d(w"<<ct<<")): ";
             for(int v=1; v<=n; v++)
                if(i!=v){
                   if(first != 0)
                      myFile<<" + ";
                   if(v>i)
                       myFile<<"Y"<<i<<v;
                    else
                      myFile<<"Y"<<v<<i;
                    if(first == 0)
                       first = 1;
                  }

              myFile<<" >= "<<con[i]<<endl;
             }
            sum=sum+ct;
            cout<<endl<<"The count for set 1 is = "<<ct<<endl<<endl;
            ct=0;
```

```
                myFile<<endl;
            break;
    case 2 : for(int i=1; i<n; i++)
            for(int j=2; j<=n; j++){
            if(count==6){
                    count=0;
                    cout<<endl;
                   }
                  if(i<j){
                   ct++;
              cout<<" W"<<ct+sum<<" = {"<<i<<", "<<j<<"}";
                   count++;
                  }
                 }
              ct=0;
              for(int i=1; i<n; i++)
                for(int j=2; j<=n; j++)
                  if(i<j){
                      ct++;
                      first=0;
                      myFile<<" y(δ (W"<<ct+sum<<")): ";
                      for(int v=1; v<=n; v++)
                       if((i!=v)&&(j!=v)){
                          if(first != 0)
                            myFile<<" + ";
                          if(v>i)
                            myFile<<"Y"<<i<<v;
                          else
                           myFile<<"Y"<<v<<i;
                          if(first == 0)
                           first = 1;
```

```
                    if(v>j)
                      myFile<<" + Y"<<j<<v;
                    else
                      myFile<<" + Y"<<v<<j;
                  }
                  if(((i==1)&&(j!=2))||(i==2))
                   myFile<<" >= "<<con[i]<<endl;
                  else
                   myFile<<" >= 1"<<endl;
                 }
          sum=sum+ct;
          count=0;
          cout<<endl<<"The count for set 2 is = "<<ct<<endl<<endl;
          ct=0;
               myFile<<endl;
          break;
    default : cout<<"Sorry try with a smaller number below 14 nodes!"<<endl;
  }
myFile<<endl<<"\\\\ Node cut constraints."<<endl;
sum=0;
ct=0;
for (int t=3; t<=n; t++){
  switch (t){
    case 3 : for(int i=1; i<=2; i++)
               for(int j=4; j<=n; j++){
                 if(count==6){
                  count=0;
                  cout<<endl;
                 }
                if(i<j){
                 ct++;
```

```
      cout<<" N3W"<<ct+sum<<" = {"<<i<<", "<<j<<"}";
        count++;
       }
     }
ct=0;
for(int i=1; i<=2; i++)
 for(int j=4; j<=n; j++)
    if(i<j){
       ct++;
       first=0;
       myFile<<" y(δ (N3W"<<ct+sum<<")): ";
       for(int v=1; v<=n; v++)
        if((i!=v)&&(j!=v)&&(v!=3)){
           if(first != 0)
             myFile<<" + ";
           if(v>i)
             myFile<<"Y"<<i<<v;
           else
            myFile<<"Y"<<v<<i;
           if(first == 0)
            first = 1;
           if(v>j)
             myFile<<" + Y"<<j<<v;
           else
             myFile<<" + Y"<<v<<j;
        }
       myFile<<" >= 1"<<endl;
      }
sum=sum+ct;
count=0;
cout<<endl<<"The count for set 2 is = "<<ct<<endl<<endl;
```

```
ct=0;
for(int i=1; i<=2; i++)
 for(int j=4; j<n; j++)
  for(int k=5; k<=n; k++){
   if(count==6){
    count=0;
    cout<<endl;
   }
   if((i<j)&&(j<k)){
    ct++;
cout<<" N3W"<<ct+sum<<" = {"<<i<<", "<<j<<", "<<k<<"}";
    count++;
   }
  }
 ct=0;
 for(int i=1; i<=2; i++)
  for(int j=4; j<n; j++)
   for(int k=5; k<=n; k++)
    if((i<j)&&(j<k)){
     ct++;
     first=0;
     myFile<<" y(δ (N3W"<<ct+sum<<")): ";
     for(int v=1; v<=n; v++)
      if((i!=v)&&(j!=v)&&(k!=v)&&(v!=3)){
       if(first != 0)
        myFile<<" + ";
       if(v>i)
        myFile<<"Y"<<i<<v;
       else
        myFile<<"Y"<<v<<i;
       if(first == 0)
```

```
                    first = 1;
                   if(v>j)
                     myFile<<" + Y"<<j<<v;
                   else
                     myFile<<" + Y"<<v<<j;
                   if(v>k)
                     myFile<<" + Y"<<k<<v;
                   else
                     myFile<<" + Y"<<v<<k;
                 }
                myFile<<" >= 1"<<endl;
               }
            sum=0;
            count=0;
            cout<<endl<<"The count for set 3 is = "<<ct<<endl<<endl;
            ct=0;
            myFile<<endl;
            break;
case 4 : for(int i=1; i<=2; i++)
              for(int j=3; j<=n; j++){
                  if(count==6){
               count=0;
               cout<<endl;
              }
             if((i<j)&&(j!=4)){
              ct++;
              cout<<" N4W"<<ct+sum<<" = {"<<i<<", "<<j<<"}";
              count++;
             }
           }
         ct=0;
```

```
for(int i=1; i<=2; i++)
        for(int j=3; j<=n; j++)
         if((i<j)&&(j!=4)){
         ct++;
         first=0;
         myFile<<" y(δ (N4W"<<ct+sum<<")): ";
         for(int v=1; v<=n; v++)
          if((i!=v)&&(j!=v)&&(v!=4)){
             if(first != 0)
               myFile<<" + ";
             if(v>i)
               myFile<<"Y"<<i<<v;
             else
              myFile<<"Y"<<v<<i;
             if(first == 0)
              first = 1;
             if(v>j)
               myFile<<" + Y"<<j<<v;
             else
              myFile<<" + Y"<<v<<j;
          }
        myFile<<" >= 1"<<endl;
       }
sum=sum+ct;
count=0;
cout<<endl<<"The count for set 2 is = "<<ct<<endl<<endl;
ct=0;
for(int i=1; i<=2; i++)
           for(int j=3; j<n; j++)
    for(int k=5; k<=n; k++){
              if(count==6){
```

```
        count=0;
        cout<<endl;
       }
      if((i<j)&&(j<k)&&(j!=4)){
       ct++;
       cout<<" N4W"<<ct+sum<<" = {"<<i<<", "<<j<<", "<<k<<"}";
       count++;
      }
    }
ct=0;
for(int i=1; i<=2; i++)
 for(int j=3; j<n; j++)
   for(int k=5; k<=n; k++)
    if((i<j)&&(j<k)&&(j!=4)){
      ct++;
      first=0;
      myFile<<" y(δ (N4W"<<ct+sum<<")): ";
      for(int v=1; v<=n; v++)
       if((i!=v)&&(j!=v)&&(k!=v)&&(v!=4)){
         if(first != 0)
           myFile<<" + ";
         if(v>i)
           myFile<<"Y"<<i<<v;
         else
           myFile<<"Y"<<v<<i;
         if(first == 0)
           first = 1;
         if(v>j)
           myFile<<" + Y"<<j<<v;
         else
           myFile<<" + Y"<<v<<j;
```

```
                    if(v>k)
                      myFile<<" + Y"<<k<<v;
                    else
                      myFile<<" + Y"<<v<<k;
                  }
                myFile<<" >= 1"<<endl;
                }
              sum=0;
              count=0;
              cout<<endl<<"The count for set 3 is = "<<ct<<endl<<endl;
              ct=0;
              myFile<<endl;
              break;
case 5 : for(int i=1; i<=2; i++)
                    for(int j=3; j<=n; j++){
                      if(count==6){
                 count=0;
                 cout<<endl;
                }
               if((i<j)&&(j!=5)){
                ct++;
             cout<<" N5W"<<ct+sum<<" = {"<<i<<", "<<j<<"}";
                count++;
               }
             }
           ct=0;
           for(int i=1; i<=2; i++)
                for(int j=3; j<=n; j++)
                    if((i<j)&&(j!=5)){
                 ct++;
                 first=0;
```

```
        myFile<<" y(δ (N5W"<<ct+sum<<")): ";
        for(int v=1; v<=n; v++)
         if((i!=v)&&(j!=v)&&(v!=5)){
           if(first != 0)
             myFile<<" + ";
           if(v>i)
             myFile<<"Y"<<i<<v;
           else
            myFile<<"Y"<<v<<i;
           if(first == 0)
            first = 1;
           if(v>j)
             myFile<<" + Y"<<j<<v;
           else
            myFile<<" + Y"<<v<<j;
         }
       myFile<<" >= 1"<<endl;
      }
sum=sum+ct;
count=0;
cout<<endl<<"The count for set 2 is = "<<ct<<endl<<endl;
ct=0;
for(int i=1; i<=2; i++)
          for(int j=3; j<n; j++)
   for(int k=4; k<=n; k++){
             if(count==6){
      count=0;
      cout<<endl;
     }
    if((i<j)&&(j<k)&&(j!=5)&&(k!=5)){
     ct++;
```

```
        cout<<" N5W"<<ct+sum<<" = {"<<i<<", "<<j<<", "<<k<<"}";
        count++;
       }
     }
ct=0;
for(int i=1; i<=2; i++)
 for(int j=3; j<n; j++)
   for(int k=4; k<=n; k++)
    if((i<j)&&(j<k)&&(j!=5)&&(k!=5)){
       ct++;
       first=0;
       myFile<<" y(δ (N5W"<<ct+sum<<")): ";
       for(int v=1; v<=n; v++)
        if((i!=v)&&(j!=v)&&(k!=v)&&(v!=5)){
          if(first != 0)
            myFile<<" + ";
          if(v>i)
            myFile<<"Y"<<i<<v;
          else
            myFile<<"Y"<<v<<i;
          if(first == 0)
            first = 1;
          if(v>j)
            myFile<<" + Y"<<j<<v;
          else
            myFile<<" + Y"<<v<<j;
          if(v>k)
            myFile<<" + Y"<<k<<v;
          else
            myFile<<" + Y"<<v<<k;
        }
```

```
                    myFile<<" >= 1"<<endl;
                    }
                sum=0;
                count=0;
                cout<<endl<<"The count for set 3 is = "<<ct<<endl<<endl;
                ct=0;
                myFile<<endl;
                break;
      default : cout<<"Sorry try nodes below 6!"<<endl;
  }
 }
}
myFile<<"Bounds"<<endl;
for(int i=1; i<n; i++)
 for(int j=2; j<=n; j++){
  if(i < j){
   myFile<<" 0 <= Y"<<i<<j<<" <= 1"<<endl;
  }
 }
myFile<<"Binaries"<<endl;
for(int i=1; i<n; i++)
 for(int j=2; j<=n; j++){
  if(count==8){
   count=0;
   myFile<<endl;
  }
  if(i < j){
   myFile<<" Y"<<i<<j<<" ";
   count++;
  }
 }
```

```
 myFile<<endl<<"End"<<endl;
 myFile.close();
}

int Comb(int num, int setn)
{
 return Fact(num) / (Fact(num-setn) * Fact(setn));
}

double Fact(int numb)
{
 if (numb <= 1)
   return 1;
 return numb * Fact(numb-1);
}
```

C.4 Network2Econ.lp problem

```
\\ Problem Name : TwoEconNetwork
Minimize
Obj: 2Y12 + 0.5Y13 + Y14 + 2Y15 + 0.5Y23 + 2Y24
   + Y25 + 0.5Y34 + 0.5Y35 + 2Y45
Subject To
\\ Edge cut constraints.
 y(δ (W1)): Y12 + Y13 + Y14 + Y15 >= 2
 y(δ (W2)): Y12 + Y23 + Y24 + Y25 >= 2
 y(δ (W3)): Y13 + Y23 + Y34 + Y35 >= 1
 y(δ (W4)): Y14 + Y24 + Y34 + Y45 >= 1
 y(δ (W5)): Y15 + Y25 + Y35 + Y45 >= 1
 y(δ (W6)): Y13 + Y23 + Y14 + Y24 + Y15 + Y25 >= 1
 y(δ (W7)): Y12 + Y23 + Y14 + Y34 + Y15 + Y35 >= 2
 y(δ (W8)): Y12 + Y24 + Y13 + Y34 + Y15 + Y45 >= 2
 y(δ (W9)): Y12 + Y25 + Y13 + Y35 + Y14 + Y45 >= 2
 y(δ (W10)): Y12 + Y13 + Y24 + Y34 + Y25 + Y35 >= 2
 y(δ (W11)): Y12 + Y14 + Y23 + Y34 + Y25 + Y45 >= 2
 y(δ (W12)): Y12 + Y15 + Y23 + Y35 + Y24 + Y45 >= 2
```

