Heuristic Approaches for Survivable Network Optimization

by

Ahmed Kasem

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In

Communications

Electrical and Computer Engineering
University of Alberta

Abstract

Over the last two decades, many important developments have been made in the field of communication networks, opening the door to new applications in various fields such as transportation, banking and financial services, e-health services, etc. *Wavelength division multiplexing* (WDM) is one technique used to transfer these vast amounts of data over networks of fibre optic cables. But because WDM permits such high bandwidth, any cut in a fibre optic cable can result in a major financial loss due to the volume of traffic at risk. Therefore, survivability is an important consideration in the design of modern large-scale telecommunication networks. Several survivability schemes have been developed for use in WDM networks. Our work will focus on three of these schemes, namely meta-mesh span restoration, *p*-cycles, and node-encircling *p*-cycles (NEPCs).

Span restoration and path restoration are well known survivability schemes, widely addressed in the open literature. While path restoration is more capacity efficient, span restoration is simpler and faster. The meta-mesh scheme had previously been proposed to bridge the gap between the two approaches, because it enhances the capacity efficiency of the span restorable mesh networks while maintaining much of its simplicity. In the present work, we introduce a node-arc ILP model to investigate the effect of the meta-mesh scheme on the capacity efficiency of the span-restorable networks. Then we modify an existing *integer linear programming* (ILP) design model to allow for incremental network evolution. In this work, we formulate two models, one of which applies the

conventional span restoration technique and another that uses the meta-mesh span restoration scheme.

p-Cycle restoration has attracted considerable interest in the network survivability literature in recent years. However, most of the existing work assumes a known network topology upon which to apply *p*-cycle restoration. In the present work, we develop an incremental topology optimization ILP model for *p*-cycle network design, where a known topology can be amended with new fibre links selected from a set of eligible spans. The ILP model proves to be relatively easy to solve for small test case instances, but becomes computationally intensive on larger networks. We then follow with a relaxation-based decomposition approach to overcome this challenge. In our test cases, the decomposition approach significantly reduces computational complexity of the problem, allowing the ILP model to be solved in reasonable time with a minimal impact on the solution optimality.

NEPCs have been proposed as a technique for node and span protection. A key step in obtaining an optimal solution in most existing NEPC network design methods is the enumeration of candidate cycles. The present work introduces two novel algorithms for enumerating an efficient set of candidate cycles for an NEPC network design: the *node-disjoint path partitioning* (NDPP) method and the *level partitioning* method. Furthermore, we develop a *capacitated iterative design algorithm* (CIDA) approach for providing fully capacitated NEPC networks, as well as a genetic algorithm model to investigate the best values for the weighting factors in the key a priori efficiency equation used in the NEPC-CIDA algorithm.

Finally, we propose a node-encircling *p*-cycle ILP design with enhanced span-failure protection to further minimize the total network design cost.

Preface

This thesis is an original work by Ahmed Kasem.

Chapter 5 of this thesis has been published as A. Kasem, J. Doucette, "Incremental Optical Network Topology Optimization Using Meta-Mesh Span Restoration", *Design of Reliable Communication Networks* (DRCN 2011), Krakow, Poland, 10-12 October 2011.

Chapter 6 of this thesis has been published as Md. Noor-E-Alam, A. Kasem, J. Doucette "ILP Model and Relaxation-Based Decomposition Approach for Incremental Topology Optimization in *p*-Cycle Networks," *Journal of Computer Networks and Communications*, Vol. 2012, pp. 1-10, 2012. This work is collaboration with Dr. Md. Noor-E-Alam. He helped me with implementing the Relaxation-Based Decomposition Approach. I proposed the paper idea and developed the ILP model.

Chapter 7 of this thesis has been published as A. Kasem, J. Doucette, "Algorithmic Approaches for Efficient Enumeration of Candidate Node-Encircling *p*-Cycles", *(INFORMS Telecommunication*) Conference, Montreal, Quebec, 5 – 7 May 2010.

Chapter 8 of this thesis has been published as A. Kasem, R. Gallardo, J. Doucette, "An Enhanced ILP Design Model for Node-Encircling *p*-Cycle Networks", *Design of Reliable Communication Networks (DRCN 2014),* Ghent, Belgium, 1-3 April 2014. This work is collaboration with Roberto Gallardo. He helped me with collect the data and analyzes the results. I proposed the paper idea and developed the ILP model and the enhanced ILP model.

Acknowledgement

My deepest gratitude goes to my supervisor Dr. John Doucette. He not only supervised me throughout the course of my PhD program, but also guided my attitude towards career and life. His extensive knowledge and exceptional ability to find new approaches for difficult problems were pivotal in this work and my development as a PhD student. This work would not have been completed without his encouragement and patience. I feel privileged to have had the opportunity to be supervised by him.

I am grateful to Professors Dr. Ivan Fair, Dr. Michael Lipsett, Dr. Hooman Askari-Nasab, and Dr. Gangxiang Shen for serving on my PhD exam committee. My understanding of the subject materials benefited immensely through my interaction with them.

I thank the faculty of Electrical and Computer Engineering at University of Alberta and TRLabs in Edmonton for their superb guidance and assistance. I thank the office staff and the systems support staff in them.

I thank all of the members of my research group especially Dr. Md. Noor-E-Alam and Brody Todd.

To my parents, Dr. Mohamed Kasem and Dr. Mervat Bahgat, who through personal sacrifices permitted me to complete my PhD successfully, from whom I inherited determination and spirit of never give up. I thank my sister and my brother, my wife and my kids Eyad and Esraa for their constant support and encouragement in my life.

Contents

List of Figures

List of Tables

List of Abbreviations

APS	Automatic Protection Switching
BLSR	Bidirectional Line-Switched Rings
BIP	Binary Integer Programming
CIDA	Capacitated Iterative Design Algorithm
DFS	Depth First Search Algorithm
FIPP	Failure Independent Path-Protecting
GA	Genetic Algorithm
Gb\s	Giga bit per second
ILP	Integer Linear Program
JCA	Joint Capacity Allocation
LAN	Local Area Network
LCMA	Local-map Cycles Mining Algorithm
LP	Linear Program
NEPC	Node-Encircling *p*-Cycle
NCMA	Node-Encircling *p*-Cycles Mining Algorithm
NDPP	Node-Disjoint Partitioning Path
OR	Operations Research
OXC	Optical Cross Connect
SCA	Space Capacity Allocation

UPSR Unidirectional Path Switched Rings

WAN Wide Area Network

WDM Wavelength Division Multiplexing

Chapter 1. Introduction

Over the last two decades, telecommunications systems have achieved unprecedented growth in nearly all aspects of our lives. This growth increases the need for fast and reliable communicationss networks. Thanks to fibre optic technologies, wavelength division multiplexing (WDM) networks have provided a tremendous bandwidth. WDM is the technique used in the fibre optic realm, where multiple wavelengths are transmitted simultaneously over a single fibre. This technology reaches up to 320 wavelengths per fiber; each one could carry 10Gb/s [1].

In addition to their numerous benefits, there are potential risks in the case of failure occurrence. On February 2013, a failure that persisted more than eight hours occurred while engineers of Ethio Telecom were upgrading the system of the main mobile telephone network controlling station. This failure wreaked havoc on many activities including business activities throughout the Ethiopian capital [2]. On July 2012, a ten-hour failure in the France Telecom network caused by a software bug prevented its subscribers from making calls [3]. On October 21, 2006, due to a small fire, more than 100,000 phone customers in St. John's and Internet clients all over Newfoundland lost service. There was no 911 services or working automated bank machines. Services were restored after five hours [4]. On February 1, 2008, three of the primary fibre cables connecting Asia and the Mideast to Europe were cut by ship's anchors. This outage led to degradation in the communication ability between the two regions to less than 30% of its normal performance [5]. On April 9, 2009, vandals cut two sets of fibre optic cables in San Jose and San Carlos which led to disruption in telecommunication across Silicon Valley and 911 emergency services in Santa Clara County [6]. On January 27, 2010, vandals also cut fibre optic cables to Selah in Seattle which led to disruption of phone, TV and Internet services [7]. On average, the failure rate for long-haul telecommunication links is three failures per 1000 km per year and for the metropolitan links is twelve failures per 1000 km per year. Restoration time on average is twelve hours per cut [8]. The failure cost

is very expensive, more than 50% of the Fortune 500 companies suffer at least IT system failure of 1.6 hours per week, which leads to a financial loss of on average $46 million dollar per each company [9], [10].

Network failure sources are various and can be classified into two different categories: node failures and cable failures. As illustrated in Figure 1-1, the most common causes of cable failure are digging, cuts during construction, maintenance in the surrounding area, and vandalism, while the main sources of node failures are software failure or fire [11].

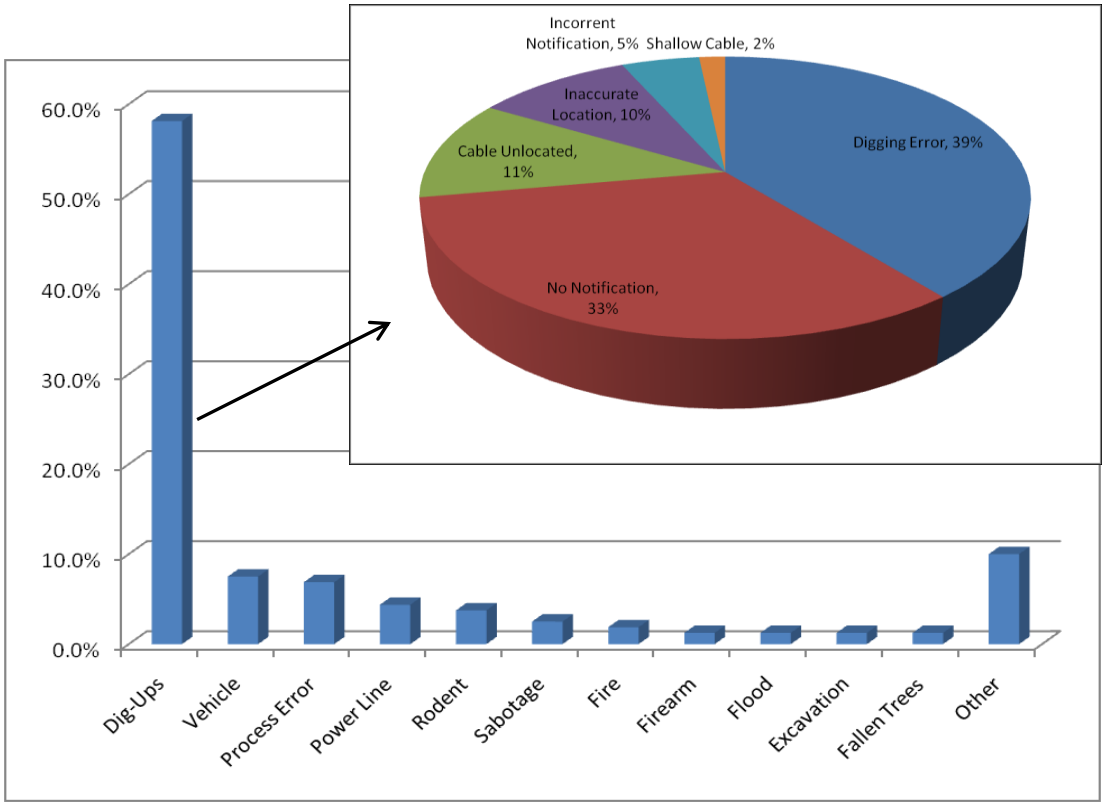


Figure 1-1. Network failure common sources [12] [13] [8]

Typically, network equipment operating at the nodes is contained in a safe environment that is carefully monitored by the network administrator, avoiding any environmental factors that might affect the performance of the equipment and providing speedy recovery in the case of a node failure. However, network cables traverse through diverse environments, often outside the control of the network

administrator. This increases the susceptibility of disruptions and increases the restoration time [14]. Our work is relevant to the network disruptions that occur due to failure in nodes or cables.

As the dependence on the telecommunication infrastructure increased, network survivability became a significant issue for all aspects of telecommunication networks. Their failure can have a major impact, including enormous economic and social consequences. As the use of technology increases, dependence on the telecommunication infrastructure to provide highly reliable communication is growing. Two recent areas that will put considerable pressure on telecommunication networks are cloud computing [15] and e-health [16]. Cloud computing is providing computing and storage capacity as a service to a group of end users. All users' services and calculations can be delivered through cloud computing over the internet. There are three types of cloud computing: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). In using SaaS, users also rent application software and databases. The cloud providers administer the infrastructure and platforms on which the applications run. End users access cloud-based applications through a web browser or mobile application while the business software and users' data are kept on servers at a remote location in the cloud. Cloud computing permits incorporations to get their software programs up and run them faster, with enhanced manageability and less maintenance, and enables IT to adjust resources more rapidly to meet variable and unpredictable business request [17]. E-health is a tag for the suite of applications that are being developed for the healthcare industry to enhance patient care and lower costs. These multiple services include electronic health records, remote doctor patient consultations, digital imaging, and tele-surgery. All these technologies are built on having a reliable telecommunication infrastructure, which is a must for the next generation of business, social and government applications.

1.1 Motivation and Goals

Our main goal in this thesis is to provide the telecommunications network operators with an efficient solution for a survivable network design. In our work, we mean by an efficient solution that it should be faster and lower cost. Reducing the run time for a design instance will help to run so many instances with different parameters to reach more informative decision. In this work, we also try to provide the designer with more insights and understanding of the different survivability schemes used throughout this thesis.

1.2 Thesis Outline

In this section, each chapter of the thesis will be outlined.

Chapter 2: This chapter is the first part of the introductory chapters of the thesis. It covers a variety of topics related to telecommunication networks. Its aim is to provide the reader with the tools, concepts, and terminologies used throughout this thesis. The first section provides a telecommunication network overview examining the three layer model access network, metropolitan-area network, and backbone network. Then it provides many basic definitions and concepts from graph theory those are related to transport networks. The last section provides illustrations to many common network design algorithms.

Chapter 3: This chapter illuminates many survivability schemes such as automatic protection switching (APS), unidirectional path switched rings (UPSR), bidirectional line-switched rings (BLSR), pre-configured protection cycles *(p*-cycle), node-encircling *p*-cycles (NEPC), and meta-mesh span restoration. The survivability techniques in WDM networks can be classified into two main categories, the first type is ring protection where the network is protected by rings of spare capacity included in the network, such as UPSR, and BLSR. The second type of survivability technique is mesh restoration, which tries to exploit some of the spare capacity in the spans of the network. Examples of mesh restoration are APS, span restoration, path restoration, *p*-cycles, and NEPC.

Chapter 4: This chapter describes the different methodologies used for solving the research optimization problems in this thesis. Typically, there are two main approaches to tackling any optimization problem, the first of which are exact methods to find optimal solutions. The time complexity and the memory required for these methods increase exponentially with the increase of the problem size. The second are approximate methods that provide a reasonably good solution within a shorter period of time, but these do not guarantee finding the optimal solution.

Chapter 5: In this chapter, we study the meta-mesh span restoration technique. While path restoration is more efficient with respect to capacity, span restoration is simpler and faster to implement and design. The meta-mesh scheme was proposed a number of years ago to bridge that gap in sparse network topologies, providing more capacity-efficient designs with a simple span-restoration-like mechanism. We have developed a new node-arc meta-mesh ILP formulation and further extended that formulation to allow for incremental topology optimization.

Chapter 6: In this second study**,** we have developed a new ILP model for incremental topology optimization in a *p*-cycle network that is capable of selecting an optimal subset of potential spans to add to an existing *p*-cycle network. While the ILP proves to be relatively simple to solve for small test case networks, it is computationally intricate to solve for larger networks. We then developed a relaxation-based decomposition heuristic that considerably decreases runtime of the ILP in our large test networks, while having no statistical influence on optimality.

Chapter 7: The third study has been devoted to looking at the algorithmic approaches for solving NEPC network design problems. In the integer linear programming of *p*-cycles and node-encircling *p*-cycles network design, the first and most time-consuming step is to enumerate a number of eligible cycles that could be used in the final solution. Enumerating all cycles in the network is an impractical approach specifically in large networks because the number of generated cycles grows exponentially with the increase of the number of nodes

and spans. Many algorithms have been proposed for enumerating a respectively small set of candidate *p*-cycles without degrading the optimality of the final ILP solution significantly. However, few algorithms have been developed for NEPC design. We have developed two algorithms for this problem, the *node-disjoint path partitioning algorithm* (NDPP) and *level partitioning algorithm*. To release a complete network design, an algorithm should be used to select the least cost combination of cycles that will fully protect the network working capacities. A CIDA-like algorithmic approach, for providing fully capacitated NEPC networks, has been developed. We then proposed a genetic algorithm model to enhance the CIDA-like algorithm by determining the best values for its factors.

Chapter 8: In this last study, we have developed a new enhanced ILP design model that optimally designs a node-encircling *p*-cycle network. The new ILP model takes advantage of the observation that NEPCs assigned solely for node-failure protection will inherently protect all two-hop segments of every multi-hop working lightpath. As a result, only single-hop working lightpaths need explicit span-failure protection in the conventional manner. The new ILP model shows a significant reduction in capacity requirements. We have developed an enhanced version of the ILP design model for node-encircling *p*-cycle networks that provides only the explicit span-failure protection needed.

Chapter 9: This chapter summarizes the four studies those have been introduced in this thesis. The contributions of the thesis are illustrated.

Chapter 2. Background

2.1 Introduction

This chapter is the first part of the introductory chapters of the thesis and covers a variety of topics related to telecommunication networks. Its aim is to provide the reader with the concepts and terminologies used throughout this thesis.

The first section provides telecommunication networks overview examining the three-layer model of the access network, metropolitan-area network, and backbone network. Many functions, examples, and implementations are discussed for each layer. Then, basic background on WDM concepts, terminologies, and switching elements is presented to support our subsequent studies. The third section provides many basic definitions and concepts from graph theory that are related to transport networks. The last section introduces some common network algorithms used in telecommunication network optimization problems, such as graph search algorithms, shortest path algorithms, and maximum flow algorithms.

2.2 Overview of the Telecommunication Networks

Telecommunication networks considered in this work consist of three parts, an access network, a metropolitan-area network, and a backbone network [18], [19]. Each one of these parts has its own role, coverage area, standards, and technologies. Figure 2-1 shows a schematic of a telecom network. In the following sections, each of the telecommunication network hierarchical layers will be described in detail.

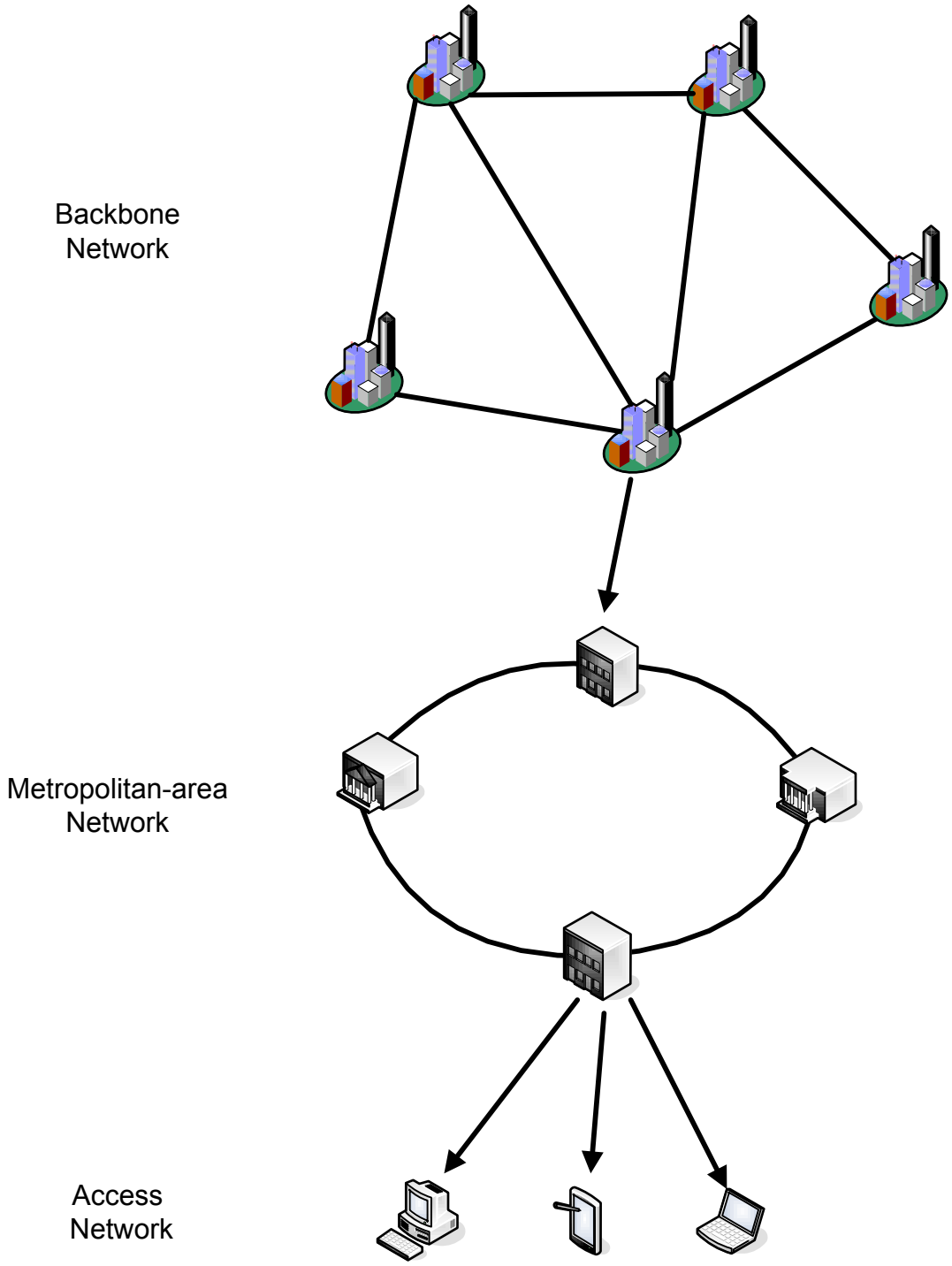


Figure 2-1. Schematic of telecommunication network

The main function of the access network layer is to connect the end users, whether residential or business users, to the metropolitan-area network. Typically, the access network spans a few kilometers between the central office (CO) and

the end users. Many technologies have been deployed in access networks such as *digital subscriber loop* (DSL), cable modems, *passive optical networks* (PON), and wireless connections. A PON cuts the amount of fibre and central office equipment needed compared to point-to-point architectures. At the service provider's CO a PON utilizes an *optical line terminal* (OLT), and close to end users uses a number of *optical network units* (ONUs). A PON is a point-to-multipoint fibre network architecture in which passive optical splitters are involved to allow a one optical fibre to serve many end users [20]. DSL is a group of technologies that offers digital data transmission over the local loop telephone network wires. An *asymmetric digital subscriber line* (ADSL) is widely known as digital subscriber line in the telecommunication market, and it is the most commonly deployed technical variety of DSL. DSL service is transported simultaneously with fixed terrestrial telephone service on the same line. This is possible because DSL uses a higher frequency. These frequency bands are then separated by a splitter at the end user site. Cable television service providers present another type of access network called Cable Modem. The Internet signal is passed on the same coaxial cable that transports cable television service. A special modem splits the Internet signal from the other signal transported on the line and delivers an Ethernet link to the end user. Satellite service providers deliver satellite services. The computer connects *point of presence* (POP) within the satellite network [21]. All of the access networks are linked together through the metropolitan-area network.

The term metropolitan refers to a city consisting of a heavily populated urban centre and its less-populated adjacent territories, sharing industry, infrastructure, and housing. A metropolitan layer usually includes multiple jurisdictions, municipalities, and public services. Normally, the metropolitan-area network spans a metropolitan region, covering distances of a few tens to a few hundreds of kilometers [18]. Metropolitan-area network technologies encompass fibre optics, *worldwide interoperability for microwave access* (WiMAX), and Ethernet.

In hierarchical telecommunication networks the backbone layer consists of the midway connections between the core network elements; each connection could span a few thousand kilometers. For example, while landline phones communicate with a local exchange office, the link between this office and the rest of the world is considered a backbone connection to the core of the telephone company's network. Backbone network technologies include fibre optics, point-to-point microwave radio links, and Ethernet.

There are two main designs of interconnections deployed in backbone networks. Most current telecommunication networks are interconnected as *synchronous optical networking/synchronous digital hierarchy* (SONET/SDH) rings. Nonetheless, their inefficient network resource utilization. Mesh networks, the likely choice of the future, are more efficient in utilizing network resources [18]. The design of the backbone network should consider many parameters such as capacity, cost, and the need for resources such as frequency spectrum, optical fibre, or rights of way. Backbone resources such as link capacity can be shared among different network operators. Our main focus in this research is the design of fibre optic backbone networks.

2.2.1 Wavelength Division Multiplexing (WDM)

Before we start talking about *Wavelength Division Multiplexing* (WDM), we need to know some important information about optical networking. First, what is an optical network, and second, why is an optical network an ideal technology for the backbone network? An optical network is a set of nodes connected by fibre optic cables, where the data travels in the form of the light signal on these cables. Optical fibre technology has many advantages, such as [22]:

a) Optical fibre is immune to electromagnetic interference.
b) Optical fibre can transport tremendous bandwidth (i.e. more than fifty terabits per second) which meets the massive information demands that our current life requires.

c) It has low signal attenuation (as low as 0.2 dB/km), so it does not need many repeaters over a long distance, as is the case of electrical networks.
d) It has low power consumption.
e) Crosstalk and interference are not an issue between adjacent fibre cables as is the case of electrical wires.
f) It is light weight and non-flammable.
g) It is very hard to eavesdrop on, because of its lack of electromagnetic radiation.

Wavelength division multiplexing [23] is a multiplexing technique that can transmit a number of non-overlapping wavelengths over a single fibre cable, as illustrated in Figure 2-2. At the multiplexer end, several wavelengths are aggregated over one trunk, and at the demultiplexer end, the aggregated signals are distributed over several different links. To make sure that all WDM components produced by different vendors will operate with each other, the *International Telecommunication Union* (ITU) has presented a standard for WDM systems called the ITU Wavelength Grid [24]. WDM systems are classified into two wavelength patterns, namely conventional/coarse and dense. *Coarse wavelength division multiplexing* (CWDM) uses enlarged channel spacing to allow less complicated and thus inexpensive transceiver designs. The same transmission spacing but with denser channel window is used in *dense wavelength division multiplexing* (DWDM).

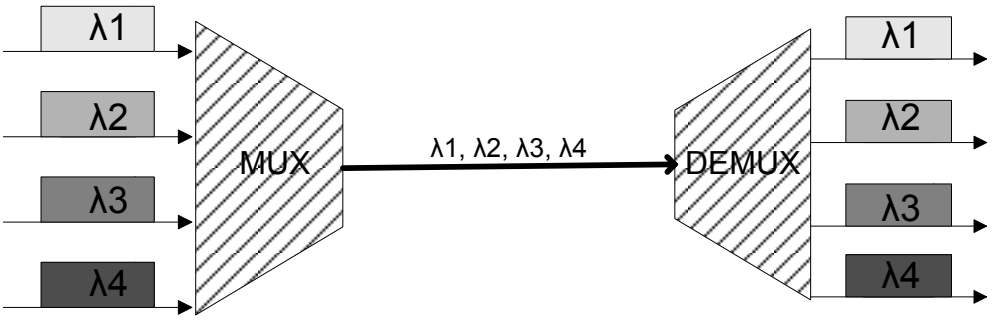


Figure 2-2. Example of WDM

2.2.2 Switching Elements

One of the main components of any telecommunication network is the switching elements. Conventionally, today's optical networks are still widely using optical-electronic-optical (o-e-o) conversion. This means that, electronic processing is still employed in the majority of the modern optical networks and uses the optical fibre only as a transmission medium. Data processing and switching are accomplished by transforming an optical form to its equivalent electronic signal. Such a network relies on electronic switches [18]. The other type of optical switches is called transparent, where the signal is switched in the form of light without the need to convert it to electrical signal. The rest of this section reviews various optical switching elements that are usually deployed in today's optical networks.

An optical add/drop multiplexer (OADM) is a switching element that can add and drop traffic in the network [11]. As illustrated in Figure 2-3, an OADM receives a fibre link with several wavelengths, then some wavelengths are dropped, and new wavelengths are added to the fibre, while other wavelengths pass through the OADM without any modification. The reduction of unnecessary optoelectronic conversions through the use of OADM provides major cost savings in the network. An OADM can work in both the fixed and reconfigurable modes, and it can be used in either mesh or ring networks.

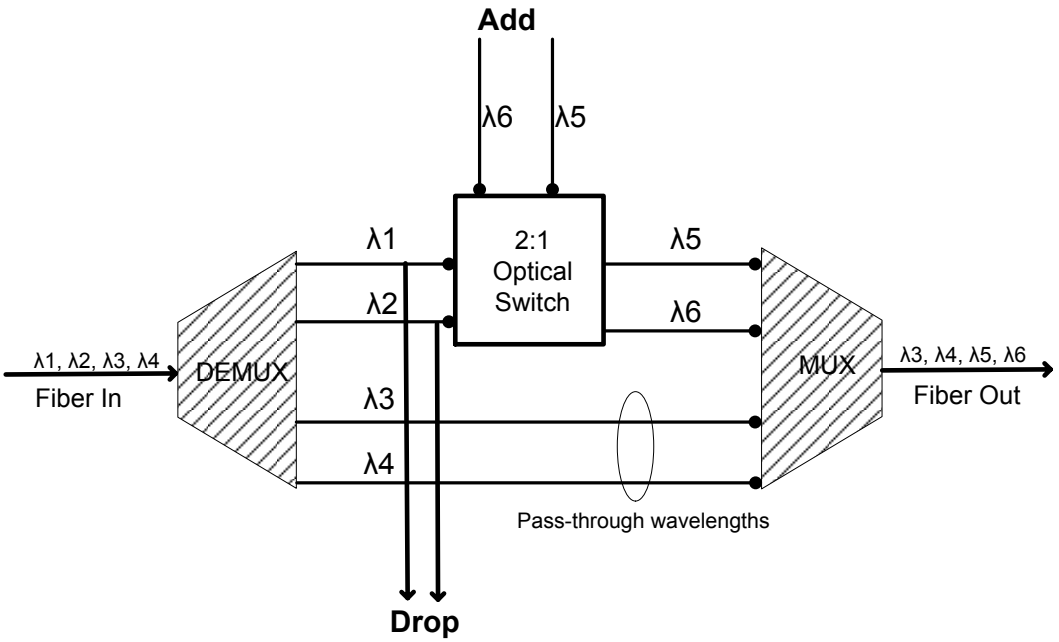


Figure 2-3. Example of OADM

In fixed OADM, the add/drop and pass through wavelengths are predefined and can be manually reordered following the installation. In reconfigurable OADM, the wavelengths that are added/dropped or pass all the way through the node can be dynamically reconfigured as needed by the network operator. This structure is more sophisticated but more flexible because it can offer an automatic on-demand provisioning; consequently, they can be set up in the production environment.

The other optical switch that we are going to discuss is the *optical cross-connect* (OXC) switch [23]. It switches optical signals from input ports to output ports. This kind of the switching element is typically considered to be wavelength insensitive, i.e., incapable of multiplexing or demultiplexing various wavelength signals on a single fibre. A basic cross-connect element is the 2 x 2 crosspoint element which transfers optical signals from two input ports to two output ports and has two states: cross state and straight through state, as shown in Figure 2-4. In the cross state, the signal from the upper input port is transferred to the lower output port, and the signal from the lower input port is transferred to the upper output port. In the straight through state, the signal from the upper input port is

transferred to the upper output port, and the signal from the lower input port is transferred to the lower output port.

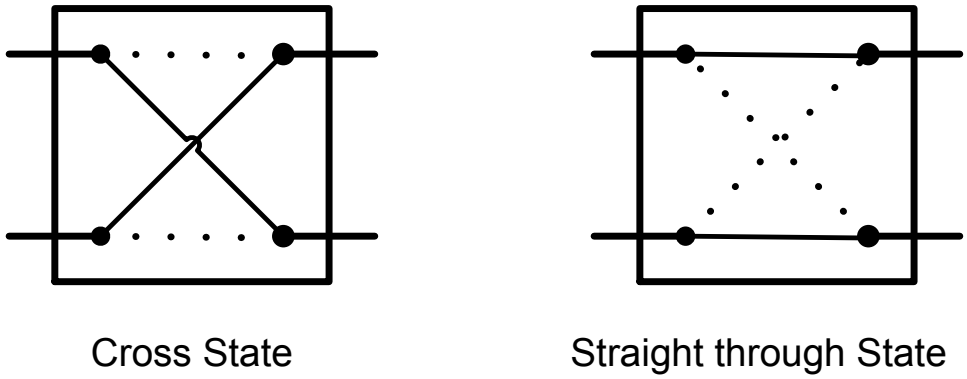


Figure 2-4. 2 × 2 crosspoint element

2.2.3 Routing and Wavelength Assignment

In this section, we will discuss the *routing and wavelength assignment* (RWA) problem in WDM optical networks. Typically, any optical network consists of a set of nodes interconnected by point-to-point optical fiber links. Every optical link can accommodate a limited number of wavelengths. The nodes in the optical network can route a wavelength coming in an input interface to one output interface, independent of the other wavelengths. For example, Figure 2-5 illustrates a simple optical WDM network. In this example, a light signal of wavelength λ_1, that connects node A and C, enters a node B at an interface and is routed to another output interface. A second optical signal flows from node B into node D on a second wavelength, λ_2. A third optical signal flows from node C into the node E on a third wavelength, λ_3. Such end-to-end connections are called lightpaths. It offers a high speed transparent channel to its end users: a virtual conduit is established between the source and the destination nodes. At the same time, another lightpath can reutilize the same wavelength in some other area of the network (this feature is called wavelength reuse), as long as both lightpaths do not pass the same link. Many lightpaths can share the same fiber as long as they do not use the same wavelengths in the shared fibers. A lightpath from any

incoming fiber with a particular wavelength can be routed to any outgoing fiber with the same wavelength. In this case, a lightpath uses a single wavelength from the source to the destination. This is referred to as the *Routing and Wavelength Assignment* (RWA) problem [25]. A feasible assignment for the problem must satisfy two conditions:

- Wavelength continuity condition. The same wavelength must be assigned to all the links along the path traversed by a lightpath. In Figure 2-5, this condition is represented by using a single color for each lightpath.
- Distinct wavelength condition. Two or more lightpaths share a common link; each must be assigned a unique wavelength. In Figure 2-5. Routing and wavelength assignment, this condition requires that the two lightpaths are sharing a link be represented by two different colors.

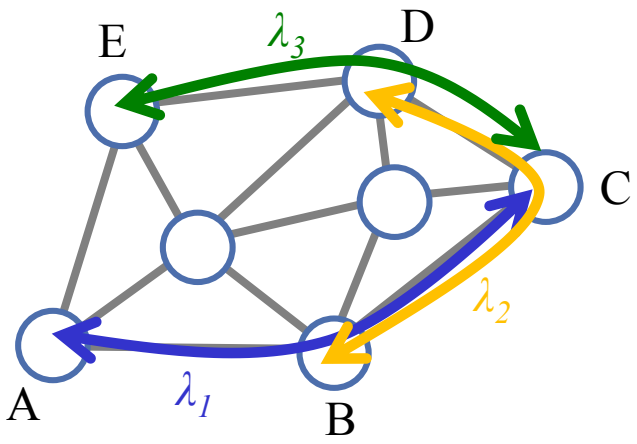


Figure 2-5. Routing and wavelength assignment

However, if a wavelength-converter is utilized in the nodes, a lightpath from any incoming fiber can be switched to any outgoing fiber with any wavelength as long as the wavelength is vacant. As illustrated in Figure 2-6, if we are able to use a wavelength converter to convert the data arriving on one wavelength on a link into another wavelength at an intermediary node before forwarding it on the next link, the wavelength continuity condition can be removed. This approach is possible and is known as wavelength conversion. Networks with this feature are known as wavelength convertible networks. In this type of networks, one

lightpath can utilize multiple wavelengths along its path on each span. In this case, wavelength assignment condition is not a concern, and only the routing problem is considered [26]. The other reason for relaxing the continuity constraints is that most of the current optical switches are o-e-o switches; this means that all the received light signals will be converted to electrical signals before transmitting them. So any wavelength can be converted to any other wavelength without any special processing.

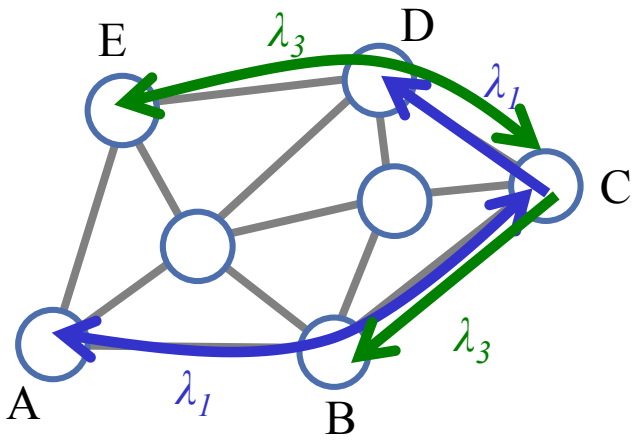


Figure 2-6. Wavelength conversion

Typically, for any routing algorithm, using a wavelength conversion results in a lower solution, in terms of the number of wavelengths used. For example, in Figure 2-6 wavelength conversion was utilized to route the same demand in Figure 2-5, two wavelengths were used to satisfy the demand instead of three wavelengths [26]. In this thesis, we assume that, there is a wavelength converter in every node.

2.3 Graph Theory

Typically, telecommunication networks are represented as a set of nodes and spans between these nodes. Nodes could represent buildings or cities in the real world and spans could represent engineering cables or ducts. This section provides many basic definitions and concepts from graph theory those are related to transport networks.

2.3.1 Terminologies

This section introduces some basic concepts and definitions from graph theory [27] and set theory [28] as a foundation for more advanced treatment of many network algorithms. A *graph* (*network*) G = (N, S) consists of a finite set of *nodes* (*vertices*) N = {N_1, N_2, …..} and a set of *spans* (*edges*) S = {S_1, S_2, …..}, such that each span in S connects a pair of nodes in N. Nodes {N_1, N_2} of a graph are adjacent (neighbours) if they are connected by span S_1 = {N_1, N_2} ∈ S, and span S_1 is said to be incident on nodes N_1 and N_2. In the same way, spans S_1, S_2 are adjacent if they are incident on a common node N_1. Two spans are parallel if they have the same end nodes.

A graph is said to be a *directed graph* if its spans are ordered in pairs of distinct nodes called directed spans. A directed span is represented by a line segment with an arrowhead indicating the direction from the source to the destination. Figure 2-7 illustrates an example of a directed graph. In this graph, N = { 1, 2, 3, 4, 5, 6, 7} and S = {(1, 2), (3, 1), (4, 1), (4, 3), (3, 7), (3, 5), (4, 6), (5, 2), (5, 7), (7, 6)}. Similarly, a graph is said to be an undirected graph if its spans are unordered pairs of distinct nodes. An *undirected span* is represented by a line segment without an arrowhead. Figure 2-8 depicts an example of an undirected graph. In this graph, N = { 1, 2, 3, 4, 5, 6, 7} and S = {(1, 2), (1, 3), (1, 4), (2, 5), (3, 5), (3, 7), (4, 3), (4, 6), (5, 7), (6,7)}. In an undirected graph, any span that connects node pair N_1 and N_2 can be referred as either (N_1, N_2) or (N_2, N_1). In undirected spans, the flow is permitted in two ways while in directed spans it is permitted in one direction.