```
 y(δ (W13)): Y13 + Y14 + Y23 + Y24 + Y35 + Y45 >= 1
 y(δ (W14)): Y13 + Y15 + Y23 + Y25 + Y34 + Y45 >= 1
 y(δ (W15)): Y14 + Y15 + Y24 + Y25 + Y34 + Y35 >= 1
Bounds
 0 <= Y12 <= 1
 0 <= Y13 <= 1
 0 <= Y14 <= 1
 0 <= Y15 <= 1
 0 <= Y23 <= 1
 0 <= Y24 <= 1
 0 <= Y25 <= 1
 0 <= Y34 <= 1
 0 <= Y35 <= 1
 0 <= Y45 <= 1
Binaries
 Y12 Y13 Y14 Y15 Y23 Y24 Y25 Y34
 Y35 Y45
End
```

C.5 Network2Ncon.lp problem

```
\\ Problem Name : TwoNconNetwork

Minimize

Obj: 2Y12 + 0.5Y13 + Y14 + 2Y15 + 0.5Y23 + 2Y24
   + Y25 + 0.5Y34 + 0.5Y35 + 2Y45
Subject To

\\ Edge cut constraints.
 y(δ (W1)): Y12 + Y13 + Y14 + Y15 >= 2
 y(δ (W2)): Y12 + Y23 + Y24 + Y25 >= 2
 y(δ (W3)): Y13 + Y23 + Y34 + Y35 >= 1
 y(δ (W4)): Y14 + Y24 + Y34 + Y45 >= 1
 y(δ (W5)): Y15 + Y25 + Y35 + Y45 >= 1
 y(δ (W6)): Y13 + Y23 + Y14 + Y24 + Y15 + Y25 >= 1
 y(δ (W7)): Y12 + Y23 + Y14 + Y34 + Y15 + Y35 >= 2
 y(δ (W8)): Y12 + Y24 + Y13 + Y34 + Y15 + Y45 >= 2
 y(δ (W9)): Y12 + Y25 + Y13 + Y35 + Y14 + Y45 >= 2
 y(δ (W10)): Y12 + Y13 + Y24 + Y34 + Y25 + Y35 >= 2
 y(δ (W11)): Y12 + Y14 + Y23 + Y34 + Y25 + Y45 >= 2
 y(δ (W12)): Y12 + Y15 + Y23 + Y35 + Y24 + Y45 >= 2
 y(δ (W13)): Y13 + Y14 + Y23 + Y24 + Y35 + Y45 >= 1
 y(δ (W14)): Y13 + Y15 + Y23 + Y25 + Y34 + Y45 >= 1
 y(δ (W15)): Y14 + Y15 + Y24 + Y25 + Y34 + Y35 >= 1
\\ Node cut constraints.
 y(δ (N3W1)): Y12 + Y24 + Y15 + Y45 >= 1
 y(δ (N3W2)): Y12 + Y25 + Y14 + Y45 >= 1
```

```
 y(δ (N3W3)): Y12 + Y14 + Y25 + Y45 >= 1
 y(δ (N3W4)): Y12 + Y15 + Y24 + Y45 >= 1
 y(δ (N3W5)): Y12 + Y24 + Y25 >= 1
 y(δ (N3W6)): Y12 + Y14 + Y15 >= 1
 y(δ (N4W1)): Y12 + Y23 + Y15 + Y35 >= 1
 y(δ (N4W2)): Y12 + Y25 + Y13 + Y35 >= 1
 y(δ (N4W3)): Y12 + Y13 + Y25 + Y35 >= 1
 y(δ (N4W4)): Y12 + Y15 + Y23 + Y35 >= 1
 y(δ (N4W5)): Y12 + Y23 + Y25 >= 1
 y(δ (N4W6)): Y12 + Y13 + Y15 >= 1
 y(δ (N5W1)): Y12 + Y23 + Y14 + Y34 >= 1
 y(δ (N5W2)): Y12 + Y24 + Y13 + Y34 >= 1
 y(δ (N5W3)): Y12 + Y13 + Y24 + Y34 >= 1
 y(δ (N5W4)): Y12 + Y14 + Y23 + Y34 >= 1
 y(δ (N5W5)): Y12 + Y23 + Y24 >= 1
 y(δ (N5W6)): Y12 + Y13 + Y14 >= 1
Bounds
 0 <= Y12 <= 1
 0 <= Y13 <= 1
 0 <= Y14 <= 1
 0 <= Y15 <= 1
 0 <= Y23 <= 1
 0 <= Y24 <= 1
 0 <= Y25 <= 1
 0 <= Y34 <= 1
 0 <= Y35 <= 1
 0 <= Y45 <= 1
Binaries
 Y12 Y13 Y14 Y15 Y23 Y24 Y25 Y34
 Y35 Y45
End
```

C.6 NetTest2ncon.lp problem

```
\\Problem Name: NetTest
Minimize

OBJ: 216.24X010301T0 + 508.40y00103
 + 27.48X010301T1 + 947.60y10103
 + 199.92X010401T0 + 496.80y00104
 + 58.88X010401T1 + 877.20y10104
 + 206.88X010501T0 + 459.60y00105
 + 24.31X010501T1 + 822.80y10105
 + 293.52X010601T0 + 408.00y00106
 + 125.68X010601T1 + 606.80y10106
 + 215.84X010701T0 + 520.00y00107
 + 56.92X010701T1 + 989.20y10107
 + 308.32X010801T0 + 449.60y00108
 + 153.36X010801T1 + 812.80y10108
```

```
 + 242.52X030201T0 + 468.00y00302
 + 121.12X030201T1 + 918.80y10302
 + 76.48X030401T0 + 76.48X040301T0 + 56.51y00304
 + 36.60X030401T1 + 36.60X040301T1 + 110.50y10304
 + 240.64X030501T0 + 240.64X050301T0 + 542.00y00305
 + 64.96X030501T1 + 64.96X050301T1 + 476.80y10305
 + 70.56X030601T0 + 70.56X060301T0 + 103.70y00306
 + 27.86X030601T1 + 27.86X060301T1 + 130.60y10306
 + 238.36X030701T0 + 238.36X070301T0 + 464.00y00307
 + 48.72X030701T1 + 48.72X070301T1 + 898.00y10307
 + 287.56X030801T0 + 287.56X080301T0 + 590.80y00308
 + 37.32X030801T1 + 37.32X080301T1 + 938.00y10308
 + 298.08X040201T0 + 467.20y00402
 + 44.80X040201T1 + 758.80y10402
 + 56.64X040501T0 + 56.64X050401T0 + 112.50y00405
 + 8.76X040501T1 + 8.76X050401T1 + 200.70y10405
 + 79.19X040601T0 + 79.19X060401T0 + 79.63y00406
 + 34.44X040601T1 + 34.44X060401T1 + 188.80y10406
 + 74.58X040701T0 + 74.58X070401T0 + 109.30y00407
 + 39.44X040701T1 + 39.44X070401T1 + 209.50y10407
 + 76.39X040801T0 + 76.39X080401T0 + 86.28y00408
 + 6.10X040801T1 + 6.10X080401T1 + 119.10y10408
 + 316.24X050201T0 + 568.40y00502
 + 60.92X050201T1 + 744.00y10502
 + 190.16X050601T0 + 190.16X060501T0 + 258.84y00506
 + 113.76X050601T1 + 113.76X060501T1 + 779.60y10506
 + 308.12X050701T0 + 308.12X070501T0 + 257.20y00507
 + 102.04X050701T1 + 102.04X070501T1 + 579.20y10507
 + 62.63X050801T0 + 62.63X080501T0 + 95.32y00508
 + 16.74X050801T1 + 16.74X080501T1 + 141.30y10508
 + 316.12X060201T0 + 539.60y00602
 + 40.96X060201T1 + 730.40y10602
 + 281.00X060701T0 + 281.00X070601T0 + 268.72y00607
 + 57.36X060701T1 + 57.36X070601T1 + 553.20y10607
 + 196.88X060801T0 + 196.88X080601T0 + 298.20y00608
 + 139.56X060801T1 + 139.56X080601T1 + 892.40y10608
 + 190.32X070201T0 + 537.20y00702
 + 110.48X070201T1 + 654.80y10702
 + 221.24X070801T0 + 221.24X080701T0 + 488.80y00708
 + 21.97X070801T1 + 21.97X080701T1 + 839.20y10708
 + 281.64X080201T0 + 488.00y00802
 + 84.08X080201T1 + 952.00y10802
 + 1.00SRC + 1.00DST + 1.00Node03 + 1.00Node04
 + 1.00Node05 + 1.00Node06 + 1.00Node07 + 1.00Node08

Subject To
 SRC01C01: X010301T0 + X010301T1 + X010401T0 + X010401T1 + X010501T0
                + X010501T1 + X010601T0 + X010601T1 + X010701T0 + X010701T1
                + X010801T0 + X010801T1 = 1
 DST02C01: X030201T0 + X030201T1 + X040201T0 + X040201T1 + X050201T0
                + X050201T1 + X060201T0 + X060201T1 + X070201T0 + X070201T1
                + X080201T0 + X080201T1 = 1
 TRN03C01: X030201T0 + X030201T1 + X030401T0 + X030401T1 + X030501T0
               + X030501T1 + X030601T0 + X030601T1 + X030701T0 + X030701T1
               + X030801T0 + X030801T1 - X010301T0 - X010301T1 - X040301T0
               - X040301T1 - X050301T0 - X050301T1 - X060301T0 - X060301T1
               - X070301T0 - X070301T1 - X080301T0 - X080301T1 = 0
```

```
TRN04C01: X040301T0 + X040301T1 + X040201T0 + X040201T1 + X040501T0
              + X040501T1 + X040601T0 + X040601T1 + X040701T0 + X040701T1
              + X040801T0 + X040801T1 - X010401T0 - X010401T1 - X030401T0
              - X030401T1 - X050401T0 - X050401T1 - X060401T0 - X060401T1
              - X070401T0 - X070401T1 - X080401T0 - X080401T1 = 0
TRN05C01: X050301T0 + X050301T1 + X050401T0 + X050401T1 + X050201T0
              + X050201T1 + X050601T0 + X050601T1 + X050701T0 + X050701T1
              + X050801T0 + X050801T1 - X010501T0 - X010501T1 - X030501T0
              - X030501T1 - X040501T0 - X040501T1 - X060501T0 - X060501T1
              - X070501T0 - X070501T1 - X080501T0 - X080501T1 = 0
TRN06C01: X060301T0 + X060301T1 + X060401T0 + X060401T1 + X060501T0
              + X060501T1 + X060201T0 + X060201T1 + X060701T0 + X060701T1
              + X060801T0 + X060801T1 - X010601T0 - X010601T1 - X030601T0
              - X030601T1 - X040601T0 - X040601T1 - X050601T0 - X050601T1
              - X070601T0 - X070601T1 - X080601T0 - X080601T1 = 0
TRN07C01: X070301T0 + X070301T1 + X070401T0 + X070401T1 + X070501T0
              + X070501T1 + X070601T0 + X070601T1 + X070201T0 + X070201T1
              + X070801T0 + X070801T1 - X010701T0 - X010701T1 - X030701T0
              - X030701T1 - X040701T0 - X040701T1 - X050701T0 - X050701T1
              - X060701T0 - X060701T1 - X080701T0 - X080701T1 = 0
TRN08C01: X080301T0 + X080301T1 + X080401T0 + X080401T1 + X080501T0
              + X080501T1 + X080601T0 + X080601T1 + X080701T0 + X080701T1
              + X080201T0 + X080201T1 - X010801T0 - X010801T1 - X030801T0
              - X030801T1 - X040801T0 - X040801T1 - X050801T0 - X050801T1
              - X060801T0 - X060801T1 - X070801T0 - X070801T1 = 0
UB0304T0: 4y00304 - 40X030401T0 - 40X040301T0 >= 0
UB0304T1: 16y10304 - 40X030401T1 - 40X040301T1 >= 0
UB0305T0: 4y00305 - 40X030501T0 - 40X050301T0 >= 0
UB0305T1: 16y10305 - 40X030501T1 - 40X050301T1 >= 0
UB0306T0: 4y00306 - 40X030601T0 - 40X060301T0 >= 0
UB0306T1: 16y10306 - 40X030601T1 - 40X060301T1 >= 0
UB0307T0: 4y00307 - 40X030701T0 - 40X070301T0 >= 0
UB0307T1: 16y10307 - 40X030701T1 - 40X070301T1 >= 0
UB0308T0: 4y00308 - 40X030801T0 - 40X080301T0 >= 0
UB0308T1: 16y10308 - 40X030801T1 - 40X080301T1 >= 0
UB0405T0: 4y00405 - 40X040501T0 - 40X050401T0 >= 0
UB0405T1: 16y10405 - 40X040501T1 - 40X050401T1 >= 0
UB0406T0: 4y00406 - 40X040601T0 - 40X060401T0 >= 0
UB0406T1: 16y10406 - 40X040601T1 - 40X060401T1 >= 0
UB0407T0: 4y00407 - 40X040701T0 - 40X070401T0 >= 0
UB0407T1: 16y10407 - 40X040701T1 - 40X070401T1 >= 0
UB0408T0: 4y00408 - 40X040801T0 - 40X080401T0 >= 0
UB0408T1: 16y10408 - 40X040801T1 - 40X080401T1 >= 0
UB0506T0: 4y00506 - 40X050601T0 - 40X060501T0 >= 0
UB0506T1: 16y10506 - 40X050601T1 - 40X060501T1 >= 0
UB0507T0: 4y00507 - 40X050701T0 - 40X070501T0 >= 0
UB0507T1: 16y10507 - 40X050701T1 - 40X070501T1 >= 0
UB0508T0: 4y00508 - 40X050801T0 - 40X080501T0 >= 0
UB0508T1: 16y10508 - 40X050801T1 - 40X080501T1 >= 0
UB0607T0: 4y00607 - 40X060701T0 - 40X070601T0 >= 0
UB0607T1: 16y10607 - 40X060701T1 - 40X070601T1 >= 0
UB0608T0: 4y00608 - 40X060801T0 - 40X080601T0 >= 0
UB0608T1: 16y10608 - 40X060801T1 - 40X080601T1 >= 0
UB0708T0: 4y00708 - 40X070801T0 - 40X080701T0 >= 0
UB0708T1: 16y10708 - 40X070801T1 - 40X080701T1 >= 0
DD0103: 40X010301T0 + 40X010301T1 = 9
DD0104: 40X010401T0 + 40X010401T1 = 3
```