Any given directed, span S_1 = (N_1, N_2) has two end nodes N_1 and N_2. N_1 is called an *origin* of span S_1while N_2 is referred to as a t*arget* node. The *degree* of a node represents the total number of spans connected to this node. This definition is complete in the case of an undirected graph, while in a directed graph the degree could be divided into two parts; the *indegree* of a node is the number of the incoming span of that node, and the *outdegree* is the number of outgoing spans.

For example, in Figure 2-7, node 3 has an indegree of 2, an outdegree of 2, and degree of 4. G′ is a s*ubgraph* of a graph G if all the nodes and spans that construct G′ belong to graph G. A *weighted graph* is a graph in which each span is associated with a number. This number often represents cost, distance, or capacities in telecommunication networks. A graph, where this number represents capacity, is said to be a *capacitated graph*. Figure 2-9 gives an example of a capacitated graph.

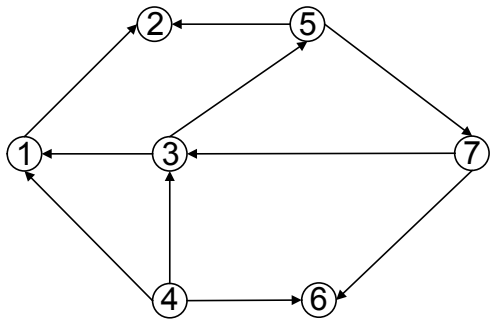


Figure 2-7. Directed graph

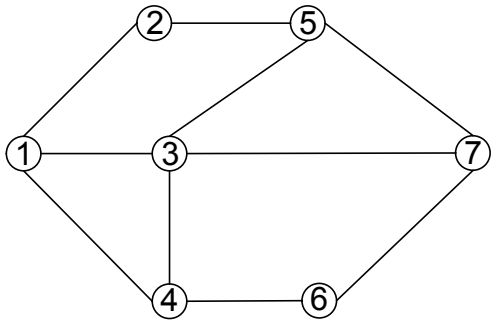


Figure 2-8. Undirected graph

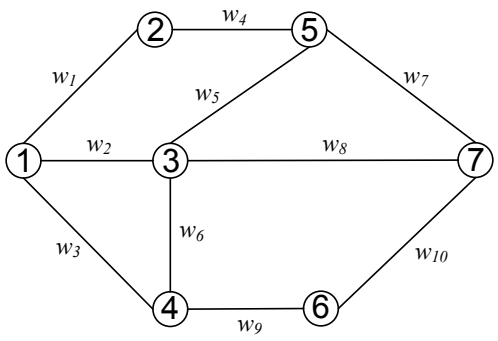


Figure 2-9. Capacitated graph

A *walk* is a subgraph of graph G consisting of a series of nodes and spans. Figure 2-10(a) and (b) show two walks in this graph: 1-2-5-7-3-5 and 1-2-3-7 respectively. A walk without any repetition of nodes is called a *route (path)*. So the walk in Figure 2-10(b) is also a route, but the walk illustrated in Figure 2-10(a) is not considered as a route because it repeats node 5 twice. A *cycle* is a route with same start and end node. Figure 2-11 illustrates two potential cycles from Figure 2-8: 1-2-5-3-7-6-4 and 2-1-5-3. If the graph G contains no cycles, it is an *acyclic graph*. For any graph, there could be two special cases of cycles. First, the *Hamiltonian cycle* traverses all nodes in the graph exactly once. Second, the *Eulerian cycle* passes all spans in the graph one time only.

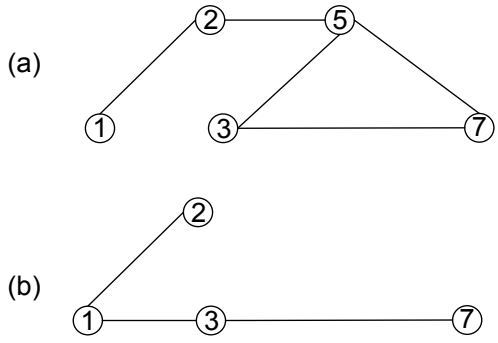


Figure 2-10. Examples of walks

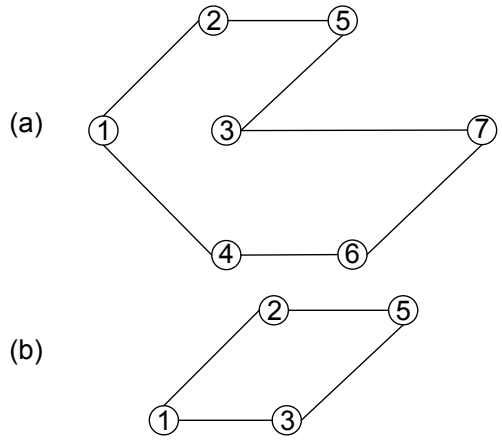


Figure 2-11. Examples of cycles

There are many types of relationships between routes that could connect two distinct nodes. First, the *node-disjoint (fully disjoint) paths* have no element in common between these paths. Figure 2-12(a) presents an example of two node-disjoint routes between nodes 4 and 5, namely 5-2-1-4 and 5-7-6-4. Second, the *span-disjoint paths* have no span in common between these paths. Figure 2-12(b) illustrates an example of two span-disjoint routes linking nodes 4 and 5: 5-2-1-3-4 and 5-3-7-6-4. So every two node-disjoint paths are considered span-disjoint paths, but not every two span-disjoint paths are considered node-disjoint paths. Third, any two routes are considered distinct routes if at least one but not all spans are common among them. Figure 2-12(c) depicts an example of two distinct routes connecting nodes 4 and 5: 5-3-4 and 5-3-7-6-4, respectively.

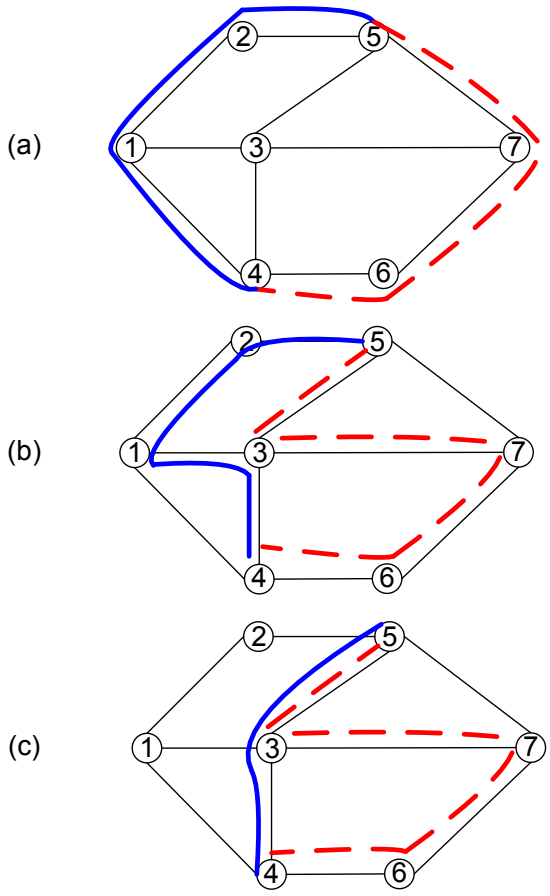


Figure 2-12. Routes have (a) node-disjoint routes, (b) span-disjoint routes and (c) distinct routes

For any given graph, if there is at least one route connecting nodes N_2 and N_1, we can say that these two nodes are *connected*. If every two nodes in the graph are connected then this graph is connected. Or else the graph is *disconnected*. A graph is said to be *bi-connected* if there are at least two node-disjoint routes linking every pair of nodes. This definition is illustrated in Figure 2-13(a). A graph is called *two-connected* if there are at least two span-disjoint routes linking every pair of nodes. Figure 2-13(b) is an example of a two-connected graph. Every bi-connected graph is a two-connected one. Graph connectivity properties are quite essential for telecommunication networks. In order to survive all single span failures, the graph must be at least two-connected, and in order to survive all single node failures, the graph must be bi-connected.

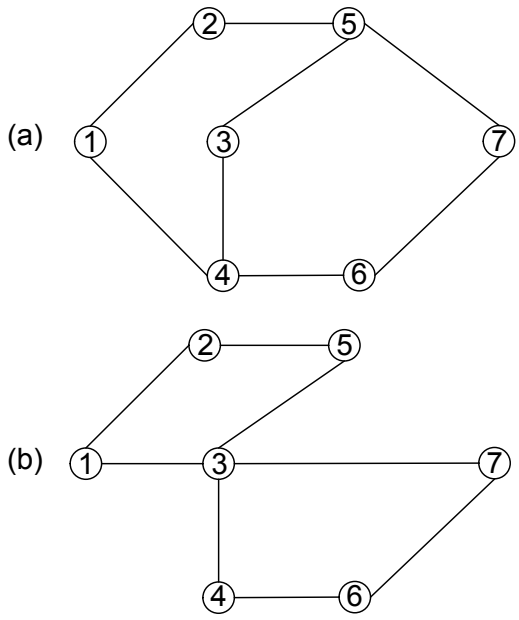


Figure 2-13. (a) bi-connected graph and (b) two-connected graph

A cut is a division of a graph into two partitions. Each cut is defined by a set of spans through which the cut passes. Figure 2-14 illustrates a cut in a graph. The set of spans in this cut are (2, 5), (3, 5), (3, 7), and (4, 6). A graph is said to be a *tree* if it has no cycle. Trees have several distinct properties. First, if a tree has N nodes, it contains N-1 spans. Second, a tree has at least two nodes with degree one, which are called leaves. Finally, every two nodes on a tree are linked by a single distinct route. A spanning tree is a tree that crosses all nodes, and a typical graph can generate many spanning trees. Figure 2-15 provides one spanning tree for the graph in Figure 2-8. A tree is a fundamental concept that arises in the range of network algorithms introduced in this chapter. A group of trees is called a *forest*. Figure 2-16 gives an example of a forest.

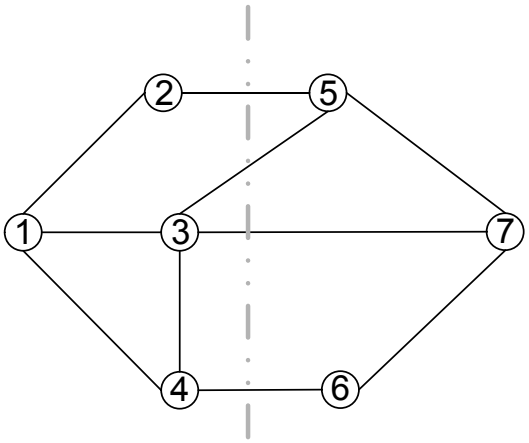


Figure 2-14. An example of a cut

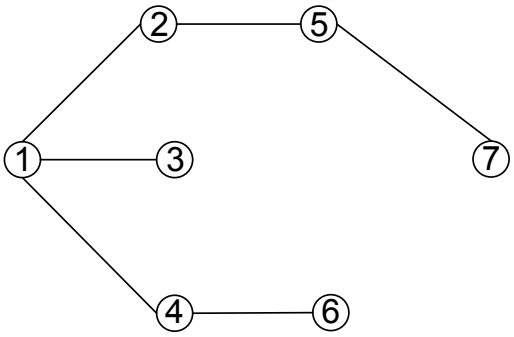


Figure 2-15. An example of a spanning tree

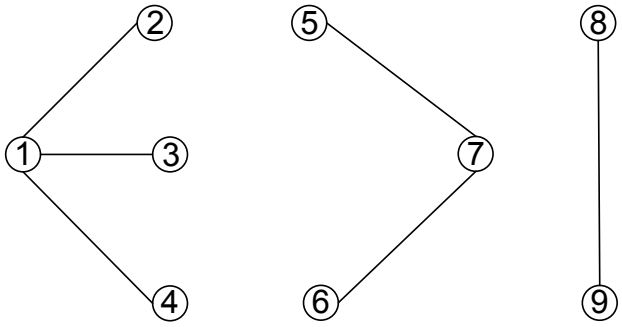


Figure 2-16. An example of a forest

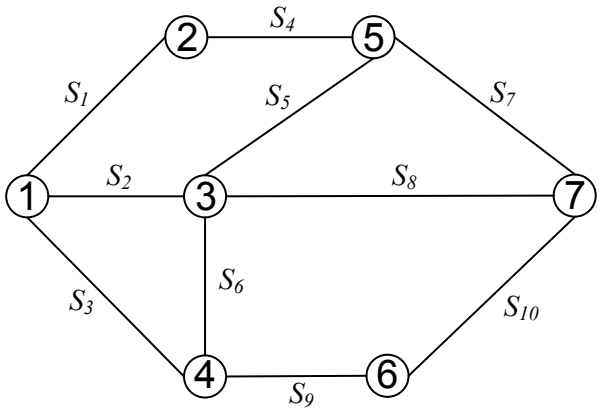
2.3.2 Network Representation

The method used to represent a graph within a computer, or mathematical process is essential in determining the performance of a network algorithm. In this section, we study some techniques for representing a graph. To represent a graph,

two main types of information need to be stored: first, the network topology that determines which node connects to which span in the graph, and second, data such as demands, costs, and capacities associated with each span in the network.

A node-span incidence matrix representation or simply incidence matrix, saves the graph as an N × S matrix that contains one column for each span of the graph and one row for each node. Each column contains two non-zero values (i.e., 1) because each span connects only two nodes. Figure 2-17 shows this representation for a network example. The matrix has a row n and a column s corresponding to every node and every span, respectively, and its $(n,s)^{th}$ value equals 1 if node n is incident on span s and equals 0 otherwise.

A node-node adjacency matrix representation, or simply adjacency matrix, stores the network as an N × N matrix that has one column and one row for each node of the graph. Figure 2-18 gives this representation for the network example in Figure 2-17. The matrix contains a column and a row for every node, and its $(x,y)^{th}$ value equals 1 if $(x,y) \in S$ and equals 0 otherwise. The incidence matrix representation of a graph is not computationally efficient because it contains so few nonzero coefficients. However, the adjacency matrix is space efficient only if the network is dense while in the case of a sparse graph it wastes significant space. Nonetheless, the straight forwardness of the adjacency matrix allows us to implement most graph algorithms simply. This kind of matrix has been utilized extensively in the programs developed through the course of this thesis.



	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
N_1	1	1	1	0	0	0	0	0	0	0
N_2	1	0	0	1	0	0	0	0	0	0
N_3	0	1	0	0	1	1	0	1	0	0
N_4	0	0	1	0	0	1	0	0	1	0
N_5	0	0	0	1	1	0	1	0	0	0
N_6	0	0	0	0	0	0	0	0	1	1
N_7	0	0	0	0	0	0	1	1	0	1

Figure 2-17. Node-span incidence matrix of the graph example

	N_1	N_2	N_3	N_4	N_5	N_6	N_7
N_1	0	1	1	1	0	0	0
N_2	1	0	0	0	1	0	0
N_3	1	0	0	1	1	0	1
N_4	1	0	1	0	0	1	0
N_5	0	1	1	0	0	0	1
N_6	0	0	0	1	0	0	1
N_7	0	0	1	0	1	1	0

Figure 2-18. Node-node adjacency matrix of the graph example

2.4 Network Algorithms

The term algorithm refers to a sequence of steps for solving a computational problem; each one of these steps requires one or more computational operations. At the end of this sequence, a solution for this problem will be generated. To implement an algorithm a few conditions should be met: (1) a clear objective must be defined, (2) each step of the proposed sequence must be implemented in a finite amount of time, and (3) the overall number of these steps must be limited.

Algorithms play a significant role in solving many network computational problems, for example:

1) Shortest path problems: What is the best way to traverse a network to get from one node to another as economically as possible?
2) Maximum flow problems: If you have a capacitated network, how can you send as much flow as possible between two nodes in the network while respecting the span flow capacities?
3) Minimum cost flow problems: If a cost per unit flow on a network is placed on each span and units of a demand located at one or more nodes in the network need to be sent to one or more other nodes, how can we send this demand at lowest possible cost?

Many network algorithms have been proposed throughout the open literature to solve the preceding network problems. In the subsequent section, we will discuss several network algorithms. The network graph of every one of the following algorithms will be examined.

2.4.1 Graph Search Algorithms

Graph search algorithms are a fundamental type of network algorithm; they arise regularly as sub-problems when tackling many network optimization problems. The applications of graph search algorithms comprise: (1) determining the graph connectivity, and (2) finding all the reachable nodes in a network from a specific node *n*. In this thesis, the depth first search algorithm [29] will be discussed in more details.

2.4.1.1 Depth First Search Algorithm

Depth-first search (DFS) is a technique for searching graphs. One begins at the root and explores as far as needed to open a new branch before going into reverse. A version of the depth-first search was explored in the nineteenth century by French mathematician Charles Pierre as a strategy for solving mazes [29]. Algorithms that use depth-first search as a building block include finding connected components and finding bi-connectivity in graphs. The DFS algorithm is essential for the work in this thesis because it is used as a step in developing a variety of cycle enumeration algorithms [30], [31], [32], [33].

A simple version of a practical demonstration of depth first search algorithm to discover the DFS tree is depicted on graphs in Figure 2-19 to Figure 2-33. The sequence of the algorithm as follows:

a) Unmark all nodes and spans in the network.
b) Label node 1 and score it in the stack.
c) Pick the last node in the stack, which is currently node 1. Mark any scanned span that connects it to any unvisited node and score this node (node 2) in the stack and mark it.
d) Similarly, pick the last node in the stack, which is currently node 2. Mark any scanned span that connects it to any unvisited node and score this node (node 5) in the stack and mark it.
e) Pick the last node in the stack, which is currently node 5. Mark any scanned span that connects it to any unvisited node and score this node (node 7) in the stack and mark it.
f) Pick the last node in the stack, which is currently node 7. If there are no spans that are connected to an unvisited node, delete this node from the stack and pick the one before (node 5).
g) Keep repeating this process as in Figure 2-25 until the stack is emptied, and the depth first search tree of the graph is generated in Figure 2-33.

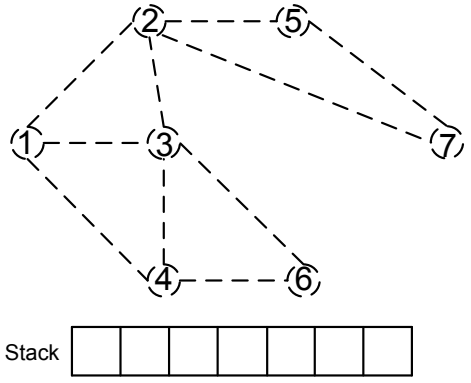


Figure 2-19. Unmark all nodes and spans in the network.

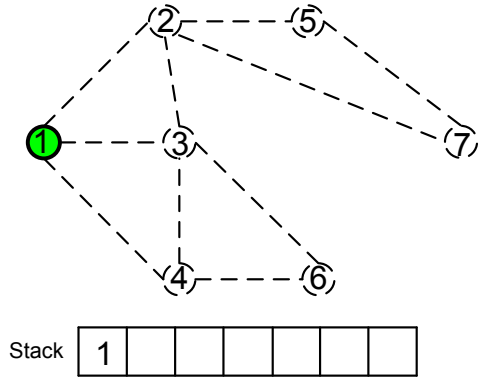


Figure 2-20. Label node 1 and score it in the stack.

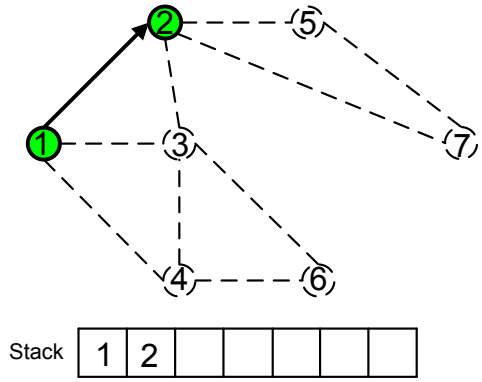


Figure 2-21. Pick the last node in the stack, which is currently node 1. Mark any scanned span that connects it to any unvisited node and score this node (node 2) in the stack and mark it.

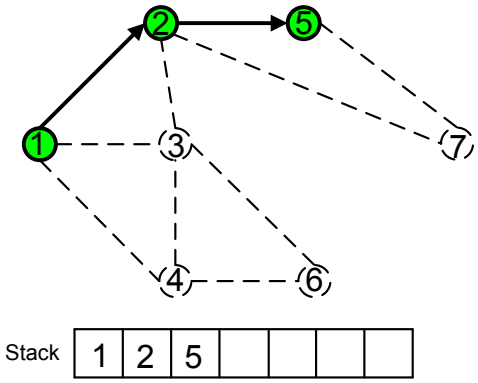


Figure 2-22. Similarly, pick the last node in the stack, which is currently node 2. Mark any scanned span that connects it to any unvisited node and score this node (node 5) in the stack and mark it.

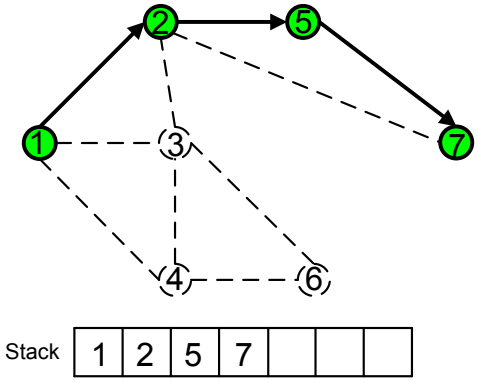


Figure 2-23. Pick the last node in the stack, which is currently node 5. Mark any scanned span that connects it to any unvisited node and score this node (node 7) in the stack and mark it.

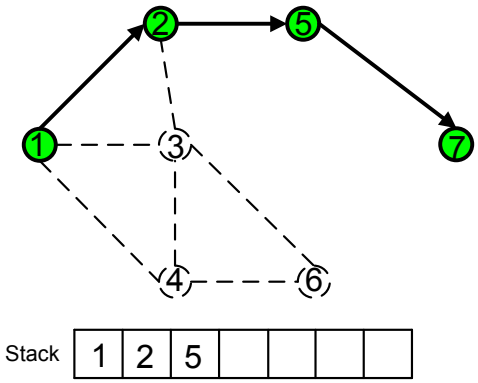


Figure 2-24. Pick the last node in the stack, which is currently node 7. If there are no spans that are connected to any unvisited node, delete this node from the stack.

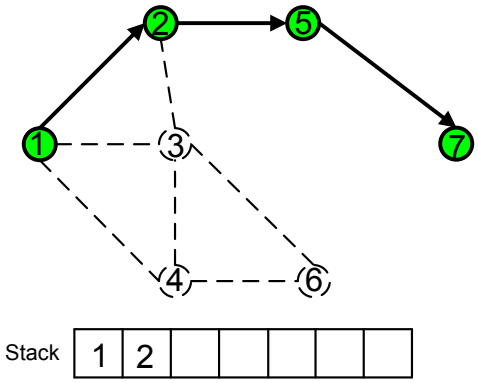


Figure 2-25. Pick the last node in the stack, which is currently node 5. If there are no spans that are connected to any unvisited node, delete this node from the stack.

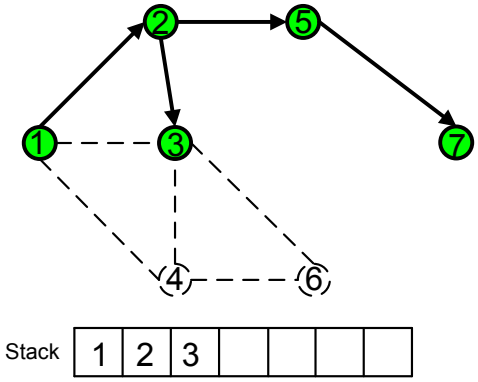


Figure 2-26. Pick the last node in the stack, which is currently node 2. Mark any scanned span that connects it to any unvisited node and score this node (node 3) in the stack and mark it.

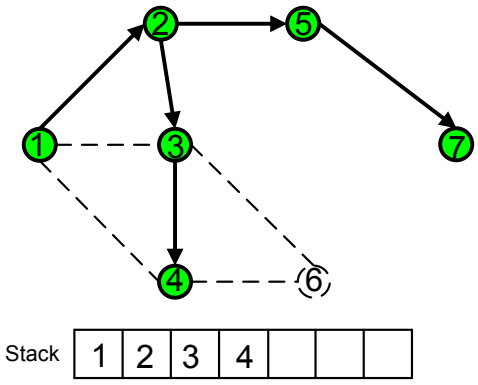


Figure 2-27. Pick the last node in the stack, which is currently node 3. Mark any scanned span that connects it to any unvisited node and score this node (node 4) in the stack and mark it.

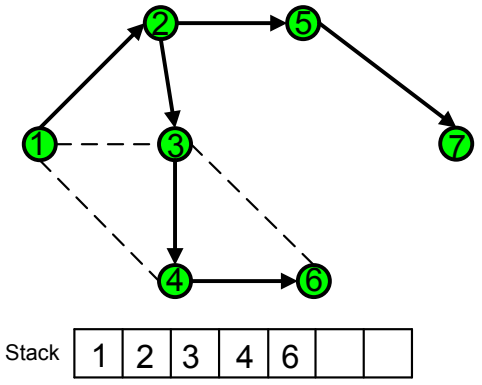


Figure 2-28. Pick the last node in the stack, which is currently node 4. Mark any scanned span that connects it to any unvisited node and score this node (node 6) in the stack and mark it.

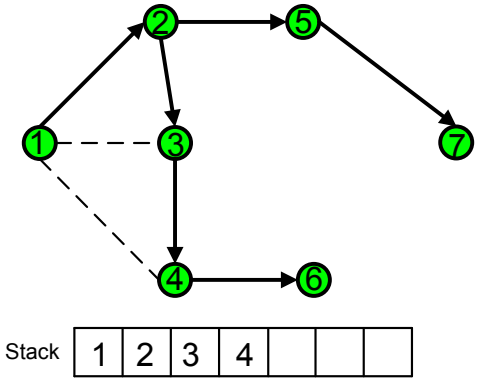


Figure 2-29. Pick the last node in the stack, which is currently node 6. If there are no spans that are connected to any unvisited node, delete this node from the stack.

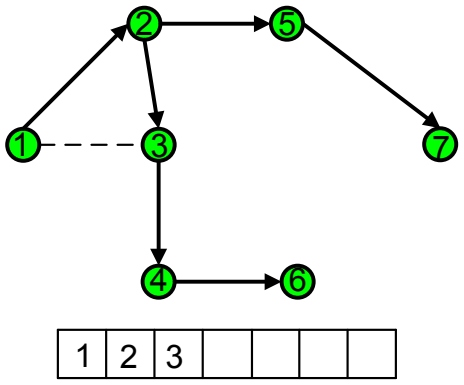


Figure 2-30. Pick the last node in the stack, which is currently node 4. If there are no spans that are connected to any unvisited node, delete this node from the stack.

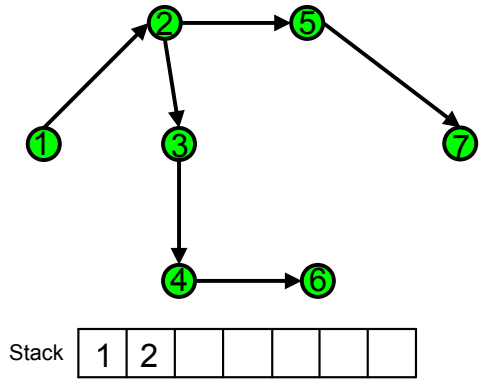


Figure 2-31. Pick the last node in the stack, which is currently node 3. If there are no spans that are connected to any unvisited node, delete this node from the stack.

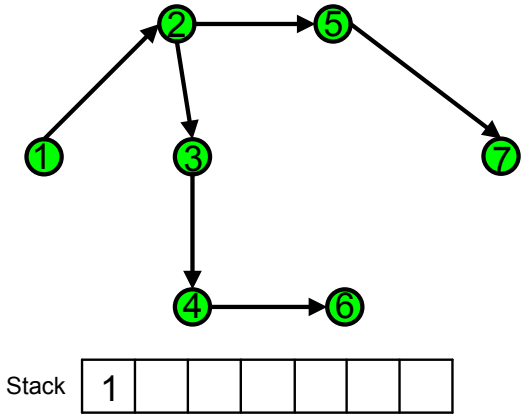


Figure 2-32. Pick the last node in the stack, which is currently node 2. If there are no spans that are connected to any unvisited node, delete this node from the stack.

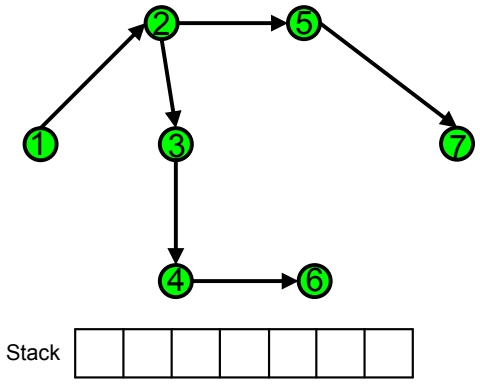


Figure 2-33. Pick the last node in the stack, which is currently node 1. If there are no spans that are connected to any unvisited node, delete this node from the stack.

2.4.2 Shortest Path Algorithms

Finding the shortest path from one node to another in a network is an important step in many network optimization problems, such as routing problems. Officially described, given a cost capacitated network G = (N, S), the shortest path from node *x* in N is the set S' of edges in S that connect *x* to *y* at the least cost. Shortest path algorithms have many applications, such as finding driving directions on web mapping websites like Google Maps.

2.4.2.1 Dijkstra's Algorithms

One of the main algorithms to solve the single-source multi-destination shortest-path problem is Dijkstra's algorithm published by Dutch computer scientist Edsger Dijkstra in 1959 [34], [35]. Dijkstra's shortest-paths tree is developed during the process of this algorithm. Its root is the start node, and its branches are the shortest paths from the root node to all other nodes in the network. Dijkstra's algorithm is a greedy algorithm that generates an optimal solution. The algorithm is greedy because it adds spans to the shortest-paths tree depending on which is best at this point in time. It works for positive capacitated graphs, such that all the values (cost and distance) associated with all spans must be nonnegative numbers.

Figure 2-34 to Figure 2-43 provide a realistic example of implementing Dijkstra's algorithm for a small network. This algorithm tries to find the shortest routes from node 1 to every other node in the network using the distances (costs) of the network spans. The algorithm proceeds as follows:

a) Unmark all nodes and spans in the network and capacitate the network with the distance of the spans. Now the number written beside each span represents the cost of it.
b) Mark node 1 and label it with 0 and label every other node in the network with ∞. These numbers (labels) beside each node represent the costs from these nodes to node 1.
c) Find the node with the lowest label, which is currently node 1. Mark every scanned span that links node 1 to an adjacent node, if this adjacent node will have a new label with a value lower than the current one. Update the labels using spans' costs.
d) Similarly, find the unmarked node with the lowest label (node 3 because its label is 1) and mark it in the network. Mark every scanned span that links node 3 to an adjacent node if this adjacent node will have a new label with a value lower than the current one. Update the labels.
e) Find the unmarked node with the lowest label (node 4 or node 2 because their labels are 2) and mark any one of them. (In our example we picked node 4.) Mark every scanned span that links node 4 to an adjacent node if this adjacent node will have a new label with a value lower than the current one. Update the labels.
f) Similarly, find the unmarked node with the lowest label (node 2 because its label is 2) and mark it. Mark every scanned span that links node 2 to an adjacent node, if this adjacent node will have a new label with a value lower than the current one. Update the labels.
g) Iterate the previous step until all nodes are marked, and the Dijkstra's tree with root node 1 of the graph is generated. The label associated with each node represents the cost from this node to node 1. For example, to reach node 1 from node 7, it costs 7 units, and its route is {7 - 5 - 2 - 1}.

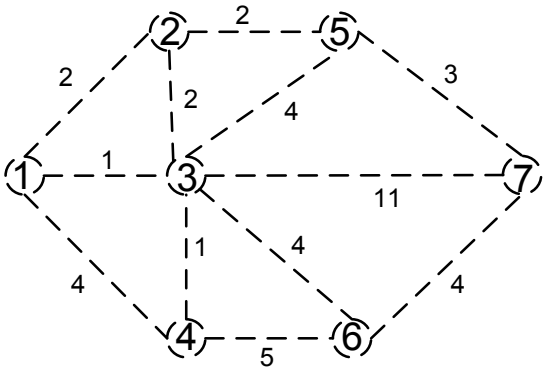


Figure 2-34. Unmark all nodes and spans in the network and capacitate the network with the distance of the spans. Now the number written beside each span represents the cost of it.

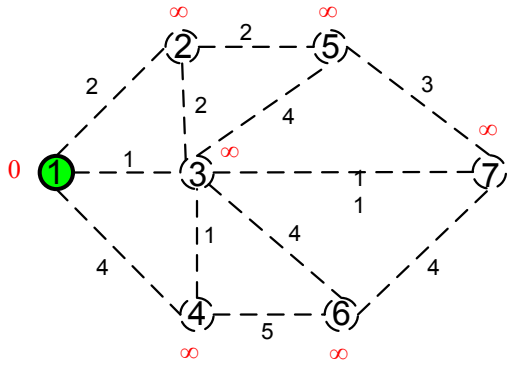


Figure 2-35. Mark node 1 and label it with 0 and label every other node in the network with ∞. These numbers (labels) beside each node represent the costs from these nodes to node 1.

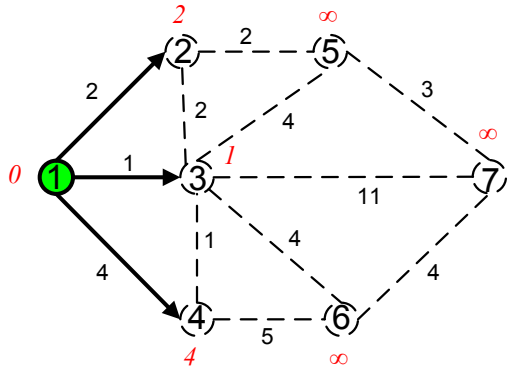


Figure 2-36. Find the node with the lowest label, which is currently node 1. Mark every scanned span that links node 1 to an adjacent node if this adjacent node will have a new label with a value lower than the current one. Update the labels using spans' costs.

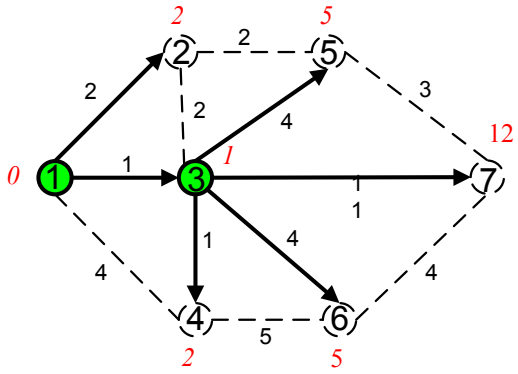


Figure 2-37. Similarly, find the unmarked node with the lowest label (node 3 because its label is 1) and mark it in the network. Mark every scanned span that links node 3 to an adjacent node if this adjacent node will have a new label with a value lower than the current one. Update the labels.

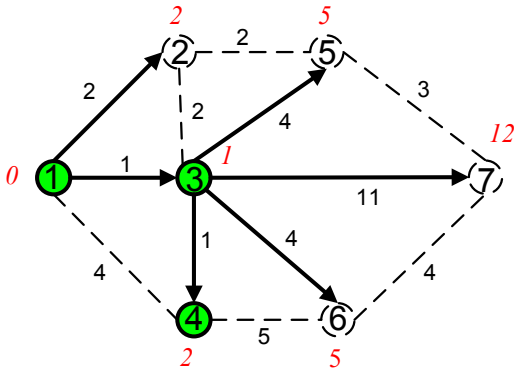


Figure 2-38. Find the unmarked node with the lowest label (node 4 or node 2 because their labels are 2) and mark any one of them. (In our example we picked node 4.) Mark every scanned span that links node 4 to an adjacent node if this adjacent node will have a new label with a value lower than the current one. Update the labels.

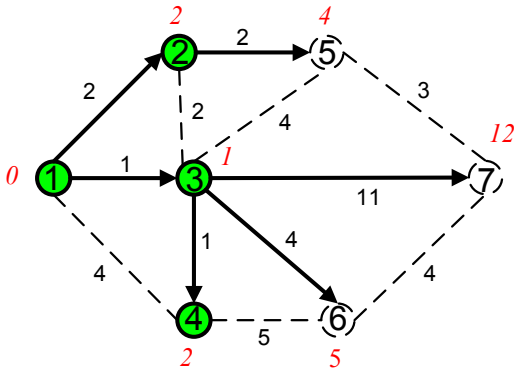


Figure 2-39. Similarly, find the unmarked node with the lowest label (node 2 because its label is 2) and mark it. Mark every scanned span that links node 2 to an adjacent node if this adjacent node will have a new label with a value lower than the current one. Update the labels.

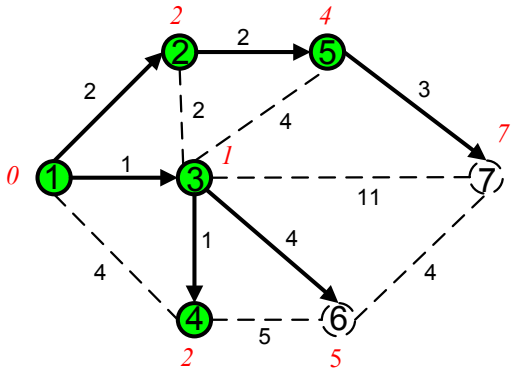


Figure 2-40. Find the unmarked node with the lowest label (node 5 because its label is 5) and mark it. Mark every scanned span that links node 5 to an adjacent node if this adjacent node will have a new label with a value lower than the current one. Update the labels.

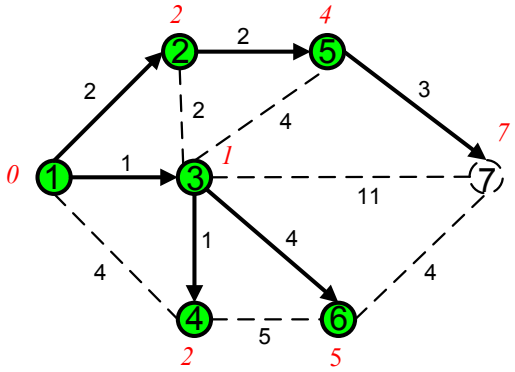


Figure 2-41. Find the unmarked node with the lowest label (node 6 because its label is 6) and mark it. Mark every scanned span that links node 6 to an adjacent node if this adjacent node will have a new label with a value lower than the current one. Update the labels.

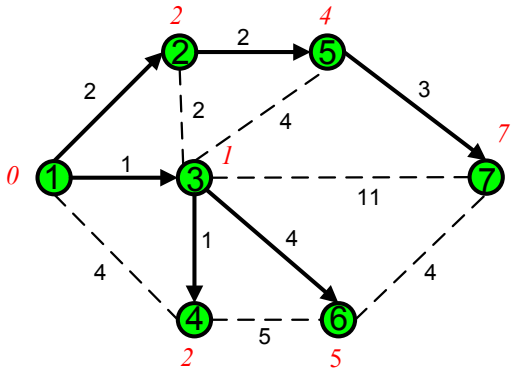


Figure 2-42. Find the unmarked node with the lowest label (node 7 because its label is 7) and mark it. Mark every scanned span that links node 7 to an adjacent node if this adjacent node will have a new label with a value lower than the current one. Update the labels.

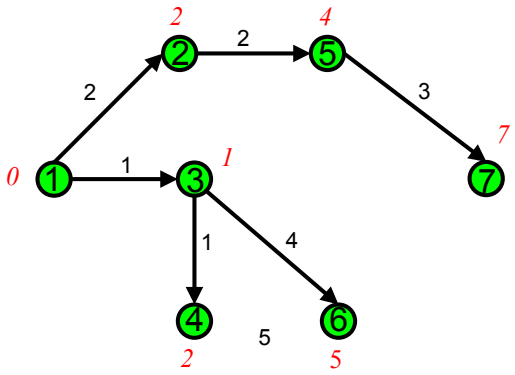


Figure 2-43. All nodes are marked, and the Dijkstra's tree with root node 1 of the graph is generated. The label associated with each node represents the cost from this node to node 1. For example, to reach node 1 from node 7, it costs 7 units, and its route is {7 - 5 - 2 - 1}.

2.4.3 Maximum Flow Algorithms

Many problems arise in telecommunications, and transportation fields require maximizing the flows (i.e., data flows and cars flows) from a set of supply nodes to a set of demand nodes. These problems are called maximum flow problems. Formally stated, given a maximum flow capacitated network G = (N, S), find a flow of maximum value connecting nodes *x* to *y*.

2.4.3.1 K Shortest Path Routing Algorithm

The *k-shortest path* routing algorithm (KSP) is an extension algorithm of the shortest path routing algorithm in a given graph. It is sometimes essential to have more than one path between two nodes in a given network. In the event, there are additional restrictions, other paths different from the shortest path can be computed. To find the shortest path one can use shortest path algorithms such as Dijkstra's algorithm and extend them to find more than one path. The algorithm not only finds the shortest path, but also K other paths in order of increasing cost. K is the number of shortest paths to find. The K shortest path routing is a good alternative for geographic path planning and network routing, especially in optical mesh networks where there are additional restrictions that cannot be solved by using ordinary shortest path algorithms.

A practical demonstration of the KSP algorithm is illustrated on graphs in Figure 2-44 to Figure 2-50. The main objective of this example is to find the maximum flow from node 1 to node 7 without violating the spans' maximum flows in Figure 2-44. The sequence of the algorithm is as follows:

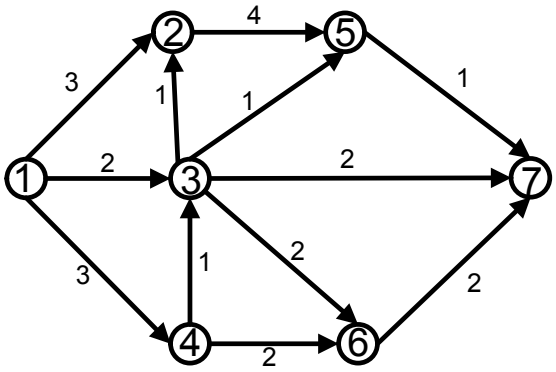


Figure 2-44. This is the original graph capacitated with the spans' maximum flows. For instance, the maximum flow from node 2 to node 5 is 4 capacity units.

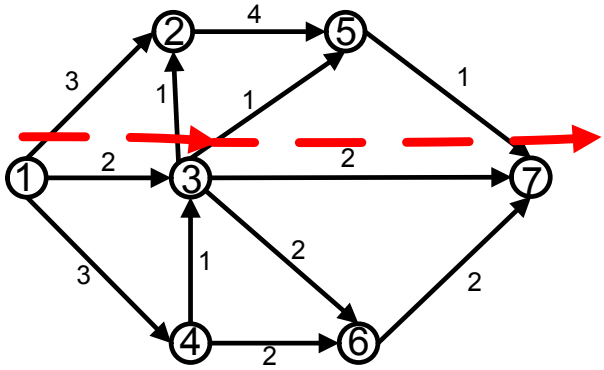


Figure 2-45. Find the shortest path using Dijkistra's algorithm from node 1 to node 7 in the graph, which is path {1 – 3 – 7}.

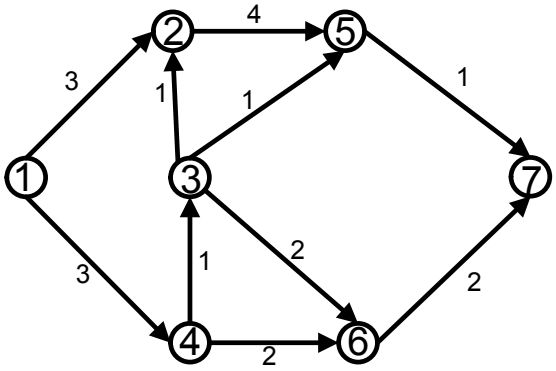


Figure 2-46. Determine the maximum flow of the path, which is 2 units, and then deduct the same number of units from the path.

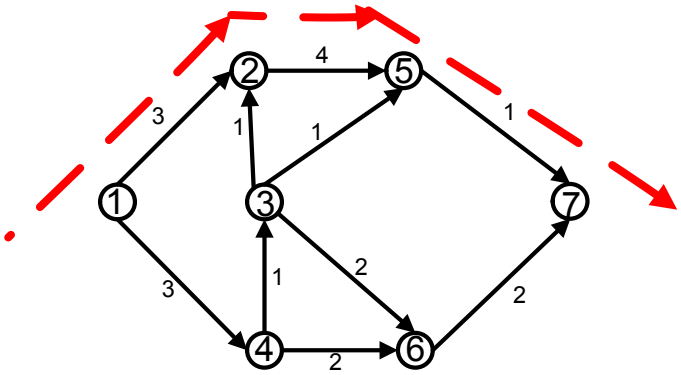


Figure 2-47. Find the shortest path using Dijkstra's algorithm from node 1 to node 7 in the residual graph, which is path {1 – 2 – 5 – 7}.

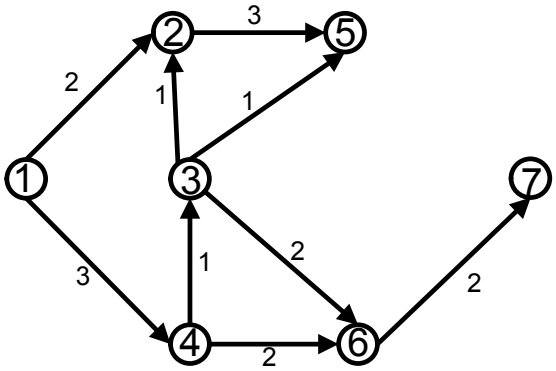


Figure 2-48. Determine the maximum flow of the path, which is 1 unit, and then deduct the same number of units from the path.

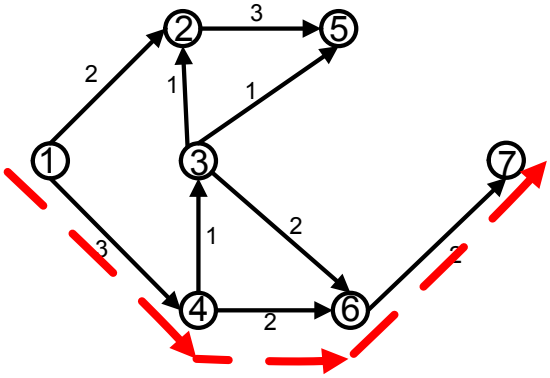


Figure 2-49. Find the shortest path using Dijkstra's algorithm from node 1 to node 7 in the residual graph, which is path {1 – 4 – 6 – 7}. Determine the maximum flow.

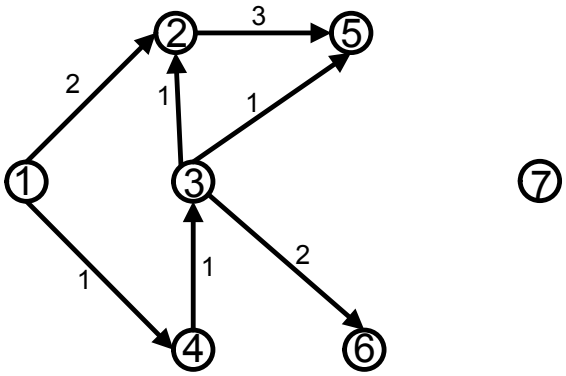


Figure 2-50. Stop when there are no more paths to be found. The maximum flow is the summation of the 3 shortest paths flows found so far, which is 5 units.

2.4.3.2 Trap Topology

In the following example, we will show how the KSP is trapped by its greedy nature. The original graph in Figure 2-51 illustrates the trap situation. For simplicity, each span has a maximum capacity of 1 unit. For instance, the maximum flow from node 2 to node 5 is 1 capacity unit. As shown in Figure 2-51 KSP will take the shortest path {1 – 3 – 7 – 8}, making it impossible to discover any more paths, as depicted in Figure 2-52. To solve this problem, the Ford-Fulkerson algorithm (FFA) was developed to compute the maximum flow in a flow network. It was published in 1956 [36], [29].

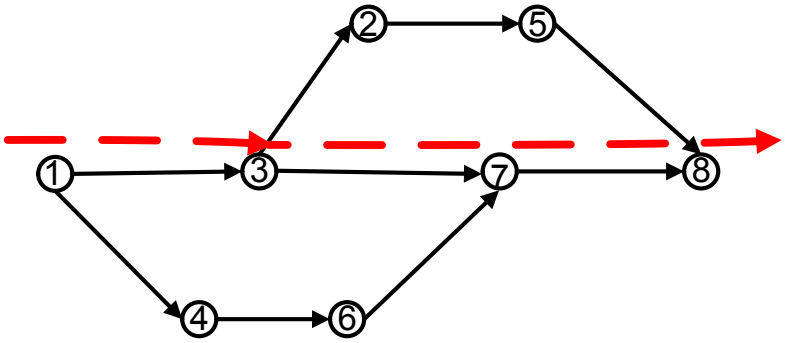


Figure 2-51. Find the shortest path using Dijkstra's algorithm from node 1 to node 8 in the graph, which is path {1 – 3 – 7 – 8}.

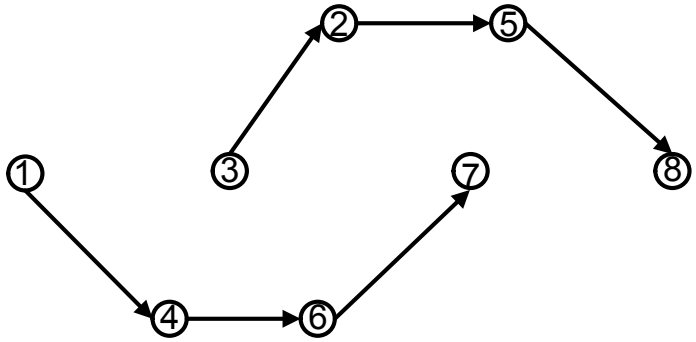


Figure 2-52. Determine the maximum flow of the path, which is 1 unit, and then deduct the same number of units from the path.

Chapter 3. Survivability Schemes in WDM Networks

3.1 Introduction

As we introduced in the previous chapter, any telecommunication network comprises a number of elements [37] A *node* is any device that can transmit or receive data. In the context of WDM networks, this could mean optical cross-connects (OXCs) or add/drop multiplexers. A *link* is a channel that connects two nodes. A *span* is a group of links that connects two nodes and can be exposed to one physical failure. A *path* is a series of links that makes a connection between the source node and the destination node. A *route* is a series of cascaded spans from one node to another. A *demand* is the amount of data that would be transferred between any pair of nodes in the network.

Our main focus in this research is survivability mechanisms for single-failure accident. Other mechanisms were proposed in the open literature for dual failure, such as in [38], [39]. As the global traffic was increased more than fivefold in the last five years, an accident failure could result in a huge loss of data, and as a consequence, significant financial loss [40]. To avoid such a disaster, survivability mechanisms must be deployed in such networks. The survivability techniques in WDM networks can be classified into two main categories. First is ring protection, where the network is protected by rings of spare capacities included in the network design so that in the case of a span failure, the affected signal will be rerouted through the spare capacities in these rings. Second, mesh restoration exploits some of the spare capacities in the spans of the network to construct new routes. Mesh restoration is more flexible than ring restoration, in that, it can be more easily adapted for different traffic loads. One type of mesh restoration technique called *p*-cycles incorporates aspects of ring networks [41].

The two main types of ring restoration are:

a) Unidirectional path switched ring (UPSR) [42].
b) Bidirectional line-switched rings (BLSR) [11].

The mesh restoration types include:

a) Automatic protection switching (APS) [43].
b) Span restoration [11].
c) Path restoration [44].
d) Shared backup path protection (SBPP) [45], [46].
e) *p*-Cycles [47], [48].
f) Node-encircling *p*-cycle (NEPC) [41].

3.2 Ring Restoration

3.2.1 Unidirectional Path Switched Ring

A *unidirectional path switched ring* (UPSR) is the most widely used scheme in SONET networks. Because of its low cost and simplicity, it can be used for access networks and metropolitan networks. As shown in Figure 3-1, in this scheme, the nodes are arranged in a closed ring containing two sets of fibre optic cables. The demand will use one of these cables to transmit its signal between its nodes. If a failure happens in a certain span on the working path, the demand will be rerouted to use the fibre on the other side of the ring between its nodes. The spare capacity in a UPSR equals the total of all demands between the nodes in this ring [32]. Protection switching times for a UPSR are normally within the 50 ms range [42].

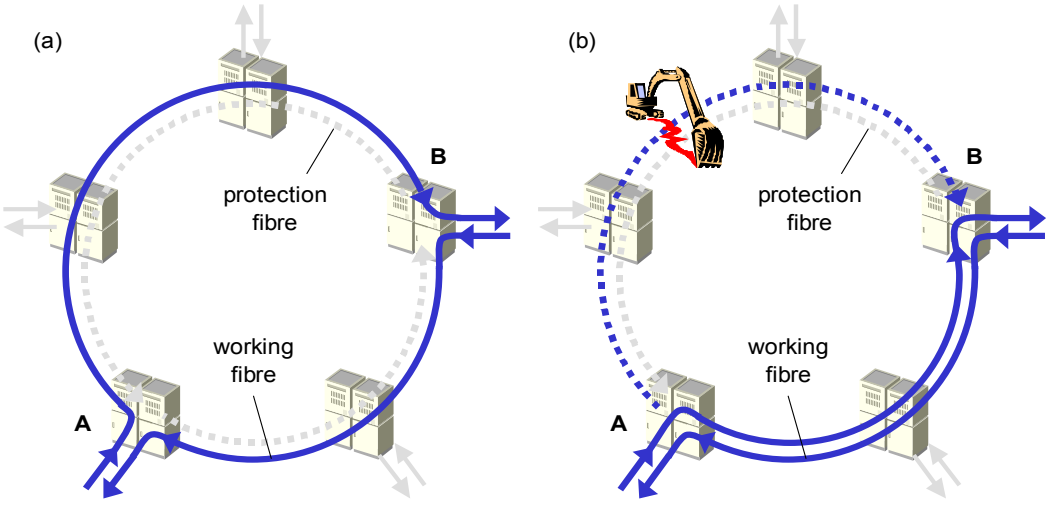


Figure 3-1. Basic operation of a UPSR (a) before failure, and (b) after failure [37], used with permission

3.2.2 Bidirectional Line-Switched Rings

In *bidirectional line-switched rings* (BLSR), as shown in Figure 3-2, the nodes are also organized in a ring topology, except that the ring usually contains four sets of fibre optic cables. Working traffic uses two sets of cables on one side of the ring to connect its nodes. When a failure occurs in a span, the nodes of this span change the signal from the working sets of cable to the spare sets on the other side of the ring. This method requires less capacity than in UPSRs. The spare capacity in BLSR equals the maximum load between any two nodes in that ring [32], [11], [42].

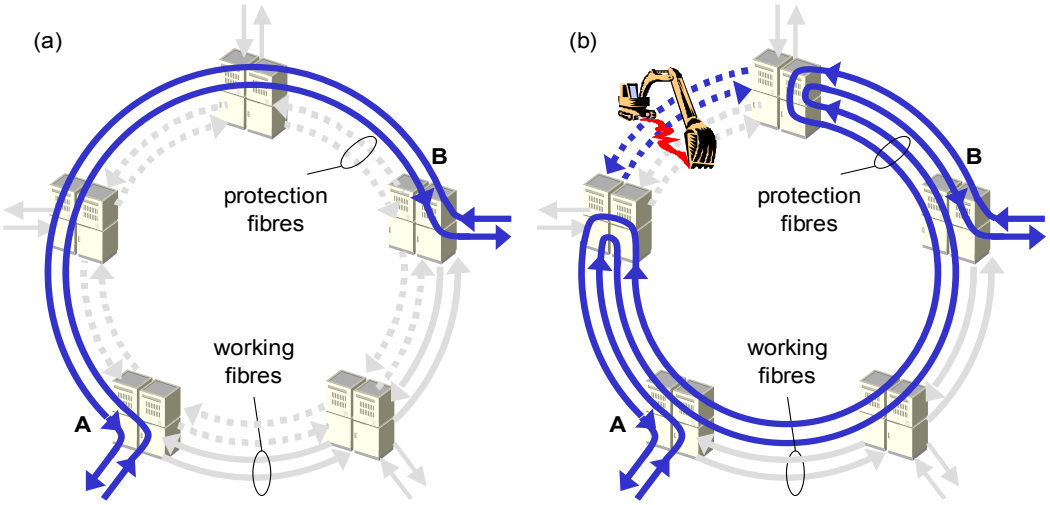


Figure 3-2. Basic operation of a BLSR (a) before failure, and (b) after failure [37], used with permission

3.3 Mesh Restoration

3.3.1 Automatic Protection Switching (APS)

In *automatic protection switching* (APS) systems, for each working route, there is a predetermined protection route. In the case of the failure of a working route; the protection route will enter in the service. The protection switching can be done whether the signal disappears entirely, or there is simply a degradation in the signal strength. It can also be done manually by a user. Depending on the specific implementation, traffic can revert back to the original working route after the failure is repaired or in the non-reverting case, the traffic remains on the spare route after repair (effectively becoming the new working route) [43].

There are three main types of APS, 1+1 APS, 1:1 APS, and 1:N APS. In 1+1 APS systems, a copy of the signal is sent on both routes. One of these routes is called the working route and the other is called the protection route. The receiver monitors both routes and takes the signal from the route with the higher performance. Because of that, this scheme provides the minimum restoration time of schemes.

In 1:1 APS systems, during normal operations the working signal is carried by the working route while the protection route can carry another signal, which is not related to the signal on the original working route. When the working route fails, the signal of the working route will be transmitted on the protection route canceling the transmission of any other signal on this route. Thus, the signal in the working route should have higher priority than the signal on the protection route. It is clear that the restoration time of this system is greater than the restoration time of the previous one.

1:N APS is a more capacity-efficient scheme where the goal is to protect only against individual channel failures rather than entire cable failures, as illustrated in Figure 3-3. In 1:N APS, a single backup channel is shared amongst N working channels. In the case of a failure of a working channel, the receiving end first confirms that the protection channel is available and then sends a signal to the other end to launch a head-end bridge of the failed working channel onto the backup channel before performing a tail-end handover. *k*:N APS is a more general system of 1:N APS where there are *k* backup channels available instead of only one [37].

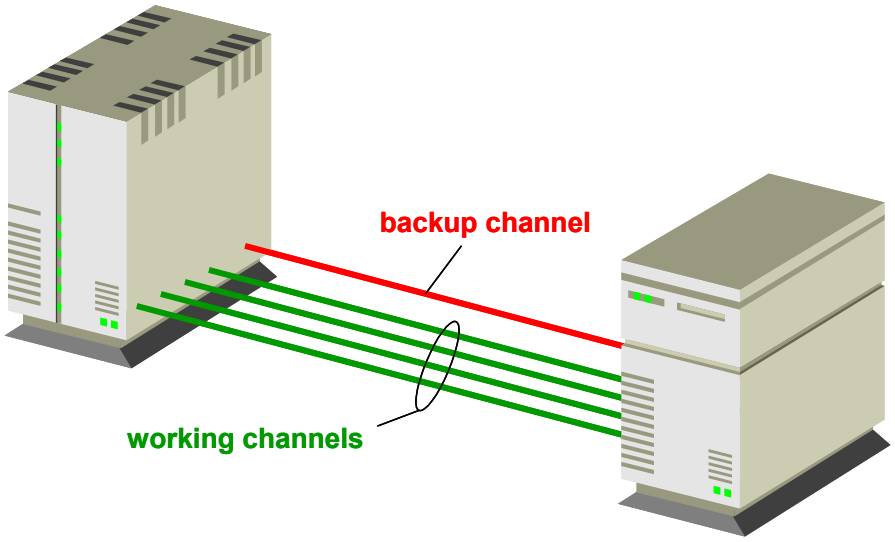


Figure 3-3. Automatic protection switching [37], used with permission.

3.3.2 Path Restoration

Path restoration is an end-to-end technique, in that, when a span fails, the affected working route is replaced by a new one from the origin node to the destination (see Figure 3-4). The surviving spans on an affected working route will be released as spare capacity to be used in the restoration process for any of the affected working paths. This operation is called stub release. Stub release helps the path restoration to achieve better capacity efficiency, but it also complicates the reversion process after repairing the failure. The new path is not fixed; it can be changed from time to time depending on the spare capacity units on the spans of the networks. Path restoration can be considered a multi-commodity max-flow problem, where we have multiple groups of paths and want to arrange them with the maximum flow [11]. A path restoration scheme is also called *failure dependent path protection* (FDPP), in that, the restoration paths for any given failed working route will depend on the failed span [44], [49], [11].

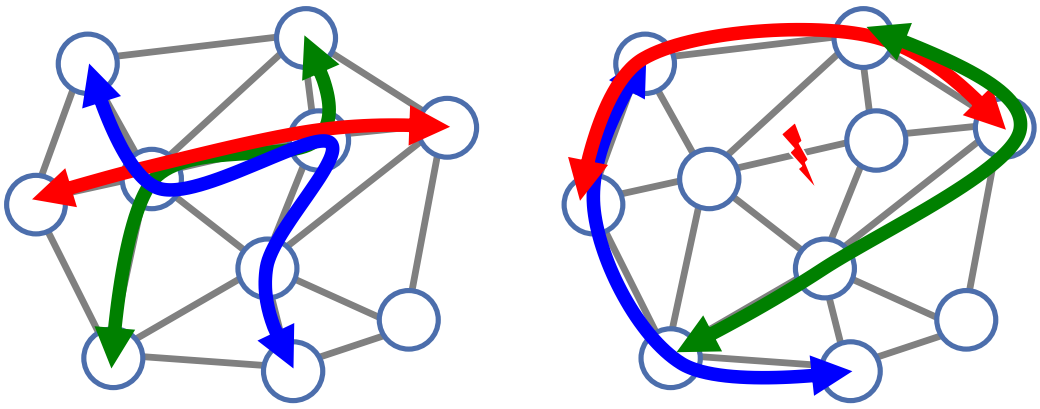


Figure 3-4. Path restoration before failure (left panel) and after failure (right panel)

3.3.3 Shared Backup Path Protection

Shared backup path protection (SBPP) is another end-to-end protection scheme similar to 1:1 APS, except the spare capacity for a restoration path is also available to any other restoration path from any other demand, so long as the working paths are link-disjoint paths. As show in Figure 3-5, SBPP operates in a similar manner to path restoration, except that it does not utilize the stub release mechanism and the backup route for a working path is not dependent on the location of the failure [37]. In addition, the selection of restoration paths for each failed working path does not require knowledge of where the failure occurred, as in the case of path restoration. SBPP is less capacity efficient than path restoration, but it is also less complex than path restoration [45], [50].

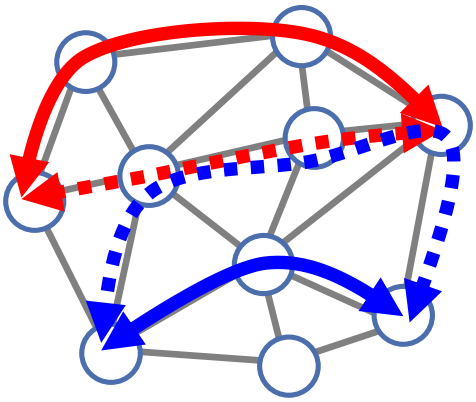


Figure 3-5. Shared backup path protection [37], used with permission

3.3.4 Span Restoration

Span restoration is the replacement of the failed span by paths between the nodes of failure through the rest of the network. These paths use spare capacity distributed throughout the spans of the network, as shown in Figure 3-6. Span restoration can be considered a single-commodity max-flow problem where we have a single group of paths and want to arrange them with the maximum flow [51], [52], [11].

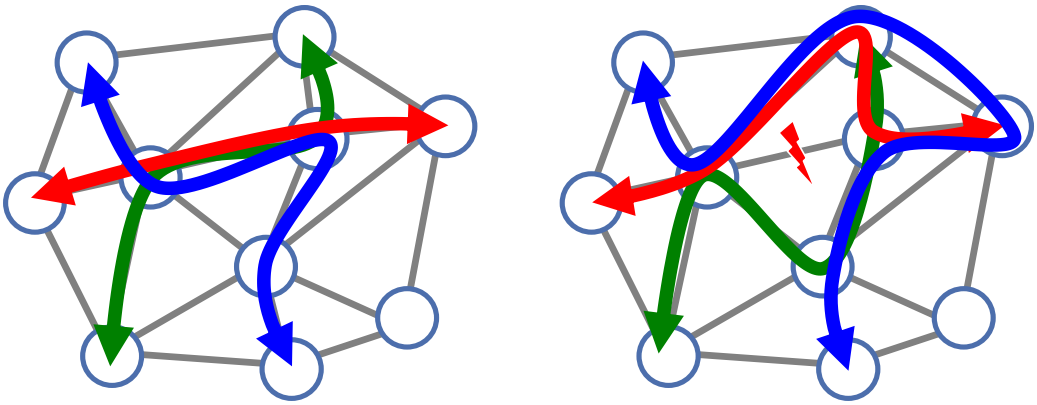


Figure 3-6. Span restoration

While path restoration is more capacity efficient, span restoration is simpler and faster. The *meta-mesh* scheme, to be discussed later, has been proposed to bridge the gap between the two [53], because it enhances the capacity efficiency for the span restorable mesh networks while maintaining much of its simplicity.

3.3.5 *p*-Cycles

p-Cycles are a network survivability approach that was developed as a compromise between the faster ring-survivability approaches and the much more capacity-efficient span restoration [47]. *p*-Cycles are cyclic structures of pre-connected spare capacity, as illustrated in Figure 3-7. Each unit-sized copy of the *p*-cycle can protect one unit of capacity on each of the on-cycle spans. In straddling span protection, each unit-sized copy of a *p*-cycle can be used to protect two units of capacity on each and every straddling span. *p*-Cycle is a well-known survivability scheme that was originally intended to protect against span failure. In the current literature, four *p*-cycles approaches have been developed for span and node protection, namely, *node-encircling p-cycles* (NEPC) [41], *failure-independent path-protecting* (FIPP) *p-cycles* [54], *path-segment protecting p-cycles* [55], and *2-hops node protection p-cycles* [56]. Of these variants, we will discuss only NEPCs in details.

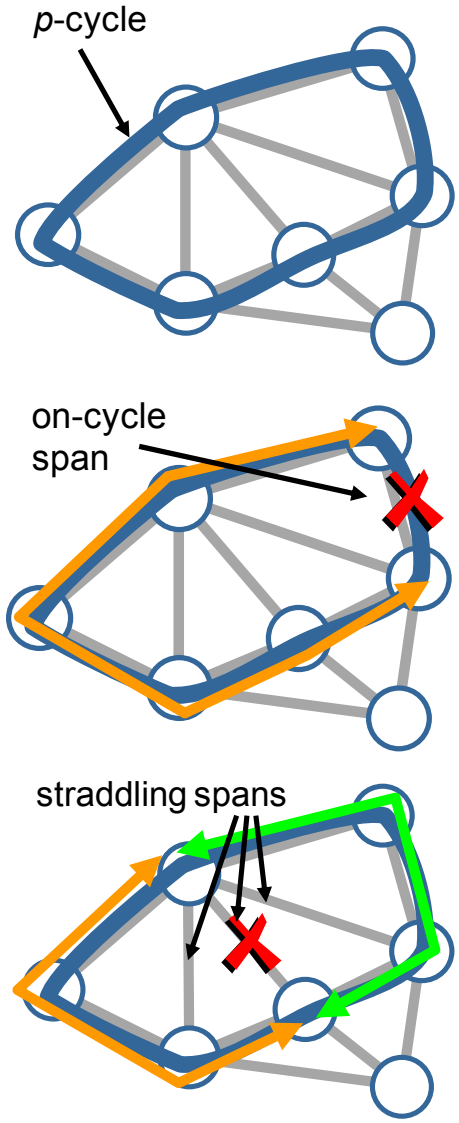


Figure 3-7. Illustration of *p*-cycle restoration

3.3.6 Node-Encircling *p*-Cycles

Since conventional *p*-cycles are fundamentally a form of span restoration, they are capable of protecting only span failures, not node failures. The idea of node-encircling *p*-cycles was introduced in the context of Internet Protocol (IP) layer restoration to protect against router failures [57], but the concept is also applicable to optical layer restoration. An NEPC functions by providing protection for any lightpaths transiting the failed node. In NEPC survivability, when a node fails, an

NEPC is able to protect any flow passing through that node (i.e., transiting flow) that does not originate or terminate at it. By definition, transiting flow through an encircled node also passes at least two other nodes on the NEPC. This NEPC protects the node by routing the transiting flow around the *p*-cycle in either direction. Figure 3-8 illustrates this behavior. The cycle A-B-C-D-E is an NEPC for node G. If node H fails, transiting flow passing through the A-G-D route segment can be re-routed around the failed node, through A-B-C-D and/or A-E-D (one unit of transiting flow in each direction).

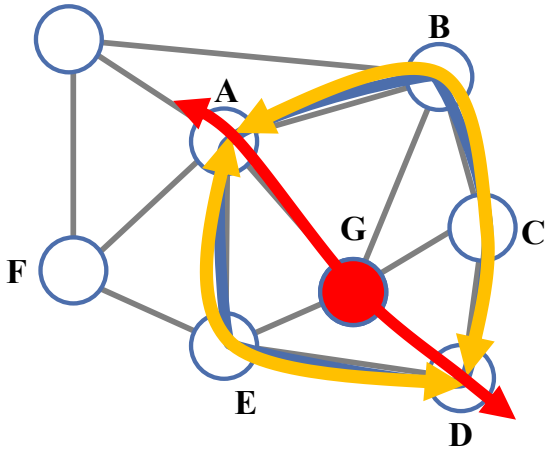


Figure 3-8. NEPC intercepting on node transiting flow

Chapter 4. Network Optimization Methods

Typically, there are two main approaches for tackling any optimization problem, as illustrated in Figure 4-1. First, exact methods find the optimal solutions. The time complexity and the memory required for these methods increase exponentially with the increase of the problem size. Second, approximate methods provide a respectively good solution within a polynomial time, but they do not guarantee finding the optimal solution.

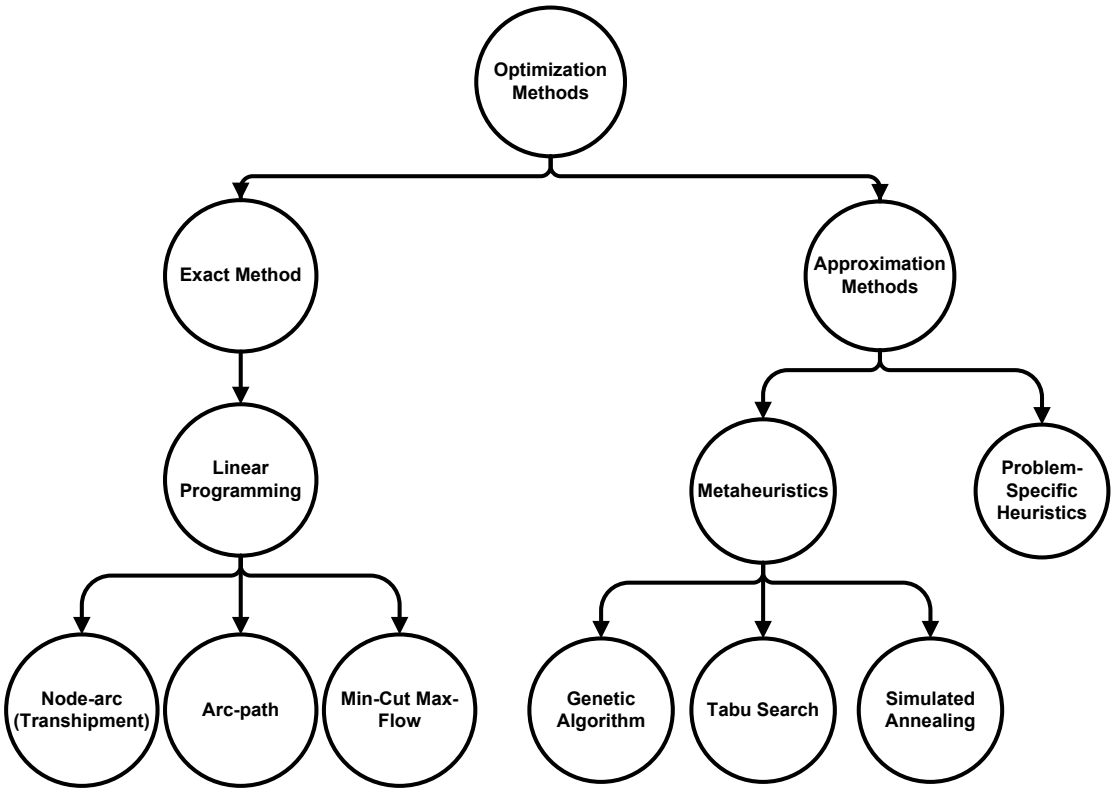


Figure 4-1. Network optimization methods

4.1 Integer Linear Programming

Operations research (OR) is the science of how to formulate a mathematical model for any complex engineering or management problems. One of its analytical methods is the use of exact optimization tools. Optimization handles the maximization or minimization of an objective function subject to given constraints. Decision variables are the values assigned by the decision maker to optimize the mathematical model. The objective function and its constraints are

called a *mathematical optimization model*. A *linear program* (LP) is a mathematical optimization model in which the objective function and all constraints are linear. A typical LP model can be formulated as:

Maximize:

$$\sum_{\forall i} c_i \, x_i \tag{4.1}$$

Subject to:

$$\sum_{\forall i \in I} a_{ij} \, x_i \; \leq \; \mathrm{b}_j \qquad \forall j \in J \tag{4.2}$$

$$x_i \geq 0 \qquad \forall i \in I \tag{4.3}$$

Where x_i represents the decision variables and c_i is the vector of coefficients used in the objective function. $\sum_{\forall i \in I} a_{ij} x_i \leq b_j$ represents a set of constraints that should be satisfied. An LP is called an *integer linear programming* (ILP) problem if its decision variables are discrete. If some of its decision variables are discrete and the rest are linear, this model is called *mixed integer linear programming* (MIP) model. An LP model is called a *binary integer programming* (BIP) model if some of its variables are binary.

An ILP technique is one of the main tools used in telecommunication networks design, owing to the fact that it provides the optimal solution within a reasonable amount of time for small instances of problems. Moreover, ILP models could work as accurate baselines when designing an approximate algorithm for the same problem. The following section provides some ILP models for major survivability schemes demonstrated in the previous chapter, such as span restoration, path restoration, *p*-cycle, and NEPCs.

4.2 ILP Models for Major Survivability Schemes

Network optimization can take one of two forms when considering survivability techniques. First is the *spare capacity allocation* (SCA) form, where the working

capacities have been calculated previously. Typically, the working capacities are determined from the shortest path routing of the demand matrix. The main target of this problem is to determine the spare capacity units on each span while maintaining 100% restorability for a single failure [58], [59], [60]. Second is the *joint capacity allocation* (JCA) form. Its aim is to calculate the optimal working and spare capacities simultaneously. Several studies [49], [61], [62] have shown that JCA can yield a considerable total capacity reduction compared to SCA.

Two fundamentally different ILP models have been developed for most network survivability problems in the literature [49], [61], [62]. First is the arc-path model where restoration flow is assigned to a set of predefined eligible routes. Second is the node-arc model [63], which originates from transshipment problems. In such models, the flow variables are associated with spans rather than specific paths in the network. For any given demand, its origin and its destination nodes represent source and sink of a commodity, respectively, while the transshipment nodes act as transit nodes that pass the commodity to another node until the commodity reaches the destination node.

In most of the current research, the physical topology has been given as an input to the network model. This means that it has only considered the variable cost that corresponds to the addition of a new unit of spare or working capacity. However, the fixed cost that represents the rights-of-way and lease acquisitions, excavation, duct installation, amplifiers, etc., has not been taken into the account. In these kinds of problems, the main target is to route the working and the spare capacity flows with the least cost on a legacy network.

4.2.1 Span Restoration ILP Models

In this section, we will introduce the ILP models used in span restoration. The first model is designed using the arc-path technique. The second model is designed using the node-arc technique.

4.2.1.1 Arc-Path SCA Model for Span Restoration Technique

In this section, we will introduce the arc-path SCA model for span restoration. The following notations will be used in the SCA arc-path model:

Sets:

S Set of all spans in the network typically indexed by two symbols *i,j,* where *i* refers to a failing span and *j* refers to a surviving span.

P_i Set of all unique potential routes available to carry the restoration flow for span *i*. It is indexed by *p*.

Parameters*:*

c_j The parameter which represents the incremental cost of adding one unit of working or spare capacity on span *j*.

w_i The number of working capacity units assigned on span *i* and that need to be protected.

$\delta_{i,j}^{p} \in \{0,1\}$ The parameter that represents the relation between the failed span *i*, the restoration route *p*, and a surviving span *j*. $\delta_{i,j}^{p} = 1$ if restoration route *p* used for restoration of span *i* passes through span *j*. $\delta_{i,j}^{p} = 0$ if restoration route *p* used for restoration of span *i* does not pass through span *j*.

Decision Variables:

s_j This decision variable represents the total number of spare capacity units deployed on span *j*.

$f_{i,p} \geq 0$ This decision variable denotes the number of spare capacity units assigned on route *p* to restore span *i.*

The objective is to minimize the cost of the network subject to a set of constraints.

Minimize:

$$\sum_{\forall j \in S} c_j \cdot s_j \tag{4.4}$$

Subject to:

$$\sum_{\forall p \in P} f_{i,p} = w_i \qquad \forall i \in S \tag{4.5}$$

$$\sum_{\forall p \in P} \delta_{i,j}^{p} \cdot f_{i,p} \leq s_j \qquad \forall i, j \in S \,|\, i \neq j \tag{4.6}$$

In the above SCA model, the objective function (4.4) minimizes the total cost results from the total spare capacities cost. To optimize this objective function, we need to define the feasible region over which the search will be performed. Equation (4.5) ensures that the total number of spare capacity units assigned on all routes satisfies the working capacities on each span. Equation (4.6) guarantees that enough spare capacity units are provided on each span to protect any single failed span against.

4.2.1.2 Arc-Path JCA ILP Model for the Span Restoration Technique

In this section, the arc-path model for span restoration will be introduced by modifying the previous SCA model. This means that the working capacity that was given as an input parameter in the previous model will be considered output variables in this model. In addition to the previous notation used in the SCA model, the following additional notation will be used in the JCA model:

Additional Sets:

D Set of all demands in the network. It is indexed by *r*.

Q^r Set of all unique potential routes available to carry the working flow for demand *r*. It is indexed by *q*.

Additional Parameters*:*

d_r The parameter that represents the number of demand units for

demand *r*.