```
DD0105: 40X010501T0 + 40X010501T1 = 8
DD0106: 40X010601T0 + 40X010601T1 = 11
DD0107: 40X010701T0 + 40X010701T1 = 0
DD0108: 40X010801T0 + 40X010801T1 = 9
DD0302: 40X030201T0 + 40X030201T1 = 4
DD0402: 40X040201T0 + 40X040201T1 = 15
DD0502: 40X050201T0 + 40X050201T1 = 3
DD0602: 40X060201T0 + 40X060201T1 = 5
DD0702: 40X070201T0 + 40X070201T1 = 4
DD0802: 40X080201T0 + 40X080201T1 = 9
NDC01: 40f3434 + 40f3435 + 40f3436
              + 40f3437 + 40f3438 = 4
NDC02: 40f3534 + 40f3535 + 40f3536
              + 40f3537 + 40f3538 = 0
NDC03: 40f3634 + 40f3635 + 40f3636
              + 40f3637 + 40f3638 = 0
NDC04: 40f3734 + 40f3735 + 40f3736
              + 40f3737 + 40f3738 = 0
NDC05: 40f3834 + 40f3835 + 40f3836
              + 40f3837 + 40f3838 = 5
NDC06: 40f4343 + 40f4345 + 40f4346
              + 40f4347 + 40f4348 = 2
NDC07: 40f4543 + 40f4545 + 40f4546
              + 40f4547 + 40f4548 = 0
NDC08: 40f4643 + 40f4645 + 40f4646
              + 40f4647 + 40f4648 = 1
NDC09: 40f4743 + 40f4745 + 40f4746
              + 40f4747 + 40f4748 = 0
NDC10: 40f4843 + 40f4845 + 40f4846
              + 40f4847 + 40f4848 = 0
NDC11: 40f5353 + 40f5354 + 40f5356
              + 40f5357 + 40f5358 = 0
NDC12: 40f5453 + 40f5454 + 40f5456
              + 40f5457 + 40f5458 = 4
NDC13: 40f5653 + 40f5654 + 40f5656
              + 40f5657 + 40f5658 = 2
NDC14: 40f5753 + 40f5754 + 40f5756
              + 40f5757 + 40f5758 = 1
NDC15: 40f5853 + 40f5854 + 40f5856
              + 40f5857 + 40f5858 = 1
NDC16: 40f6363 + 40f6364 + 40f6365
              + 40f6367 + 40f6368 = 1
NDC17: 40f6463 + 40f6464 + 40f6465
              + 40f6467 + 40f6468 = 4
NDC18: 40f6563 + 40f6564 + 40f6565
              + 40f6567 + 40f6568 = 3
NDC19: 40f6763 + 40f6764 + 40f6765
              + 40f6767 + 40f6768 = 0
NDC20: 40f6863 + 40f6864 + 40f6865
              + 40f6867 + 40f6868 = 3
NDC21: 40f7373 + 40f7374 + 40f7375
              + 40f7376 + 40f7378 = 0
NDC22: 40f7473 + 40f7474 + 40f7475
              + 40f7476 + 40f7478 = 0
NDC23: 40f7573 + 40f7574 + 40f7575
              + 40f7576 + 40f7578 = 0
NDC24: 40f7673 + 40f7674 + 40f7675
```

```
                + 40f7676 + 40f7678 = 0
NDC25: 40f7873 + 40f7874 + 40f7875
                + 40f7876 + 40f7878 = 0
NDC26: 40f8383 + 40f8384 + 40f8385
                + 40f8386 + 40f8387 = 1
NDC27: 40f8483 + 40f8484 + 40f8485
                + 40f8486 + 40f8487 = 3
NDC28: 40f8583 + 40f8584 + 40f8585
                + 40f8586 + 40f8587 = 0
NDC29: 40f8683 + 40f8684 + 40f8685
                + 40f8686 + 40f8687 = 2
NDC30: 40f8783 + 40f8784 + 40f8785
                + 40f8786 + 40f8787 = 3
NDC31: 40f3434 + 40f3454 + 40f3464
                + 40f3474 + 40f3484 = 4
NDC32: 40f3535 + 40f3545 + 40f3565
                + 40f3575 + 40f3585 = 0
NDC33: 40f3636 + 40f3646 + 40f3656
                + 40f3676 + 40f3686 = 0
NDC34: 40f3737 + 40f3747 + 40f3757
                + 40f3767 + 40f3787 = 0
NDC35: 40f3838 + 40f3848 + 40f3858
                + 40f3868 + 40f3878 = 5
NDC36: 40f4343 + 40f4353 + 40f4363
                + 40f4373 + 40f4383 = 2
NDC37: 40f4535 + 40f4545 + 40f4565
                + 40f4575 + 40f4585 = 0
NDC38: 40f4636 + 40f4646 + 40f4656
                + 40f4676 + 40f4686 = 1
NDC39: 40f4737 + 40f4747 + 40f4757
                + 40f4767 + 40f4787 = 0
NDC40: 40f4838 + 40f4848 + 40f4858
                + 40f4868 + 40f4878 = 0
NDC41: 40f5343 + 40f5353 + 40f5363
                + 40f5373 + 40f5383 = 0
NDC42: 40f5434 + 40f5454 + 40f5464
                + 40f5474 + 40f5484 = 4
NDC43: 40f5636 + 40f5646 + 40f5656
                + 40f5676 + 40f5686 = 2
NDC44: 40f5737 + 40f5747 + 40f5757
                + 40f5767 + 40f5787 = 1
NDC45: 40f5838 + 40f5848 + 40f5858
                + 40f5868 + 40f5878 = 1
NDC46: 40f6343 + 40f6353 + 40f6363
                + 40f6373 + 40f6383 = 1
NDC47: 40f6434 + 40f6454 + 40f6464
                + 40f6474 + 40f6484 = 4
NDC48: 40f6535 + 40f6545 + 40f6565
                + 40f6575 + 40f6585 = 3
NDC49: 40f6737 + 40f6747 + 40f6757
                + 40f6767 + 40f6787 = 0
NDC50: 40f6838 + 40f6848 + 40f6858
                + 40f6868 + 40f6878 = 3
NDC51: 40f7343 + 40f7353 + 40f7363
                + 40f7373 + 40f7383 = 0
NDC52: 40f7434 + 40f7454 + 40f7464
                + 40f7474 + 40f7484 = 0
```

```
NDC53: 40f7535 + 40f7545 + 40f7565
              + 40f7575 + 40f7585 = 0
NDC54: 40f7636 + 40f7646 + 40f7656
              + 40f7676 + 40f7686 = 0
NDC55: 40f7838 + 40f7848 + 40f7858
              + 40f7868 + 40f7878 = 0
NDC56: 40f8343 + 40f8353 + 40f8363
              + 40f8373 + 40f8383 = 1
NDC57: 40f8434 + 40f8454 + 40f8464
              + 40f8474 + 40f8484 = 3
NDC58: 40f8535 + 40f8545 + 40f8565
              + 40f8575 + 40f8585 = 0
NDC59: 40f8636 + 40f8646 + 40f8656
              + 40f8676 + 40f8686 = 2
NDC60: 40f8737 + 40f8747 + 40f8757
              + 40f8767 + 40f8787 = 3
NDC61: f3435 + f3445 + f3465 + f3475 + f3485 - f3453
              - f3454 - f3456 - f3457 - f3458 = 0
NDC62: f3436 + f3446 + f3456 + f3476 + f3486 - f3463
              - f3464 - f3465 - f3467 - f3468 = 0
NDC63: f3437 + f3447 + f3457 + f3467 + f3487 - f3473
              - f3474 - f3475 - f3476 - f3478 = 0
NDC64: f3438 + f3448 + f3458 + f3468 + f3478 - f3483
              - f3484 - f3485 - f3486 - f3487 = 0
NDC65: f3534 + f3554 + f3564 + f3574 + f3584 - f3543
              - f3545 - f3546 - f3547 - f3548 = 0
NDC66: f3536 + f3546 + f3556 + f3576 + f3586 - f3563
              - f3564 - f3565 - f3567 - f3568 = 0
NDC67: f3537 + f3547 + f3557 + f3567 + f3587 - f3573
              - f3574 - f3575 - f3576 - f3578 = 0
NDC68: f3538 + f3548 + f3558 + f3568 + f3578 - f3583
              - f3584 - f3585 - f3586 - f3587 = 0
NDC69: f3634 + f3654 + f3664 + f3674 + f3684 - f3643
              - f3645 - f3646 - f3647 - f3648 = 0
NDC70: f3635 + f3645 + f3665 + f3675 + f3685 - f3653
              - f3654 - f3656 - f3657 - f3658 = 0
NDC71: f3637 + f3647 + f3657 + f3667 + f3687 - f3673
              - f3674 - f3675 - f3676 - f3678 = 0
NDC72: f3638 + f3648 + f3658 + f3668 + f3678 - f3683
              - f3684 - f3685 - f3686 - f3687 = 0
NDC73: f3734 + f3754 + f3764 + f3774 + f3784 - f3743
              - f3745 - f3746 - f3747 - f3748 = 0
NDC74: f3735 + f3745 + f3765 + f3775 + f3785 - f3753
              - f3754 - f3756 - f3757 - f3758 = 0
NDC75: f3736 + f3746 + f3756 + f3776 + f3786 - f3763
              - f3764 - f3765 - f3767 - f3768 = 0
NDC76: f3738 + f3748 + f3758 + f3768 + f3778 - f3783
              - f3784 - f3785 - f3786 - f3787 = 0
NDC77: f3834 + f3854 + f3864 + f3874 + f3884 - f3843
              - f3845 - f3846 - f3847 - f3848 = 0
NDC78: f3835 + f3845 + f3865 + f3875 + f3885 - f3853
              - f3854 - f3856 - f3857 - f3858 = 0
NDC79: f3836 + f3846 + f3856 + f3876 + f3886 - f3863
              - f3864 - f3865 - f3867 - f3868 = 0
NDC80: f3837 + f3847 + f3857 + f3867 + f3887 - f3873
              - f3874 - f3875 - f3876 - f3878 = 0
NDC81: f4335 + f4345 + f4365 + f4375 + f4385 - f4353
```

```
                - f4354 - f4356 - f4357 - f4358 = 0
NDC82: f4336 + f4346 + f4356 + f4376 + f4386 - f4363
                - f4364 - f4365 - f4367 - f4368 = 0
NDC83: f4337 + f4347 + f4357 + f4367 + f4387 - f4373
                - f4374 - f4375 - f4376 - f4378 = 0
NDC84: f4338 + f4348 + f4358 + f4368 + f4378 - f4383
                - f4384 - f4385 - f4386 - f4387 = 0
NDC85: f4543 + f4553 + f4563 + f4573 + f4583 - f4534
                - f4535 - f4536 - f4537 - f4538 = 0
NDC86: f4536 + f4546 + f4556 + f4576 + f4586 - f4563
                - f4564 - f4565 - f4567 - f4568 = 0
NDC87: f4537 + f4547 + f4557 + f4567 + f4587 - f4573
                - f4574 - f4575 - f4576 - f4578 = 0
NDC88: f4538 + f4548 + f4558 + f4568 + f4578 - f4583
                - f4584 - f4585 - f4586 - f4587 = 0
NDC89: f4643 + f4653 + f4663 + f4673 + f4683 - f4634
                - f4635 - f4636 - f4637 - f4638 = 0
NDC90: f4635 + f4645 + f4665 + f4675 + f4685 - f4653
                - f4654 - f4656 - f4657 - f4658 = 0
NDC91: f4637 + f4647 + f4657 + f4667 + f4687 - f4673
                - f4674 - f4675 - f4676 - f4678 = 0
NDC92: f4638 + f4648 + f4658 + f4668 + f4678 - f4683
                - f4684 - f4685 - f4686 - f4687 = 0
NDC93: f4743 + f4753 + f4763 + f4773 + f4783 - f4734
                - f4735 - f4736 - f4737 - f4738 = 0
NDC94: f4735 + f4745 + f4765 + f4775 + f4785 - f4753
                - f4754 - f4756 - f4757 - f4758 = 0
NDC95: f4736 + f4746 + f4756 + f4776 + f4786 - f4763
                - f4764 - f4765 - f4767 - f4768 = 0
NDC96: f4738 + f4748 + f4758 + f4768 + f4778 - f4783
                - f4784 - f4785 - f4786 - f4787 = 0
NDC97: f4843 + f4853 + f4863 + f4873 + f4883 - f4834
                - f4835 - f4836 - f4837 - f4838 = 0
NDC98: f4835 + f4845 + f4865 + f4875 + f4885 - f4853
                - f4854 - f4856 - f4857 - f4858 = 0
NDC99: f4836 + f4846 + f4856 + f4876 + f4886 - f4863
                - f4864 - f4865 - f4867 - f4868 = 0
NDC100: f4837 + f4847 + f4857 + f4867 + f4887 - f4873
                - f4874 - f4875 - f4876 - f4878 = 0
NDC101: f5334 + f5354 + f5364 + f5374 + f5384 - f5343
                - f5345 - f5346 - f5347 - f5348 = 0
NDC102: f5336 + f5346 + f5356 + f5376 + f5386 - f5363
                - f5364 - f5365 - f5367 - f5368 = 0
NDC103: f5337 + f5347 + f5357 + f5367 + f5387 - f5373
                - f5374 - f5375 - f5376 - f5378 = 0
NDC104: f5338 + f5348 + f5358 + f5368 + f5378 - f5383
                - f5384 - f5385 - f5386 - f5387 = 0
NDC105: f5443 + f5453 + f5463 + f5473 + f5483 - f5434
                - f5435 - f5436 - f5437 - f5438 = 0
NDC106: f5436 + f5446 + f5456 + f5476 + f5486 - f5463
                - f5464 - f5465 - f5467 - f5468 = 0
NDC107: f5437 + f5447 + f5457 + f5467 + f5487 - f5473
                - f5474 - f5475 - f5476 - f5478 = 0
NDC108: f5438 + f5448 + f5458 + f5468 + f5478 - f5483
                - f5484 - f5485 - f5486 - f5487 = 0
NDC109: f5643 + f5653 + f5663 + f5673 + f5683 - f5634
                - f5635 - f5636 - f5637 - f5638 = 0
```

```
NDC110: f5634 + f5654 + f5664 + f5674 + f5684 - f5643
              - f5645 - f5646 - f5647 - f5648 = 0
NDC111: f5637 + f5647 + f5657 + f5667 + f5687 - f5673
              - f5674 - f5675 - f5676 - f5678 = 0
NDC112: f5638 + f5648 + f5658 + f5668 + f5678 - f5683
              - f5684 - f5685 - f5686 - f5687 = 0
NDC113: f5743 + f5753 + f5763 + f5773 + f5783 - f5734
              - f5735 - f5736 - f5737 - f5738 = 0
NDC114: f5734 + f5754 + f5764 + f5774 + f5784 - f5743
              - f5745 - f5746 - f5747 - f5748 = 0
NDC115: f5736 + f5746 + f5756 + f5776 + f5786 - f5763
              - f5764 - f5765 - f5767 - f5768 = 0
NDC116: f5738 + f5748 + f5758 + f5768 + f5778 - f5783
              - f5784 - f5785 - f5786 - f5787 = 0
NDC117: f5843 + f5853 + f5863 + f5873 + f5883 - f5834
              - f5835 - f5836 - f5837 - f5838 = 0
NDC118: f5834 + f5854 + f5864 + f5874 + f5884 - f5843
              - f5845 - f5846 - f5847 - f5848 = 0
NDC119: f5836 + f5846 + f5856 + f5876 + f5886 - f5863
              - f5864 - f5865 - f5867 - f5868 = 0
NDC120: f5837 + f5847 + f5857 + f5867 + f5887 - f5873
              - f5874 - f5875 - f5876 - f5878 = 0
NDC121: f6334 + f6354 + f6364 + f6374 + f6384 - f6343
              - f6345 - f6346 - f6347 - f6348 = 0
NDC122: f6335 + f6345 + f6365 + f6375 + f6385 - f6353
              - f6354 - f6356 - f6357 - f6358 = 0
NDC123: f6337 + f6347 + f6357 + f6367 + f6387 - f6373
              - f6374 - f6375 - f6376 - f6378 = 0
NDC124: f6338 + f6348 + f6358 + f6368 + f6378 - f6383
              - f6384 - f6385 - f6386 - f6387 = 0
NDC125: f6443 + f6453 + f6463 + f6473 + f6483 - f6434
              - f6435 - f6436 - f6437 - f6438 = 0
NDC126: f6435 + f6445 + f6465 + f6475 + f6485 - f6453
              - f6454 - f6456 - f6457 - f6458 = 0
NDC127: f6437 + f6447 + f6457 + f6467 + f6487 - f6473
              - f6474 - f6475 - f6476 - f6478 = 0
NDC128: f6438 + f6448 + f6458 + f6468 + f6478 - f6483
              - f6484 - f6485 - f6486 - f6487 = 0
NDC129: f6543 + f6553 + f6563 + f6573 + f6583 - f6534
              - f6535 - f6536 - f6537 - f6538 = 0
NDC130: f6534 + f6554 + f6564 + f6574 + f6584 - f6543
              - f6545 - f6546 - f6547 - f6548 = 0
NDC131: f6537 + f6547 + f6557 + f6567 + f6587 - f6573
              - f6574 - f6575 - f6576 - f6578 = 0
NDC132: f6538 + f6548 + f6558 + f6568 + f6578 - f6583
              - f6584 - f6585 - f6586 - f6587 = 0
NDC133: f6743 + f6753 + f6763 + f6773 + f6783 - f6734
              - f6735 - f6736 - f6737 - f6738 = 0
NDC134: f6734 + f6754 + f6764 + f6774 + f6784 - f6743
              - f6745 - f6746 - f6747 - f6748 = 0
NDC135: f6735 + f6745 + f6765 + f6775 + f6785 - f6753
              - f6754 - f6756 - f6757 - f6758 = 0
NDC136: f6738 + f6748 + f6758 + f6768 + f6778 - f6783
              - f6784 - f6785 - f6786 - f6787 = 0
NDC137: f6843 + f6853 + f6863 + f6873 + f6883 - f6834
              - f6835 - f6836 - f6837 - f6838 = 0
NDC138: f6834 + f6854 + f6864 + f6874 + f6884 - f6843
```

```
                - f6845 - f6846 - f6847 - f6848 = 0
NDC139: f6835 + f6845 + f6865 + f6875 + f6885 - f6853
                - f6854 - f6856 - f6857 - f6858 = 0
NDC140: f6837 + f6847 + f6857 + f6867 + f6887 - f6873
                - f6874 - f6875 - f6876 - f6878 = 0
NDC141: f7334 + f7354 + f7364 + f7374 + f7384 - f7343
                - f7345 - f7346 - f7347 - f7348 = 0
NDC142: f7335 + f7345 + f7365 + f7375 + f7385 - f7353
                - f7354 - f7356 - f7357 - f7358 = 0
NDC143: f7336 + f7346 + f7356 + f7376 + f7386 - f7363
                - f7364 - f7365 - f7367 - f7368 = 0
NDC144: f7338 + f7348 + f7358 + f7368 + f7378 - f7383
                - f7384 - f7385 - f7386 - f7387 = 0
NDC145: f7443 + f7453 + f7463 + f7473 + f7483 - f7434
                - f7435 - f7436 - f7437 - f7438 = 0
NDC146: f7435 + f7445 + f7465 + f7475 + f7485 - f7453
                - f7454 - f7456 - f7457 - f7458 = 0
NDC147: f7436 + f7446 + f7456 + f7476 + f7486 - f7463
                - f7464 - f7465 - f7467 - f7468 = 0
NDC148: f7438 + f7448 + f7458 + f7468 + f7478 - f7483
                - f7484 - f7485 - f7486 - f7487 = 0
NDC149: f7543 + f7553 + f7563 + f7573 + f7583 - f7534
                - f7535 - f7536 - f7537 - f7538 = 0
NDC150: f7534 + f7554 + f7564 + f7574 + f7584 - f7543
                - f7545 - f7546 - f7547 - f7548 = 0
NDC151: f7536 + f7546 + f7556 + f7576 + f7586 - f7563
                - f7564 - f7565 - f7567 - f7568 = 0
NDC152: f7538 + f7548 + f7558 + f7568 + f7578 - f7583
                - f7584 - f7585 - f7586 - f7587 = 0
NDC153: f7643 + f7653 + f7663 + f7673 + f7683 - f7634
                - f7635 - f7636 - f7637 - f7638 = 0
NDC154: f7634 + f7654 + f7664 + f7674 + f7684 - f7643
                - f7645 - f7646 - f7647 - f7648 = 0
NDC155: f7635 + f7645 + f7665 + f7675 + f7685 - f7653
                - f7654 - f7656 - f7657 - f7658 = 0
NDC156: f7638 + f7648 + f7658 + f7668 + f7678 - f7683
                - f7684 - f7685 - f7686 - f7687 = 0
NDC157: f7843 + f7853 + f7863 + f7873 + f7883 - f7834
                - f7835 - f7836 - f7837 - f7838 = 0
NDC158: f7834 + f7854 + f7864 + f7874 + f7884 - f7843
                - f7845 - f7846 - f7847 - f7848 = 0
NDC159: f7835 + f7845 + f7865 + f7875 + f7885 - f7853
                - f7854 - f7856 - f7857 - f7858 = 0
NDC160: f7836 + f7846 + f7856 + f7876 + f7886 - f7863
                - f7864 - f7865 - f7867 - f7868 = 0
NDC161: f8334 + f8354 + f8364 + f8374 + f8384 - f8343
                - f8345 - f8346 - f8347 - f8348 = 0
NDC162: f8335 + f8345 + f8365 + f8375 + f8385 - f8353
                - f8354 - f8356 - f8357 - f8358 = 0
NDC163: f8336 + f8346 + f8356 + f8376 + f8386 - f8363
                - f8364 - f8365 - f8367 - f8368 = 0
NDC164: f8337 + f8347 + f8357 + f8367 + f8387 - f8373
                - f8374 - f8375 - f8376 - f8378 = 0
NDC165: f8443 + f8453 + f8463 + f8473 + f8483 - f8434
                - f8435 - f8436 - f8437 - f8438 = 0
NDC166: f8435 + f8445 + f8465 + f8475 + f8485 - f8453
                - f8454 - f8456 - f8457 - f8458 = 0
```