$\zeta_j^{r,q} \in \{0,1\}$ The parameter that represents the relation between the span *j*, the working route *q*, and the demand *r*. $\zeta_j^{r,q} = 1$ if working route *q* used for restoration of demand *r* passes through span *j*. $\zeta_j^{r,q} = 0$ if working route *q* used for routing of demand *r* does not pass through span *j*.

Additional Decision Variables:

w_j This decision variable represents the total number of working capacity units deployed on span *j*.

$g^{r,q} \geq 0$ This decision variable denote the number of spare capacity units assigned to route *q* to route demand *r*.

Minimize:

$$\sum_{\forall j \in S} c_j \cdot (w_j + s_j) \tag{4.7}$$

In addition to the constraints applied in the previous section, the following constraints will be used:

$$\sum_{\forall q \in Q} g^{r,q} = d_r \qquad \forall r \in D \tag{4.8}$$

$$\sum_{\forall r \in D} \sum_{\forall q \in Q} \zeta_j^{r,q} \cdot g^{r,q} \leq w_j \qquad \forall j \in S \tag{4.9}$$

In the above arc-path JCA model, the new objective function (4.7) ensures minimizing the total cost results from the total working and spare capacity cost. To optimize this new objective function, we need to define additional of constraints for controlling the working capacities. Equation (4.8) ensures that the total number of working capacity units assigned to all routes satisfies the working capacities for each demand. Equation (4.9) guarantees that enough working capacity units are provided in each span to route all the demands.

4.2.1.3 Node-arc SCA ILP model for the Span Restoration Technique

In this section, we will introduce the node-arc SCA model for the span restoration. The following notation will be used in the node-arc SCA model:

Sets:

N Set of all nodes in the network. It can be indexed by *i, j, n, k, l, b, or q*.

S Set of all spans in the network typically indexed by two pair of nodes like *i,j,* which represents a directional span from node *i* to node *j*.

Parameters*:*

$c_{i,j} = c_{j,i}$ The parameter that represents the incremental cost of adding one unit of capacity on span *i,j*.

$w_{i,j}$ The number of working capacity units assigned to span *i,j*.

Decision Variables:

$s_{i,j}^{i,k}$ This decision variable denote the number of spare capacity units assigned to span *i,k* to restore span *i,j*.

Minimize:

$$\sum_{\forall i,j \in S} c_{i,j} \cdot \quad (4.10)$$

In addition to the constraints applied in the previous section, the following constraints will be used:

$$\sum_{\forall i,k \in S | j \neq k} s_{i,j}^{i,k} = w_{i,j} \qquad \forall i,j \in S \quad (4.11)$$

$$\sum_{\forall i,k \in S | j \neq k} s_{i,j}^{k,i} = 0 \qquad \forall i,j \in S \quad (4.12)$$

$$\sum_{\forall j,k \in S | i \neq k} s_{i,j}^{k,j} = w_{i,j} \qquad \forall i,j \in S \quad (4.13)$$

$$\sum_{\forall j,k \in S | i \neq k} s_{i,j}^{j,k} = 0 \qquad \forall i, j \in S \tag{4.14}$$

$$\sum_{\forall n,k \in S} s_{i,j}^{n,k} - \sum_{\forall n,k \in S} s_{i,j}^{k,n} = 0 \qquad \forall i, j \in S, \forall n \in N \mid n \notin \{i, j\} \tag{4.15}$$

$$s_{k,i} \geq s_{i,j}^{k,i} \qquad \forall i, j \in S \tag{4.16}$$

The objective is to minimize the total cost of the network spare capacity units subject to technical constraints. The six constraints will be used to assign the number of spare units over the network spans to ensure that the network is fully restorable. Each span considered as a demand, where the working capacity on this span will be treated as a demand. One of the nodes incident on this span will be considered an origin and the other a virtual target. Constraints (4.11) and (4.12) denote that, the summation of the flows units out of *i* (i.e., the virtual origin) must equal the working capacity units on this span, and the flows in *i* must equal zero, respectively. Equations (4.13) and (4.14) say that the summation of the flow units in *j* (i.e., the virtual target) must equal the working capacity on this span, and the flows out of *j* must equal zero respectively. The constraint in (4.15) represents the conservation law, in that, for any demands at any node other than the virtual origin or the virtual target, the summation of flows out should equal the summation of flows in. The inequality in (4.16) guarantees that the number of spare units deployed on any span will be sufficient for restoring any failed span.

4.2.1.4 Node-arc JCA ILP Model for the Span Restoration Technique

In this section, the node-arc model for the span restoration will be introduced by modifying the node-arc SCA model. This means the working capacity that was given as an input parameter in the previous model will be considered output variables in this model. In addition to the previous notation used in the node-arc SCA model, the following additional notation will be used:

Additional Sets:

D Set of all demands in the network. It is indexed by *r*.

Additional Parameters:

d_r The parameter that represents the number of demand units for demand *r*.

O_r, T_r Two symbolic parameters used to determine the origin and the target, respectively, of a demand *r*. Their value must be belong to the set *N*.

Additional Decision Variables:

$w_{i,j}$ The number of working capacity units assigned to span *i,j*.

$w_{i,j}^r$ The decision variables that represent the number of working capacity units deployed on span *i,j* for demand *r*.

$s_{i,j}$ The number of the spare capacity units deployed on *i,j* span that are used for restoration.

Minimize:

$$\sum_{\forall i,j \in S} c_{i,j}(s_{i,j} + w_{i,j}) \tag{4.17}$$

Subject to:

$$\sum_{\forall i,j \in S | i=n} w_{i,j}^r = d_r \quad \forall r \in D, \forall n \in N \mid n = O_r \tag{4.18}$$

$$\sum_{\forall i,j \in S | j=n} w_{i,j}^r = 0 \quad \forall r \in D, \forall n \in N \mid n = O_r \tag{4.19}$$

$$\sum_{\forall i,j \in S | j=n} w_{i,j}^r = d_r \quad \forall r \in D, \forall n \in N \mid n = T_r \tag{4.20}$$

$$\sum_{\forall i,j \in S | i=n} w_{i,j}^r = 0 \quad \forall r \in D, \forall n \in N \mid n = T_r \tag{4.21}$$

$$\sum_{\forall i,j \in S | j=n} w_{i,j}^r - \sum_{\forall i,j \in S | j=n} w_{i,j}^r = 0 \quad \forall r \in D, \forall n \in N \mid n \notin \{O_r, T_r\} \tag{4.22}$$

$$w_{i,j} = \sum_{\forall r \in D} w_{i,j}^r \quad \forall i,j \in S \tag{4.23}$$

The objective is to minimize the cost of the network, including the working and spare capacity costs, subject to technical constraints. The constraint in equation (4.18) says that for any demand, the number of working capacities unities that flow out of the demand's origin node must equal the number of demand units for this demand. The condition in (4.19) requires that all flows into a demand's origin should be zero for this demand. The equation in (4.20) requires that for any demand, the summation of flows into the target node should equal the demand units for this demand. The condition in (4.21) denotes that all flows out of the demand's target should be zero for this demand. The constraint in (4.22) represents the conservation law, where for any demand at any node other than its target or the destination, the summation of the flows out should equal the summation of the flows in. Equation (4.23) guarantees that the number of working units deployed on any span will be sufficient to accommodate all demands passing through this span.

4.2.2 *p*-Cycle Protection ILP Models

In this section, we present SCA and JCA ILP formulations for the *p*-cycle network design problems [11].

4.2.2.1 SCA ILP Formulation for *p*-Cycle

In this sub-section, we present the formulation of the network design for determining the spare capacity for the fixed amount of working capacity using *p*-cycles.

We use the following notation:

Sets:

S	The set of all spans in the network, typically indexed by i or j.
P	The set of all eligible p-cycles in the network, typically indexed by p.

Parameters:

$c_{i,j}$ The cost of each unit of capacity (working or spare) placed on span *i,j*.

$x_{i,j,p} \in \{0,1,2\}$ An input parameter that encodes the number of protection relationships provided to span *i,j* by each unit-sized copy of eligible *p*-cycle *p*. $x_{i,j,p} = 2$ if span *i* straddles cycle *p*, $x_{i,j,p} = 1$ if span *i,j* is on cycle *p*, and $x_{i,j,p} = 0$ in all other cases.

$w_{i,j}$ The number of working units placed on span *i,j*.

Decision Variables:

$s_{i,j}$ The number of the spare units deployed on span *i,j* used for restoration.

n_p An integer decision variable that represents the number of copies of *p*-cycle *p* that will be used in this design.

The ILP formulation is as follows:

Minimize:

$$\sum_{\forall i,j \in S} c_{i,j} s_{i,j} \qquad \forall i,j \in \boldsymbol{S} \qquad (4.24)$$

Subject to:

$$w_{i,j} \le \sum_{\forall p \in \boldsymbol{P}} x_{i,j,p} \cdot n_p \qquad \forall i,j \in \boldsymbol{S} \qquad (4.25)$$

$$s_{i,j} = \sum_{\forall p \in \boldsymbol{P} | x_{i,j,p}=1} n_p \qquad \forall i,j \in \boldsymbol{S} \qquad (4.26)$$

In the above SCA model, the objective function (4.24) minimizes the total spare capacity cost. To optimize this objective function, we need to define the feasible region over which the search will be performed. Equation (4.25) ensures that the number of *p*-cycles satisfy the working capacities on each span. Equation (4.26) guarantees that enough spare capacity units are provided in each span to be protected by *p*-cycles.

4.2.2.2 JCA ILP Formulation Using *p*-Cycles

In this sub-section, we present the formulation of the network design for determining the working and spare capacities using *p*-cycles techniquees.

Additional Sets:

N	The set of all nodes in the network, typically indexed by i or j.
D	Set of all demands in the network. It is indexed by r.

Additional Parameters:

d_r	The parameter that represents the number of demand units for demand r.
O_r, T_r	Two symbolic parameters used to determine the origin and the target, respectively, of a demand r. Their value must be belong to the set N.

Additional Decision Variables:

$w_{i,j}$	The working units placed on span i,j.
$s_{i,j}$	The number of the spare units deployed on i,j span, used for restoration.
n_p	The number of the spare units deployed on i,j span, used for restoration.

The ILP formulation is as follows:

Minimize:

$$\sum_{\forall i,j \in S} c_{i,j}(s_{i,j} + w_{i,j}) \qquad \forall i,j \in \boldsymbol{S} \tag{4.27}$$

In addition to the constraints applied in the previous section, the following constraints will be used:

$$\sum_{\forall i,j \in \boldsymbol{S} | i=n} w_{i,j}^{r} = d_r \qquad \forall r \in D, \forall n \in N \mid n = O_r \tag{4.28}$$

$$\sum_{\forall i,j \in \boldsymbol{S} | j=n} w_{i,j}^{r} = 0 \qquad \forall r \in D, \forall n \in N \mid n = O_r \tag{4.29}$$

$$\sum_{\forall i,j \in \boldsymbol{S} | j=n} w_{i,j}^{r} = d_r \qquad \forall r \in D, \forall n \in N \mid n = T_r \tag{4.30}$$

$$\sum_{\forall i,j \in \boldsymbol{S} | i=n} w_{i,j}^{r} = 0 \qquad \forall r \in D, \forall n \in N \mid n = T_r \tag{4.31}$$

$$\sum_{\forall i,j \in \boldsymbol{S} | j=n} w_{i,j}^{r} - \sum_{\forall i,j \in \boldsymbol{S} | j=n} w_{i,j}^{r} = 0 \qquad \forall r \in D, \forall n \in N \mid n \notin \{O_r, T_r\} \tag{4.32}$$

$$w_{i,j} = \sum_{\forall r \in D} w_{i,j}^{r} \qquad \forall i, j \in S \tag{4.33}$$

The objective function in (4.27) seeks to minimize the cost of working and spare capacity. To optimize this new objective function, we need to define more constraints for controlling the working capacities. The constraints in equation (4.28) ensure that for any demand, the number of working capacity units flowing out of the demand's origin node must equal the number of demand units for this demand. The constraints in (4.29) require that all flows into or out of a demand's origin equal zero for this demand. Equations (4.30) and (4.31) are the equivalent constraints for the target node of a demand. The constraints in (4.32) represent the conservation of the flow requirement, where the working traffic flows into and out of the transshipment nodes for a demand are equal. Equation (4.33) guarantees that the number of working capacity units deployed on any span will be sufficient to accommodate all of the traffic passing through it.

4.2.3 NEPC Restoration ILP Models

In this section, we present SCA and JCA ILP formulations for the NEPC network design problems.

4.2.3.1 SCA Formulation Using NEPC

The notation used in the NEPC SCA model are described next.

Sets:

S The set of all spans in the network, typically indexed by *i* or *j*.

N The set of all nodes in the network, typically indexed by n.

P The set of all eligible p-cycles in the network, typically indexed by p. Note that we make no distinction here between conventional span-protecting p-cycles and NEPCs.

Parameters:

c_j The cost of each unit of capacity (working or spare) placed on span j.

$x_{i,p} \in \{0,1,2\}$ An input parameter that encodes the number of protection relationships provided to span i by each unit-sized copy of eligible p-cycle p. $x_{i,p}$= 2 if span i straddles cycle p, $x_{i,p}$ = 1 if span i is on cycle p, and $x_{i,p}$= 0 in all other cases. For the special case of non-simple cycles, $x_{i,p}$= 0 for on-cycle spans that are crossed twice by the cycle.

$x_p^n \in \{0,1\}$ An input parameter that encodes whether or not eligible p-cycle p can act as an NEPC for node n. $x_p^n = 1$ if it can and $x_p^n = 0$ if it cannot.

w_i The working units placed on span i.

τ_n Transiting flow, in units, for node n.

Decision Variables:

s_j The spare capacity allocation in unit-copies on span j.

n_p An integer decision variable that represents the number of copies of p-cycle p that will be used in this design.

The ILP formulation for the SCA model is:

Minimize:

$$\sum_{\forall j \in \boldsymbol{S}} c_j \cdot s_j \tag{4.34}$$

Subject to:

$$w_i \le \sum_{\forall p \in \boldsymbol{P}} x_{i,p} \cdot n_p \qquad \forall i \in \boldsymbol{S} \qquad (4.35)$$

$$\tau_n \le \sum_{\forall p \in \boldsymbol{P} \mid x_p^n = 1} 2 \cdot n_p \qquad \forall n \in \boldsymbol{N} \qquad (4.36)$$

$$s_j = \sum_{\forall p \in \boldsymbol{P} \mid x_{i,p} = 1} n_p \qquad \forall j \in \boldsymbol{S} \qquad (4.37)$$

The objective function in equation (4.34) minimizes the total spare capacity cost subject to the technical constraints (4.35), (4.36) and (4.37). Constraint (4.35) ensures a full span restoration in case of a single span failure. Constraint (4.36) guarantees a full node restoration, in that all transiting flows passing through this node will be restored over its NEPCs. Constraint (4.37) ensures sufficient spare capacity allocation on each span for every *p*-cycle selected in the design.

4.2.3.1 JCA Formulation Using NEPC

In this subsection, we will introduce the JCA approach for NEPC network design, which simultaneously determines optimal working and restoration routing (and working and spare capacity). The model is provided with a set of eligible working routes and eligible *p*-cycles, which are then optimally selected such that capacity costs are minimized. In addition to the notation provided in the previous SCA model, we use the following additional notation:

Sets

D	The set of all demands in the network, typically indexed by r.
Q^r	The set of all distinct eligible working routes capable of routing lightpaths for demand r, typically indexed by q.

Parameters:

$\zeta_i^{r,q} \in \{0,1\}$	A binary parameter that defines the relationship between working routes and the network spans for each demand. It equals 1 if working route q used for demand r passes through a

span *i*. Otherwise it equals 0.

$\phi_r^n \in \{0,1\}$ A binary parameter that equals 1 if node *n* is the origin or the destination of demand *r*. Otherwise, it equals 0.

$z_n^{r,q} \in \{0,1\}$ A binary parameter that describes the relationship between working routes and the network nodes for each demand. $z_n^{r,q} = 1$ if working route *q* used for demand *r* crosses node *n*. Otherwise $z_n^{r,q} = 0$.

Decision Variables:

$g^{r,q} \geq 0$ The integer number of working lightpaths assigned to working route *q* used for demand relation *r*.

The ILP formulation is as follows:

Minimize:

$$\sum_{\forall j \in \boldsymbol{S}} c_j \cdot \left(s_j + w_j \right) \tag{4.38}$$

In addition to the constraints applied in the previous section, the following constraints will be used:

$$\sum_{\forall q \in \boldsymbol{Q}^r} g^{r,q} = d_r \qquad \forall r \in D \tag{4.39}$$

$$\sum_{\forall r \in \boldsymbol{D}} \sum_{\forall q \in \boldsymbol{Q}^r} \zeta_i^{r,q} \cdot g^{r,q} = w_i \qquad \forall i \in S \tag{4.40}$$

$$w_i \leq \sum_{\forall p \in \boldsymbol{P}} x_{i,p} \cdot n_p \qquad \forall i \in S \tag{4.41}$$

$$\tau_n = \sum_{\substack{\forall r \in \boldsymbol{D} \mid \\ \phi_r^n = 0}} \sum_{\forall q \in \boldsymbol{Q}^r \mid \mathrm{Z}_n^{r,q} = 1} g^{r,q} \qquad \forall n \in N \tag{4.42}$$

$$\tau_n \leq \sum_{\forall p \in \boldsymbol{P} \mid x_p^n = 1} 2 \cdot n_p \qquad \forall n \in N \tag{4.43}$$

$$s_j = \sum_{\forall p \in \boldsymbol{P} \mid \boldsymbol{x_{i,p}} = 1} n_p \qquad \forall j \in S \tag{4.44}$$

The objective function in (4.38) seeks to minimize the total cost of placing working and spare capacity in the network. The constraints in (4.39) guarantee that all demands will be provided with a sufficient number of working lightpaths,

and the constraints in equation (4.40) assign a sufficient amount of working capacity on each span *i* to accommodate all working lightpaths routed over it. Equation (4.41) assigns sufficient copies of the various eligible *p*-cycles to provide restoration of all working capacity on each span. Equations (4.42) and (4.43), respectively, determine the number of working lightpaths transiting through each node, and ensure that there are sufficient copies of the various eligible *p*-cycles (acting as NEPCs) to protect all transiting lightpaths through each node in the event of failure of that node. Note that the 2× multiplier in equation (4.43) is due to the fact that each copy of an NEPC can protect two transiting lightpaths from failure of node *n*, one in each direction around the *p*-cycle. Finally, the constraints in equation (4.44) place spare capacity on each span *j* to accommodate all copies of eligible *p*-cycles assigned to the network.

4.3 Metaheuristics

Approximate algorithms can be classified into two main categories. First is the custom case-specific algorithm, which is designed to solve a particular optimization problem. In our current research, the NDPP algorithm is an example of a custom algorithm to solve the NEPC enumeration problem in chapter 7. Second are metaheuristics, which are a general method which can be used for any optimization problem. Unlike the ILP, both of these approaches do not guarantee finding the optimal solution.

The term metaheuristic was firstly presented by F. Glover [64]. The suffix "meta" is a Greek word meaning upper level methodologies, and "heuristic" has its Greek origin *heuriskein,* which means the art of finding new ways to solve problems. In the literature, various metaheuristics techniques have been proposed, such as genetic algorithms, particle swarm optimization, ant colony optimization, Tabu search, simulated annealing, and local search. They can be classified in several ways as follows [65], [66]. The first is population-based search versus a single-solution based search. In single-solution based search, such as simulated annealing and Tabu search, a single point is tracked during the search. In a population-based search, such as genetic algorithms and particle swarm

optimization, a group of points (i.e., population) is tracked during the course of the search. The population-based solution is more diversity oriented while the single-solution based search is more intensity oriented.

The second means of categorization are nature inspired versus non-nature inspired. Several metaheuristic techniques originated from natural processes that could be biological, social, or physical. The genetic algorithm metaheuristics, for example, are based on evolution theory, while simulated annealing is inspired by the annealing process in metallurgy. A final categorization is deterministic versus stochastic. In stochastic techniques, some random rules are used to move from one solution to another, such as in the case of genetic algorithms while in the deterministic techniques no random decisions are used at all. Tabu search is an example of this kind of techniques.

4.3.1 Genetic Algorithm

Genetic algorithms are one of the main naturel-inspired meta-heuristics utilized in network design problems [11], [67]. It is a kind of population-based technique based on the natural selection concept of Darwin's theory. The genetic algorithm proceeds as follows. First, it begins its operation with an initial generation of sub-optimal solutions. This generation constitutes an acceptable subset of potential solutions of the investigated problem. Next, the objective function is used to assign a fitness value for each member of that generation.

The fitness value for each individual solution (i.e., member) represents the quality of that solution the fitness value is used in the process of electing the individual parents that generate the consecutive population. As the fitness level of a specific individual improves, the probability to be selected as a parent increases. Third, the next generation (i.e., offspring) is produced from the parents by using various operators such as crossover and mutation. According to this mating process the offspring inherit some of characteristics of each participating parent.

Finally, the next generation is produced by replacing the current one with the offspring or by replacing a subset of it. This whole operation is iterated repeatedly until a predefined condition is reached. This condition could represent a maximum number of generations, a certain time limit, or a degree of convergence. This process tips the scale in favor of the individuals with better fitness value to survive from generation to generation, which makes the genetic algorithm often converge to a near-optimal solution in a reasonable amount of time.

4.3.2 Tabu Search

Tabu search was introduced by Glover in 1986 [64]. In this approach, the search process is guided by imposing certain restrictions. These restrictions can have several methods, but they primarily function by "forbidding" certain search alternatives. This is why it is called Tabu "taboo" search. The most common tabu restrictions are put in place to prevent being trapped in a local optima. Even though a move may worsen a current solution, or make it infeasible, it can still be accepted in a tabu search algorithm. In this way, a tabu search algorithm can move away from a local optima and move towards the global optimum, if available.

Adaptive memory is the essence of the tabu search approach. Memory works by storing information about the search process, such as the fitness of certain moves, as the search algorithm is running. This information is utilized to guide the search operation during the next moves, by intensifying the search around promising solution areas, or diversifying it away from not so promising ones. During the search process, it is quite possible that some solutions will be re-examined. To avoid this (and in the worst-case, transform into cycling), a tabu list is used.

The Tabu list constitutes the short-term memory part of the algorithm, and it records recently examined moves. For a certain number of iterations, the moves in the Tabu list are forbidden. After each iteration, the counter of the moves in the Tabu list is decremented. When the counter of a move reaches 0, it is taken off the Tabu list.

A fitness function is a standard part of Tabu search algorithms. This function is utilized to evaluate the performance of each move. If a move provides the best solution for a certain area and if it is not on the Tabu list, it is executed. This way, the algorithm proceeds from one best solution to the next in each area while recording the best overall solution obtained during the whole search process. Note that, as discussed earlier, the best solution to an area may be worse than the preceding area's best solution. It can also be an infeasible solution. In both cases, it is accepted as the present solution. This is a major dissimilarity between a Tabu search and a greedy algorithm. Sometimes, a move that is on the Tabu list can provide the best overall solution. In this case, an aspiration factor is used to accept the move even though it is on the tabu list. Once a predetermined factor is reached, such as the number of iterations or the speed of progress in obtaining better solutions, the search is ended. At the end of the search, not only the best overall solution is recorded, but also all the appropriate information about the search operation.

4.3.3 Simulated Annealing

Simulated annealing (SA) is a generic probabilistic metaheuristic approach for locating a good approximate solution to the global optimum solution of a specific function in a large search area. It is often used when the search space is discrete (i.e., the number of communication links used in the network topology design). In some situations, the main goal of the optimization problem is to get a good solution in a reasonable amount of time, instead of finding the optimal one. For these problems, simulated annealing may be more efficient than exhaustive enumeration

This idea of slow cooling is utilized in the simulated annealing algorithm as a slow reduction in the probability of accepting worse solutions as it discovers the solution area. Accepting worse solutions is an essential property of metaheuristics because it allows for a more extensive search for the optimal solution.

4.3.4 Other Metaheuristics

There are many other metaheuristics that have been utilized in solving optimization problems, such as ant colony optimization. The ant colony optimization algorithm is a probabilistic approach for solving large optimization problems that can be reduced to discover good paths through graphs. In 1992, ant colony optimization was introduced by Marco Dorigo in his PhD dissertation, based on the behaviour of ants looking for a path between their colony and a food source. The target of the first algorithm was to discover an optimal path in a graph. The same idea has been expanded to solve a broader class of optimization problems.

Another kind of metaheuristics is local search algorithms, which move from one solution to another in the search space by applying local changes, until a solution is considered as optimal is found or a time limit is reached.

Particle swarm optimization is another type of metaheuristics that optimizes a problem by having a population of potential solutions, which are called particles, and moving these particles in the search space according to simple mathematical calculations. Each particle's move is determined by its local best known position but, is also controlled by the best known positions in the search space, which are updated as better positions are discovered by other particles. This is expected to guide the swarm towards the optimal solutions.

4.4 Efficient Approaches for Solving Larger Linear Programs

4.4.1 Column Generation

Column generation is an effective technique for solving complex linear programs. The main idea is that several linear programs are very complex to take into consideration all the decision variables at the same time. Assuming that most of the decision variables will have a value of zero in the optimal solution, only a subset of variables needs to be considered in theory when solving the problem.

Column generation utilizes this idea to generate only the variables with the potential to improve the objective function.

The given optimization model is divided into two models: the master model and the sub-model. The master model is the original one with only few decision variables. The sub-model is a new one built to find a new variable. The process works as follows. The master model is solved from this solution; we are able to obtain dual prices for each of the constraints in the master model. This information is then utilized in the objective function of the sub-model. The sub-model is solved. If the objective value of the sub-problem is negative, a variable with negative reduced cost has been found. This variable is then added to the master model, and the master model is resolved. Resolving the master problem will generate a new set of dual values, and the process is repeated until no negative reduced cost variables are identified. The sub-model returns a solution with a non-negative reduced cost, and we can conclude that the solution to the master model is optimal.

In many cases, this allows large linear programs those have been previously considered intractable to be solved, such as the cutting stock problem. One particular technique in linear programming that uses this kind of approach is the Dantzig–Wolfe decomposition algorithm. Additionally, column generation has been applied to many problems such as crew scheduling, vehicle routing, and the capacitated *p*-median problem.

4.4.2 Lagrangian Relaxation Techniques

Lagrangian relaxation is a mathematical optimization relaxation approach that replaces a complex optimization problem of constrained optimization with a simpler problem. A solution to the relaxed problem is an approximate solution to the original problem.

The method penalizes violations of inequality difficult constraints by using a Lagrange multiplier, which enforces a cost on violations. These added costs are

used instead of the strict inequality constraints in the optimization. In practice, this new problem can often be solved more easily than the original one.

There were a number of trials before 1970 to utilize Lagrangian technique in discrete optimization, including the Lorie-Savage (1955) approach to capital budgeting, Everett's proposal for "generalizing" Lagrange multipliers (1963). However, the inception of the Lagrangian technique as it is used today was introduced in 1970 when Held and Karp (1970, 1971) used a Lagrangian problem based on minimum spanning trees to propose a vividly successful algorithm for the traveling salesman problem. Motivated by this success, other researchers later used the Lagrangian approach in scheduling problems and the general integer programming problem. In 1974, when Geoffrion devised the perfect name for this approach "Lagrangian relaxation", it had taken on considerable currency. Since then the list of applications of Lagrangian relaxation has increased to contain over a dozen of the most well-known combinatorial optimization problems. For most of these problems, Lagrangian relaxation has delivered the best present solution for the problem [68].

In the following example we will demonstrate the general concept of the Lagrangian relaxation technique:

$$\begin{aligned} \max \quad & z = cx \\ \text{S.T.} \quad & Ax \leq b \qquad \text{easy constraints} \\ & Dx \leq d \qquad \text{hard constraints} \end{aligned}$$

This problem would be solved quickly, if the integer linear programming model contained only the easy constraints. But the existence of the hard constraints makes it much harder to solve. The hard constraints can be eliminated and replaced with a penalty function in the objective function:

$$\begin{aligned} \max \quad & zLR = cx + \lambda (d - Dx) \\ \text{S.T.} \quad & Ax \leq b \qquad \text{easy constraints} \end{aligned}$$

The new non-negative λ parameters (λ = {λ1, λ2, λ3,…}) are Lagrange multipliers, or weights by which we are penalized for not considering the hard constraints.

4.4.3 Relaxation-Based Decomposition Technique

It is sometimes observed that computational complexity arises from a set of constraints or integrality properties of specific sets of variables of the optimization problem. The decomposition technique works as follows. Decompose the original problem into two easy sub-problems by relaxing the integrality property of some variables rather than relaxing the set of constraints. With some insights into the problem at hand, insights that a general approach might not have, we can decompose the ILP problem into two sub-problems. First, we use a partially relaxed version of the original, which is more easily solved. We can then use the solution from that problem to set fixed values for a subset of integer variables and resolve the original with that subset of integer variables acting as parameters. While the solution is not guaranteed to be optimal, proper selection of the integer variables to relax in the first sub-problem and of the integer variables for which we can fix their values in the second sub-problem can permit near-optimal solutions. As with most near-optimal algorithms, the quality of the solution (in terms of both the objective function value and the runtime improvement) will depend on careful selection of those subsets of variables.

Chapter 5. Incremental Network Topology Optimization Using Meta-Mesh Span Restoration

This chapter represents the following paper: "**Incremental Optical Network Topology Optimization Using Meta-Mesh Span Restoration**," *Design of Reliable Communication Networks (DRCN 2011)*, Krakow, Poland, 10-12 October 2011.

5.1 Introduction and Background

Survivability is an important consideration when designing large-scale communication networks, enabling them to mitigate the negative effects of span or node failure [11], [43]. There are many survivability techniques in use, and most can be classified as either mesh restoration or ring restoration [61], [69]. Mesh restorable networks are particularly efficient in the case of dense networks with average nodal degrees ranging from 3 to 4.5. This is typical for many European networks. In contrast, ring restoration is often thought of to be economically preferable in sparse network topologies such as those more commonly seen in North America. Mesh and ring techniques can be further classified into path (or end-to-end) restoration [49], [50], and span (or link) restoration approaches [53]. Unlike path restoration, where a working path is generally replaced in its entirety (i.e., by a backup or restoration path end-to-end between its origin and destination nodes), span restoration approaches simply reroute between two nodes on either end of the failed span, leaving the surviving portions of the working lightpath intact [18]. Path restoration is usually more capacity efficient than span restoration, however, span restoration is simpler and often faster [11]. The meta-mesh scheme was proposed a number of years ago to bridge that gap in sparse network topologies, providing more capacity-efficient designs with a simple span-restoration-like mechanism. We develop a new node-arc meta-mesh ILP formulation and further extend that formulation to allow for incremental topology optimization. In our network test cases, results show that even where topology is flexible, thereby allowing a span-restorable network to

use a higher-connectivity topology, meta-mesh restoration can outperform span restoration in terms of capacity and number of spans required.

5.1.1 Meta-Mesh Restoration

Meta-mesh survivability is a modification to conventional span restoration, introduced in [53], [37] to improve capacity efficiency of sparse network topologies. In sparse networks, one is likely to find a large number of degree-2 nodes, including many chains, with portions of the network consisting of a number of degree-2 nodes in sequence. The chain's anchor nodes are the two degree-3 or higher nodes at the ends of the chain. In Figure 5-1, a chain consisting of two degree-2 nodes is anchored by nodes 1 and 4. Here, with the working capacities (w_i) indicated on each span, full restoration via conventional span restoration would require the spare capacities (s_i) shown on each span for loopback of all working capacity.

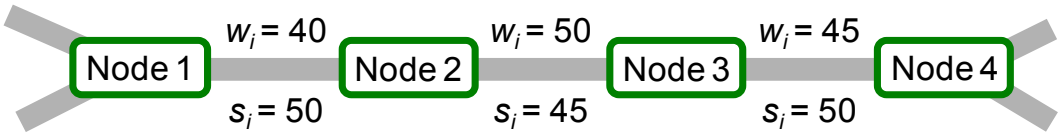


Figure 5-1. Spare capacity requirements in a chain employing span restoration (and loopback)

However, when we look closely at the makeup of those 50 units of working capacity on each span, we would typically see something like that shown in Figure 5-2. A portion of the working capacity, results from working lightpaths that fully transit the chain in its entirety, or express flow. The remaining, results from working capacity destined for one of the degree-2 nodes within the chain, so-called local flow. While the latter type of working capacity still requires full loopback spare capacity within the chain, express working capacity does not. The restoration routes for that working capacity have no need to re-enter the chain. In fact, doing so would generally force an excess of spare capacity within the chain. Consider a failure of the span between nodes 2 and 3. If its express working capacity is restored as in conventional span restoration via a restoration route

between nodes 2 and 3, then the fully restored lightpath would enter the chain at node 4, continue on to node 3, and then re-exit the chain again at node 4. The same would occur at the other end of the chain as well. Clearly, adding spare capacity on the span between nodes 3 and 4 simply to permit that restored lightpath to enter and exit the chain is unnecessary.

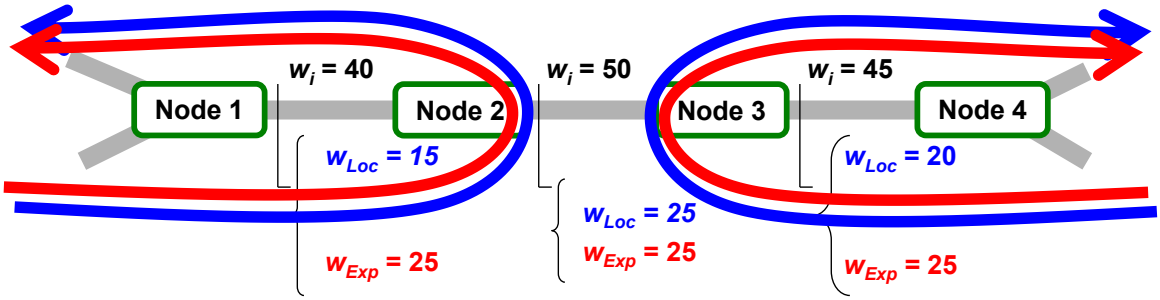


Figure 5-2. Breakdown of working channels in a chain into those that arise from local flow and express flow

In meta-mesh restoration, the idea is to treat the express working capacity as if it was routed on a single bypass span directly connecting the anchor nodes of the chain, as shown in Figure 5-3. We then provide restoration in the conventional span restoration approach, with the express working capacity restored directly between the anchor nodes of the chain as if the logical bypass span was real. The result is that only the local working capacity actually requires spare capacity within the chain itself.

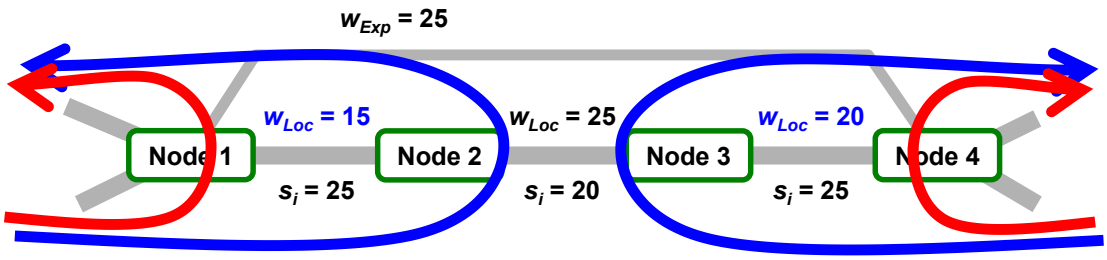


Figure 5-3. Spare capacity requirements in a chain employing meta-mesh restoration (no loopback for express lightpaths).

The theoretical foundation of capacity efficiency improvement is related to the network's so-called meta-mesh topology [70], [37]. The network in Figure 5-4(a) contains three chains, but if we redraw that topology as it would be seen only by

working lightpaths that full transit chains, then the meta-mesh topology would be as shown in Figure 5-4(b). In that topology, each chain has been replaced by its logical bypass. Effectively, it is this topology that is seen by the transiting lightpaths, and so it is this topology's theoretical redundancy that is achievable by those transiting lightpaths. Consider that the original network in Figure 5-4(a) has an average nodal degree of $\bar{d} = 2.5$. Calculating the $1/(\bar{d} - 1)$ lower bound on span restoration redundancy [71], [72], we observe that we can achieve 67% redundancy at best. However, the meta-mesh topology in Figure 5-4 (b) has $\bar{d} = 3$, giving a lower bound on redundancy of 50%. In practical, the bypass chain could be implemented by running separate fibres between the chain's anchor nodes or it could be routed using glass-throughs at chain's nodes [37]. This will lead to a reduction of the number of ports at every node in the chain.

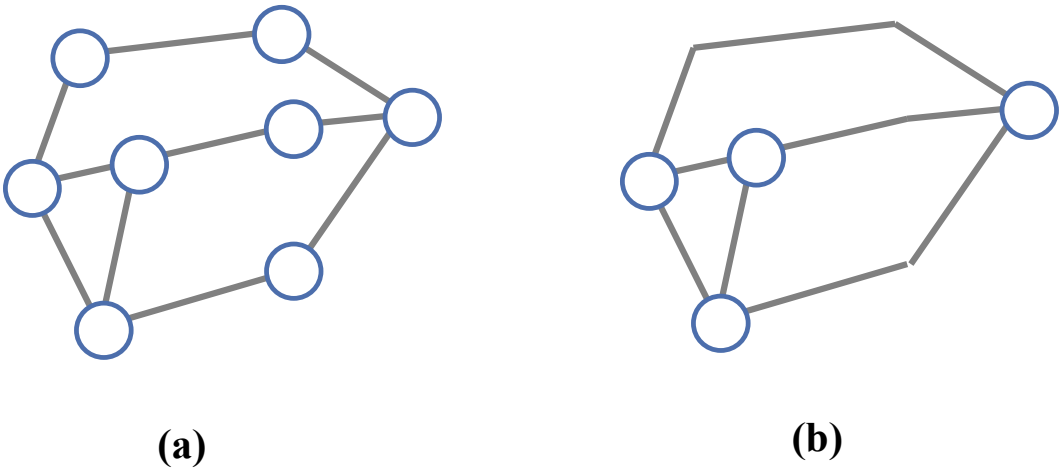


Figure 5-4. (a) A sparse network topology, and (b) its corresponding meta-mesh topology

5.1 Topology Optimization

The prior work on meta-mesh network design has assumed a known network topology, however, there are many cases where the topology itself may not be known (and various techniques for dealing with those situations) [73], [74]. The goal of the present chapter will be to extend the meta-mesh concept to networks with indeterminate topology, either via incremental topology design or green-fields design where no previous topology exists. As in the prior approaches in the

literature, we will assume a large set of eligible spans, and select from amongst them for an optimal design.

5.2 Meta-Mesh ILP Models

The current meta-mesh network design models use an *integer linear programming* (ILP) model, [53], [37]. However, that ILP model follows an arc-path approach like that in [75], where the ILP model chooses amongst a set of eligible working and restoration routes and places capacity appropriately to reduce costs. As we mentioned in chapter 2, we assume there is a wavelength converter in every node, and the wavelength continuity constraint can be relaxed in all ILP models used in this thesis.

5.2.1 Meta-Mesh Arc-path ILP Model

This model is similar to the model used for conventional span restoration design [37]. The main difference, in this model, that an extra bypass span is associated with every chain and a single failure of any span within a chain will lead to a failure of its associated logical span between the anchor nodes of that chain. In this ILP model, we will use the following notations:

N	is the set of all nodes in the network, indexed by n or m.
S	is the set of all spans in the network, typically indexed by i, b, or j. This includes eligible "optional" spans as well as existing spans.
D	is the set of all demands in the network, indexed by r.
d_r	is the number of demand units of demand r.
P_i	is the set of all distinct potential routes available to restore the flow for failure of span i, and is typically indexed by p.
c_j	The incremental cost of adding one unit of capacity on span j.
w_j	An integer decision variable for the total number of working

capacity units assigned to span *j*. (typically shortest path routing).

$g^{r,q} \geq 0$ is the amount of working flow assigned to working route *q* used for demand *r*.

$\delta_{i,j}^{p} \in \{0,1\}$ is a parameter that encodes restoration routes. If $\delta_{i,j}^{p} = 1$, restoration route *p* used for restoration of span *i* crosses span *j*. If $\delta_{i,j}^{p} = 0$, restoration route *p* used for restoration of span *i* does not cross span *j*.

$s_j \geq 0$ is the amount of spare capacity that is placed on span *j*.

$f_i^p \geq 0$ is the amount of restoration flow assigned to restoration route *p,* in the case of the failure of span *i*.

Q^r is the set of all distinct eligible working routes capable of routing lightpaths for demand *r*, typically indexed by *q*.

$\zeta_j^{r,q}$ is a binary parameter that defines the relationship between working routes and the network spans for each demand. It equals 1 if working route *q* used for demand *r* passes through a span *j*, otherwise it equals 0.

$S_d \subseteq S$ is the set of all remaining spans in the network (referred to as *direct spans* in [70]).

$S_b \subseteq S$ is the set of all logical *bypass spans* in the network. From the point of view of the ILP model, these are real spans, but from a practical perspective, this is simply a mean of permitting express working lightpaths to fully transit a chain.

$S_c \subseteq S$ is the set of all *chain spans* in the network (i.e., those spans with at least one degree-2 end node).

$k_i \in S_B$ is the relation between individual spans of the complete network and an associated logical bypass *k*. For example, if a

chain consists of spans *S*8, *S*9, and *S*15 and has a bypass span *B*4, then $k_{S8} = k_{S9} = k_{S15} = B4$.

The formulation itself is expressed as follows:

Minimize:

$$\sum_{\forall j \in S} c_j \cdot \left(s_j + w_j \right) \tag{5.1}$$

Subject to:

$$\sum_{\forall q \in Q^r} g^{r,q} = d_r \qquad \forall r \in D \tag{5.2}$$

$$\sum_{\forall r \in D} \sum_{\forall q \in Q^r} \zeta_j^{r,q} \cdot g^{r,q} = w_j \qquad \forall j \in S \tag{5.3}$$

$$\sum_{\forall p \in P_i} f_i^p = w_i \qquad \forall i \in S \tag{5.4}$$

$$s_j \geq \sum_{\forall p \in P} \delta_{i,j}^p \cdot f_i^p \qquad \forall i \in S_d \quad \forall j \in S \left| i \neq j \right. \tag{5.5}$$

$$s_j \geq \sum_{\forall p \in P_i} \delta_{i,j}^p \cdot f_i^p + \sum_{\forall p \in P_{k_i}} \delta_{k_i,j}^p \cdot f_{k_i}^p \qquad \forall i \in S_c \quad \forall j \in S \left| i \neq j \neq k_i \right. \tag{5.6}$$

The objective function in equation (5.1) seeks to minimize the total working and spare capacity cost of the network subject to following technical constraints. Constraint (5.2) guarantees that the overall working units deployed on all eligible working routes for demand relation *r* is sufficient to completely route it. In the equation (5.3) it ensures that the total working flow assigned to all eligible working routes for demand relation *r* is sufficient to fully route it. The constraints in equation (5.4) ensure that the overall restoration flow assigned to all potential restoration routes for failure of span *i* is sufficient to completely recover all of the working capacity on the failed span. Equation (5.5) deploys enough spare capacity on each surviving span *j* to accommodate the total restoration flow assigned to all restoration flows crossing it for restoration of any failed span *i*.

Equation (5.6) ensures that there is sufficient spare capacity on any span j to support all the restoration flows routed over it for the simultaneous failure of any chain span i as well as its associated bypass span k_i.

This model is not suitable for a topology-optimization problem, because the enumeration of possible routes requires a definite topology on which they can be deployed. While this model can technically be formulated, any reasonably large set of eligible spans will make enumeration of potential routes and solution of the subsequent problem intractable. Therefore, we have to remodel the meta-mesh network design problem as a node-arc (i.e., transshipment) problem, same as the ILP in [74] for a conventional span restoration.

5.2.2 New Meta-Mesh Node-Arc ILP Model

This ILP model can also be used to produce a strictly optimal solution to the original meta-mesh network design problem; the existing arc-path model will generally provide a sub-optimal solution since, in practice, an arc-path ILP model is difficult to solve with a complete eligible route set [76].

In this new model, we use the following notations, in addition to the notations used in the previous ILP model:

O_r and T_r	are the origin and target nodes of demand r.
$w^r_{j,n} \geq 0$	A decision variable for the number of working capacity units assigned for demand r on span j and flow into node n.
$w^r_{n,j} \geq 0$	A decision variable for the number of working capacity units assigned for demand r on span j and flow out of node n.
$s^j_{i,n} \geq 0$	is a decision variable representing the number of spare capacity units on span i and flow into node n. to restore span j.
λ^b_i	is a parameter that defines the relation between chain spans and bypass spans. $\lambda^b_i = 1$ if a chain span i is associated with a bypass

span b, $\lambda_i^b = 0$ if not.

The ILP model will allow working lightpaths to be routed either through chains or over their associated logical bypass spans. For ordinary (i.e., direct) spans, which are neither bypass spans nor chain spans, restoration will be carried out between the end-nodes of the failed span as normal for span restoration. However, for failure of a chain span, restoration will be carried out by two mechanisms. First, any working capacity arising from working lightpaths routed on the chain span itself will be restored between the end-nodes of the failed chain span, as normal for span restoration. And in addition, any working capacity arising from working lightpaths routed on the chain's associated logical bypass span will be restored between the end-nodes of the bypass span (i.e., between the anchor nodes of the chain).

The ILP formulation is constructed as follows:

Minimize:

$$\sum_{\forall j \in S} c_j \left(s_j + w_j \right) \tag{5.7}$$

Subject to:

$$\sum_{\forall j \in S_n} w_{n,j}^r = d_r \qquad \forall r \in D, \forall n \in N \mid n = O_r \tag{5.8}$$

$$\sum_{\forall j \in S_n} w_{j,n}^r = 0 \qquad \forall r \in D, \forall n \in N \mid n = O_r \tag{5.9}$$

$$\sum_{\forall j \in S_n} w_{j,n}^r = d_r \qquad \forall r \in D, \forall n \in N \mid n = T_r \tag{5.10}$$

$$\sum_{\forall j \in S_n} w_{n,j}^r = 0 \qquad \forall r \in D, \forall n \in N \mid n = T_r \tag{5.11}$$

$$\sum_{\forall j \in S_n} w_{n,j}^r - \sum_{\forall j \in S_n} w_{j,n}^r = 0 \qquad \forall r \in D, \forall n \in N \mid n \notin \{O_r, T_r\} \tag{5.12}$$

$$w_{n,j}^r = w_{j,m}^r \qquad \forall r \in D, \forall j \in S_n, \forall j \in S_m \mid n \neq m \tag{5.13}$$

$$w_{j,n}^r = w_{m,j}^r \qquad \forall r \in D, \forall j \in S_n, \forall j \in S_m \mid n \neq m \tag{5.14}$$

$$w_j \geq \sum_{\forall r \in D, \forall n \in N | j \in S_n} w_{n,j}^r + \sum_{\forall r \in D, \forall n \in N | j \in S_n} w_{j,n}^r \qquad \forall j \in S \tag{5.15}$$

$$\sum_{\forall i, j \in S | i \neq j} s_{n,i}^j = w_j \qquad \forall j \in S, \forall n \in N \,|\, n = O_j \tag{5.16}$$

$$\sum_{\forall i, j \in S | i \neq j} s_{i,n}^j = 0 \qquad \forall j \in S, \forall n \in N \,|\, n = O_j \tag{5.17}$$

$$\sum_{\forall i, j \in S | i \neq j} s_{i,m}^j = w_j \qquad \forall j \in S, \forall n \in N \,|\, m = T_j \tag{5.18}$$

$$\sum_{\forall i, j \in S | i \neq j} s_{m,i}^j = 0 \qquad \forall j \in S, \forall n \in N \,|\, m = T_j \tag{5.19}$$

$$\sum_{\forall i \in S_n} s_{n,i}^j - \sum_{\forall i \in S_n} s_{i,n}^j = 0 \qquad \forall j \in S, \forall n \in N \,|\, n \notin \{O_j, T_j\} \tag{5.20}$$

$$s_{n,i}^j = s_{i,m}^j \qquad \forall j \in S, \forall i \in S_n, \forall i \in S_m \,|\, n \neq m \tag{5.21}$$

$$s_{i,n}^j = s_{m,i}^j \qquad \forall j \in S, \forall i \in S_n, \forall i \in S_m \,|\, n \neq m \tag{5.22}$$

$$s_i \geq s_i^j \qquad \forall i \in S, \forall j \in S_d \,|\, i \neq i, j \tag{5.23}$$

$$s_b^i = 0 \qquad \forall i \in S, \forall b \in S_b \tag{5.24}$$

$$s_i^b = 0 \qquad \forall i \in S_c, \forall b \in S_b \,\big|\, \lambda_i^b = 1 \tag{5.25}$$

$$s_i \geq s_i^j + \sum_{\forall b \in S_b} \lambda_i^b \times s_i^b \qquad \forall j \in S_c, \quad \forall i \in S \tag{5.26}$$

The objective function in (5.7) seeks to minimize the total cost of the network subject to technical constraints. The constraints in equation (5.8) ensure that, for any demand, the number of working capacity units flowing out of the demand's origin node must equal the number of demand units for this demand. The constraints in (5.9) require that all flows into a demand's origin equal zero for this demand. Equations (5.10) and (5.11) are the equivalent constraints for the target node of a demand. The constraints in (5.12) represent the conservation of flow requirement, where the working traffic flows into and out of the transshipment nodes for a demand are equal, while constraints (5.13) and (5.14) ensure conservation of flow for all spans (i.e., any traffic flow into a span for a particular demand equals the flow out of that span for that demand). Constraints (5.15) guarantee that the number of working capacity units deployed on any span will be sufficient for accommodating all of the traffic demands passing it.

The next six constraint equations (5.16)-(5.23) are the equivalent constraints for span restoration and spare capacity allocation, except that, the working capacity on each span is treated in the same way as a demand treated in equations (5.8)-(5.15). In addition, constraint (5.24) ensures that restoration flow over any bypass span for one of its associated chain spans is zero. In other words, any chain span will not be restored over its associated bypass span. Equation (5.25) ensures that any bypass span is not restored over its chain spans. Finally, constraints (5.26) ensure that the number of spare units on span *i* are sufficient to support the restoration flows of chain span *j* and its associated bypass span *b* simultaneously.

5.3 Meta-Mesh Topology Optimization ILP Model

Although the ILP formulation in the preceding section is simply a new formulation for meta-mesh survivable network design, it does represent an easier approach for obtaining a strictly optimal solution, while the prior ILP model from the literature generally doesn't. But that is not the main reason for that new ILP. As described above, this new node-arc ILP model will permit us to add topology optimization to the problem, which was not easily done with the existing ILP model. We now introduce some new notation and add a number of new constraints so that we can perform topology optimization. More specifically, this new formulation will allow for incremental topology optimization, as we will designate existing and new (eligible) spans for the ILP model to select from.

In this model, we use the following new notation:

$S_e \subseteq S$	is the set of all new eligible spans that can be added to the network as needed.
h_b	is the establishment cost for eligible span *b*.
L	is the set of all degree-2 nodes in the original network.
$\mu_i^n = \{0,1\}$	is a binary parameter defining the relationship between bypass spans and intermediate nodes within its associated chain. $\mu_i^n = 1$ if bypass span *i* bypasses a chain that includes node *n*, $\mu_i^n = 0$

otherwise.

$\delta_i \in \{0,1\}$ is a binary decision variable, where $\delta_i = 1$ if eligible span *i* is selected for use, $\delta_i = 0$ otherwise.

$\zeta_n \in \{0,1\}$ is a binary decision variable, where $\zeta_n = 1$ if no eligible spans connect to node *n*, $\zeta_n = 0$ otherwise.

Note that S_c and S_d remain as defined before, without including any new eligible spans. S_b, on the other hand, now represents the set of all possible bypass spans that already exist or can exist in the network. Consider what happens to a chain if there are some eligible spans whose end nodes are on one of the intermediate nodes within the chain, as shown in Figure 5-5. That chain will no longer exist, and rather, would be partitioned into up to two new chains, with that intermediate node now acting as an anchor node to the new chain(s). In order to properly implement meta-mesh restoration in such a network with a flexible topology, we must enumerate all possible chains that can arise depending on what combination of eligible spans are selected for use in the final design.

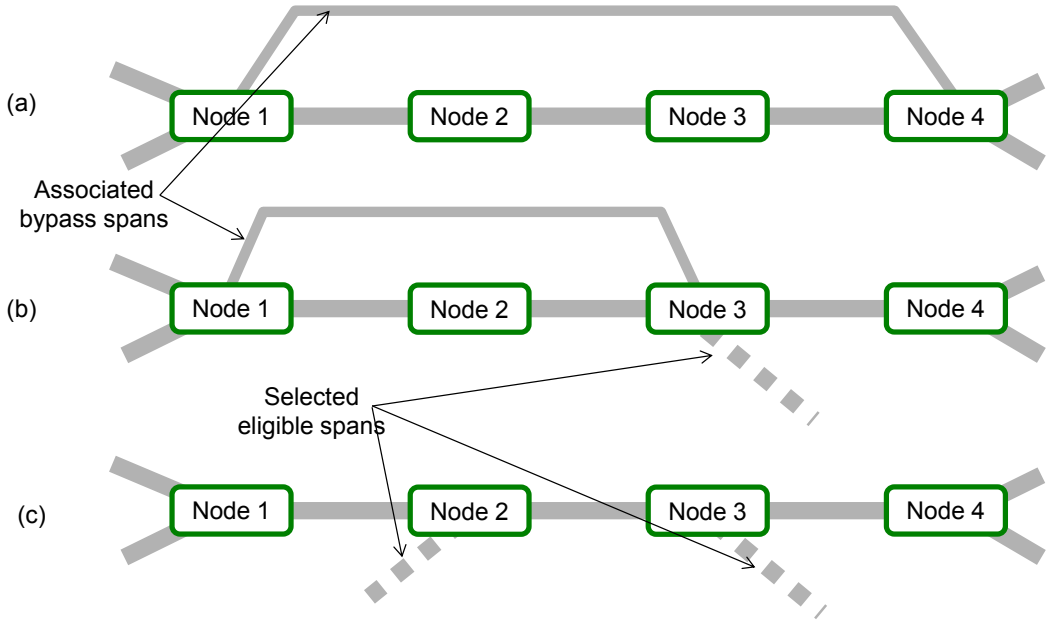


Figure 5-5. (a) Original chain and associated bypass span in existing network, (b) new shorter chain when an eligible span incident on N3 is

selected, (c) elimination of chain altogether when eligible spans incident on N2 and N3, respectively, are selected

The meta-mesh incremental topology optimization ILP model includes all equations from the ILP formulation introduced above, except for equation (5.23), which is removed. In addition, the new objective function in (5.27) replaces the one in (5.7), and we add the constraints in (5.28)-(5.32).

Minimize:

$$\sum_{\forall i \in S} c_i (s_i + w_i) + \sum_{b \in S_e} h_b \times \delta_b \tag{5.27}$$

Subject to (additional constraints):

$$\sum_{\forall b \in S_b} \delta_b \times \mu_b^n \leq (1 - \zeta_n) \qquad \forall n \in L \tag{5.28}$$

$$\sum_{\forall i \in S_e | n \in \{T_i,\ O_i\}} \delta_i \leq M \times \zeta_n \qquad \forall n \in L \tag{5.29}$$

$$2 \times \sum_{\forall i \in S_e | n \in \{T_i,\ O_i\}} \delta_i \geq \sum_{\forall i \in S_b | n \in \{T_i,\ O_i\}} \delta_i \qquad \forall n \in L \tag{5.30}$$

$$s_i \geq s_i^j \qquad \forall i \in S, \forall j \in S_d \cup S_e \ | i \neq j \tag{5.31}$$

$$s_i + w_i \leq M \delta_i \qquad \forall i \in S_e \cup S_b \tag{5.32}$$

The objective (5.27) has been modified from (5.7) to include the fixed span establishment costs for all those eligible spans those were selected for use in the final design. Constraints (5.28) and (5.29) ensure that, for any intermediate node, *n*, if any eligible span incident on *n* has been selected, all the bypass spans passing over this node will not be used. Note that *M* is simply some arbitrarily large number, at least larger than the left hand side of the equation could possibly get. Constraint (5.30) guarantees that, for any intermediate node *n*, the bypass spans connected to it (i.e., those for which *n* will act as an anchor node) could only be used if one of the eligible spans connected to *n* has been selected for use in the final solution, and furthermore, it limits the number of such bypasses to two (only up to two can exist in the case where a selected eligible span splits the original chain into two). Equation (5.31) guarantees that, for any direct span or eligible span, the number of the spare capacity installed on spans along its restoration

route will satisfy its working capacity. Equation (5.32) ensures that, for any eligible span or bypass span *i*, the corresponding binary variable δ_i is set to 1 if this span is used in the design. As in (5.29), *M* is an arbitrarily large number that is at least larger than the left hand side of the equation could possibly get.

5.4 Experimental Study

We carried out our experiments on the 15-node and 35-node network families from [37], because these two families show clearly the effect of using meta-mesh. Each subsequent network within a family is identical to the previous one except that a single span has been added. This gives us a range of related networks with the same underlying nodal arrangements and allows us to better study the effects of network connectivity on the performance of our ILP models [77]. The 15-node networks used herein ranged from 16 spans to 30 spans, while the 35-node networks ranged in size from 37 to 70 spans. In both network families, each node pair exchanges random amount of demand from 1 to 10 (using a simple uniform random distribution), and all members of a network family use the same demand matrix. Benchmark network designs solved using the arc-path ILP formulation in [70] were provided with at least 5 working routes and 10 restoration routes using the same procedure in that prior work. In all cases, both of the fixed and incremental working and spare capacity costs for each span are proportional to the length of the span. The fixed cost comprises rights-of-way, excavation, duct installation, equipment housing for amplification, while incremental capacity costs includes all per-channel costs such as requirements for adding additional fibres. Furthermore, each h_b value is equivalent to cost of placing 50 units of capacity on the span.

The ILP models formulated have been modeled in the AMPL modeling language and solved using CPLEX 11.0 on a Quad core Xeon CPU running 64 bit Windows Server 2003. All solutions were run with the default mipgap of 0.0001, meaning they are guaranteed to be within 0.01% of optimal.

5.5 Results and Discussion

The first set of results in Figure 5-6 and Figure 5-7 compares the capacity costs of the 15-node and 35-node network families, respectively, designed using the benchmark arc-path approach in [70] with those of the new node-arc approach developed above (without topology optimization). In each figure, a data point represents the total (working and spare) capacity cost of an optimally designed meta-mesh network of the indicated size and connectivity ($\bar{d}$), designed using the indicated ILP formulation. Also note that the figures only show results for networks up to $\bar{d} = 3.2$ (i.e., 24 spans) in the 15-node family, and $\bar{d} = 3.4$ (i.e., 59 spans) in the 35-node family. This is because all of the more highly connected members of those families had no chains in which meta-mesh can be effective. In such networks, the meta-mesh restoration mechanism defaults to conventional span restoration [70]. In both networks, capacity costs are normalized to that of the lowest-cost network solved. In the 15-node family, that corresponds to the $\bar{d} = 3.1$ member, and in the 15-node family, it corresponds to the $\bar{d} = 3.4$ member.

As expected, solutions obtained from the node-arc ILP formulation are at worst equivalent to the arc-path solutions, but in most cases, significantly lower. This is because the arc-path approach is limited in its capability to find efficient solutions because of an incomplete set of eligible routes to select from. In general, the complexity (and therefore, the runtime) of a node-arc approach is higher than that for an arc-path approach [11], but the latter will increase significantly with larger eligible route sets [76].

In the 15-node networks, the greatest reduction in capacity is observed in the $\bar{d} = 3.2$ network (6.5% lower than the arc-path approach), while in the 35-node networks, the greatest reduction (12%) at $\bar{d} = 3.1$. In both network families, the benefits of the node-arc approach are less in sparser networks than in more richly connected networks. This is because when we enumerated eligible route sets for

the arc-path approach, those eligible route sets in sparser networks are more likely to be a near-complete set of all possible eligible routes.

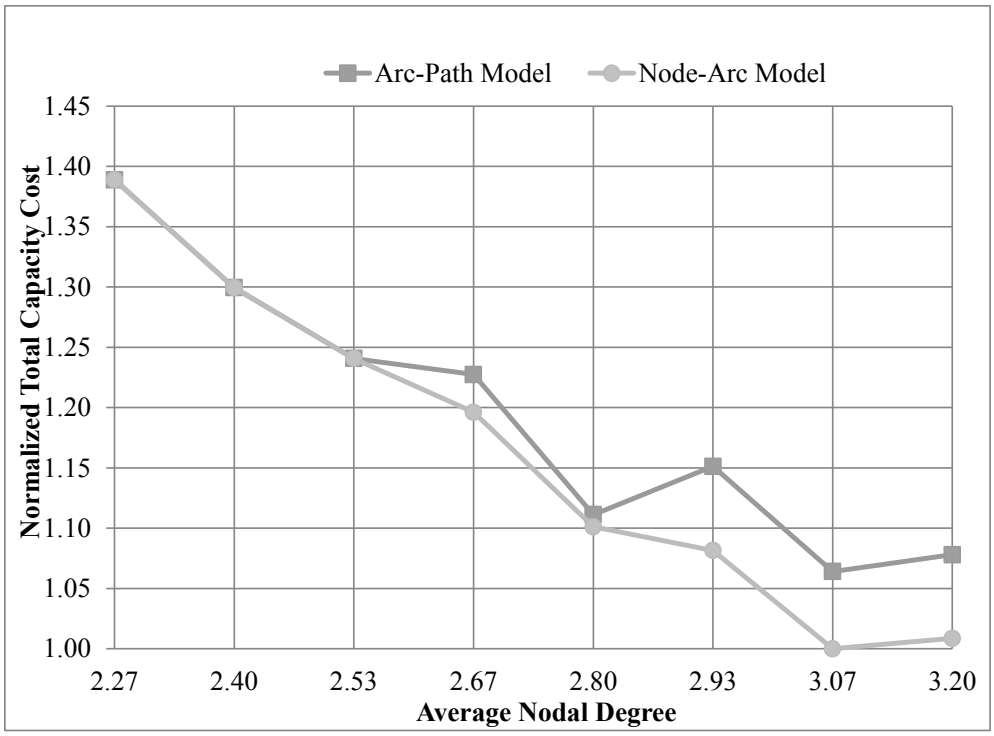


Figure 5-6. Normalized capacity costs of node-arc and arc-path meta-mesh ILP formulations in the 15-node network family

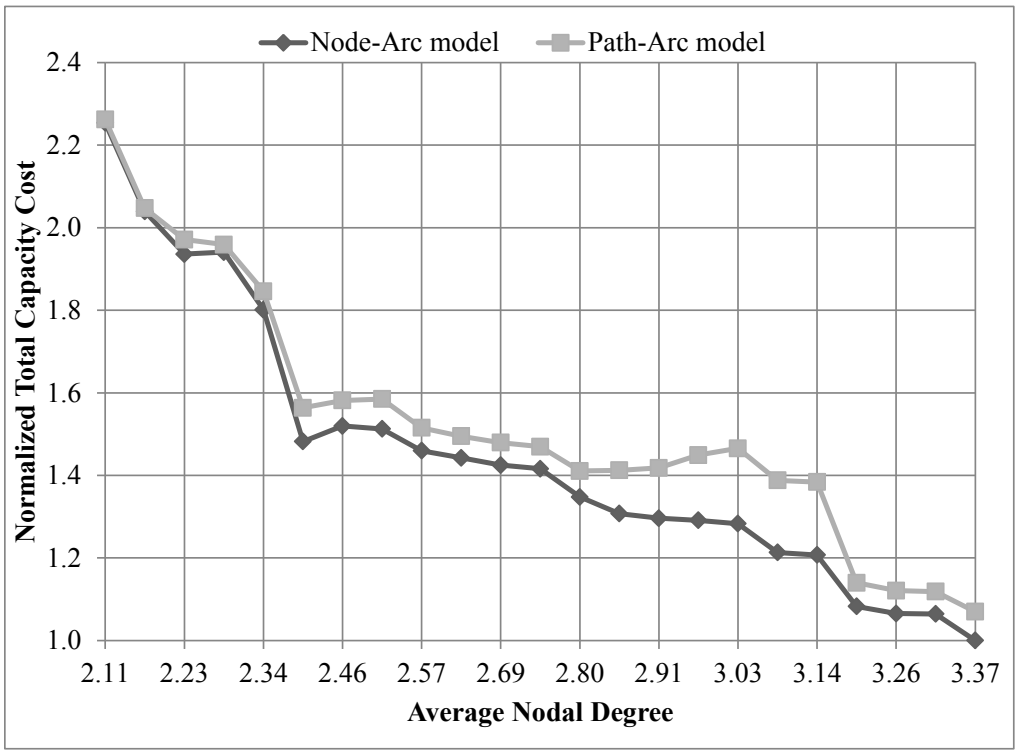


Figure 5-7. Normalized capacity costs of node-arc and arc-path meta-mesh ILP formulations in the 35-node network family

In Figure 5-8, we compare the costs of topology optimization in a meta-mesh network with topology optimization for span restoration. Each data point represents the total cost of the most sparse member of the 15-node network family solved for the indicated survivability mechanism (either meta-mesh or span restoration) when provided with the indicated number of extra eligible spans. In other words, the x-axis represents the size, $|S_e|$, of the eligible span set, S_e, provided to the solver. In all cases, the eligible span set corresponds to the set of spans that would have been added in sequence as we constructed the network family.

We can see that for both survivability mechanisms, the total cost (which is made up of the working and spare capacity costs plus the span establishment costs, h_b, of the eligible spans used in the resulting solution) decreases as the number of eligible spans increases. This is expected, since the more eligible spans available to select from, the more likely the solver will be able to configure an efficient network design. What is surprising, perhaps, is that the decreasing cost appears to follow a roughly linear relationship with $|S_e|$. This will be explored further in future work as we continue to study the effects of topology optimization in meta-mesh networks. Note that we only provide solutions for the 15-node network herein.

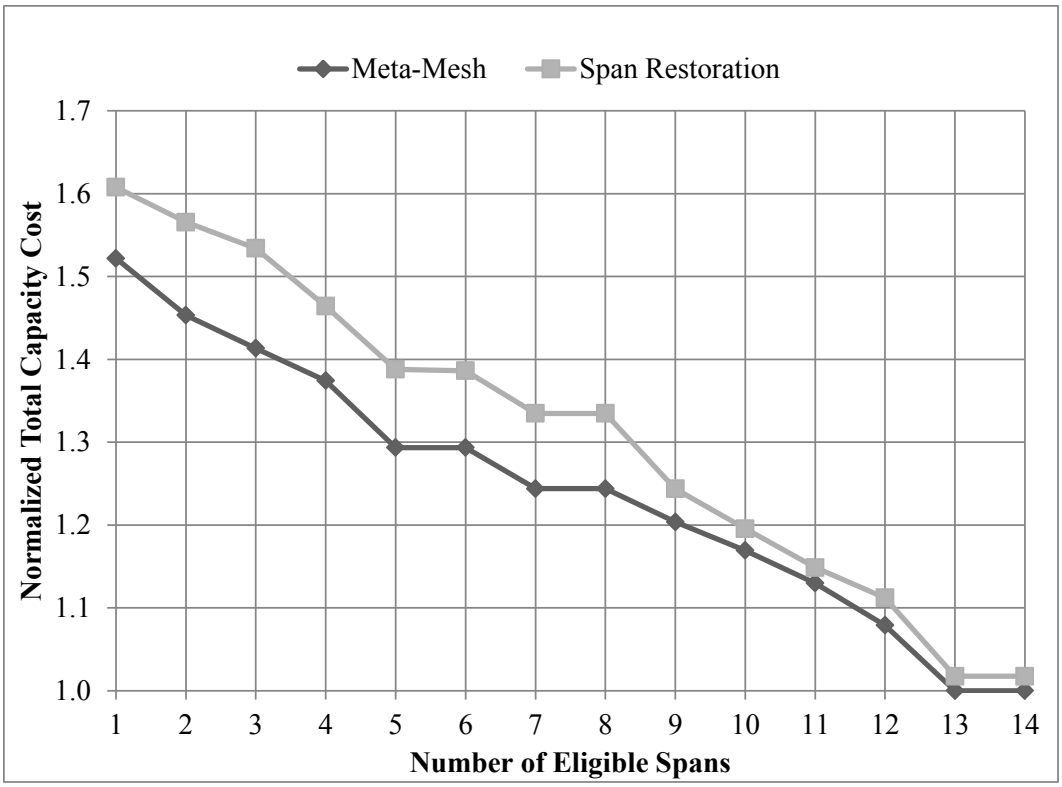


Figure 5-8. Meta-mesh and span restoration incremental topology optimization costs in the 15-node network family

In Figure 5-8, we can also observe that the connectivity of the resulting optimally designed networks increases as we provide the solver with a larger eligible span set. In other words, when we have a larger number of possible spans to add to our existing network, we select more of them for inclusion in our final design. While it is not necessarily surprising that this relationship would exist as we provide just a few eligible spans, it is obvious that the solver continued to select more and more spans as we provided a greater number of spans to pick from. We expect if we keep adding more spans the solver will at some point max out the number of spans to use in the optimal solutions. We suspect that this is a result of the h_b values those we used. Preliminary studies suggested that h_b of values 50 times the unit capacity cost on each span would provide a good trade-off between capacity costs and span establishment costs. However, the results in Figure 5-9 seem to indicate that the solver continues to add more and more spans to the network as eligible spans become available. We will explore this in future work as well, including experimenting with various h_b values. Larger h_b values will likely have

a mitigating effect on the observation above (more eligible spans results in more selected spans), though we will endeavor to select a range of h_b values that better represent a wide range of real networks.

Another observation we can make from the data in Figure 5-9 is that the solver selects fewer additional spans when the network utilizes meta-mesh restoration than span restoration. This is also expected since the meta-mesh approach allows for better capacity efficiency in more sparse networks than would be achievable with span restoration in the same network. From the same figure, another observation for span restoration curve. When the number of eligible spans is nine, the solver selects less number of spans than in the previous two cases. The reason is that the solver found adding this last span can replace two of the previously chosen eligible spans while reducing the cost of the network.

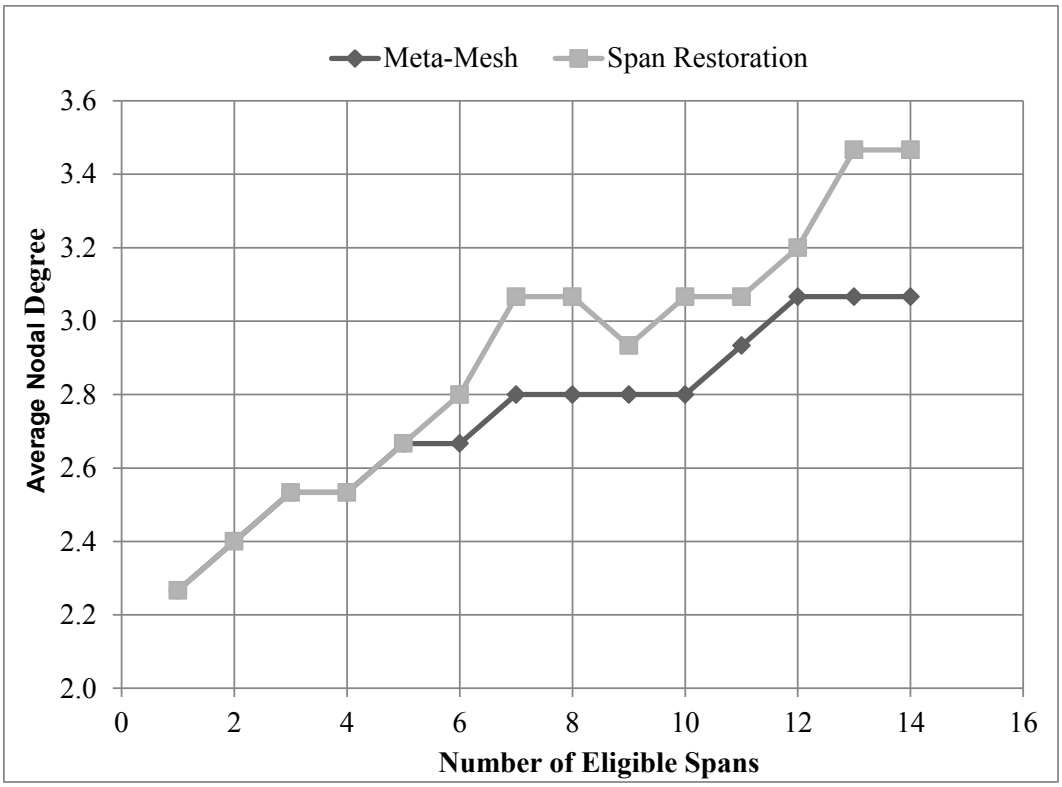


Figure 5-9. Network connectivity of meta-mesh and span restoration incremental topology optimization versus eligible span set in the 15-node network family

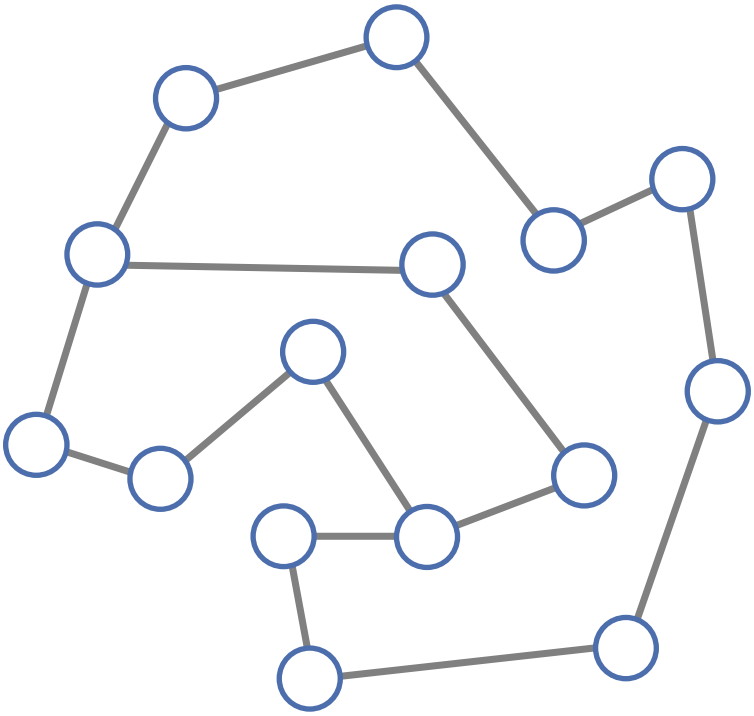
Table 5-1 shows the runtime of a sample of five of our test cases. In this table, the second column identifies the network that the test cases correspond to, the third column indicates the number of eligible spans considered, and the fourth column shows how many of those eligible spans were actually selected in the optimally-designed solutions. Finally, the runtime for each test is shown in the fifth column. As observed, that the runtime increases in an exponential fashion with increasing complexity of the problem, but even the runtime for the largest test case was still easily solved, taking just over two days to solve to optimality.

Table 5-1. Runtime data for our a sample of five test case problems

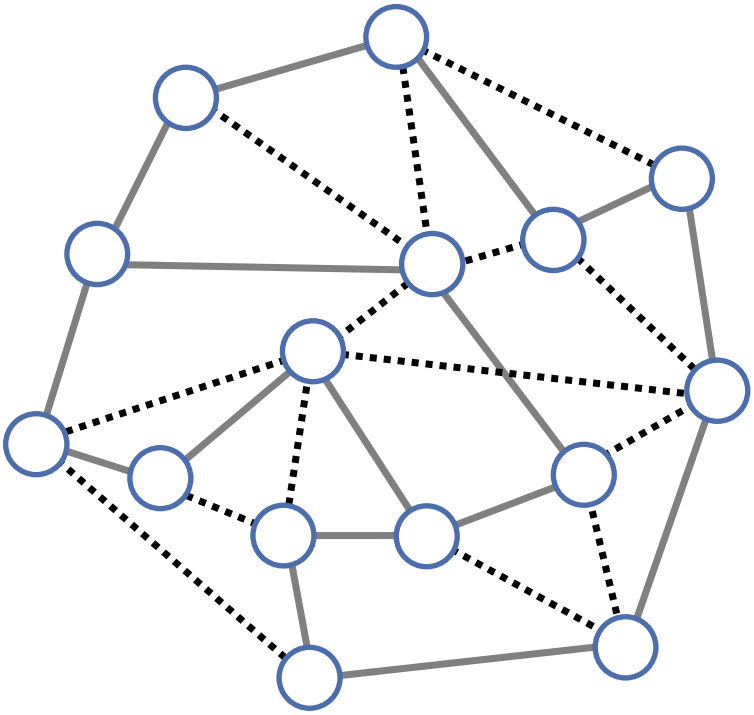
Test Case Sample	Existing Network	# Eligible Spans	# Eligible Spans Selected	Runtime (hours)
1	15 nodes 16 spans	6	4	0.005
2	15 nodes 16 spans	14	7	1.4
3	35 nodes 37 spans	11	11	0.2
4	35 nodes 37 spans	15	15	4.2
5	35 nodes 37 spans	19	17	50.7

Finally, for the interested reader, we take a closer look at the actual graph topologies of the two sample test cases in Figure 5-10 and Figure 5-11. Within each figure, the topology (a) represents the original existing network (i.e., it does not consider any of the eligible spans. Topology (b) illustrates the base topology plus the set of all eligible spans provided to the solver. The topology of the optimally designed network is shown in topology (c). More specifically, Figure 5-10 shows the noted topologies for the 15-node test case network and a maximal set of 14 eligible spans, while Figure 5-11 shows the noted topologies for the 35-node test case network and a maximal set of 19 eligible spans.