```
NDC167: f8436 + f8446 + f8456 + f8476 + f8486 - f8463
              - f8464 - f8465 - f8467 - f8468 = 0
NDC168: f8437 + f8447 + f8457 + f8467 + f8487 - f8473
              - f8474 - f8475 - f8476 - f8478 = 0
NDC169: f8543 + f8553 + f8563 + f8573 + f8583 - f8534
              - f8535 - f8536 - f8537 - f8538 = 0
NDC170: f8534 + f8554 + f8564 + f8574 + f8584 - f8543
              - f8545 - f8546 - f8547 - f8548 = 0
NDC171: f8536 + f8546 + f8556 + f8576 + f8586 - f8563
              - f8564 - f8565 - f8567 - f8568 = 0
NDC172: f8537 + f8547 + f8557 + f8567 + f8587 - f8573
              - f8574 - f8575 - f8576 - f8578 = 0
NDC173: f8643 + f8653 + f8663 + f8673 + f8683 - f8634
              - f8635 - f8636 - f8637 - f8638 = 0
NDC174: f8634 + f8654 + f8664 + f8674 + f8684 - f8643
              - f8645 - f8646 - f8647 - f8648 = 0
NDC175: f8635 + f8645 + f8665 + f8675 + f8685 - f8653
              - f8654 - f8656 - f8657 - f8658 = 0
NDC176: f8637 + f8647 + f8657 + f8667 + f8687 - f8673
              - f8674 - f8675 - f8676 - f8678 = 0
NDC177: f8743 + f8753 + f8763 + f8773 + f8783 - f8734
              - f8735 - f8736 - f8737 - f8738 = 0
NDC178: f8734 + f8754 + f8764 + f8774 + f8784 - f8743
              - f8745 - f8746 - f8747 - f8748 = 0
NDC179: f8735 + f8745 + f8765 + f8775 + f8785 - f8753
              - f8754 - f8756 - f8757 - f8758 = 0
NDC180: f8736 + f8746 + f8756 + f8776 + f8786 - f8763
              - f8764 - f8765 - f8767 - f8768 = 0
NDC181: f3434 + f3534 + f3634 + f3734 + f3834 + f4334
              + f4534 + f4634 + f4734 + f4834 + f5334 + f5434
              + f5634 + f5734 + f5834 + f6334 + f6434 + f6534
              + f6734 + f6834 + f7334 + f7434 + f7534 + f7634
              + f7834 + f8334 + f8434 + f8534 + f8634 + f8734
            - X030401T0 - X030401T1 = 0
NDC182: f3435 + f3535 + f3635 + f3735 + f3835 + f4335
              + f4535 + f4635 + f4735 + f4835 + f5335 + f5435
              + f5635 + f5735 + f5835 + f6335 + f6435 + f6535
              + f6735 + f6835 + f7335 + f7435 + f7535 + f7635
              + f7835 + f8335 + f8435 + f8535 + f8635 + f8735
            - X030501T0 - X030501T1 = 0
NDC183: f3436 + f3536 + f3636 + f3736 + f3836 + f4336
              + f4536 + f4636 + f4736 + f4836 + f5336 + f5436
              + f5636 + f5736 + f5836 + f6336 + f6436 + f6536
              + f6736 + f6836 + f7336 + f7436 + f7536 + f7636
              + f7836 + f8336 + f8436 + f8536 + f8636 + f8736
            - X030601T0 - X030601T1 = 0
NDC184: f3437 + f3537 + f3637 + f3737 + f3837 + f4337
              + f4537 + f4637 + f4737 + f4837 + f5337 + f5437
              + f5637 + f5737 + f5837 + f6337 + f6437 + f6537
              + f6737 + f6837 + f7337 + f7437 + f7537 + f7637
              + f7837 + f8337 + f8437 + f8537 + f8637 + f8737
            - X030701T0 - X030701T1 = 0
NDC185: f3438 + f3538 + f3638 + f3738 + f3838 + f4338
              + f4538 + f4638 + f4738 + f4838 + f5338 + f5438
              + f5638 + f5738 + f5838 + f6338 + f6438 + f6538
              + f6738 + f6838 + f7338 + f7438 + f7538 + f7638
              + f7838 + f8338 + f8438 + f8538 + f8638 + f8738
```

```
              - X030801T0 - X0308C1T1 = 0
NDC186: f3443 + f3543 + f3643 + f3743 + f3843 + f4343
              + f4543 + f4643 + f4743 + f4843 + f5343 + f5443
              + f5643 + f5743 + f5843 + f6343 + f6443 + f6543
              + f6743 + f6843 + f7343 + f7443 + f7543 + f7643
              + f7843 + f8343 + f8443 + f8543 + f8643 + f8743
              - X040301T0 - X040301T1 = 0
NDC187: f3445 + f3545 + f3645 + f3745 + f3845 + f4345
              + f4545 + f4645 + f4745 + f4845 + f5345 + f5445
              + f5645 + f5745 + f5845 + f6345 + f6445 + f6545
              + f6745 + f6845 + f7345 + f7445 + f7545 + f7645
              + f7845 + f8345 + f8445 + f8545 + f8645 + f8745
              - X040501T0 - X040501T1 = 0
NDC188: f3446 + f3546 + f3646 + f3746 + f3846 + f4346
              + f4546 + f4646 + f4746 + f4846 + f5346 + f5446
              + f5646 + f5746 + f5846 + f6346 + f6446 + f6546
              + f6746 + f6846 + f7346 + f7446 + f7546 + f7646
              + f7846 + f8346 + f8446 + f8546 + f8646 + f8746
              - X040601T0 - X040601T1 = 0
NDC189: f3447 + f3547 + f3647 + f3747 + f3847 + f4347
              + f4547 + f4647 + f4747 + f4847 + f5347 + f5447
              + f5647 + f5747 + f5847 + f6347 + f6447 + f6547
              + f6747 + f6847 + f7347 + f7447 + f7547 + f7647
              + f7847 + f8347 + f8447 + f8547 + f8647 + f8747
              - X040701T0 - X040701T1 = 0
NDC190: f3448 + f3548 + f3648 + f3748 + f3848 + f4348
              + f4548 + f4648 + f4748 + f4848 + f5348 + f5448
              + f5648 + f5748 + f5848 + f6348 + f6448 + f6548
              + f6748 + f6848 + f7348 + f7448 + f7548 + f7648
              + f7848 + f8348 + f8448 + f8548 + f8648 + f8748
              - X040801T0 - X040801T1 = 0
NDC191: f3453 + f3553 + f3653 + f3753 + f3853 + f4353
              + f4553 + f4653 + f4753 + f4853 + f5353 + f5453
              + f5653 + f5753 + f5853 + f6353 + f6453 + f6553
              + f6753 + f6853 + f7353 + f7453 + f7553 + f7653
              + f7853 + f8353 + f8453 + f8553 + f8653 + f8753
              - X050301T0 - X050301T1 = 0
NDC192: f3454 + f3554 + f3654 + f3754 + f3854 + f4354
              + f4554 + f4654 + f4754 + f4854 + f5354 + f5454
              + f5654 + f5754 + f5854 + f6354 + f6454 + f6554
              + f6754 + f6854 + f7354 + f7454 + f7554 + f7654
              + f7854 + f8354 + f8454 + f8554 + f8654 + f8754
              - X050401T0 - X050401T1 = 0
NDC193: f3456 + f3556 + f3656 + f3756 + f3856 + f4356
              + f4556 + f4656 + f4756 + f4856 + f5356 + f5456
              + f5656 + f5756 + f5856 + f6356 + f6456 + f6556
              + f6756 + f6856 + f7356 + f7456 + f7556 + f7656
              + f7856 + f8356 + f8456 + f8556 + f8656 + f8756
              - X050601T0 - X050601T1 = 0
NDC194: f3457 + f3557 + f3657 + f3757 + f3857 + f4357
              + f4557 + f4657 + f4757 + f4857 + f5357 + f5457
              + f5657 + f5757 + f5857 + f6357 + f6457 + f6557
              + f6757 + f6857 + f7357 + f7457 + f7557 + f7657
              + f7857 + f8357 + f8457 + f8557 + f8657 + f8757
              - X050701T0 - X050701T1 = 0
NDC195: f3458 + f3558 + f3658 + f3758 + f3858 + f4358
              + f4558 + f4658 + f4758 + f4858 + f5358 + f5458
```

```
              + f5658 + f5758 + f5858 + f6358 + f6458 + f6558
              + f6758 + f6858 + f7358 + f7458 + f7558 + f7658
              + f7858 + f8358 + f8458 + f8558 + f8658 + f8758
             - X050801T0 - X050801T1 = 0
NDC196: f3463 + f3563 + f3663 + f3763 + f3863 + f4363
              + f4563 + f4663 + f4763 + f4863 + f5363 + f5463
              + f5663 + f5763 + f5863 + f6363 + f6463 + f6563
              + f6763 + f6863 + f7363 + f7463 + f7563 + f7663
              + f7863 + f8363 + f8463 + f8563 + f8663 + f8763
             - X060301T0 - X060301T1 = 0
NDC197: f3464 + f3564 + f3664 + f3764 + f3864 + f4364
              + f4564 + f4664 + f4764 + f4864 + f5364 + f5464
              + f5664 + f5764 + f5864 + f6364 + f6464 + f6564
              + f6764 + f6864 + f7364 + f7464 + f7564 + f7664
              + f7864 + f8364 + f8464 + f8564 + f8664 + f8764
             - X060401T0 - X060401T1 = 0
NDC198: f3465 + f3565 + f3665 + f3765 + f3865 + f4365
              + f4565 + f4665 + f4765 + f4865 + f5365 + f5465
              + f5665 + f5765 + f5865 + f6365 + f6465 + f6565
              + f6765 + f6865 + f7365 + f7465 + f7565 + f7665
              + f7865 + f8365 + f8465 + f8565 + f8665 + f8765
             - X060501T0 - X060501T1 = 0
NDC199: f3467 + f3567 + f3667 + f3767 + f3867 + f4367
              + f4567 + f4667 + f4767 + f4867 + f5367 + f5467
              + f5667 + f5767 + f5867 + f6367 + f6467 + f6567
              + f6767 + f6867 + f7367 + f7467 + f7567 + f7667
              + f7867 + f8367 + f8467 + f8567 + f8667 + f8767
             - X060701T0 - X060701T1 = 0
NDC200: f3468 + f3568 + f3668 + f3768 + f3868 + f4368
              + f4568 + f4668 + f4768 + f4868 + f5368 + f5468
              + f5668 + f5768 + f5868 + f6368 + f6468 + f6568
              + f6768 + f6868 + f7368 + f7468 + f7568 + f7668
              + f7868 + f8368 + f8468 + f8568 + f8668 + f8768
             - X060801T0 - X060801T1 = 0
NDC201: f3473 + f3573 + f3673 + f3773 + f3873 + f4373
              + f4573 + f4673 + f4773 + f4873 + f5373 + f5473
              + f5673 + f5773 + f5873 + f6373 + f6473 + f6573
              + f6773 + f6873 + f7373 + f7473 + f7573 + f7673
              + f7873 + f8373 + f8473 + f8573 + f8673 + f8773
             - X070301T0 - X070301T1 = 0
NDC202: f3474 + f3574 + f3674 + f3774 + f3874 + f4374
              + f4574 + f4674 + f4774 + f4874 + f5374 + f5474
              + f5674 + f5774 + f5874 + f6374 + f6474 + f6574
              + f6774 + f6874 + f7374 + f7474 + f7574 + f7674
              + f7874 + f8374 + f8474 + f8574 + f8674 + f8774
             - X070401T0 - X070401T1 = 0
NDC203: f3475 + f3575 + f3675 + f3775 + f3875 + f4375
              + f4575 + f4675 + f4775 + f4875 + f5375 + f5475
              + f5675 + f5775 + f5875 + f6375 + f6475 + f6575
              + f6775 + f6875 + f7375 + f7475 + f7575 + f7675
              + f7875 + f8375 + f8475 + f8575 + f8675 + f8775
             - X070501T0 - X070501T1 = 0
NDC204: f3476 + f3576 + f3676 + f3776 + f3876 + f4376
              + f4576 + f4676 + f4776 + f4876 + f5376 + f5476
              + f5676 + f5776 + f5876 + f6376 + f6476 + f6576
              + f6776 + f6876 + f7376 + f7476 + f7576 + f7676
              + f7876 + f8376 + f8476 + f8576 + f8676 + f8776
```

```
              - X070601T0 - X070601T1 = 0
NDC205: f3478 + f3578 + f3678 + f3778 + f3878 + f4378
              + f4578 + f4678 + f4778 + f4878 + f5378 + f5478
              + f5678 + f5778 + f5878 + f6378 + f6478 + f6578
              + f6778 + f6878 + f7378 + f7478 + f7578 + f7678
              + f7878 + f8378 + f8478 + f8578 + f8678 + f8778
             - X070801T0 - X070801T1 = 0
NDC206: f3483 + f3583 + f3683 + f3783 + f3883 + f4383
              + f4583 + f4683 + f4783 + f4883 + f5383 + f5483
              + f5683 + f5783 + f5883 + f6383 + f6483 + f6583
              + f6783 + f6883 + f7383 + f7483 + f7583 + f7683
              + f7883 + f8383 + f8483 + f8583 + f8683 + f8783
             - X080301T0 - X080301T1 = 0
NDC207: f3484 + f3584 + f3684 + f3784 + f3884 + f4384
              + f4584 + f4684 + f4784 + f4884 + f5384 + f5484
              + f5684 + f5784 + f5884 + f6384 + f6484 + f6584
              + f6784 + f6884 + f7384 + f7484 + f7584 + f7684
              + f7884 + f8384 + f8484 + f8584 + f8684 + f8784
             - X080401T0 - X080401T1 = 0
NDC208: f3485 + f3585 + f3685 + f3785 + f3885 + f4385
              + f4585 + f4685 + f4785 + f4885 + f5385 + f5485
              + f5685 + f5785 + f5885 + f6385 + f6485 + f6585
              + f6785 + f6885 + f7385 + f7485 + f7585 + f7685
              + f7885 + f8385 + f8485 + f8585 + f8685 + f8785
             - X080501T0 - X080501T1 = 0
NDC209: f3486 + f3586 + f3686 + f3786 + f3886 + f4386
              + f4586 + f4686 + f4786 + f4886 + f5386 + f5486
              + f5686 + f5786 + f5886 + f6386 + f6486 + f6586
              + f6786 + f6886 + f7386 + f7486 + f7586 + f7686
              + f7886 + f8386 + f8486 + f8586 + f8686 + f8786
             - X080601T0 - X080601T1 = 0
NDC210: f3487 + f3587 + f3687 + f3787 + f3887 + f4387
              + f4587 + f4687 + f4787 + f4887 + f5387 + f5487
              + f5687 + f5787 + f5887 + f6387 + f6487 + f6587
              + f6787 + f6887 + f7387 + f7487 + f7587 + f7687
              + f7887 + f8387 + f8487 + f8587 + f8687 + f8787
             - X080701T0 - X080701T1 = 0
Node03 - y00302 >= 0
Node03 - y10302 >= 0
Node03 - y00304 >= 0
Node03 - y10304 >= 0
Node03 - y00305 >= 0
Node03 - y10305 >= 0
Node03 - y00306 >= 0
Node03 - y10306 >= 0
Node03 - y00307 >= 0
Node03 - y10307 >= 0
Node03 - y00308 >= 0
Node03 - y10308 >= 0
Node04 - y00403 >= 0
Node04 - y10403 >= 0
Node04 - y00402 >= 0
Node04 - y10402 >= 0
Node04 - y00405 >= 0
Node04 - y10405 >= 0
Node04 - y00406 >= 0
Node04 - y10406 >= 0
```