(a)



(b)



(c)

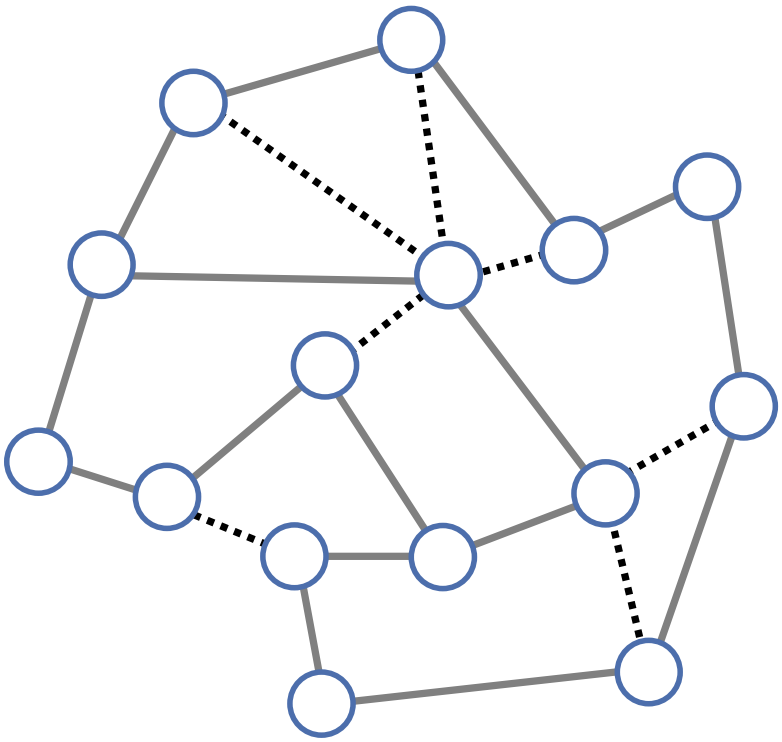
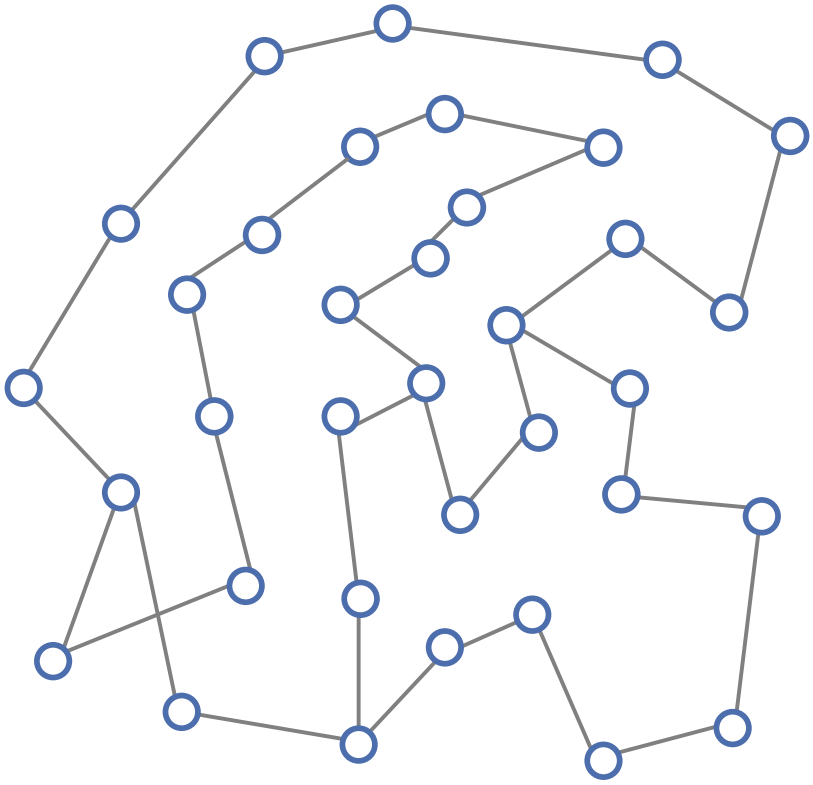
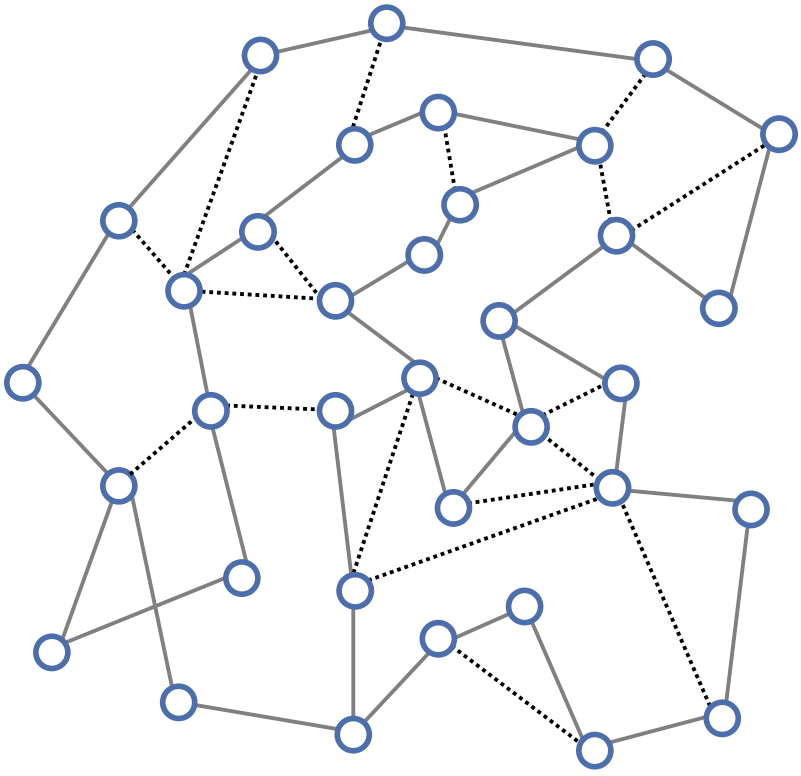


Figure 5-10. a) Existing 15-node 16-span network, b) the original network plus the maximal set of 14 eligible spans, and c) the optimal topology

(a)



(b)



(c)

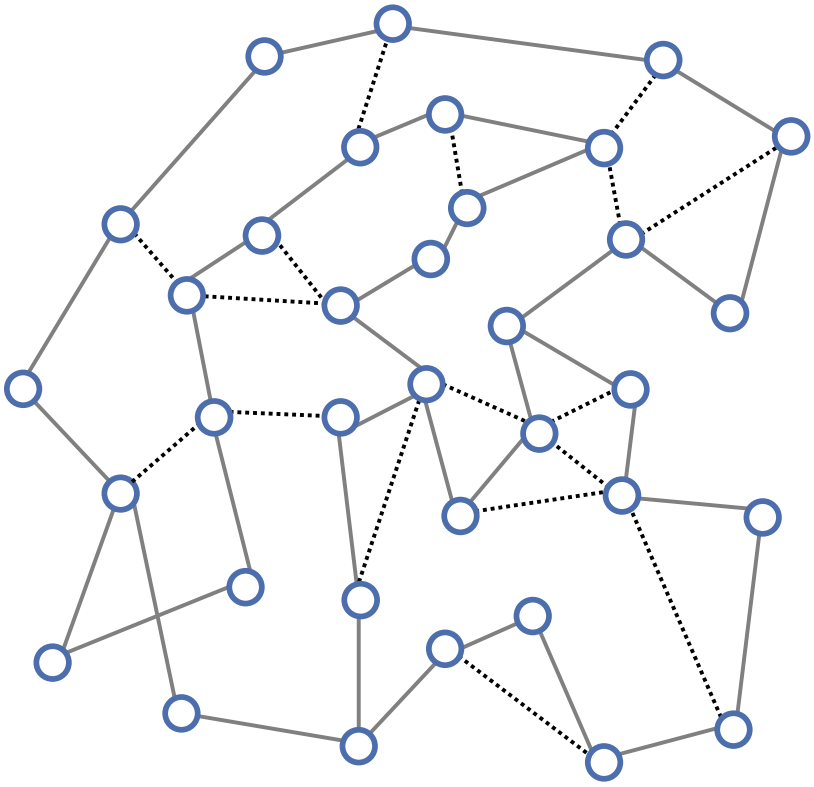


Figure 5-11. a) Existing 35-node 37-span network, b) the original network plus a maximal set of 19 eligible spans, and c) the optimal topology

5.6 Conclusion

We have developed a new node-arc ILP formulation for optimal meta-mesh network design and demonstrated that it can be used to provide very capacity-efficient network designs in reasonable amount of time. We also developed an extension to that ILP formulation that allows for topology optimization. Results show that this design approach can be used to optimally select fewer spans than needed in span restoration, and can achieve lower overall network design costs.

We can note here that, while the complexity of the model presented herein is such that we can solve reasonably-sized networks of the type seen in the optical core, the model is currently intractable for extremely large networks of hundreds or thousands of nodes. Future work will investigate heuristic approaches for solving larger instances of the problem.

Chapter 6. Incremental Network Topology Optimization Using *p*-Cycle Technique

This chapter represents the following paper: "**ILP Model and Relaxation-Based Decomposition Approach for Incremental Topology Optimization in *p*-Cycle Networks**," *published in Journal of Computer Networks and Communication, Vol. 2012, pp. 1-10, 2012.*

6.1 Introduction and Background

High-availability networks have become integral to our everyday lives, used for banking, financial transactions, voice and data communications, entertainment, etc. While much effort has been made to make them as reliable as possible, failures, and more critically, service outages still occur with alarming frequency. The vast majority of such failures are a result of fibre cuts, with most of those failures due to cable dig-ups and similar construction accidents [11].

As the frequency of failures has increased, researchers have developed many approaches for ensuring survivability of the network even in the face of cable cuts or other equipment failures, including a number of mechanisms that allow the network to actively respond to a failure by rerouting affected traffic onto one or more backup routes. Survivability mechanisms are often thought of as being either *restoration* or *protection* [37]. Although the differences between the two are often blurred, and some mechanisms can be considered to be either type, the general idea is that restoration techniques are those in which a backup route is formed post-failure, while protection techniques are those in which a backup route is formed pre-failure. Each individual survivability mechanism has its own advantages and disadvantages, and requires differing amounts of spare capacity distributed throughout the network to accommodate backup routes.

In most of the work in the literature, the underlying network topology is known in advance, but there have been several approaches developed that include at least some aspects of an unknown or variable topology in the network design process [78], [79], [74], [80]. In [78], and [81], the design methods for tree topologies optimization in communication and data networks did not consider restoration or

reliability. In [82], and [79], bi-connected network topologies were considered as a transition from tree topologies. In [74], and [80], survivability itself was included in the design approach. In these approaches, fixed costs are typically associated with establishment of a span as well as with placement of working and spare capacities on those spans. Fixed establishment costs represent rights-of-way and lease acquisitions, excavation, duct installation, amplifiers, etc., that are not generally dependent on the capacity or bandwidth of the spans.

Relatively little effort has been made on the investigation of the incremental topology optimization problem. Therefore, the goal of the present work is to develop a JCA *p*-cycle network topology optimization ILP formulation that will minimize the overall design cost (capacity and fixed span establishment costs) of a *p*-cycle network along with its underlying topology such that all single span failures are restorable. Due to the significant computational complexity of this problem (as will be discussed later), we will consider only incremental topology design, where a pre-existing initial topology already exists but which is amended through span additions. Even this less complex problem becomes intractable for large networks, and so we further develop a problem-specific relaxation-based decomposition technique to solve this large scale ILP.

6.2 *p*-Cycle ILP Model

In this section, we present our ILP formulation for incremental topology optimization for *p*-cycle network design problem. Prior topology optimization ILP models generally make use of the node-arc approach, as enumeration of eligible restoration routes becomes a challenging combinatorial problem when the underlying topology is not known; a separate set of eligible routes is needed for every combination of selected eligible spans. In chapter 4, we introduced the *p*-cycle ILP model, both of working routing and *p*-cycle selection placement done via arc-path approach (i.e., selection from an enumerated set of eligible *p*-cycles). While we will utilize a node-arc approach for our new ILP model with respect to working routing, our overall approach will be a hybrid, with the *p*-cycle selection placement still done via an arc-path approach. There has been a few notable

works in recent literature that develop methods for *p*-cycle network design without enumeration of eligible cycles [83], [84], but these approaches have proven challenging to incorporate into our topology-optimization ILP. In order to formulate our ILP model, we first define the following notation:

N	Set of all nodes in the network topology, indexed by *n* or *m*.
S	Set of all spans in the network topology, typically indexed by *i* or *j*. This includes eligible spans as well as existing spans.
S_n	Set of all spans incident on node *n*, indexed by *i* or *j*.
Q	Set of all eligible spans that can be added to the network, indexed by *i* or *j*.
D	Set of all demands in the network, indexed by *r*.
d_r	The parameter that represents the number of demand units for demand *r*.
$O_r \in N$	The origin node of demand *r*.
$T_r \in N$	The target node of demand *r*.
c_j	The incremental cost of adding one unit of capacity on span *j*.
f_i	The fixed establishment cost for eligible span *i*.
$x_{j,p} \in \{0,1,2\}$	A parameter that enumerates eligible *p*-cycles by representing the relationship between span *j* and *p*-cycle *p*, where $x_{j,p} = 2$ if it is a straddling span, $x_{j,p} = 1$ if it is an on-cycle span, and 0 otherwise.
$w^r_{n,j} \geq 0$	A decision variable for the number of working capacity units assigned for demand *r* on span *j* and flow out from node *n.*
$w^r_{j,n} \geq 0$	A decision variable for the number of working capacity units assigned for demand *r* on span *j* and flow into node *n.*
$w_j \geq 0$	An integer decision variable for the total number of working capacity units assigned to span *j.*
$s_j \geq 0$	An integer decision variable for the number of spare units deployed on span *j*.

$\delta_i \in \{0,1\}$ A binary decision variable that equals 1 if the eligible span *i* will be used in the design, and 0 otherwise.

$n_p \geq 0$ An integer decision variable that represents the number of copies of *p*-cycle *p* that will be used in this design.

M A large number (in our case, the summation of all demands plus one).

Note that strictly speaking, the $w^r_{n,j} \geq 0$ and $w^r_{j,n} \geq 0$ decisions variables are integer variables. However, as was shown in [85], as long as the capacity variables themselves are integer, integrality can be relaxed on the underlying flow variables. We then define the problem as follows:

Minimize:

$$\sum_{\forall j \in S} c_j (s_j + w_j) + \sum_{i \in Q} f_i \times \delta_i \tag{6.1}$$

Subject to:

$$\sum_{\forall j \in S_n} w^r_{n,j} = d_r \qquad \forall r \in D, \forall n \in N \mid n = O_r \tag{6.2}$$

$$\sum_{\forall j \in S_n} w^r_{j,n} = 0 \qquad \forall r \in D, \forall n \in N \mid n = O_r \tag{6.3}$$

$$\sum_{\forall j \in S_n} w^r_{j,n} = d_r \qquad \forall r \in D, \forall n \in N \mid n = T_r \tag{6.4}$$

$$\sum_{\forall j \in S_n} w^r_{n,j} = 0 \qquad \forall r \in D, \forall n \in N \mid n = T_r \tag{6.5}$$

$$\sum_{\forall j \in S_n} w^r_{n,j} - \sum_{\forall j \in S_n} w^r_{j,n} = 0 \qquad \forall r \in D, \forall n \in N \mid n \notin \{O_r, T_r\} \tag{6.6}$$

$$w^r_{n,j} = w^r_{j,m} \qquad \forall r \in D, \forall j \in S_n, \forall j \in S_m \mid n \neq m \tag{6.7}$$

$$w^r_{j,n} = w^r_{m,j} \qquad \forall r \in D, \forall j \in S_n, \forall j \in S_m \mid n \neq m \tag{6.8}$$

$$w_j \geq \sum_{\forall r \in D, \forall n \in N \mid j \in S_n} w^r_{n,j} + \sum_{\forall r \in D, \forall n \in N \mid j \in S_n} w^r_{j,n} \qquad \forall j \in \boldsymbol{S} \tag{6.9}$$

$$w_j \leq \sum_{\forall p \in \boldsymbol{P}} x_{j,p} \cdot n_p \qquad \forall j \in \boldsymbol{S} \tag{6.10}$$

$$s_j \geq \sum_{\forall p \in \boldsymbol{P} \mid \boldsymbol{x_{j,p}} = 1} n_p \qquad \forall j \in \boldsymbol{S} \tag{6.11}$$

$$s_i + w_i \leq M \times \delta_i \qquad \forall i \in Q \qquad (6.12)$$

The objective function in equation (6.1) seeks to minimize the total cost of the network, including the variable costs incurred for placing working and spare capacities on all spans, and the fixed costs incurred by adding any additional spans to the existing topology (i.e., selecting one or more of the eligible spans). Equations (6.2) to (6.6) are the node-arc constraints that determine working routing and working capacity placement, similar to the approach in [74]. The constraints in equation (6.2) ensure that, for any demand, the total number of working capacity units flowing out from the origin node must equal to the number of demand units for this demand, while constraints (6.3) ensures that all network flows into the origin node for a particular demand equal zero. Equations (6.4) and (6.5) are the related target node constraints. The constraints in (6.6) ensure the conservation of flow requirement for all transshipment nodes (i.e., not the origin or target nodes) for each demand, while constraints (6.7) and (6.8) ensure conservation of flow for all spans (i.e., any traffic flow into a span for a particular demand equals the flow out of that span for that demand). Equation (6.9) guarantees that the total number of working capacity units deployed on any span will be sufficient to accommodate all of the working traffic routed through it.

Equations (6.10) and (6.11) are the arc-path *p*-cycle placement constraints like those in the original *p*-cycle paper [41]. Constraints (6.10) ensure that for each failed span, the total number protection routes available from *p*-cycles deployed in the network will be sufficient for restoring the working capacity on each span; each copy of a *p*-cycle can restore one working capacity on each of its on-cycle spans and two units of working capacities on each of its straddling spans. Constraints in (6.11) place sufficient spare capacity to accommodate all deployed *p*-cycles. Finally, the constraints in (6.12) force all span selection variables to equal one if the associated span is assigned any working and/or spare capacity.

6.3 Experimental Methodology

We used a set of seven test case networks of 10 nodes, 15 nodes, 20 nodes, 25 nodes, 30 nodes, 35 nodes, and 40 nodes. The base networks we used herein (i.e.,

defining the existing topologies) are the most sparse members of the network families from [37], while their so-called master networks (i.e., those with average nodal degree $\bar{d} = 4.0$) represent the set of eligible spans for each of our respective networks. The set of demands for each of those networks were also used herein; each node pair in a network exchanges a number of lightpaths drawn from a uniform random distribution. While one might argue that demands in reality are not known in advance with any precision and are not static, this treatment of demands is common in the literature, as the demands used can represent upper limits on the expected demands.

Eligible *p*-cycles were enumerated via a custom designed C++ algorithm that performed a depth-first search type of algorithm to enumerate at least the shortest 10 thousand possible cycles that can be drawn in the graph to protect each single span failure, including eligible spans.

We solved all instances of the problem on an 8 processor ACPI multiprocessor X64-based PC with Intel Xeon® CPU X5460 running at 3.16GHz with 32 GB memory. The ILP models were implemented in AMPL [86] and solved with the CPLEX 11.2 solver [87]. We used a CPLEX mipgap setting of 0.001, which means all test cases solved to full termination are provably within 0.1% of optimality.

6.4 Preliminary Result Analysis

Figure 6-1 through Figure 6-7 show the relationship between total network design cost and the number of eligible restoration routes with various establishment cost multipliers. Each square, diamond and triangular data point represents the normalized total cost (working and spare capacity plus fixed span establishment costs) of the network indicated with the specified number of eligible spans and with the specified span establishment cost multiplier. The cost multiplier is the ratio of the spans' fixed establishment cost to its per-unit capacity cost (i.e., it equals f_i / c_i); the same cost multiplier is applied uniformly on all spans in the network. In our case, we used cost multipliers of 10, 20, and 50, denoted in the

charts as low, medium, and high, respectively. We remind the reader that the fixed establishment costs represent rights-of-way costs associated with the span's fibre facility route, installation of the conduit and fibre cables, and all other one-time costs that might be incurred to establish a new span. The network design cost curves for the medium and high establishment cost factors are not shown for the three larger networks, as problem complexity becomes exceedingly problematic for these test cases (see further discussion below).

As we expect, the ILP model is better able to perform working and restoration routing and allocate the associated working capacity and *p*-cycles as we introduce more eligible spans, so that overall capacity costs are reduced as the eligible span set gets larger. The rate of the cost reductions varies from network to network, but the trend spears to be that cost reductions slow as the number of eligible spans becomes large. The interpretation here is that as we provide the network with a greater and greater number of eligible spans to select from, it becomes more difficult for the network to make use of these eligible spans.

We can also note that the establishment cost factor doesn't appear to have a significant bearing on the behaviour of the relationship between network costs and the number of eligible spans. For each network, the differences between the three curves themselves (corresponding to the low, medium, and high establishment cost factors) is primarily due to the fact that the sum of the selected spans' fixed costs will be larger with a higher establishment cost factor (i.e., the second summation in the objective function), irrespective of the actual number of selected spans. In addition, as will be discussed later, the differences between the three curves is partially a function of the differences in the spans selected the by the solver for the various cost factors. However, since the higher establishment cost factors generally result in selection of fewer eligible spans (see the discussion below), this will have a negative effect on the design costs at higher establishment costs. The total network design costs tend to become closer (i.e., the differences between them become less, relatively speaking) as the networks become larger, though this is primarily due to the fact that the capacity costs represent a

proportionally greater share of the overall network cost as the networks become larger. In hindsight, this suggests that perhaps our establishment cost multipliers is likely too small to adequately demonstrate the effect that is seen in smaller networks. This should not be interpreted as suggesting that the objective function itself is flawed, rather, there will be a degree of uncertainty in establishment cost factors that will need to be selected based on observed (i.e., actual) costs and perhaps also artificially through a desire to drive rich or sparse topologies (through low cost multipliers and high cost multipliers, respectively).

In any case, the ILP model effectively permits a network designer to select an optimal set of span additions (i.e., incremental topology optimization) on which to design a *p*-cycle network. Strictly speaking, this problem is NP-hard [88], [89], but like many NP-hard problems, specific instances are solvable in reasonable times. That is the case for small instances of this problem. However, we can also observe in Figure 6-1 through Figure 6-7 that the solution runtimes become prohibitively high for large test case network instances, and generally also increase with the number of eligible spans provided to a network. In those figures, the curves with cross (×) data points (read against the right hand side y axes) show the runtime required by the solver when running the corresponding low establishment cost factor results to optimality. Each data point represents the actual processor time used in total amongst all 8 processors (as recorded by CPLEX) to solve the ILP model for the indicated test network with the indicated number of eligible spans using the low establishment cost factor (though in most cases, only a single processor was utlilized).

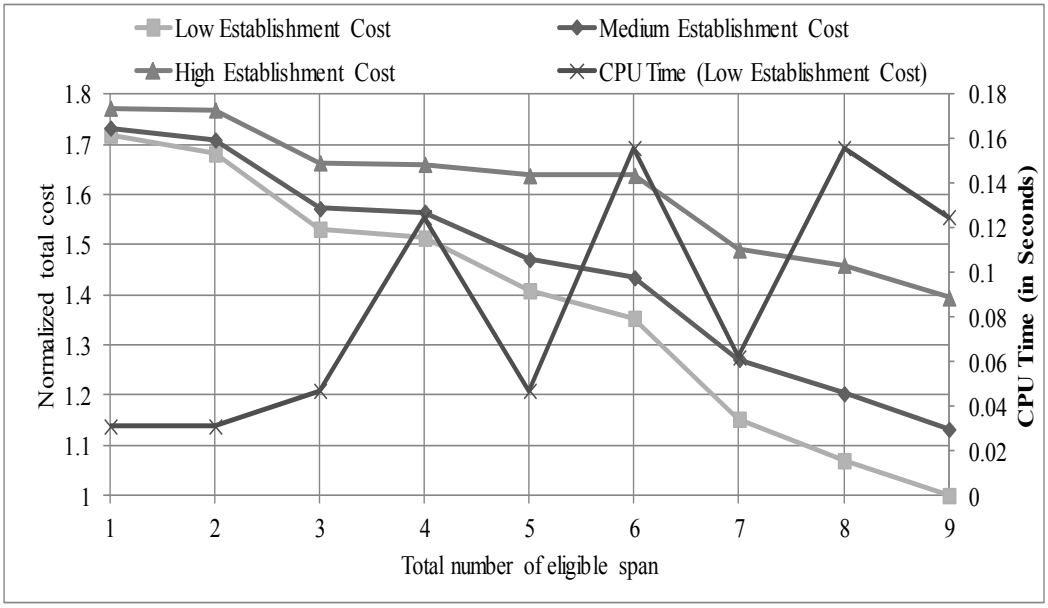


Figure 6-1. Total network costs and CPU time versus number of eligible spans for the 10-node network

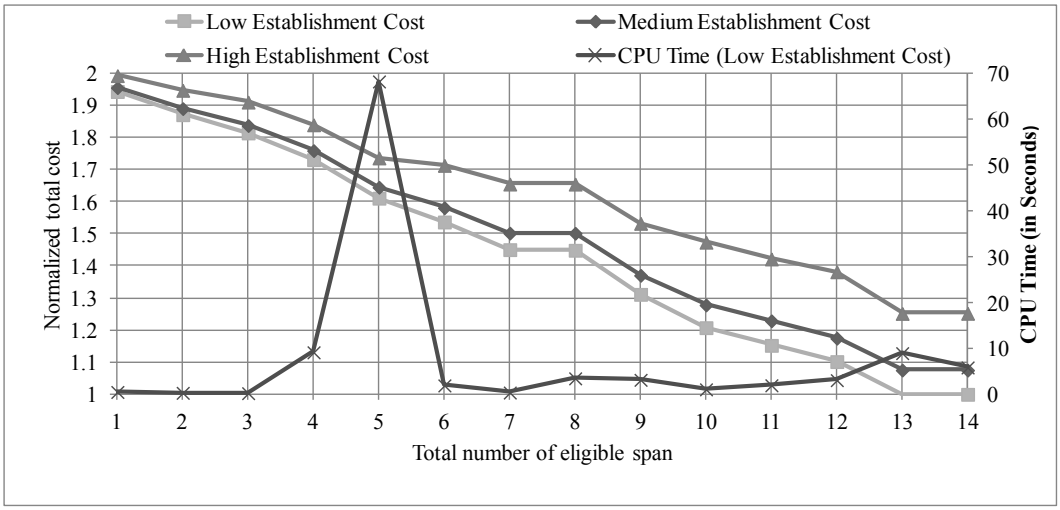


Figure 6-2. Total network costs and CPU time versus number of eligible spans for the 15-node network

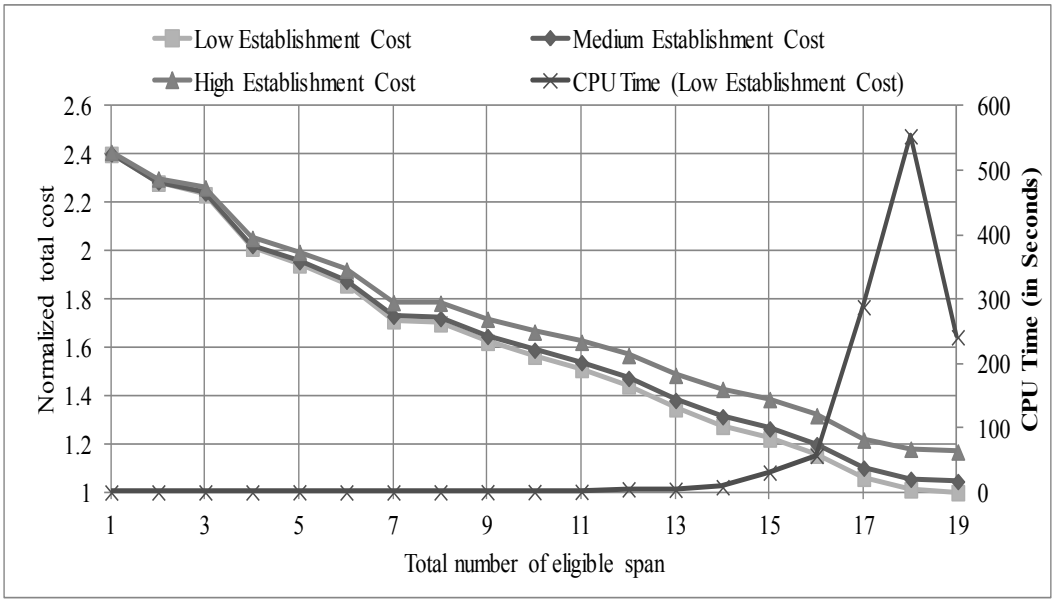


Figure 6-3. Total network costs and CPU time versus number of eligible spans for the 20-node network

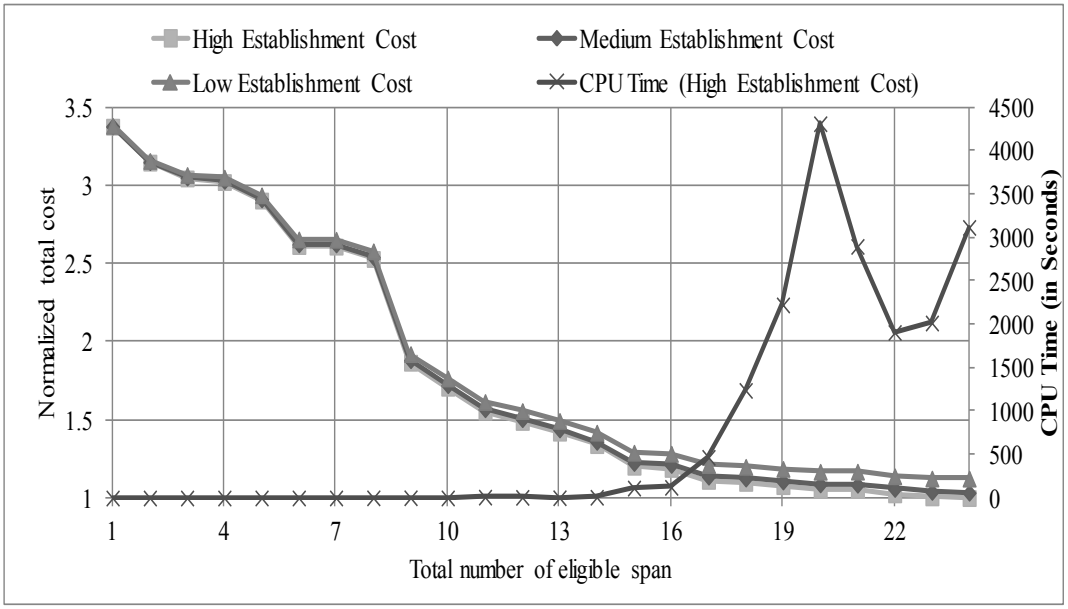


Figure 6-4. Total network costs and CPU time versus number of eligible spans for the 25-node network

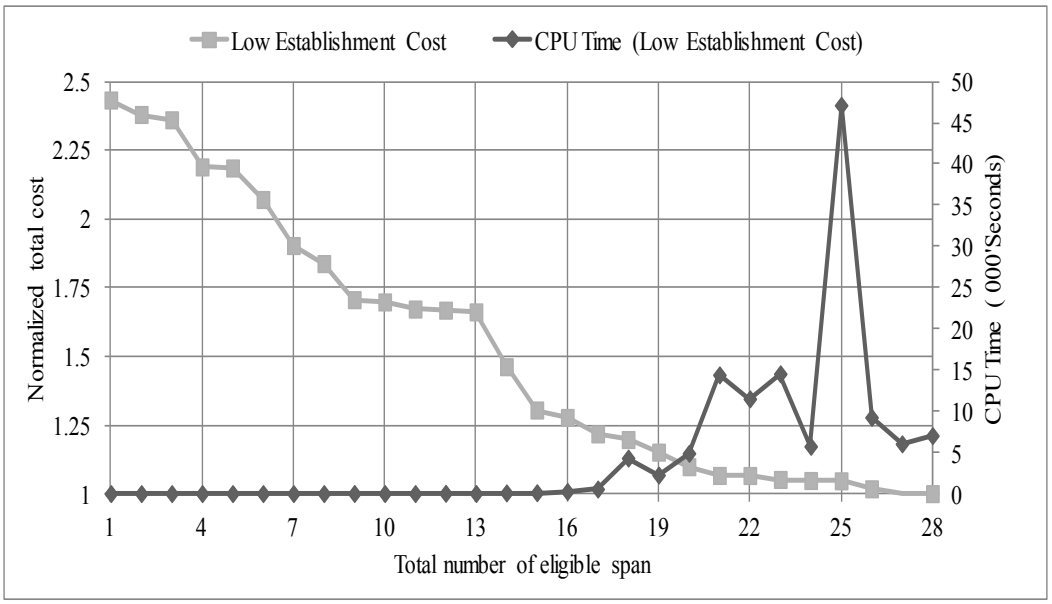


Figure 6-5. Total network costs and CPU time versus number of eligible spans for the 30-node network

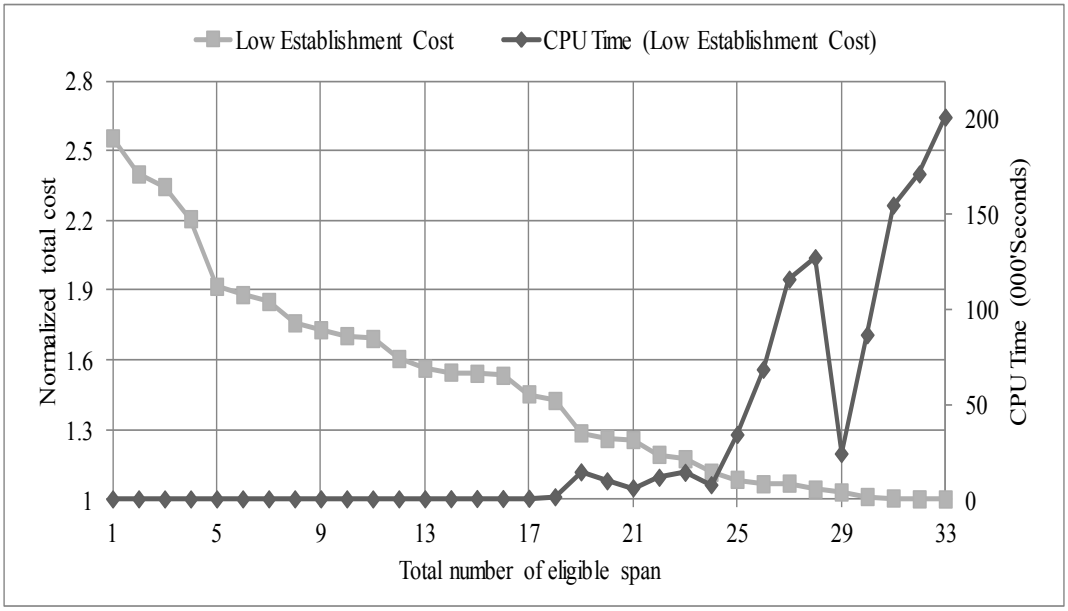


Figure 6-6. Total network costs and CPU time versus number of eligible spans for the 35-node network

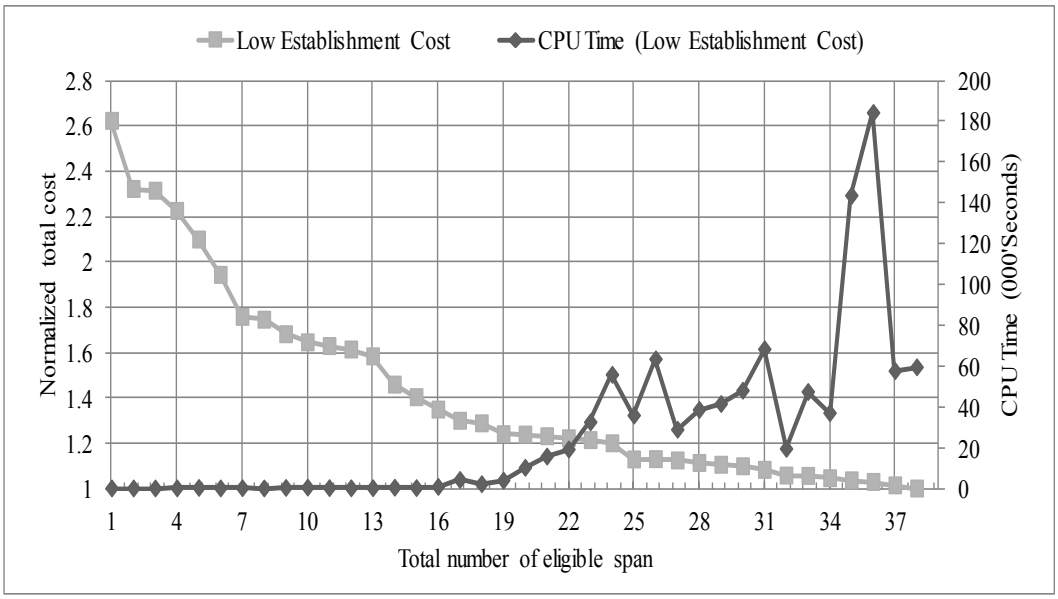


Figure 6-7. Total network costs and CPU time versus number of eligible spans for the 40-node network

As one can observe, runtimes are quite short (from fractions of a second to a few minutes) for the smaller test case networks, but become exceedingly high for larger test case networks, reaching nearly 200 thousand seconds (more than two

days) for the 40-node network with 78 eligible spans. While there is a general increasing trend in runtimes as we provide a greater number of eligible spans, we can notice that they often exhibit an irregular nature. Although it would be interesting if some useful insight could be gained from this observation, the cause is simply due to peculiarities in the network topologies and the nature of the solution approach. For instance, when the 15-node test case network is solved with 5 eligible spans (Figure 6-2), inclusion of that 5th eligible span results in enumeration of a specific set of eligible *p*-cycles that happens to be more computationally complex to solve than the test case with only 4 eligible spans or with 6 eligible spans. It might also be interesting to note that the number of branch-and-bound nodes produced by CPLEX's internal algorithm rises quite substantially in test cases corresponding to those instances with irregularly high runtimes, suggesting that simple peculiarities in the branch-and-bound tree contribute to these high runtimes. We suspect that the highly irregular nature of CPU times for those test-case networks were due to a complex interaction of the large number of spans in the network and topological effects (addition of a single span can often provide an obviously beneficial routing option that the solver takes advantage of). Such instances of the problem can create much tighter LP relaxations than other instances, and/or algorithms used by CPLEX's internal branch-and-bound procedures might be better suited to some of those specific cases. As a result, these instances see fewer branch-and-bound nodes when solving the ILP problem. It is these artifacts (i.e., the irregular nature of the runtime increases) in smaller to mid-size test case networks and more importantly, the extremely high runtimes in the large test case networks that motivates us to develop an alternative solution method for the *p*-cycle network topology optimization problem, as discussed later.