```
Node04 - y00407 >= 0
Node04 - y10407 >= 0
Node04 - y00408 >= 0
Node04 - y10408 >= 0
Node05 - y00503 >= 0
Node05 - y10503 >= 0
Node05 - y00504 >= 0
Node05 - y10504 >= 0
Node05 - y00502 >= 0
Node05 - y10502 >= 0
Node05 - y00506 >= 0
Node05 - y10506 >= 0
Node05 - y00507 >= 0
Node05 - y10507 >= 0
Node05 - y00508 >= 0
Node05 - y10508 >= 0
Node06 - y00603 >= 0
Node06 - y10603 >= 0
Node06 - y00604 >= 0
Node06 - y10604 >= 0
Node06 - y00605 >= 0
Node06 - y10605 >= 0
Node06 - y00602 >= 0
Node06 - y10602 >= 0
Node06 - y00607 >= 0
Node06 - y10607 >= 0
Node06 - y00608 >= 0
Node06 - y10608 >= 0
Node07 - y00703 >= 0
Node07 - y10703 >= 0
Node07 - y00704 >= 0
Node07 - y10704 >= 0
Node07 - y00705 >= 0
Node07 - y10705 >= 0
Node07 - y00706 >= 0
Node07 - y10706 >= 0
Node07 - y00702 >= 0
Node07 - y10702 >= 0
Node07 - y00708 >= 0
Node07 - y10708 >= 0
Node08 - y00803 >= 0
Node08 - y10803 >= 0
Node08 - y00804 >= 0
Node08 - y10804 >= 0
Node08 - y00805 >= 0
Node08 - y10805 >= 0
Node08 - y00806 >= 0
Node08 - y10806 >= 0
Node08 - y00807 >= 0
Node08 - y10807 >= 0
Node08 - y00802 >= 0
Node08 - y10802 >= 0
y00304 - y00403 = 0
y10304 - y10403 = 0
y00305 - y00503 = 0
y10305 - y10503 = 0
y00306 - y00603 = 0
```

```
y10306 - y10603 = 0
y00307 - y00703 = 0
y10307 - y10703 = 0
y00308 - y00803 = 0
y10308 - y10803 = 0
y00405 - y00504 = 0
y10405 - y10504 = 0
y00406 - y00604 = 0
y10406 - y10604 = 0
y00407 - y00704 = 0
y10407 - y10704 = 0
y00408 - y00804 = 0
y10408 - y10804 = 0
y00506 - y00605 = 0
y10506 - y10605 = 0
y00507 - y00705 = 0
y10507 - y10705 = 0
y00508 - y00805 = 0
y10508 - y10805 = 0
y00607 - y00706 = 0
y10607 - y10706 = 0
y00608 - y00806 = 0
y10608 - y10806 = 0
y00708 - y00807 = 0
y10708 - y10807 = 0
LCON0304: y00304 + y10304 - q0304 >= 0
LCON0305: y00305 + y10305 - q0305 >= 0
LCON0306: y00306 + y10306 - q0306 >= 0
LCON0307: y00307 + y10307 - q0307 >= 0
LCON0308: y00308 + y10308 - q0308 >= 0
LCON0405: y00405 + y10405 - q0405 >= 0
LCON0406: y00406 + y10406 - q0406 >= 0
LCON0407: y00407 + y10407 - q0407 >= 0
LCON0408: y00408 + y10408 - q0408 >= 0
LCON0506: y00506 + y10506 - q0506 >= 0
LCON0507: y00507 + y10507 - q0507 >= 0
LCON0508: y00508 + y10508 - q0508 >= 0
LCON0607: y00607 + y10607 - q0607 >= 0
LCON0608: y00608 + y10608 - q0608 >= 0
LCON0708: y00708 + y10708 - q0708 >= 0
C01: q0304 + q0305 + q0306 + q0307 + q0308 >= 2
C02: q0403 + q0405 + q0406 + q0407 + q0408 >=1
C03: q0503 + q0504 + q0506 + q0507 + q0508 >=1
C04: q0603 + q0604 + q0605 + q0607 + q0608 >=1
C05: q0703 + q0704 + q0705 + q0706 + q0708 >=1
C06: q0803 + q0804 + q0805 + q0806 + q0807 >= 2
C07: q0305 + q0405 + q0306 + q0406 + q0307 + q0407 + q0308 + q0408 >= 2
C08: q0304 + q0504 + q0306 + q0506 + q0307 + q0507 + q0308 + q0508 >= 2
C09: q0304 + q0604 + q0305 + q0605 + q0307 + q0607 + q0308 + q0608 >= 2
C10: q0304 + q0704 + q0305 + q0705 + q0306 + q0706 + q0308 + q0708 >= 2
C11: q0304 + q0804 + q0305 + q0805 + q0306 + q0806 + q0307 + q0807 >= 1
C12: q0403 + q0503 + q0406 + q0506 + q0407 + q0507 + q0408 + q0508 >= 1
C13: q0403 + q0603 + q0405 + q0605 + q0407 + q0607 + q0408 + q0608 >= 1
C14: q0403 + q0703 + q0405 + q0705 + q0406 + q0706 + q0408 + q0708 >= 1
C15: q0403 + q0803 + q0405 + q0805 + q0406 + q0806 + q0407 + q0807 >= 2
C16: q0503 + q0603 + q0504 + q0604 + q0507 + q0607 + q0508 + q0608 >= 1
C17: q0503 + q0703 + q0504 + q0704 + q0506 + q0706 + q0508 + q0708 >= 1
```

```
C18: q0503 + q0803 + q0504 + q0804 + q0506 + q0806 + q0507 + q0807 >= 2
C19: q0603 + q0703 + q0604 + q0704 + q0605 + q0705 + q0608 + q0708 >= 1
C20: q0603 + q0803 + q0604 + q0804 + q0605 + q0805 + q0607 + q0807 >= 2
C21: q0703 + q0803 + q0704 + q0804 + q0705 + q0805 + q0706 + q0806 >= 2
C22: q0306 + q0406 + q0506 + q0307 + q0407 + q0507 + q0308 + q0408 + q0508 >= 2
C23: q0305 + q0405 + q0605 + q0307 + q0407 + q0607 + q0308 + q0408 + q0608 >= 2
C24: q0305 + q0405 + q0705 + q0306 + q0406 + q0706 + q0308 + q0408 + q0708 >= 2
C25: q0305 + q0405 + q0805 + q0306 + q0406 + q0806 + q0307 + q0407 + q0807 >= 2
C26: q0304 + q0504 + q0604 + q0307 + q0507 + q0607 + q0308 + q0508 + q0608 >= 2
C27: q0304 + q0504 + q0704 + q0306 + q0506 + q0706 + q0308 + q0508 + q0708 >= 2
C28: q0304 + q0504 + q0804 + q0306 + q0506 + q0806 + q0307 + q0507 + q0807 >= 2
C29: q0304 + q0604 + q0704 + q0305 + q0605 + q0705 + q0308 + q0608 + q0708 >= 2
C30: q0304 + q0604 + q0804 + q0305 + q0605 + q0805 + q0307 + q0607 + q0807 >= 2
C31: q0304 + q0704 + q0804 + q0305 + q0705 + q0805 + q0306 + q0706 + q0806 >= 2
C32: q0403 + q0503 + q0603 + q0407 + q0507 + q0607 + q0408 + q0508 + q0608 >= 1
C33: q0403 + q0503 + q0703 + q0406 + q0506 + q0706 + q0408 + q0508 + q0708 >= 1
C34: q0403 + q0503 + q0803 + q0406 + q0506 + q0806 + q0407 + q0507 + q0807 >= 1
C35: q0403 + q0603 + q0703 + q0405 + q0605 + q0705 + q0408 + q0608 + q0708 >= 1
C36: q0403 + q0603 + q0803 + q0405 + q0605 + q0805 + q0407 + q0607 + q0807 >= 1
C37: q0403 + q0703 + q0803 + q0405 + q0705 + q0805 + q0406 + q0706 + q0806 >= 1
C38: q0503 + q0603 + q0703 + q0504 + q0604 + q0704 + q0508 + q0608 + q0708 >= 1
C39: q0503 + q0603 + q0803 + q0504 + q0604 + q0804 + q0507 + q0607 + q0807 >= 1
C40: q0503 + q0703 + q0803 + q0504 + q0704 + q0804 + q0506 + q0706 + q0806 >= 1
C41: q0603 + q0703 + q0803 + q0604 + q0704 + q0804 + q0605 + q0705 + q0805 >= 1
C42: q0307 + q0407 + q0507 + q0607 + q0308 + q0408 + q0508 + q0608 >= 2
C43: q0306 + q0406 + q0506 + q0706 + q0308 + q0408 + q0508 + q0708 >= 2
C44: q0306 + q0406 + q0506 + q0806 + q0307 + q0407 + q0507 + q0807 >= 2
C45: q0305 + q0405 + q0605 + q0705 + q0308 + q0408 + q0608 + q0708 >= 2
C46: q0305 + q0405 + q0605 + q0805 + q0307 + q0407 + q0607 + q0807 >= 2
C47: q0305 + q0405 + q0705 + q0805 + q0306 + q0406 + q0706 + q0806 >= 2
C48: q0304 + q0504 + q0604 + q0704 + q0308 + q0508 + q0608 + q0708 >= 2
C49: q0304 + q0504 + q0604 + q0804 + q0307 + q0507 + q0607 + q0807 >= 2
C50: q0304 + q0504 + q0704 + q0804 + q0306 + q0506 + q0706 + q0806 >= 2
C51: q0304 + q0604 + q0704 + q0804 + q0305 + q0605 + q0705 + q0805 >= 2
C52: q0403 + q0503 + q0603 + q0703 + q0408 + q0508 + q0608 + q0708 >= 1
C53: q0403 + q0503 + q0603 + q0803 + q0407 + q0507 + q0607 + q0807 >= 1
C54: q0403 + q0503 + q0703 + q0803 + q0406 + q0506 + q0706 + q0806 >= 1
C55: q0403 + q0603 + q0703 + q0803 + q0405 + q0605 + q0705 + q0805 >= 1
C56: q0503 + q0603 + q0703 + q0803 + q0504 + q0604 + q0704 + q0804 >= 1
N4C01: q0306 + q0506 + q0307 + q0507 + q0308 + q0508 >= 1
N4C02: q0305 + q0605 + q0307 + q0607 + q0308 + q0608 >= 1
N4C03: q0305 + q0705 + q0306 + q0706 + q0308 + q0708 >= 1
N4C04: q0803 + q0503 + q0806 + q0506 + q0807 + q0507 >= 1
N4C05: q0803 + q0603 + q0805 + q0605 + q0807 + q0607 >= 1
N4C06: q0803 + q0703 + q0805 + q0705 + q0806 + q0706 >= 1
N4C07: q0307 + q0507 + q0607 + q0308 + q0508 + q0608 >= 1
N4C08: q0306 + q0506 + q0706 + q0308 + q0508 + q0708 >= 1
N4C09: q0305 + q0605 + q0705 + q0308 + q0608 + q0708 >= 1
N4C10: q0803 + q0503 + q0603 + q0807 + q0507 + q0607 >= 1
N4C11: q0803 + q0503 + q0703 + q0806 + q0506 + q0706 >= 1
N4C12: q0803 + q0603 + q0703 + q0805 + q0605 + q0705 >= 1
N4C13: q0308 + q0508 + q0608 + q0708 >= 1
N4C14: q0803 + q0503 + q0603 + q0703 >= 1
N5C01: q0306 + q0406 + q0307 + q0407 + q0308 + q0408 >= 1
N5C02: q0304 + q0604 + q0307 + q0607 + q0308 + q0608 >= 1
N5C03: q0304 + q0704 + q0306 + q0706 + q0308 + q0708 >= 1
N5C04: q0803 + q0403 + q0806 + q0406 + q0807 + q0407 >= 1
```