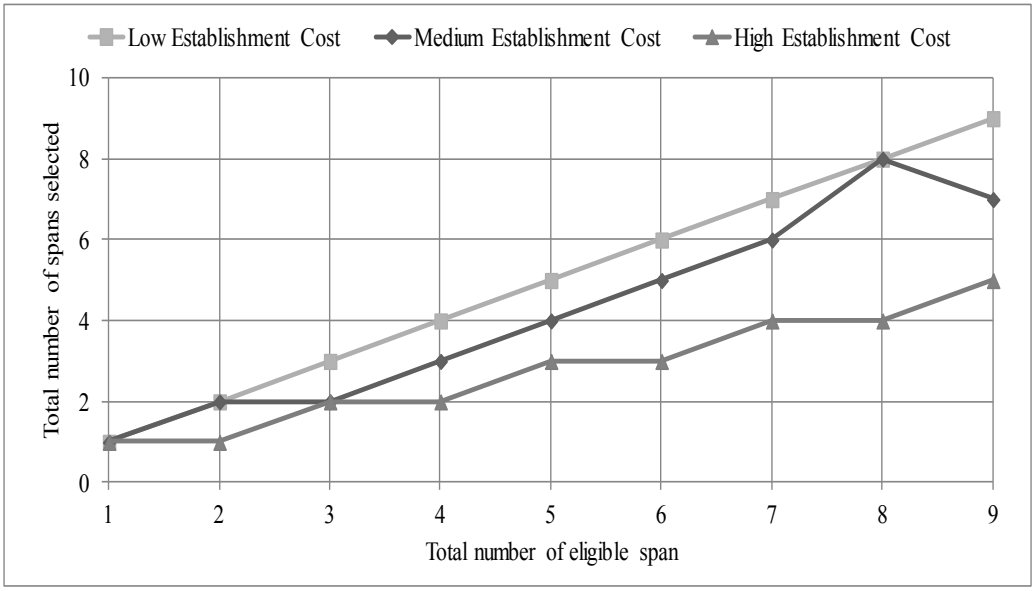


Figure 6-8. Variation of the total number of selected spans for 10node20span network

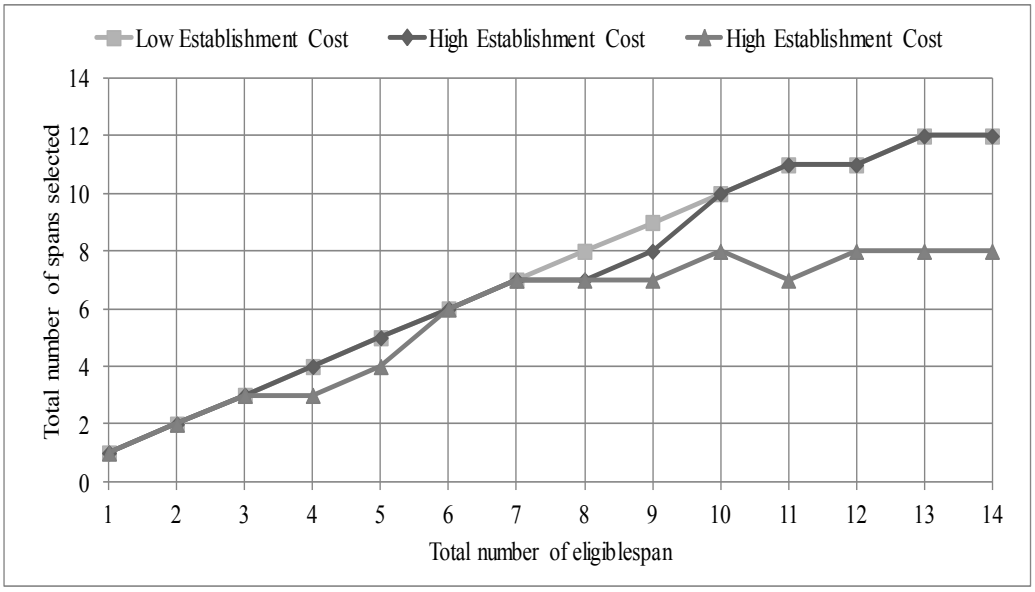


Figure 6-9. Variation of the total number of selected spans for 15node30span network

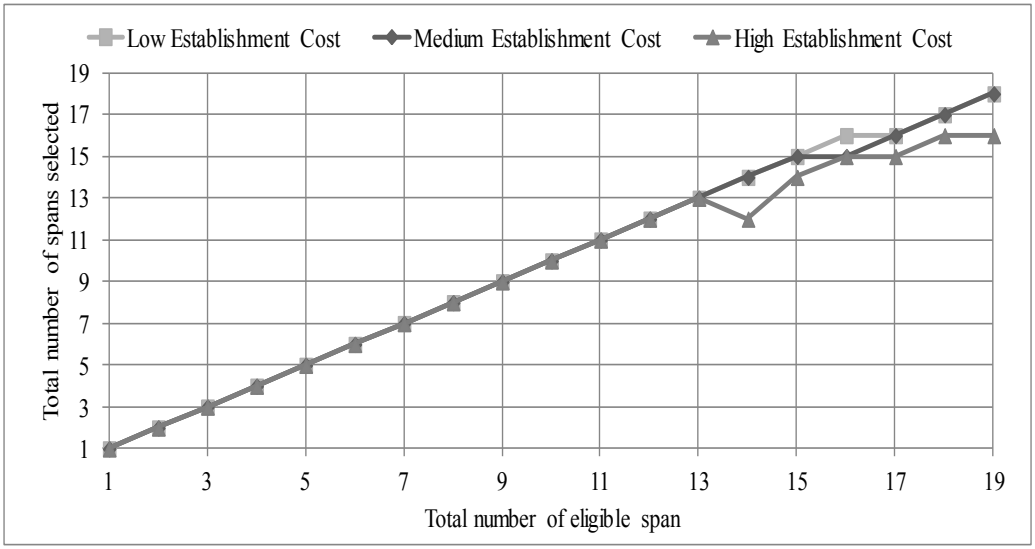


Figure 6-10. Variation of the total number of selected spans for 20node40span network

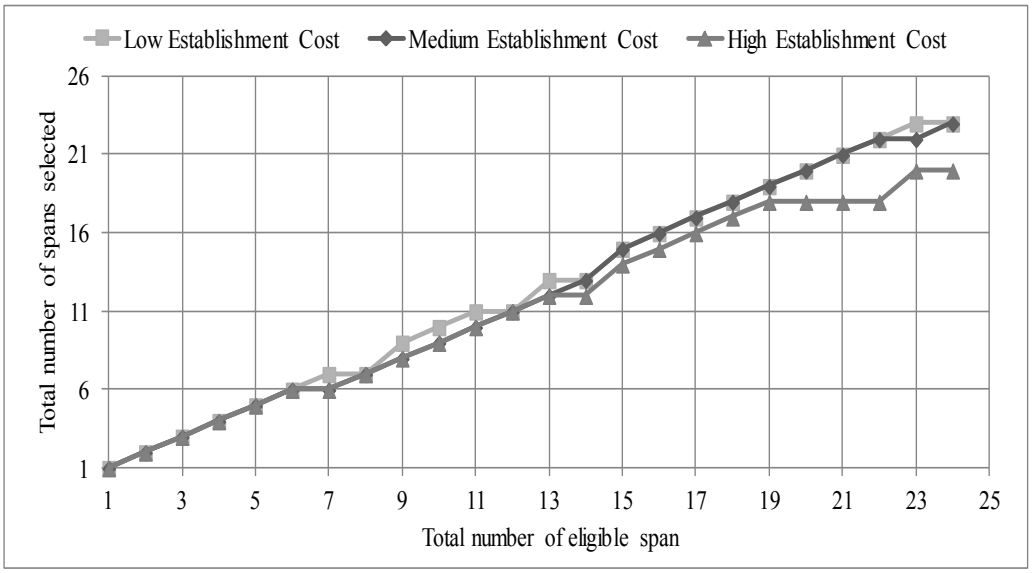


Figure 6-11. Variation of the total number of selected spans for 25node50span network

A closer look at the numbers of spans selected by the solver adds some additional interesting insights. As we can observe in Figure 6-8 through Figure 6-11, the number of spans selected decreases quite as we increase the span establishment cost factor (i.e., as spans become more expensive, relative to the per-unit capacity costs). However, for larger networks there is almost no variation initially (i.e., as

we provide only a few eligible spans), regardless of span establishment factor, but a small degree of variation arises when we provide a greater number of eligible spans. While this may initially seem indicative of some underlying phenomenon, the truth of the matter is that we happened to have selected span establishment factors that produce a lopsided objective function that is dominated by the span capacity costs in the larger networks. In hind sight, a smarter approach would have been to set higher stand establishment factors for these larger networks, so that the objective function is more balanced, with respect to the span capacity costs and the fixed establishment costs. With the span establishment factors we've used, the solver sees little disincentive to select quite a large number of the eligible spans (i.e., there is only a small cost to add extra spans, relative to the reductions in capacity that result).

6.5 Relaxation-Based Decomposition Technique

In order to be better able to solve the *p*-cycle network topology optimization problem in large test case networks, we now propose and develop a problem-specific relaxation-based decomposition technique for the ILP developed above. From the investigation of a hard ILP instance, it is sometimes observed that the computational complexity arises from a set of constraints or integrality properties of specific sets of variables. For the first scenario, we can dualize these hard constraints and create an easy sub-problem [90], [85], and the solution of this sub-problem can be used to solve the main problem. Our proposed technique is different from this approach in a sense that we decompose the original problem into two easy sub-problems by relaxing the integrality property of some variables rather than relaxing the set of constraints. While most advanced solvers, including CPLEX, utilize some form of relaxation-based approaches to speed up solution of ILP problems, such general approaches often have difficulty in properly selecting the best specific relaxations and sub-problem decompositions. With some insights into the problem at hand, insights that a general approach might not have, we can decompose the ILP problem into two sub-problems. First, we use a partially relaxed version of the original, which is more easily solved. We can then use the

solution from that problem to set fixed values for a subset of integer variables and resolve the original with that subset of integer variables acting as parameters. While the solution is not guaranteed to be optimal, proper selection of the integer variables to relax in the first sub-problem and of the integer variables for which we can fix their values in the second sub-problem can permit near-optimal solutions. As with most near-optimal algorithms, quality of the solution (in terms of both the objective function value and the runtime improvement) will depend on careful selection of those subsets of variables.

With the particular ILP problem that we developed above, we felt that if we could use a partially relaxed version of the problem to first identify which specific span additions to select, then we could fix that topology and solve the original unrelaxed problem with a known topology. We therefore decompose our problem as follows:

> **Step 1** – Relax working capacity w_j, spare capacity s_j, and *p*-cycle placement variables n_p and solve the original ILP problem. In other words, all of the integrality requirements on those decision variables are removed and the ILP model solved.
>
> **Step 2** – Fix all span establishment variables δ_i to the values obtained in Step 1. In other words, take the resulting values for all span establishment variables as solved in Step 1, and convert those variables to parameters with the same values.
>
> **Step 3** – Solve the original ILP, resetting integrality requirements in all relevant variables (but where all δ_i variables are fixed to the values in Step 2).

The main rationale for the above decomposition approach is that the span establishment variables are binary, and so fractional values would have very little meaning; if $\delta_i = 0$ then the span is not selected, and if $\delta_i = 1$ then the span is selected, but $\delta_i = 0.5$, for instance, is difficult to interpret in a manner that has any real physical meaning. The three sets of variables noted in step 1, however, can be permitted to take on fractional values and the solution can still impart some physical meaning. For instance, $w_j = 7.8$ would mean that 7.8 units of working

capacity are placed on span j, which might not strictly be feasible (one can't place a fractional unit of capacity) but is still conceptually understandable. In addition, the span establishment variables will still be driven to $\delta_i = 0$ or $\delta_i = 1$ whether the w_j, s_j, and n_p variables are integer or not. Then when we resolve the ILP model in step 3, with the span establishment variables fixed in step 2, the resultant ILP model is equivalent to the basic p-cycle network design problem.

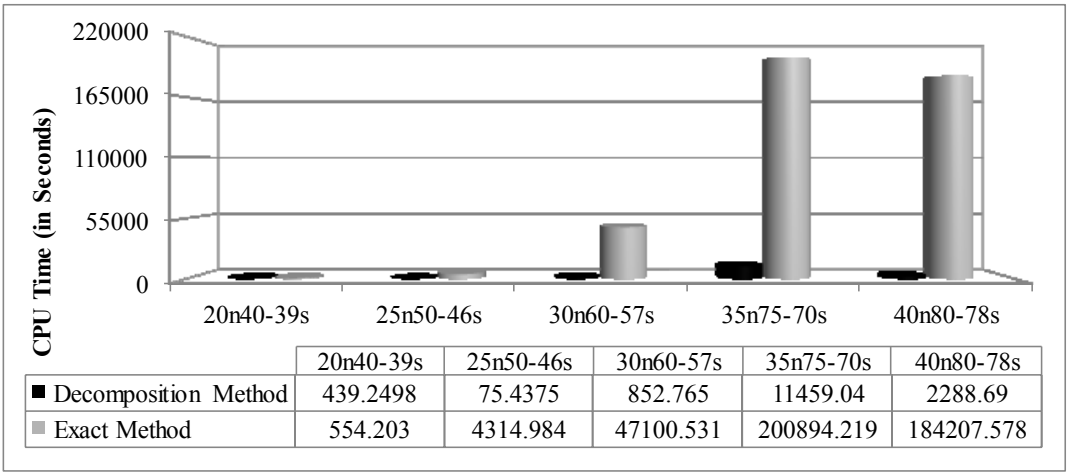
6.6 Results for Decomposition Method

To test the performance of this above technique, we selected the most computationally complex instance of the problem for each network (though we skip the 10-node and 15-node networks, as their solutions are already trivial). More specifically, we tested the decomposition technique on the 20-node network with 18 eligible spans, the 25-node network with 20 eligible spans, the 30-node network with 26 eligible spans, the 35-node network with 24 eligible spans, and the 40-node network with 37 eligible spans. And as stated earlier, our ILP models were implemented in AMPL and solved with the CPLEX 11.2 solver. We used a CPLEX mipgap setting of 0.001, which means all test cases solved to full termination are provably within 0.1% of optimality.

Figure 6-12 compares the CPU runtimes of the decomposition approach with the original ILP solution. The general trend is that runtime improvements are greater in larger test cases. As we can observe, we see only moderate runtime improvements for the mid-size networks (a 21% reduction in the 20-node test case) but significantly greater runtime improvements in the largest networks (a 99.99% reduction in the 40-node test case).

Of course, the tradeoff when implementing a heuristic approach is often a reduction in optimality of the resulting solution, so we also need to compare the solutions we obtain with the decomposition approach with the solutions from the full ILP model. As we can see from Table 6-1, the solutions obtained with the decomposition technique are at worst only 0.016% more costly than the full ILP, and in three of the five cases the decomposition approach provides a less costly

solution than the full ILP. This is counterintuitive, as the cost of the full ILP solution should serve as the lower bound on the cost of solutions obtained via our heuristic approach. The explanation is that the differences are smaller than the mipgap setting of 0.1%. In fact, we can note than in all cases, the difference between the solutions obtained from the decomposition approach and the full ILP are within the optimality gap setting of 0.1%, which means the two approaches effectively provide equivalent solutions.



	20n40-39s	25n50-46s	30n60-57s	35n75-70s	40n80-78s
Decomposition Method	439.2498	75.4375	852.765	11459.04	2288.69
Exact Method	554.203	4314.984	47100.531	200894.219	184207.578

Figure 6-12. Comparison of CPU time between decomposition method and exact method

Table 6-1. Comparison of normalized total cost between decomposition method and exact method

Networks	Normalized total cost		Percentage difference
	Exact (full ILP) Method	Decomposition Method	
20n40-39s	1	1.000137	0.013776
25n50-46s	1.76266	1.762942	0.01603
30n60-57s	2.834284	2.833077	-0.0426
35n75-70s	3.634334	3.633292	-0.02866
40n80-78s	4.468206	4.466802	-0.03142

6.7 Conclusion

We have developed a new ILP model for incremental topology optimization in a *p*-cycle network that is capable of selecting an optimal subset of eligible spans to add to an existing *p*-cycle network. While the ILP model proves to be relatively easy to solve for small test case network instances, it is computationally complex to solve for larger networks. We then developed a relaxation-based decomposition heuristic that significantly reduces runtime of the ILP models in our large test networks, while having no statistical impact on optimality. In the most computationally complex instance, the ILP runtime of over 184 thousand seconds (more than two days) was reduced to less than 2300 seconds (less than an hour), while the objective function value remained within the optimality gap. In fact, the heuristic solution was slightly better than the full ILP model (though again, we note that it was not provably better since the difference was smaller than the optimality gap). In the future work, we will investigate using Column Generation technique to solve this problem instead of using relaxation-based decomposition

heuristic. CG could be a suitable technique for this problem because the number of zero-decision variables is much more than the basic variables in all of the experiments that we have conducted.

Chapter 7. Efficient Algorithms for Node-Encircling *p*-Cycle Network Design

Part of this chapter represents the following paper: "***Algorithmic Approaches for Efficient Enumeration of Candidate Node-Encircling p-Cycles***", (10th INFORMS Telecommunications) Conference, Concordia University, Montreal, Quebec, Canada 5 – 7 May 2010.

7.1 Introduction and Background

Path restoration can be described as an end-to-end mesh restoration technique; when a span fails, the affected working route is replaced by a new restoration route routed fully back from the origin node to the destination node [11], [49]. In its most efficient variant, surviving spans of the affected working routes are released as spare capacity, ready to be used during the restoration process of any of the working paths affected by the failure. This latter operation is called *stub release*. Stub release helps path restoration achieve a greater efficiency by effectively reducing the spare capacity required (since surviving capacity on failed working paths is cannibalized as spare), but it also complicates the reversion process after repairing the failure. In general, the new path is not fixed or even necessarily known in advance. Rather, depending on the spare capacity available in the network, restoration routes will be formed wherever possible (though if properly designed, there will be sufficient spare capacities to fully restore all failed paths).

Span restoration is a form of mesh restoration where a failed span is replaced with restoration paths formed between the end-nodes of the failed span. Span restoration can be considered a single-commodity maximum flow problem, as there is only one single group of paths that are to be routed [52], [11].

7.1 Node-Encircling *p*-Cycles

Since conventional *p*-cycles are fundamentally a form of span restoration, they are capable of protecting only span failures, not node failures. The idea of node-encircling *p*-cycles was introduced in the context of IP layer restoration to protect against router failures [57], but the concept is also applicable to optical layer restoration. It was also shown in [91] that simply designing a conventional *p*-

cycle network for protection against span failures is often sufficient to protect a significant amount of node failure protection. That prior work demonstrated 15%-25% inherent node-failure restoration arising from optimally-designed conventional span-protecting *p*-cycle networks.

An NEPC functions by providing protection for any lightpaths transiting the failed node. To do so, the *p*-cycle must cross all nodes immediately adjacent to the failed node but not the node itself, as illustrated in Figure 7-1. In the upper panel, the five-node *p*-cycle in blue crosses all three nodes adjacent to the protected node in red (the adjacent nodes are shaded in gray). Any lightpaths transiting the protected node (i.e., passing through but not terminating at or originating from the node) must therefore pass through the *p*-cycle at two separate points. More specifically, this will pass the *p*-cycle at two surviving nodes immediately adjacent to the failed encircled node, so the failure will be visible to both of those surviving nodes. Upon failure of the encircled node, the transiting lightpath can be rerouted in either direction around the *p*-cycle; as with straddling spans in a conventional *p*-cycle, a unit-sized copy of an NEPC can protect two transiting lightpaths. In the lower panel, on the other hand, the six-node *p*-cycle does not pass through all nodes adjacent to the protected node, and so some lightpaths transiting the encircled node will not necessarily cross the *p*-cycle at two points.

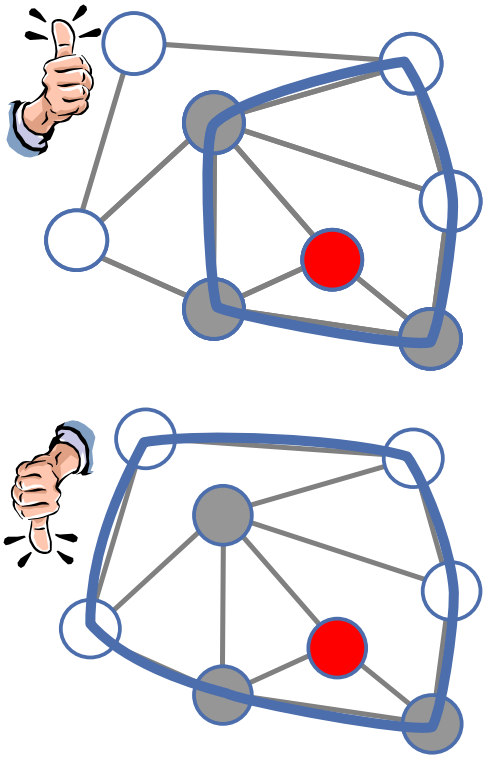


Figure 7-1. The node-encircling *p*-cycle concept

While many NEPCs are simple and visibly encircle the protected node, this is not always the case, as illustrated in Figure 7-2. In the upper panel, the *p*-cycle is simple (i.e., it does not cross the same node and/or span more than once), but it does not visibly encircle the node in question. Nonetheless, it logically encircles the protected node, as all adjacent nodes are crossed. Likewise, the two NEPCs in the center and lower panels also logically encircle the protected node, despite the fact that both are non-simple cycles and one of them (the lower one) does not visibly encircle the node.

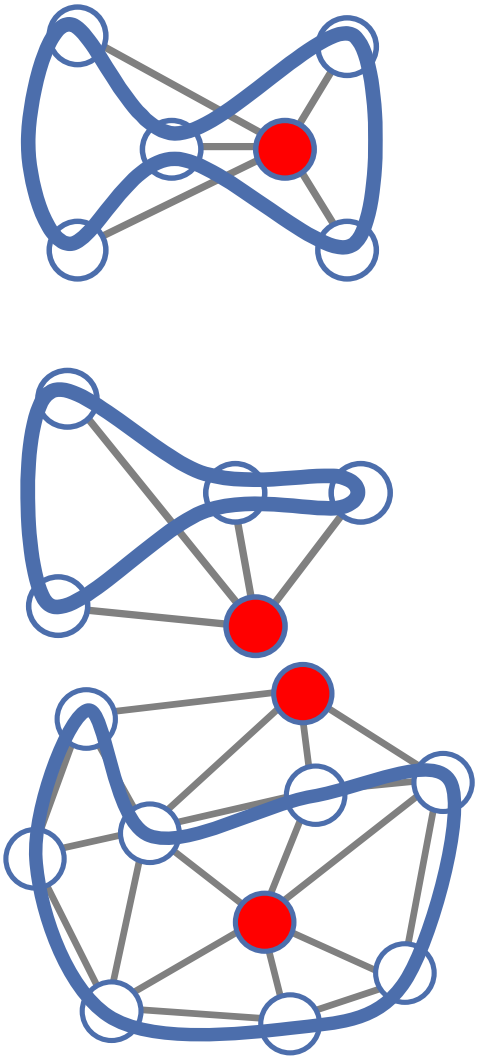


Figure 7-2. Three more node-encircling *p*-cycles

This chapter is devoted to look at the algorithmic approaches for solving the NEPC network design problem. In the Integer Linear programming of *p*-Cycles and Node-encircling *p*-cycles network design, the first and the most time consuming step is to enumerate a number of eligible cycles which could be used in the final solution. Enumerating all cycles in the network is an impractical approach specifically in the large size networks because the number of generated cycles grows exponentially with the increase of the number of nodes and spans. Many algorithms have been proposed for enumerating respectively small set of candidate *p*-cycles without degrading the optimality of the final ILP solution significantly. However, few algorithms have been developed for NEPC design namely *Node-encircling p-cycle Mining Algorithm* (NCMA) [92] and *Local-map*

Cycles Mining Algorithm (LCMA) [93]. We will develop in this chapter another two algorithms for this problem which are *Node-Disjoint Path Partitioning Algorithm* (NDPP) and *Level Partitioning Algorithm* (LPA). However, preparing the cycles with high efficiency is just a first step towards finding the least cost combination of cycles that will protect the network under investigation. Because a priori efficiency is merely an expected efficiency, it may be released as an actual efficiency if and only if the working capacities are merely existed [94]. To release a complete network design, an algorithm should be used to select the least cost combination of cycles that will fully protect the network working capacities. A CIDA-like algorithmic approach, for providing fully capacitated NEPC networks, is developed. Then, a GA model is proposed to enhance the CIDA-Like algorithm by determining the best values for its factors.

7.2 Cycle Enumeration Methods

7.2.1 Cycle Enumeration Algorithms

The benchmark approach, for designing an NEPC network, is via *integer linear programming* (ILP) formulation. When using an ILP model, the first (and the most time consuming) step is to enumerate a number of eligible cycles those are considered and ultimately selected by the ILP model. Enumerating all possible cycles in the network is not terribly complicated, as a standard *depth first search* (DFS) algorithm will suffice. For small networks, it can even be practical for the full enumeration, as it may only involve several thousand cycles. However, for any moderately sized networks and larger, it becomes intractable, as the number of generated cycles grows exponentially with the increase in the number of nodes and spans in a network. Many algorithms have been proposed for enumerating small subset(s) of candidate *p*-cycles, ideally without significantly degrading the optimality of the final ILP solution. Perhaps the simplest is to specify some desired number (say, *n*) of eligible cycles, and use DFS while incrementally paring down the maximal extent (i.e., depth) of the algorithm to match the decreasing length of

the n^{th} shortest enumerated cycle in each iteration. Additional more problem specific approaches are also discussed in the literature.

The *Straddling Link Algorithm* (SLA) was proposed for enumerating a trivially small set of cycles equal to the number of spans in the network (i.e., one eligible *p*-cycle per span) [95]. The main idea is to find a cycle for each given span such that the cycle passes through the nodes of the span in question without traversing it. Subsequently in [94], that initial primary set of *p*-cycles generated from SLA were extended by use of the *SP-Add*, *Grow*, and *Expand* algorithms, which were been developed to generate a more useful set of candidate cycles, with the intent to improve the ILP solution quality without significantly increasing it's time complexity. In [94], the *weighted DFS-based cycle search algorithm* (WDFS) was proposed to provide an efficient solution for both sparsely and densely distributed working capacities by generating two groups of cycles. The first group is short cycles while the second group consists of larger more efficient cycles. However, there is other research work that focuses on *p*-cycle design without candidate cycle enumeration [96].

Unfortunately, such algorithms are not appropriate for use in node-encircling *p*-cycles, as they do not consider the specific nature of NEPCs and the manner in which they protect transiting flows over failed nodes (in general, a set of cycles enumerated via the above methods is not guaranteed to have any NEPCs at all). Even the modified DFS approach mentioned above is not suitable, since DFS (at least in its present form) cannot consider any particular cycle characteristics (e.g., protection of some specified node's transiting flow, etc.) other than length. As such, we cannot incrementally pare down the extend (depth) of the algorithm; in the general case, it will still need to exhaustively enumerate eligible cycles to ensure enumeration of the shortest subset of eligible NEPCs.

There is a little work in the literature dealing with the efficient enumeration of good candidate node-encircling *p*-cycles. To our knowledge, only *Node-encircling p-Cycle Mining Algorithm* (NCMA), [92], *Local-map Cycles Mining Algorithm* (LCMA), [93], has been proposed. In the NCMA and LCMA algorithm, the

simplest node-encircling *p*-cycle is found first, based on a contraction algorithm and the local-map, respectively. Then several expanding algorithms (*SP-Add*, *Grow*, and *Expand* from [94]) are applied to generate more efficient candidate NEPCs. In these works, a modified a priori efficiency is used to evaluate the efficiency of each NEPC. However, preparing the cycles with high efficiency is just a first step towards finding the least cost combination of cycles that will protect the network. Because a priori efficiency is merely an *expected* efficiency, and it will be realized as an *actual* efficiency if and only if the working capacities exist as used in the calculation [94]. To properly realize a complete network design, an algorithm is required to select the least cost combination of cycles that will fully protect all the network working capacities. To do so, we propose two new algorithms, the *Node-Disjoint Path Partitioning* (NDPP) algorithm and *Level Partitioning Algorithm* (LPA).

7.2.2 Node-Disjoint Path Partitioning Algorithm

NDPP proceeds as follows (and as shown in Exhibit 1). First, we find the neighbours for a given node (we can call it the "central node") and record them as "neighbours". Second, find at least two node-disjoint paths between any two nodes in neighbours. Third, identify all additional nodes found in paths from the second step, and record them as "nodes-in-paths". Fourth, prune off all nodes in the network (and their incident spans), except for the central node, the neighbours, and the nodes-in-paths. Fifth, enumerate all the *p*-cycles found in this network partition using a standard depth first search algorithm. Finally, repeat until all nodes of the network have been considered as a central node.

Exhibit 1 – Node-Disjoint Path Partitioning pseudo-code.

```
NDPP() {
 initialize set Cycles
 for each node n {
  initialize set Nodes1
  identify the level 1 neighbours of node n and add
    them to Nodes1
```

```
  for each nodes n1 and n2 in Nodes1 {
   initialize set Nodes2
   add n, n1, and n2 to Nodes2
   prune off n and the links adjacent to it
   find two disjoint paths p1 and p2 between nodes n1
     and n2
   add all nodes in p1 and p2 to Nodes2
   use a standard depth-first search algorithm to
     enumerate all cycles in the sub-network composed
     of nodes in Nodes2 and all spans connecting them
   add the above cycles to set Cycles
  }
 }
}
```

7.2.3 Level Partitioning Algorithm

In Level Partitioning, as shown in Exhibit 2, we first need to define *level m neighbours* of a node; all nodes that are *m* hops from another node are its level *m* neighbours. Level Partitioning then proceeds as follow. First, find the level *m* neighbours (starting with $m = 1$ at the beginning) for a given central node and record them as "neighbours". Second, prune off all the nodes in the network (and their incident spans), except the central node and its neighbours. Third, enumerate all the *p*-cycles found in this network partition using a standard depth first search algorithm. Fourth, if an NEPC for the central node is not found, increase the level by one and repeat. Finally, repeat all previous steps until all nodes in the network have been considered as a central node.

Exhibit 2 – Level Partitioning pseudo-code.

```
LevelPartitioning() {
 initialize set Cycles
 for each node n {
  m = 0
  initialize set Nodes
  add n to Nodes
  until Cycles contains an NEPC of n {
   m = m + 1
```

```
    identify the level m neighbours of node n and add
      them to Nodes
    use a standard depth-first search algorithm to
      enumerate all cycles in the sub-network composed
      of nodes in Nodes and all spans connecting them
    add the above cycles to set Cycles
   }
  }
}
```

7.3 Analysis of NDPP and Level Partitioning

In order to determine the effectiveness of the NDPP and Level Partitioning algorithms, we implemented both in C++ and ran them on eight test networks of various sizes, ranging from 10 nodes and 20 spans ("10n20s") to 45 nodes and 90 spans ("45n90s"). In each network, every node-pair exchanged a uniform random number of lightpaths between 1 and 10 (inclusive). In addition, we implemented an adapted DFS algorithm in C++. The DFS algorithm, which we call the benchmark solution, was adapted as described above so that it first provided the shortest n eligible cycles, where n is the larger than the number of cycles enumerated by NDPP and the number of cycles enumerated by Level Partitioning for the network in question. However, the DFS had to continue its search until additional cycles were added as needed so that each node could be protected by at least one NEPC. While the DFS was required to go much of the way through an entire full enumeration, we felt that the modifications we made (i.e., first step finds only a small subset of cycles, paring down the depth of the algorithm as we go, and then only continuing through the standard DFS-like approach) provided the best possible runtime while still guaranteeing a suitable eligible set of cycles for NEPC protection. In other words, we feel that it is a fair benchmark to which we can compare our new proposed algorithms. All tests were run on an Intel Core 2 Duo PC running at 2.13 GHz with 4 GB memory.

Table 7-1 compares the runtimes of the NDPP and Level Partitioning algorithms with the runtime for the benchmark solution. As expected, runtimes grow in an exponential fashion with the size of the network, even for the NDPP and Level

Partitioning algorithms. For small networks, the two new algorithms do not necessarily outperform the benchmark approach, but as the network size grows, the runtime of the benchmark approach increases much more quickly.

Table 7-1. Runtime comparison of Level Partitioning and NDPP with the benchmark

Networks	Benchmark	Level Patitioning	NDPP
10n20s	46 ms	31 ms	15 ms
15n30s	62 ms	281 ms	62 ms
20n40s	1.125 sec	2 sec, and 984 ms	93 ms
25n50s	5.390 sec	1.156 sec	218 ms
30n60s	30.875 sec	1 min, 27.406 sec	625 ms
35n70s	2 min, 10.718 sec	8 min, 11.781 sec	1.156 sec
40n80s	1 hr, 46 min, 19.281 sec	1 hr, 5.328 sec	2.781 sec
45n90s	I day, 1 hr, 52 min, 56.140 sec	3 min, 3.921 sec	4.906 sec

We can also note that the runtime for the level partitioning approach did not increase as consistently as the others. For instance, the runtime for the 45-node network (approximately 3 minutes) was significantly less than that for the 35-node and 40-node networks (8 minutes and 1 hour, respectively). We suspect that this is due to the effectively pseudo-random nature in which the level *m* neighbour nodes are identified and the interaction of specific topological aspects of the networks in question.

7.4 NEPC-CIDA Algorithm

Once we have enumerated a suitable set of eligible cycles, we then need to select amongst them to produce a capacity efficient network design where all working capacity and transiting node traffic is protected by *p*-cycles (NEPC or otherwise). As mentioned above, the conventional approach is to use an ILP model. However, the Capacitated Iterative Design Algorithm (CIDA) was proposed in [94] to select from a subset from candidate cycles in order to design a *p*-cycle network that is fully protected against all single-span failures at near optimal spare capacity. The main idea behind this algorithm is to calculate the (weighted) efficiency of each candidate cycle using equation (7.1), where S is the set of spans in the network, w_i is the amount of unprotected working capacity on span i at the present time, c_i is the cost of span i, and $X_{p,i}$ is the number of protection relationships available to span i from cycle p ($X_{p,i} = 1$ if i is an on-cycle span, and $X_{p,i} = 2$ if i is a straddling span) [94]. Note that w_i values are initially set to the network's overall span working capacities as routed via shortest paths (or some other routing solution, as desired)

$$E_w(p) = \frac{\sum_{\forall i \in S} w_i \cdot X_{p,i}}{\sum_{\forall i \in S | X_{p,i}=1} c_i} \tag{7.1}$$

A copy of the most efficient eligible cycle is selected and a unit-capacity copy of the cycle is placed in the design. The working capacities on all spans are updated by subtracting one working unit from each on-cycle span for the chosen cycle and two from each straddling span, thereby allowing the w_i quantities to represent working capacity that has not yet been protected. The eligible cycles' efficiencies are recalculated and the process is repeated iteratively until all working capacities has been protected (i.e., all $w_i = 0$).

While the CIDA algorithm described above was suitable for basic *p*-cycle network design, the approach will not work for NEPC networks since there is no consideration for protection of transiting flows (and not even any consideration for transiting flows in the eligible cycles' efficiency calculations). If we wish to adapt

CIDA for the application to NEPC network design, we first need to alter the actual efficiency equation to consider for transiting flows. The new actual efficiency equation will be as follows, where f_n is the sum of the unprotected transiting flows through node n and $X_{p,n}$ is the protection relationship available to node n from cycle p (i.e., $X_{p,n} = 2$ if n is an encircled node, and otherwise $X_{p,n} = 0$):

$$E_w^{NEPC}(p) = \frac{\sum_{\forall i \in S} w_i \cdot X_{p,i} + \sum_{\forall n \in N} f_n \cdot X_{p,n}}{\sum_{\forall i \in S | X_{p,i} = 1} c_i} \tag{7.2}$$

We can then adapt the CIDA algorithm from [94], which we now call NEPC-CIDA, as follows (and as shown in Exhibit 3). First, calculate the (weighted) efficiency of each candidate cycle using equation (7.2), where w_i and f_n are initially set to the network's overall working capacities and transiting flows as routed via shortest paths (or some other routing solution, as desired). A copy of the most efficient eligible cycle (either a basic p-cycle or an NEPC, as the case may be) is selected and a unit-capacity copy of the cycle is placed in the design. The working capacities on all spans are updated by subtracting one working unit from each on-cycle span for the chosen cycle and two from each straddling span, thereby allowing the w_i quantities to represent working capacities those have not yet been protected. The transiting flows are also updated by subtracting two from the f_n values of any nodes encircled by the cycle (of course, this is only in cases where the cycle is an NEPC), thereby allowing the f_n quantities to represent transiting flows those have not yet been protected. The eligible cycles' efficiencies are recalculated and the process is repeated iteratively until all working capacities and all transiting flows have been protected (i.e., all $w_i = 0$ and all $f_n = 0$).

Exhibit 3 – NEPC-CIDA pseudo-code.

```
NEPC-CIDA() {

  Initialize CycleSet, work[], TransitFlow[] and
    CycleUse[]
  CycleSet = EnumerateCycles()
  While work[i] or TransitFlow[]  > 0 {
    BestCycle = 0
```

```
    For each cycle p in CycleSet {
     Calculate Ew(p)
     If Ew(p) > Ew(BestCycle) {
          BestCycle = p
     }
    }
    CycleUse[BestCycle] =  CycleUse[BestCycle] + 1
    For each on-cycle span i in BestCycle {
     work[i] = work[i] - 1
    }
    For each straddling span i in BestCycle
     work[i] = work[i] - 2
    }
    For each encircled node n in BestCycle
     TransitFlow [i] = TransitFlow [i] - 2
    }
  }
Return CycleSet and CycleUse
}
```

7.5 Analysis of NDPP and Level Partition in NEPC-CIDA Network Designs

Although we have already demonstrated improvements in runtime using NDPP and Level Partitioning, we now need to determine their cost effectiveness in generating overall network solutions, for which we use NEPC-CIDA. As above, we implemented NEPC-CIDA in C++ and used the benchmark DFS, NDPP, and Level Partitioning, respectively, to initialize `CycleSet`. We ran all three variants of NEPC-CIDA on the same eight test networks as above and using the same computer described above. Figure 7-3 shows the spare capacity requirements of the resultant capacitated networks. While capacity is calculated as wavelength-km units (with span lengths equivalent to the Euclidian distances between the end-nodes of each span as drawn in the network graphs), the results normalized to the most capacity efficient solution for each test-case network (level partitioning for the 30-node and 45-node networks, and NDPP for all others).

As can be seen in the figure, NDPP achieves the lowest capacity requirements for most networks. The greatest improvement relative to the benchmark method is approximately 9% in the 20-node test-case network. Overall, the NDPP approach outperforms the benchmark method by 3.5% on average in our test-case networks. A more in-depth analysis of the network designs suggests that the efficiency improvements observed in the NDPP designs arise from a more efficient set of eligible cycles that more effectively act as NEPCs.

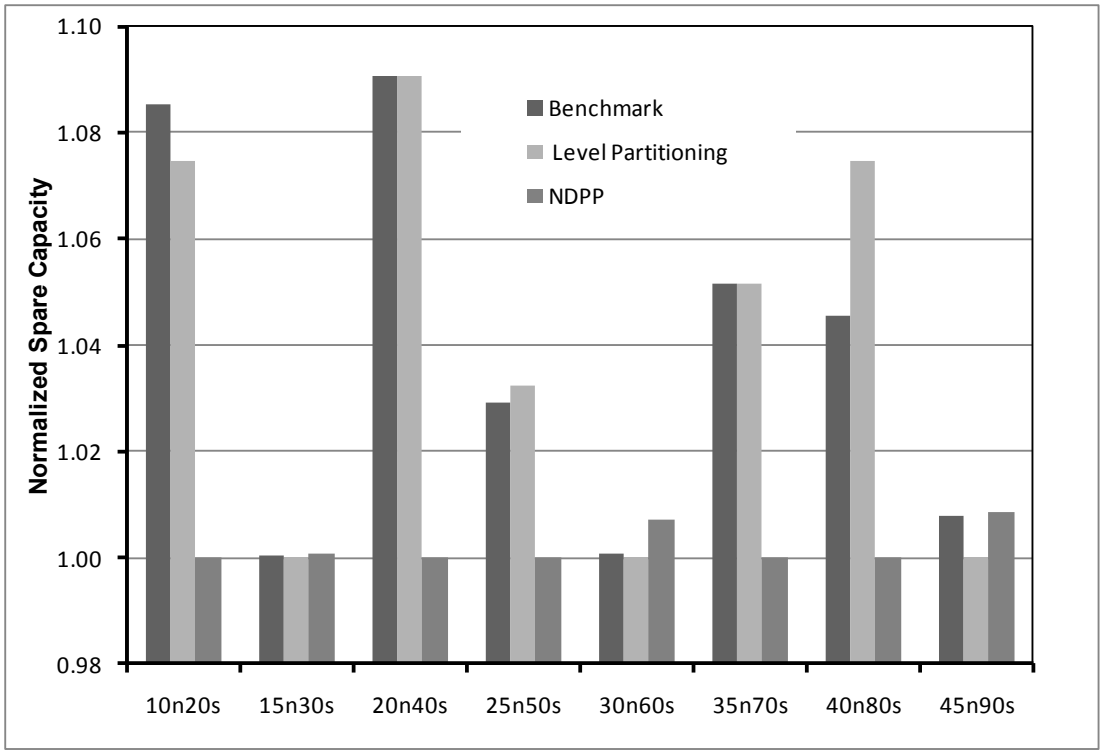


Figure 7-3. Normalized spare capacity costs of NEPC-CIDA designed networks using the benchmark, Level Partitioning, and NDPP enumeration approaches

In Figure 7-4, we compare the spare capacity requirements of NEPC networks designed by the CIDA-like algorithm using the eligible cycles enumerated with our NDPP approach to those using eligible cycles enumerated with prior enumeration approach from the literature, LCMA and NCMA (reproduced to the best of our abilities, given the description provided by the authors in [93] and [92]). As shown in Figure 7-4, the NDPP approach outperforms LCMA and NCMA for all test-case networks, in some cases by a large margin. The

improvement differs from network to network and reaches a maximum in the 40-node network with an improvement of 17%. Overall, the NDPP approach outperforms LCMA and NCMA by approximately 6%.