```
 N5C05: q0803 + q0603 + q0804 + q0604 + q0807 + q0607 >= 1
 N5C06: q0803 + q0703 + q0804 + q0704 + q0806 + q0706 >= 1
 N5C07: q0307 + q0407 + q0607 + q0308 + q0408 + q0608 >= 1
 N5C08: q0306 + q0406 + q0706 + q0308 + q0408 + q0708 >= 1
 N5C09: q0304 + q0604 + q0704 + q0308 + q0608 + q0708 >= 1
 N5C10: q0803 + q0403 + q0603 + q0807 + q0407 + q0607 >= 1
 N5C11: q0803 + q0403 + q0703 + q0806 + q0406 + q0706 >= 1
 N5C12: q0803 + q0603 + q0703 + q0804 + q0604 + q0704 >= 1
 N5C13: q0308 + q0408 + q0608 + q0708 >= 1
 N5C14: q0803 + q0403 + q0603 + q0703 >= 1
 N6C01: q0305 + q0405 + q0307 + q0407 + q0308 + q0408 >= 1
 N6C02: q0304 + q0504 + q0307 + q0507 + q0308 + q0508 >= 1
 N6C03: q0304 + q0704 + q0305 + q0705 + q0308 + q0708 >= 1
 N6C04: q0803 + q0403 + q0805 + q0405 + q0807 + q0407 >= 1
 N6C05: q0803 + q0503 + q0804 + q0504 + q0807 + q0507 >= 1
 N6C06: q0803 + q0703 + q0804 + q0704 + q0805 + q0705 >= 1
 N6C07: q0307 + q0407 + q0507 + q0308 + q0408 + q0508 >= 1
 N6C08: q0305 + q0405 + q0705 + q0308 + q0408 + q0708 >= 1
 N6C09: q0304 + q0504 + q0704 + q0308 + q0508 + q0708 >= 1
 N6C10: q0803 + q0403 + q0503 + q0807 + q0407 + q0507 >= 1
 N6C11: q0803 + q0403 + q0703 + q0805 + q0405 + q0705 >= 1
 N6C12: q0803 + q0503 + q0703 + q0804 + q0504 + q0704 >= 1
 N6C13: q0308 + q0408 + q0508 + q0708 >= 1
 N6C14: q0803 + q0403 + q0503 + q0703 >= 1
 N7C01: q0305 + q0405 + q0306 + q0406 + q0308 + q0408 >= 1
 N7C02: q0304 + q0504 + q0306 + q0506 + q0308 + q0508 >= 1
 N7C03: q0304 + q0604 + q0305 + q0605 + q0308 + q0608 >= 1
 N7C04: q0803 + q0403 + q0805 + q0405 + q0806 + q0406 >= 1
 N7C05: q0803 + q0503 + q0804 + q0504 + q0806 + q0506 >= 1
 N7C06: q0803 + q0603 + q0804 + q0604 + q0805 + q0605 >= 1
 N7C07: q0306 + q0406 + q0506 + q0308 + q0408 + q0508 >= 1
 N7C08: q0305 + q0405 + q0605 + q0308 + q0408 + q0608 >= 1
 N7C09: q0304 + q0504 + q0604 + q0308 + q0508 + q0608 >= 1
 N7C10: q0803 + q0403 + q0503 + q0806 + q0406 + q0506 >= 1
 N7C11: q0803 + q0403 + q0603 + q0805 + q0405 + q0605 >= 1
 N7C12: q0803 + q0503 + q0603 + q0804 + q0504 + q0604 >= 1
 N7C13: q0308 + q0408 + q0508 + q0608 >= 1
 N7C14: q0803 + q0403 + q0503 + q0603 >= 1
 q0304 - q0403 = 0
 q0305 - q0503 = 0
 q0306 - q0603 = 0
 q0307 - q0703 = 0
 q0308 - q0803 = 0
 q0405 - q0504 = 0
 q0406 - q0604 = 0
 q0407 - q0704 = 0
 q0408 - q0804 = 0
 q0506 - q0605 = 0
 q0507 - q0705 = 0
 q0508 - q0805 = 0
 q0607 - q0706 = 0
 q0608 - q0806 = 0
 q0708 - q0807 = 0
Bounds
 SRC = 1
 DST = 1
 0<= q0304 <= 1
```

```
0<= q0305 <= 1
0<= q0306 <= 1
0<= q0307 <= 1
0<= q0308 <= 1
0<= q0403 <= 1
0<= q0405 <= 1
0<= q0406 <= 1
0<= q0407 <= 1
0<= q0408 <= 1
0<= q0503 <= 1
0<= q0504 <= 1
0<= q0506 <= 1
0<= q0507 <= 1
0<= q0508 <= 1
0<= q0603 <= 1
0<= q0604 <= 1
0<= q0605 <= 1
0<= q0607 <= 1
0<= q0608 <= 1
0<= q0703 <= 1
0<= q0704 <= 1
0<= q0705 <= 1
0<= q0706 <= 1
0<= q0708 <= 1
0<= q0803 <= 1
0<= q0804 <= 1
0<= q0805 <= 1
0<= q0806 <= 1
0<= q0807 <= 1
0 <= y00103 <=1
0 <= y10103 <=1
0 <= y00104 <=1
0 <= y10104 <=1
0 <= y00105 <=1
0 <= y10105 <=1
0 <= y00106 <=1
0 <= y10106 <=1
0 <= y00107 <=1
0 <= y10107 <=1
0 <= y00108 <=1
0 <= y10108 <=1
0 <= y00302 <=1
0 <= y10302 <=1
0 <= y00304 <=1
0 <= y10304 <=1
0 <= y00305 <=1
0 <= y10305 <=1
0 <= y00306 <=1
0 <= y10306 <=1
0 <= y00307 <=1
0 <= y10307 <=1
0 <= y00308 <=1
0 <= y10308 <=1
0 <= y00403 <=1
0 <= y10403 <=1
0 <= y00402 <=1
0 <= y10402 <=1
```

```
 0 <= y00405 <=1
 0 <= y10405 <=1
 0 <= y00406 <=1
 0 <= y10406 <=1
 0 <= y00407 <=1
 0 <= y10407 <=1
 0 <= y00408 <=1
 0 <= y10408 <=1
 0 <= y00503 <=1
 0 <= y10503 <=1
 0 <= y00504 <=1
 0 <= y10504 <=1
 0 <= y00502 <=1
 0 <= y10502 <=1
 0 <= y00506 <=1
 0 <= y10506 <=1
 0 <= y00507 <=1
 0 <= y10507 <=1
 0 <= y00508 <=1
 0 <= y10508 <=1
 0 <= y00603 <=1
 0 <= y10603 <=1
 0 <= y00604 <=1
 0 <= y10604 <=1
 0 <= y00605 <=1
 0 <= y10605 <=1
 0 <= y00602 <=1
 0 <= y10602 <=1
 0 <= y00607 <=1
 0 <= y10607 <=1
 0 <= y00608 <=1
 0 <= y10608 <=1
 0 <= y00703 <=1
 0 <= y10703 <=1
 0 <= y00704 <=1
 0 <= y10704 <=1
 0 <= y00705 <=1
 0 <= y10705 <=1
 0 <= y00706 <=1
 0 <= y10706 <=1
 0 <= y00702 <=1
 0 <= y10702 <=1
 0 <= y00708 <=1
 0 <= y10708 <=1
 0 <= y00803 <=1
 0 <= y10803 <=1
 0 <= y00804 <=1
 0 <= y10804 <=1
 0 <= y00805 <=1
 0 <= y10805 <=1
 0 <= y00806 <=1
 0 <= y10806 <=1
 0 <= y00807 <=1
 0 <= y10807 <=1
 0 <= y00802 <=1
 0 <= y10802 <=1
Binaries
```

```
 q0304  q0305  q0306  q0307  q0308  q0403  q0405  q0406  q0407  q0408
 q0503  q0504  q0506  q0507  q0508  q0603  q0604  q0605  q0607  q0608
 q0703  q0704  q0705  q0706  q0708  q0803  q0804  q0805  q0806  q0807
 y00103 y10103 y00104 y10104 y00105 y10105 y00106 y10106 y00107 y10107
 y00108 y10108 y00302 y10302 y00304 y10304 y00305 y10305 y00306 y10306
 y00307 y10307 y00308 y10308 y00403 y10403 y00402 y10402 y00405 y10405
 y00406 y10406 y00407 y10407 y00408 y10408 y00503 y10503 y00504 y10504
 y00502 y10502 y00506 y10506 y00507 y10507 y00508 y10508 y00603 y10603
 y00604 y10604 y00605 y10605 y00602 y10602 y00607 y10607 y00608 y10608
 y00703 y10703 y00704 y10704 y00705 y10705 y00706 y10706 y00702 y10702
 y00708 y10708 y00803 y10803 y00804 y10804 y00805 y10805 y00806 y10806
 y00807 y10807 y00802 y10802 SRC DST Node03 Node04 Node05
 Node06 Node07 Node08
End
```

C.7 NetTest2ncon.sol solution

```
start
Tried aggregator 2 times.
MIP Presolve eliminated 111 rows and 206 columns.
Aggregator did 45 substitutions.
Reduced MIP has 348 rows, 855 columns, and 2811 nonzeros.
Presolve time =    0.01 sec.
Clique table members: 60
MIP emphasis: balance optimality and feasibility
Root relaxation solution time =    0.03 sec.

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Node    ItCnt     Gap

      0     0     1111.9887    15                 1111.9887      366
*     0+    0                   0     5058.4280     1111.9887      366   78.02%
                  1142.3701     9     5058.4280    Fract:  15      469   77.42%
                  1147.5783    10     5058.4280     Fract:  8      549   77.31%
                  1147.5783    10     5058.4280  Flowcuts:  1      552   77.31%
*    10+   10                   0     1269.0315     1152.3524      922    9.19%
*    34    15                   0     1249.4100     1165.3160     1733    6.73%
*    53    14                   0     1245.5482     1205.1853     2361    3.24%

Flow cuts applied:  1
Gomory fractional cuts applied: 3
Solution status 101.
Objective value 1245.54825
X010301T1       :              0.225
X010401T1       :              0.075
X010501T1       :                0.2
X010601T1       :              0.275
X010801T1       :              0.225
X030201T1       :                0.1
X030401T0       :              0.075
y00304          :                  1
X030401T1       :             0.2875
X040301T1       :             0.1125
y10304          :                  1
```

X030601T1	:	0.1375
X060301T1	:	0.2625
y10306	:	1
X040201T1	:	0.375
X040701T0	:	0.1
y00407	:	1
X040801T0	:	0.075
y00408	:	1
X040801T1	:	0.0875
X080401T1	:	0.3125
y10408	:	1
X050201T1	:	0.075
X050601T0	:	0.0375
X060501T0	:	0.0625
y00506	:	1
X050801T1	:	0.25
X080501T1	:	0.1
y10508	:	1
X060201T1	:	0.125
X070201T1	:	0.1
X080201T1	:	0.225
SRC	:	1
DST	:	1
Node03	:	1
Node04	:	1
Node05	:	1
Node06	:	1
Node07	:	1
Node08	:	1
f3434	:	0.1
f3834	:	0.125
f4343	:	0.05
f4643	:	0.025
f5458	:	0.1
f5656	:	0.0375
f5658	:	0.0125
f5758	:	0.025
f5858	:	0.025
f6363	:	0.025
f6463	:	0.1
f6563	:	0.0125
f6565	:	0.0625
f6863	:	0.075
f8384	:	0.025
f8484	:	0.075
f8684	:	0.0125
f8685	:	0.0375
f8784	:	0.075
f3848	:	0.125
f4636	:	0.025
f5484	:	0.1
f5636	:	0.0125
f5747	:	0.025
f6434	:	0.1
f6585	:	0.0125
f6848	:	0.0375
f6858	:	0.0375

f8343	:	0.025
f8636	:	0.05
f8747	:	0.075
f5663	:	0.0125
f5685	:	0.0125
f5784	:	0.025
f6536	:	0.0125
f6558	:	0.0125
f6834	:	0.0375
f6836	:	0.0375
f6885	:	0.0375
f8643	:	0.0125
f8663	:	0.0375
f8658	:	0.0375
y00403	:	1
y10403	:	1
y10603	:	1
y00605	:	1
y00704	:	1
y00804	:	1
y10804	:	1
y10805	:	1
q0304	:	1
q0306	:	1
q0407	:	1
q0408	:	1
q0506	:	1
q0508	:	1
q0403	:	1
q0603	:	1
q0605	:	1
q0704	:	1
q0804	:	1
q0805	:	1

D

SURVIVABLE NETWORK WIZARD TOOL

This section describes the functioning of the survivable network designer's wizard. This feature is designed to make it easy and fast to obtain network instances. In real life situations the function to alter the network can be employed to "build" the network problem.

D.1 Description of survivable network design tool

Figure D.1 is a graphical representation of the GUI for the network wizard. A detailed description of the screen layout is given below.

D.1.1 General Network Information

- **Network name,** this is the unique name assigned to the network. This is also the name under which the network will be saved.

- **Commodities,** this refers to the number of source/destination node pairs to be included in the design.

- **Node design cost.** This is the cost associated with the design of a node in the network. The same design cost is assigned to all the nodes in the network.

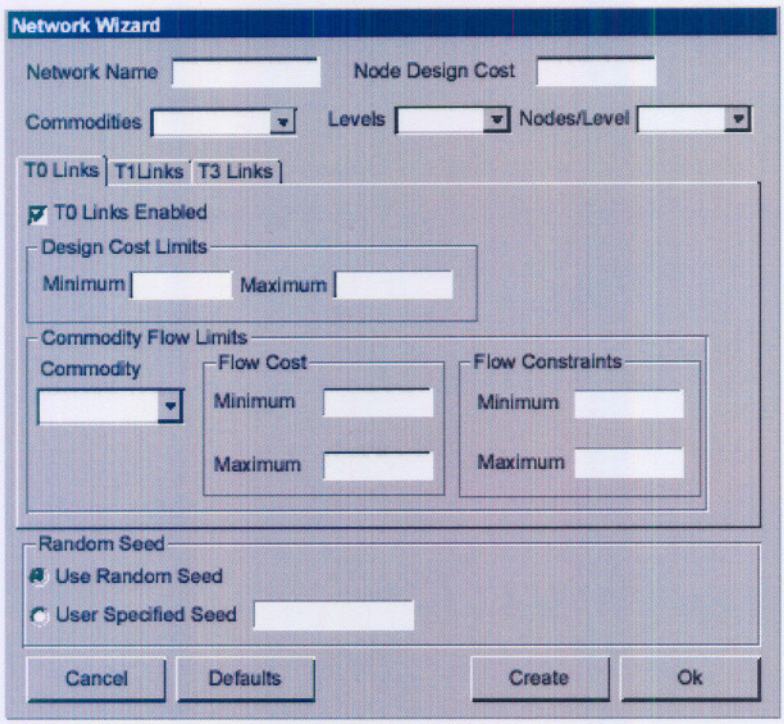


Figure D.1 Graphical representation of network designer wizard

- **Levels.** The number of transshipment node levels between the source and destination nodes (See Figure D.2).

- **Nodes / Level.** The number of transshipment nodes per level. (For the purposes of this dissertation we assumed that the wizard would always have to construct networks in which the number of nodes at all the levels is the same in number.)

D.1.2 Edge types

Three edge types namely *T0*, *T1*, *T3* edges can be selected to be included in the construction of the potential network. The edges will be connected in such a way that the source nodes are connected to all the transshipment nodes. All the transshipment nodes will be connected to the destination nodes. No edges connecting the sources directly to the destinations will be made. A more sophisticated method would be for the user to identify all potential edges.

This may, however, be a tedious activity. For the purposes of this dissertation we have assumed that the potential network configuration will most likely be as described above.

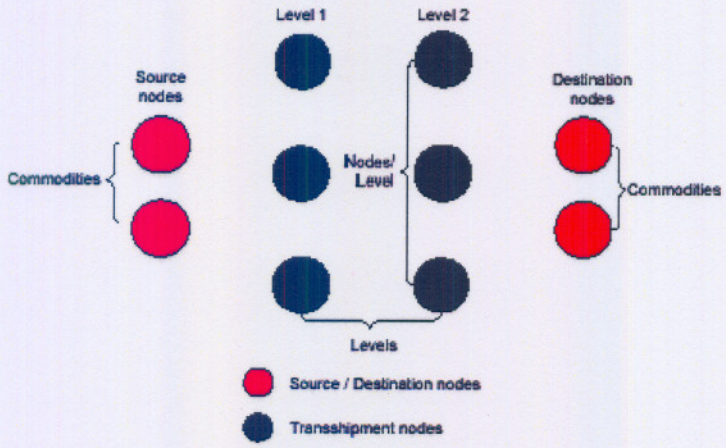


Figure D.2 Graphical representation of network wizard topology

- **Design cost limits.** The minimum and maximum bounds may be specified for the design cost of the different edge types. A random number between these bounds will be selected or the cost may be set at a fixed cost if the minimum is set equal to the maximum.

- **Commodity flow limits.** Upper and lower bounds may be specified for the flow cost and flow constraints on an edge type for all the different commodities.

D.1.3 Random seed

The Random number generator may be initialised with a hardware generated or user specified value. If the random seed is selected its value will be written into the data file. This value may be used to re-initialize the random number generator to replicate a network design with the same values as a previous design.

D.2 Value assignment for different layers

The design costs for edges are automatically assigned by the system in such a way that cost increases for edges connected to levels that are further away. For example: edges between the source nodes and level 2 will have a higher design cost than edges between the source nodes and level 1. Edges with greater capacity also have a larger design cost than those of lower capacity.

The flow costs are also assigned in such a way that they decrease as the capacity of the edge increases. It is, for example, cheaper to send a packet on a *T1* edge than on a *T0* edge.

BIBLIOGRAPHY

[1] AHUJA, R.K, MAGNANTI, T.L. & ORLIN, J.B. 1993. Network flows: theory, algorithms, and applications. Upper Saddle River, NJ: Prentice-Hall. 846p.

[2] ALLEN, A.O. 1990. Probability, statistics and queuing theory. 2nd ed. New York: Academic Press.

[3] BAZARAA, M.S., JARVIS, J.J. & SHERALI, H.D. 1990. Linear programming and network flows. 2nd ed. New York: Wiley.

[4] BELLMAN, R. 1958. On a routing problem. *Quarterly of applied mathematics,* 16:87-90.

[5] BERTSEKAS, D.P., GAFNI, E.M. & GALLAGER, R.G. 1984. Second derivative algorithms for minimum delay distributed routing in networks. *IEEE Transactions on communications,* 32:911-919.

[6] BEZALEL, G. 1995. Telecommunications - a revolution in progress. *Operations research,* 43:29-32.

[7] BOESCH, F.T. 1986. Synthesis of reliable networks - a survey. *IEEE Transactions on reliability,* R-35:240-246.

[8] CANTOR, D.G. & GERLA, M. 1974. Optimal routing in a packed-switched computer network. *IEEE Transactions on computers,* 23:1062-1069.

[9] CARDWELL, R.H., FOWLER, H., LEMBERG, H.L. & MONMA, C.L. 1988. Determining the Impact of Fiber Optic Technology on Telephone Network Design. *Bellcore Exchange Magazine*: 27-32, March/April.

[10] CARDWELL, R.H., MONMA, C.L. & WU, T.H. 1989. Computer-Aided design procedures for survivable fiber optic networks. *IEEE Selected areas in communications,* 7:1188-1197.

[11] CHRISTOFIDES, N. & WHITELOCK, C.A. 1981. Network synthesis with connectivity constraints – a survey. (*In* BRANS, J.P., *ed.* Operational research '81. IFORS: North-Holland Publishing Company.)

[12] CLARKE, L.W. & ANANDALINGAM, G. 1995. A bootstrap heuristic for designing minimum cost survivable networks. *Computers and Operations Research,* 22(9):921-934.

[13] COLBOURN, C.J. 1987. The combinatorics of network reliability. New York: Oxford University Press.

[14] COLBOURN, C.J. 1999. Reliability issues in telecommunication network planning. (*In* Sanso, B. & Soriano, P., *eds.* Telecommunications network planning. Norwell, Mass.: Kluwer Academic Publishers. p. 135-146.)

[15] COOPER, R.B. 1980. Introduction to queuing theory. 2nd ed. Cambridge: Elsevier.

[16] CROWDER, H., JOHNSON, E.L. & PADBERG, M.W. 1983. Solving large zero-one linear programming problems. *Operations research,* 31:803-834.

[17] DANTZIG, G.B. 1963. Linear programming and extensions. Princeton, New Jersey: Princeton university press.

[18] DE VILLIERS, D. & HATTINGH, J.M. 2004. Optimization of network mesh topologies and link capacities for congestion relief. *SATNAC 2004 Proceedings,* 2:95-100.

[19] DOLAN, A. & ALDOUS, J. 1993. Networks and algorithms: an introductory approach. Chichester: Wiley.

[20] FELLER, W. 1968. An introduction to probability theory and its applications. 3rd ed. New York: Wiley.

[21] FLOYD, R.W. 1962. Algorithm 97: shortest path. *Communications of the ACM,* 5:345.

[22] FORTZ, B., LABBE, M. & Maffioli, F. 2000. Solving the two-connected network with bounded meshes problem, *Operations Research,* vol. 48, no. 6, pp. 866–877.

[23] FRANK, H. & CHOU, W. 1972. Topological optimization of computer networks. *Proceedings of the IEEE,* 60:1385-1397.

[24] GAVISH, B. 1990. Backbone network design tools with economic tradeoffs. *ORSA Journal on Computing,* 2:236-252.

[25] GOMORY, R.E. 1963. An algorithm for integer solutions to linear programs. (*In* GRAVES, R.L. & WOLFE, P., *eds*. Recent advances in mathematical programming. New York: McGraw-Hill. p. 269-302.)

[26] GONDRAN, M. & MINOUX, M. 1990. Graphs and algorithms. Chichester: Wiley.

[27] GROTSCHEL, M., MONMA, C.L. & STOER, M. 1992. Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations research,* 40:309-329.

[28] HADLEY, G. 1965. Linear programming. Reading, Mass.: Addison-Wesley.

[29] HILLIER, F.S. & LIEBERMAN, G.J. 1995. Introduction to mathematical programming. New York: McGraw-Hill.

[30] HU, T.C. 1969. Integer programming and network flows. Reading, Mass.: Addison-Wesley.

[31] ILOG. 2001. Optimization. http://www.ilog.com/products/cplex Date of access: 20 Oct. 2005.

[32] KERSHENBAUM, A. 1994. Telecommunications network design algorithms. New York: McGraw-Hill.

[33] KERSHENBAUM, A., KERMANI, P. & GROVER, G.A. 1991. Mentor an algorithm for mesh network topological optimization and routing. *IEEE transactions on communications,* 39:503-513.

[34] LEE, S.M., & SHIM, J.P. 1990. Micro management science. 2nd ed. Boston: Allyn and Bacon. 528 p.

[35] LEUNG, J.M.Y., MAGNANT, T.L. & SINGHAL, V. 1990. Routing in point-to-point delivery systems: formulations and solution heuristics. *Transportation science,* 243:245-259.

[36] MAGNANTI, T.L., MIRCHANDANI, P. & VACHANI, R. 1995. Modelling and solving the two-facility capacitated network loading problem. *Operations research,* 43:142-157.

[37] MONMA, C.L. & SHALLCROSS, D.F. 1989. Methods for designing communication networks with certain two-connected survivability constraints. *Operations research,* 37:531-541.

[38] MONMA, C.L., MUNSON, B.S. & PULLEYBLANK, W.R. 1990. Minimum-weight two-connected spanning networks. *Mathematical Programming,* 46:153-171.

[39] PADBERG, M. & RINALDI, G. 1987. Optimization of a 532-city symmetric travelling salesman problem by branch-and-cut. *Operations research letters,* 6:1-7.

[40] PADBERG, M. & RINALDI, G. 1991. A branch-and-cut algorithm for the resolution of large-scale symmetric travelling salesman problem. *SIAM review,* 33:60-100.

[41] PLANE, D.R. & MCMILLAN, C. 1971. Discrete optimization. Englewood Clis, NJ: Prentice-Hall.

[42] SALKIN, M.S., K. MATHUR, & HAAS, R. 1989. Foundations of integer programming. New York: North-Holland.

[43] SCHWARTZ, M. 1977. Computer communication network design and analysis. Prentice-Hall, Englewood Cliffs, N.J.

[44] SERUMAGA-ZAKE, J.M. & HATTINGH, J.M. 2006. Experimental models for network mesh topology with designs that enhance survivability. *SATNAC 2006 Proceedings*, ISBN:0-620-37043-2 (CD).

[45] SORIANO, P., WYNANTS, C., SEGUIN, R., LABBE, M., GENDREAU, M. & FORTZ, B. 1999. Design and dimensioning of survivable SDH/SONET networks. (*In* Sanso, B. & Soriano, P., *eds*. Telecommunications network planning. Norwell, Mass.: Kluwer Academic Publishers. p. 147-167.)

[46] STALLINGS, W. 1985. Data communications. Macmillan, New York.

[47] STEIGLITZ, K., P. WEINER, & KLEITMAN, D.J. 1969. The design of minimum-cost survivable networks. *IEEE Transactions on circuit theory* CT-16, 4:455-460.

[48] TANENBAUM, A.S. 1981. Computer networks. Prentice-Hall, Englewood Cliffs, N.J.

[49] TERBLANCHE, S.E. & HATTINGH, J.M. 2004. Algorithmic considerations for the problem of dimensioning SDH. *SATNAC 2004 Proceedings*, 2:177-182.

[50] VAN DER MERWE, D.J. & HATTINGH, J.M. 2002. Solving tree knapsack problems for local access telecommunication network design issues. *SATNAC 2002 Proceedings*, p. 27-33.

[51] WIKIPEDIA. Menger's theorem. http://en.wikipedia.org/wiki/Menger's_theorem Date of access: 20 Mar. 2006.

[52] WINSTON, W.L. 2003. Operations research: applications and algorithms. 4th ed. California: Duxbury Press.