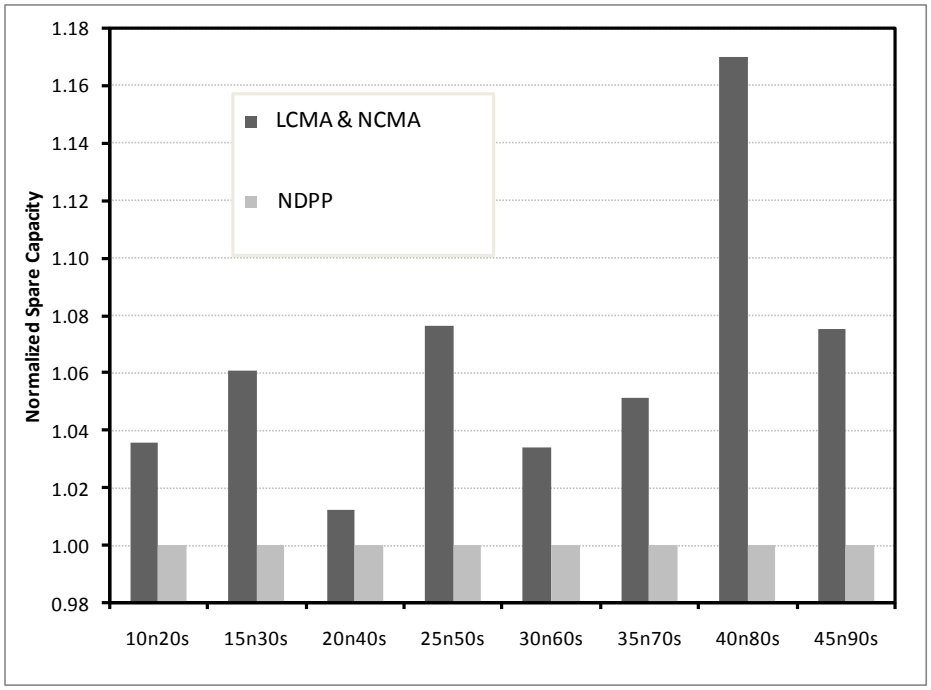


Figure 7-4. Normalized NEPC-CIDA spare capacity with NDPP and LCMA & NCMA

We now focus on equation (8-2) where we use two "factors" with specific values to evaluate the efficiency of each cycle. The first factor is $X_{p,i}$, where $X_{p,i} = 1$ if i is an on-cycle span for p-cycle p, $X_{p,i} = 2$ if i is a straddling span for p-cycle p, and $X_{p,i} = 0$ if span i is neither an on-cycle span nor a straddling span for p-cycle p. $X_{p,n}$ is the second factor, where $X_{p,n} = 2$ if n is an encircled node for p-cycle p and $X_{p,n} = 0$ if n is not an encircled node for p-cycle p. While the above values follow logically from knowledge of how p-cycles (NEPC or otherwise) protect working capacity (two units of capacities on straddling spans, one unit on on-cycle spans, and two units for transiting flows on encircled nodes), it is obvious that if we would use different values for any of those parameters, the efficiencies of the various candidate cycles would change, as would their rank order. In other words, if we would use different values for $X_{p,i}$ and $X_{p,n}$ in NEPC-CIDA, selection of `BestCycle` may be affected at any (or potentially all) iterations of the algorithm.

Furthermore, this may result in a slightly different design solution (i.e., a different set of *p*-cycles might be placed in the overall network design), which may in turn affect overall capacity requirements (even potentially resulting in reduced capacity requirements). As such, we now propose a genetic algorithm to determine an optimal set of values for $X_{p,i}$ and $X_{p,n}$.

7.6 Genetic Algorithm

In the following subsections, we will try to shed the light on some of the essential components in order to create an efficient genetic algorithm system.

7.6.1 First Population

The first population is a set of eligible solutions for the given problem. Two main factors should be considered in the first population: the representation scheme and the population size.

7.6.2 Representation Scheme

The representation scheme is the first step that has to be considered when utilizing a genetic algorithm for tackling any optimization problem. It represents the process of assigning an abstract code for any eligible solution (chromosome). Binary representation scheme and the real representation scheme are the two main schemes used in the realm of genetic algorithms. Deciding which type of representations will be used relies on the structure of the investigated problems.

In the real representation scheme, each chromosome consists of a number of decision variables called genes. As illustrated in Figure 7-5, each one could take any value between 0 and positive infinity. For example, in genetic algorithm model for NEPC-CIDA, the chromosome consists of three genes, each of which represents the value assigned to the on-cycle factor, the straddling factor, and the transit flow factor.

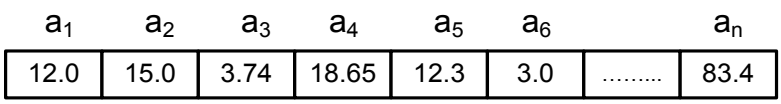


Figure 7-5. Real Number representation scheme

However, in the binary representation scheme each gene is assigned a binary value (i.e. 0 or 1). For instance, in incremental topology network optimization design, the binary code could be used to indicate if the corresponding eligible spans will be deployed in the network or not. For example, Figure 7-6 depict a 0-1 binary code, where 0 means that the corresponding eligible span will not be deployed in the network design, and 1 means that the corresponding eligible span will be deployed in the network design, and *n* represents the number of the eligible spans that are proposed to be added to the network.

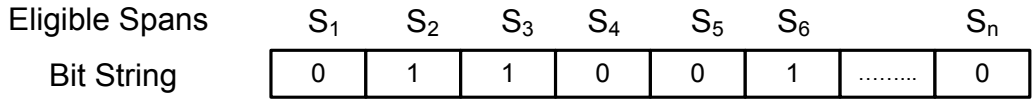


Figure 7-6. The binary representation scheme

After deciding which representation scheme should be used, the next step is to generate the initial population. Owing to the fact that the genetic algorithm is stochastic in nature [97], the uniformly distributed function could be used to produce the value assigned for each gene. Therefore, there are two approaches to feed the genetic algorithm with the initial population: first, feed it with acceptably random generated solutions as mentioned before. Second, seed it with previously known good solutions, which could be generated by another heuristic technique. On one hand, the second solution would speed up the search. On the other hand, it could trap it in the local optimal region, which could lead to a poor solution [98], [99].

7.6.3 Population size

The next step, after choosing the appropriate representation scheme and the way of populating the initial generation, is to determine the population size. As mentioned, the population is a subset of eligible solutions for the optimization problem. On the one hand, if the population size is extremely small, the premature convergences may be reached because the solution space would not be explored sufficiently. On the other hand, if it is too large, the best solution would not be reached in a reasonable amount of time [100]. Many works show that the population size should be related to the length of the chromosome (i.e. the number of genes). For instance, in [101] it suggested that the population size should be exponentially increased with the chromosome length. However, in [102]. It was proposed that it should be related linearly to the chromosome length.

Equation (7.3) has been introduced for determining the minimum acceptable population size N for binary chromosome with length l, where P_2^* represents the probability that N will provide a meaningful search.

$$N \approx |1 + \log(-l / \ln P_2^*) / \log 2| \tag{7.3}$$

The next step, after populating the first generation, is to evaluate each chromosome by using a particular objective function. For each chromosome, a fitness value is assigned depending on its objective value. The chromosome with the higher fitness value would more likely be chosen as a parent to generate the next generation of chromosomes.

7.6.4 Selection

After generating the first generation and evaluating the fitness for each chromosome individually, the selection process will start to elect the best chromosomes in terms of their fitness function and mating them to produce the offspring. This process is iterated until a predefined condition is reached and during the iterations, the fitness of the offspring will improve generation after

generation. Therefore, selection plays a pivotal role in directing the search reproduction process.

One of the main factors that affect the convergence rate is selection pressure, which is the probability of choosing the chromosomes with the higher fitness values. For instance, if the selection pressure is increased, the chance of reaching a suboptimal solution is higher, which stems from premature convergence. In contrast, low selection pressure will increase the time required to find the near optimal solution [103]. Several selection approaches have been proposed in the open literature. In the sections that follow, we will shed light on some of them.

7.6.4.1 Proportional Selection

Proportional selection is also called as roulette wheel selection. In this scheme, the circumference of the wheel is divided into parts. Their number equals the number of the chromosomes in one generation. The length of each one is proportional to the fitness value with its corresponding chromosome. Then the wheel is rotated, and a chromosome is selected randomly. The last step is iterated until the number of required parents for reproduction is reached. It is very obvious that the chromosome with the higher fitness will have a higher chance to be chosen as a parent. Furthermore, the chromosome with a very high fitness level will have a higher chance to be generated as a member of the next generation.

Equation (7.4) [104] presents the probability of choosing a chromosome x_i at generation t. N denotes the number of chromosomes per generation and $f(x_i^{(t)})$ is the fitness value for the given chromosome.

$$p\left(x_i^{(t)}\right) = \frac{f(x_i^{(t)})}{\sum_{f=1}^{N} f(x_i^{(t)})} \tag{7.4}$$

7.6.4.2 Tournament Selection

The tournament selection process starts by choosing a set of chromosomes from the current generation randomly, which is called the tour set. The chromosome with the highest fitness value in this tour set will be elected for reproduction. This process is iterated until the number of required parents is reached. The number of the chromosomes in the tour set affects the selection pressure factor for this genetic algorithm model. In that, as the size of this set increases, the probability of selecting the high fitness chromosome increases [103].

Equation (7.5) presents the probability of selecting a chromosome x_i at a generation t with a tour size q

$$p(x_i^{(t)}) = \frac{1}{N^q}\left((N-i+1)^q - (N-i)^q\right) \tag{7.5}$$

7.6.4.3 Linear Ranking Selection

In linear ranking selection, chromosomes are ranked according to their fitness value. For instance, the chromosome with the best fitness value will be assigned a rank N while the chromosome with the worst fitness value will be assigned the lowest rank [105], [104].

The next step, after deciding which parents will be selected for the reproduction, is the generating of the offspring using different kinds of operators such as the crossover and the mutation.

7.6.5 Crossover

In the realm of the genetic algorithms, there are several types of crossover methods. The usage of the crossover relies on the representation scheme. In our work, the linear crossover will be utilized.

There are three prime linear crossover approaches [106], [107]. First, the one-point crossover, where the two parents are divided into two portions at the same

point, and then the first part of the first parents is recombined with the second part of the second parent and vice-versa. Figure 7-7 illustrates this process.

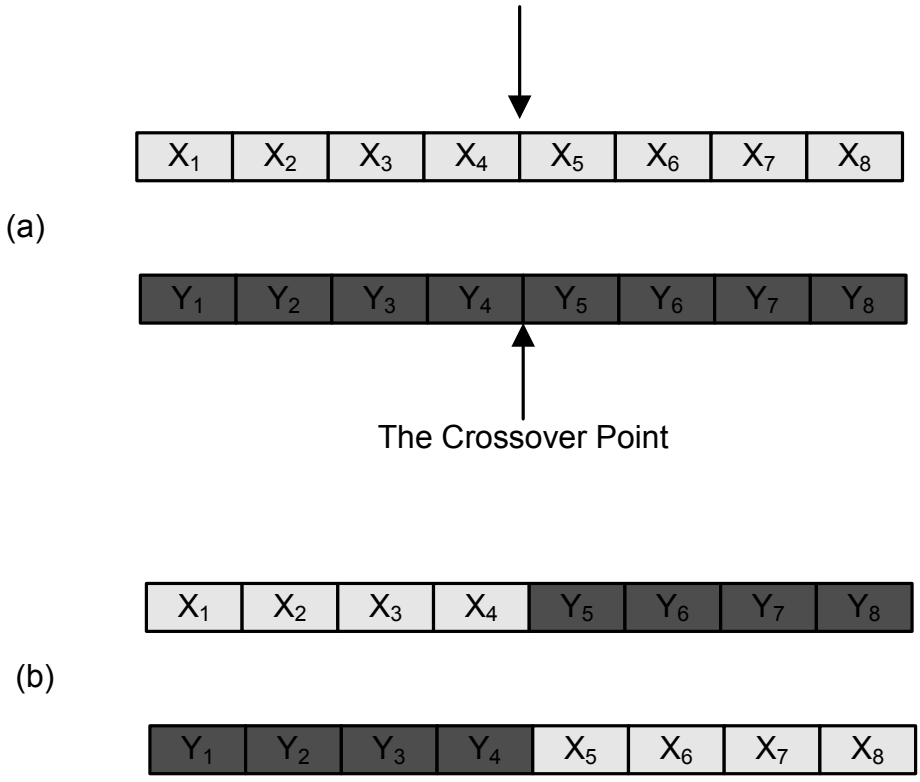


Figure 7-7. One-point Crossover (a) represents the parents, (b) the offspring

Second, the m-point crossover, more than the one crossover point, is used and a mask is required. This mask consists of a pattern of 1's and 0's which represents the parts taken from each parent. For example, Figure 7-8 depicts a 2-point crossover with a mask (1 1 1 0 0 0 1 1). The first child is generated by recombining the genes corresponding to 1's in the first parent with the genes corresponding to 0's in the second. The second child is generated vice-versa. Third, the uniform crossover, in this kind of crossover the mask is not predefined as in 2-point crossover. Every time a crossover is used, the Bernoulli function is used to generate a random pattern, which will represent the mask for the current crossover.

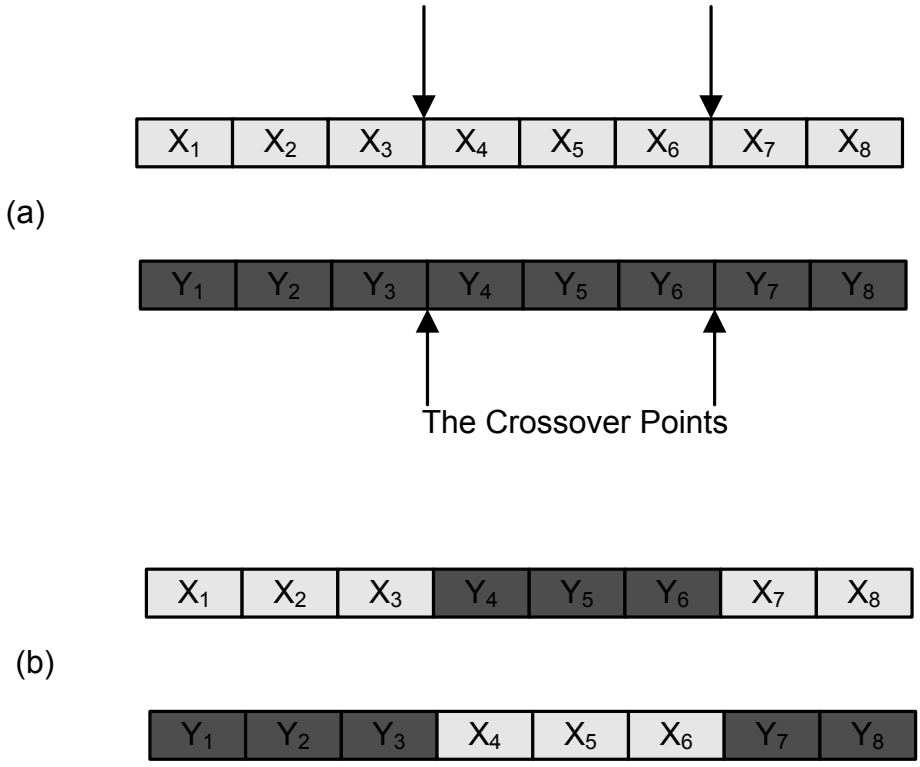


Figure 7-8. 2-point crossover, (a) the parents, (b) the offspring

7.6.6 Mutation

In addition to the crossover, mutations play a vital role in generating the offspring from the parents. The main parameter for this is the mutation rate which represents the probability of mutation for the genes. As the genetic algorithm search progresses, it will be converted to a specific search space and the crossover will become more inefficient. In contrast the mutation will be the only tool for discovering the other area to avoid the local optimal solution [107].

Once the selection, crossover, and mutation methods have been determined, the offspring is ready to be generated. The next step is to populate the next generation, which could be done by first replacing the current population by the offspring which could lead to the loss of some of good parents. Second, the next generation could be produced as a combination of the current population and its offspring. The second method outweighs the first one because it keeps the best parents from one generation to another and this is why the genetic algorithms are

called a memory usage methods. The number of offspring replacing the parents is an important factor in this iterated process [107]. Therefore, if a number of offspring is injected in the next generation, the same number of the worst parents should be deleted. The whole process is iterated until a predefined condition is reached.

7.7 Genetic Algorithm For Selection OF NEPC Weighted Efficiency Factors

The main parts of the proposed GA model in this chapter are as follows: first, the fitness function is the total spare capacity cost of the network design generated by the CIDA-Like algorithm. Second, a series of real numbers are generated to represent an individual as a chromosome. Each chromosome contains three numbers (genes): the first nmuber represents the On-cycles span factor, the second number represents the straddling factor, and the third represents the transiting flow factor. A classical crossover has been adopted as the main genetic operator. A tournament selection is applied in the selection process. Two individuals are randomly chosen from the population. Individuals having higher fitness value are chosen and inserted into the next generation. The process is iterated until the new population is obtained. The main problem is to find the factors which produce the minimum network design.

As show in Figure 7-9, the procedures of the genetic algorithm are as follows. It starts with providing GA parameters: number of individuals in each generation (PopSize), number of generation (GenSize), and crossover and mutation probabilities. PopSize is set to 10 in this work. The Initialization phase generates a random initial population. It also evaluates the fitness of each individual. On each generation, crossover and mutation mechanisms are applied to individuals, which are probabilistically selected from the population. Then, the fitness value of each new individual is calculated. The best fitness so far is also kept until the last generation. The procedure stops when the improvement in the fitness value from generation to another reaches zero.

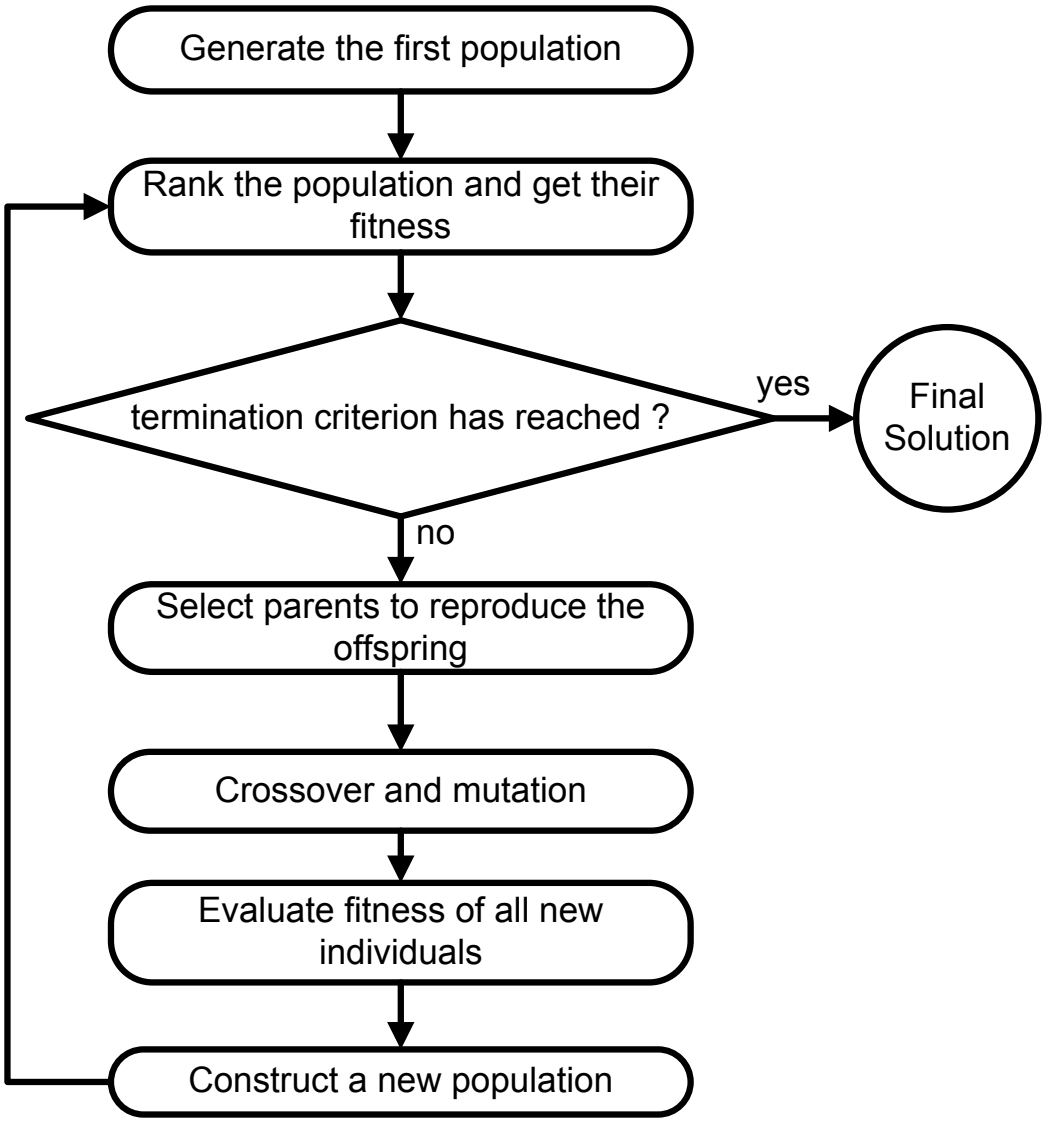


Figure 7-9. Genetic Algorithm flow chart

Table 7-2. The best factors values for CIDA-Like algorithm using the genetic algorithm model

Networks	$X_{p,i}$ (on-cycle)	$X_{p,i}$ (on-cycle)	$X_{p,n}$ (encircled)	% Improvement
10n20s	1.0	6.0	21.0	3.0%
15n30s	1.0	2.3	5.4	4.0%
20n40s	1.0	1.8	16.6	6.0%
25n50s	1.0	3.0	8.0	5.4%

The results generated from the GA model have been displayed in Table 7-2. Tests were conducted for four different networks of various sizes, ranging from 10 nodes and 20 spans ("10n20s") to 25 nodes and 50 spans ("25n50s"). The table shows that using different factors values, instead of 1, 2, and 2 for the on-Cycles, straddling, and transiting flows factors respectively, will result in a significant improvement in the spare capacity network design cost. The improvement changes from network to another and reaches a peak in the 20-node network with an improvement of 6.01%.

7.8 Conclusion

The goal of this chapter was to propose a simple algorithmic approach for the enumeration of a good set of candidate node-encircling *p*-cycles for use in NEPC network design without requiring a (nearly) exhaustive DFS enumeration of the cycle set. We proposed the NDPP and Level Partitioning algorithms, and showed that runtime is significantly shorter for both relative to the benchmark DFS approach for all of our test cases. We next adapted a prior *p*-cycle network design algorithm (CIDA) for use in NEPC network design, and showed that NDPP outperformed Level Partitioning and the benchmark DFS approach when resultant candidate cycles enumerated by each method were passed to NEPC-CIDA for overall network design. Finally, we implemented a genetic algorithm to determine

near-optimal values for parameters used within NEPC-CIDA and showed that further improvements of as much as 6% could be realized in NEPC network designs arising from NEPC-CIDA with NDPP cycle enumeration.

Chapter 8. An Enhanced ILP Design Model for Node-Encircling *p*-Cycle Networks

This chapter represents the following paper: "**An Enhanced ILP Design Model for Node-Encircling *p*-Cycle Networks**," *Design of Reliable Communication Networks (DRCN 2014)*, Gent, Belgium, 1-3 April 2014.

8.1 Introduction and Background

A number of techniques have been developed to provide network survivability. Among the simplest are survivable rings [69], [42] and 1+1 automatic protection switching (and also 1:1, 1:N) [43]. More complex approaches include span restoration [75], path restoration [61], and shared backup path protection (SBPP) [45]. The present chapter will focus on *p*-cycles [47], which have received a lot of attention in recent years, and more specifically, *node-encircling p-cycles* (NEPCs) [57], [91]. In this chapter, we develop a new enhanced ILP design model that optimally designs a node-encircling *p*-cycle network. The new model takes advantage of the observation that NEPCs assigned solely for node-failure protection will inherently protect all two-hop segments of every multi-hop working lightpath. As a result, only single-hop working lightpaths need explicit span-failure protection in the conventional manner. The new ILP model shows a significant reduction in capacity requirements.

An ILP network design model was formulated in [91]; the ILP selects an optimal combination of conventional span-protecting *p*-cycles and NEPCs so that all span failures and all node failures are fully protected. In addition to protecting node failures, NEPCs are also permitted to protect span failures in the same manner as conventional *p*-cycles (see the previous chapters for more details). In fact, the model makes no differentiation between ordinary span-protecting *p*-cycles and NEPCs. Both types are treated as *p*-cycles in general and are enumerated in a single set of eligible *p*-cycles. Just as conventional *p*-cycle protection relationships with their various on-cycle and straddling spans are encoded in a set

of $x_{i,p}$ parameters, their capabilities to protect various nodes as NEPCs are encoded in a related set of x_p^n parameters (see below for more details).

If we look closely at the makeup of the working capacity that needs protection on any individual span, we will observe that the bulk of it has arisen as a result of multi-hop working lightpaths (i.e., lightpaths whose working routes take them over at least two spans) but a small amount will be due to single-hop working lightpaths, lightpaths whose working routes cross only a single span connecting the origin to the destination. While the working capacity from single-hop working lightpaths will explicitly require protection in the conventional manner (i.e., via a *p*-cycle for which that span is an on-cycle or straddling span), working capacity from multi-hop working lightpaths will not necessarily need this explicit protection.

Consider the manner in which an NEPC protects the network from a node failure. It does so by capturing all working lightpaths transiting the protected node. This means that each two-hop segment of a multi-hop working lightpath will be protected from failure of their intermediate node by an NEPC (or perhaps several). However, failure of a node will be indistinguishable from the simultaneous failure of all spans incident on that node. From the point of view of a multi-hop working lightpath, the NEPC that protects any of its individual two-hop segments from failure of the segment's intermediate node can also protect it from failure of one or the other (or both) of the two spans of that segment. The two-hop segment will simply be routed around the NEPC whether it was the protected intermediate node that failed, or it was one (or both) of the spans of the segment. Assuming the network topology in such that all nodes have at least one NEPC capable of protecting it, then all individual two-hop segments of multi-hop working lightpaths are also inherently protected against failure of either of its spans. This leaves only working capacities arising from single-hop working lightpaths that explicitly need to be protected via conventional span-protecting *p*-cycles, rather than all working capacities.

One difficulty with this approach, however, is with regards to signaling and failure detection. In the event that two-hop segment of a working lightpath is protected against span failure by a node-encircling *p*-cycle, the failure will be visible to only one of the end nodes of that two-hop segment. Perhaps one means of overcoming that is to assume centralized control, but that may run counter to the key benefit of *p*-cycle restoration, in that *p*-cycles can act locally without an explicit need for a centralized control.

8.2 NEPC ILP Design Models

8.2.1 Benchmark ILP Design Model

Several ILP design models for NEPC network design were developed in [91]. We will use that work's Model #2 as our benchmark in the present chapter. That model utilizes a *joint capacity allocation* (JCA) approach, which simultaneously determines optimal working and restoration routing (and working and spare capacities). However, we will also consider a spare capacity allocation (SCA) approach as well, which assumes working routing is via shortest paths (or some other simple routing) and we only need to optimize restoration routing and the associated spare capacity. Rather than produce the SCA variant with a separate ILP model, we simply provide each demand with a single eligible working route, the single shortest path between the demand's end nodes. This allows both JCA and SCA designs using the JCA ILP model.

The ILP model functions as an arc-path model like that first developed in [75], where the solver is provided with a set of eligible working routes and eligible *p*-cycles, which are then optimally selected such that capacities costs are minimized. In addition, that model assumes that node-failure protection routing is evenly split in both directions around the NEPC, resulting in a slight over-provisioning of NEPCs and the resultant spare capacities (by an average of 5.8% in the test case networks in [91]). As noted therein, this benchmark model is very similar to the basic joint working routing and *p*-cycle selection network design model described

in [37], but with the addition of constraints to accommodate node-failure protection via NEPCs.

Since our enhanced model herein is based on this benchmark ILP model, we now reproduce that model here. In doing so, we use the following notations:

Sets:

S is the set of all spans in the network, typically indexed by i or j.

N is the set of all nodes in the network, typically indexed by n.

P is the set of all eligible p-cycles in the network, typically indexed by p. Note that we make no distinction here between conventional span-protecting p-cycles and NEPCs.

D is the set of all demands in the network, typically indexed by r.

Q^r is the set of all distinct eligible working routes capable of routing lightpaths for demand r, typically indexed by q.

Parameters:

d^r The parameter that represents the number of demand units for demand r.

c_j is the cost of each unit of capacity (working or spare) placed on span j. In our test cases, all c_j values were equivalent to the Euclidean distances of the spans as drawn in the network topologies.

$x_{i,p} \in \{0,1,2\}$ is an input parameter that encodes the number of protection relationships provided to span i by each unit-sized copy of eligible p-cycle p. $x_{i,p}$= 2 if span i straddles cycle p, $x_{i,p}$ = 1 if span i is on cycle p, and $x_{i,p}$= 0 in all other cases. For the special case of non-simple cycles, $x_{i,p}$= 0 for on-cycle spans that are crossed twice by the cycle.

$x_p^n \in \{0,1\}$ is an input parameter that encodes whether or not eligible p-cycle p can act as an NEPC for node n. $x_p^n = 1$ if it can and $x_p^n = 0$ if it

cannot.

$\zeta_i^{r,q} \in \{0,1\}$ is a binary parameter that defines the relationship between working routes and the network spans for each demand. It equals 1 if working route *q* used for demand *r* passes through a span *i*, otherwise it equals 0.

$\phi_r^n \in \{0,1\}$ is a binary parameter which equals 1 if node *n* is the origin or the destination of demand *r*, otherwise it equals 0.

$z_n^{r,q} \in \{0,1\}$ is a binary parameter that describes the relationship between working routes and the network nodes for each demand. $z_n^{r,q} = 1$ if working route *q* used for demand *r* crosses node *n*, otherwise $z_n^{r,q} = 0$

Decision Variables:

$g^{r,q} \geq 0$ is the integer number of working lightpaths assigned to working route *q* used for demand relation *r*.

$w_i \geq 0$ is the integer number of working capacity that is assigned to span *i* in total.

$\tau_n \geq 0$ is the integer number of transiting working flows passing through node *n*.

$n_p \geq 0$ is the integer number of unit-capacity copies of eligible *p*-cycle *p* placed in the network.

$s_j \geq 0$ is the integer number of spare capacity that is assigned to span *j*.

The benchmark NEPC ILP formulation as follows:

Minimize:

$$\sum_{\forall j \in \boldsymbol{S}} c_j \cdot \left(s_j + w_j \right) \tag{8.1}$$

Subject to:

$$\sum_{\forall q \in \boldsymbol{Q}^r} g^{r,q} = d^r \qquad \forall r \in D \tag{8.2}$$

$$\sum_{\forall r \in \boldsymbol{D}} \sum_{\forall q \in \boldsymbol{Q}^r} \zeta_i^{r,q} \cdot g^{r,q} = w_i \qquad \forall i \in S \tag{8.3}$$

$$w_i \leq \sum_{\forall p \in \boldsymbol{P}} x_{i,p} \cdot n_p \qquad \forall i \in S \tag{8.4}$$

$$\tau_n = \sum_{\substack{\forall r \in \boldsymbol{D} | \\ \phi_r^n = 0}} \sum_{\forall q \in \boldsymbol{Q}^r | Z_n^{r,q} = 1} g^{r,q} \qquad \forall n \in N \tag{8.5}$$

$$\tau_n \leq \sum_{\forall p \in \boldsymbol{P} | x_p^n = 1} 2 \cdot n_p \qquad \forall n \in N \tag{8.6}$$

$$s_j = \sum_{\forall p \in \boldsymbol{P} | \boldsymbol{x_{i,p}} = 1} n_p \qquad \forall j \in S \tag{8.7}$$

The objective function in (8.1) seeks to minimize the total cost of placing working and spare capacities in the network. Constraints (8.2) guarantee that all demands will be provided sufficient number of working lightpaths, and the constraints in equation (8.3) assign a sufficient amount of working capacities on each span *i* to accommodate all working lightpaths routed over it. Equation (8.4) assigns sufficient copies of the various eligible *p*-cycles to provide restoration of all working capacities on each span. Equations (8.5) and (8.6), respectively, determine the number of working lightpaths transiting through each node, and ensure that there are sufficient copies of the various eligible *p*-cycles (acting as NEPCs) to protect all transiting lightpaths through each node in the event of failure of that node. Note that the 2 multiplier in equation (8.6) is due to the fact that each copy of an NEPC can protect two transiting lightpaths from failure of node *n,* one in each direction around the *p*-cycle. Finally, the constraints in equation (8.7) places spare capacities on each span *j* to accommodate all copies of eligible *p*-cycles assigned to the network.

8.2.2 Enhanced JCA ILP Design Model

As discussed above in a previous section, the benchmark model explicitly provides span-failure protection for all working capacity on each span, although we need do so only for working capacity arising from single-hop working lightpaths. This means that equation (8.4) is no longer required, at least in its present form, and needs to be replaced by a new equation that asserts span-failure protection only for single-hop working capacity. We also need to designate some new decision variables to represent that specific working capacity:

$w'_i \geq 0$ is the integer number of working capacity that is assigned to span *i* arising only from single-hop working lightpaths.

We then replace equation (8.4) with a new set of constraints in equation (8.8), which is identical to the original except that we provide sufficient span-failure protection for w'_i rather than for w_i.

$$w'_i \leq \sum_{\forall p \in \boldsymbol{P}} x_{i,p} \cdot n_p \qquad \forall i \in S \tag{8.8}$$

And of course we need to introduce a new set of constraints, equation (8.9) to calculate the various w'_i values. Note that this equation is nearly identical to equation (8.3), except that we consider only those working lightpaths that follow single-hop routes. This is done by qualifying the *q* indices in the second summation such that the sum of their individual $\zeta_i^{r,q}$ values is equal to one (i.e., they only cross a single span). To reiterate, this is a new set of constraints. Equation (8.9) does not replace equation (8.3), rather it is in addition to equation (8.3), which is still needed in order to calculate the total working capacity on each span.

$$\sum_{\forall r \in \boldsymbol{D}} \sum_{\substack{\forall q \in \boldsymbol{Q}^r \\ \sum_{\forall j \in S} \zeta_j^{r,q} = 1}} \zeta_i^{r,q} \cdot g^{r,q} = w'_i \qquad \forall i \in S \tag{8.9}$$

An alternative (and perhaps simpler) expression for this constraint is to calculate the w_i' values by simply identifying the demand *r* with the same end nodes as span *i* and letting w_i' equal the number of lightpaths for that demand (or the sum of the lightpaths if there are multiple demands with the same end nodes). This will simply require enumeration of spans and demands by their end nodes, which is readily available data (in fact, we use that very data to enumerate eligible working routes, etc.). However, we would also need to introduce that additional data into the ILP model, so we elected to go with the somewhat more complex calculation of w_i' values as shown in equation (8.9) (and besides, it makes this constraint nearly identical to equation (8.3), as noted above). Furthermore, that simplified representation of this constraint makes the assumption that each demand whose end nodes are only a single hop apart will actually be routed on that single hop. In practice, that may be the case in most scenarios, however, it is possible that some such demand may elect to route its working lightpath around a longer route if it is able to take advantage of protection relationships with other cycles.

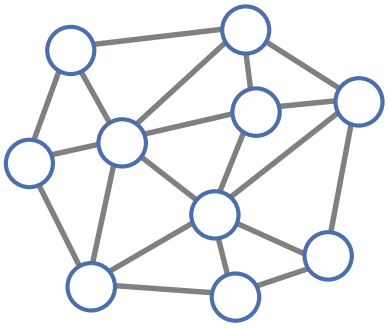
8.3 Experimental Study

We carried out our experiments on three networks, with 10 nodes and 20 spans, 15 nodes and 30 spans, and 20 nodes and 40 spans, respectively, from [37], and shown in Figure 8-1. In all three networks, each node pair exchanged lightpath demands, where the number of such lightpaths for each node pair was drawn from a uniform random number between 1 and 10, inclusive.

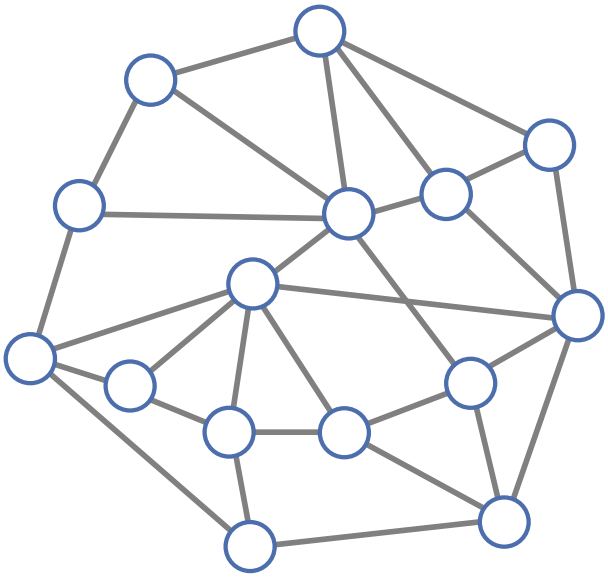
Our ILP models (the benchmark and our enhanced version) were modeled in the AMPL modeling language and solved with the CPLEX solver on an ACPI multiprocessor X64-based PC with an 8-processor Intel Xeon CPU X5460 3.16 GHz server with 32 GB on RAM. All the models were run with the default mipgap setting of 0.0001, which means all results are guaranteed to be within 0.01% of optimal. Pre-processing to enumerate eligible *p*-cycles and eligible working routes, and their associated $x_{i,p}$, x_p^n, $\zeta_i^{r,q}$, ϕ_r^n, and $z_n^{r,q}$ parameters was completed with custom software written in C. The five shortest eligible working

routes (by span length) were enumerated for each demand, and the shortest cycles that can be drawn in the network were used as eligible *p*-cycles (1000 for the 10-node network, 5000 for the 15-node network, and 10000 for the 20-node network), which was supplemented with the shortest cycles possible, such that each node has 10 eligible *p*-cycles that can act as NEPCs. Solution runtimes completed in several minutes. (The longest runtime was 343 seconds for the 20-node network's benchmark NEPC solution). Interestingly, solution runtimes were almost an order of magnitude shorter for the ENEPC despite having a greater number of integer variables.

10-node network



15-node network



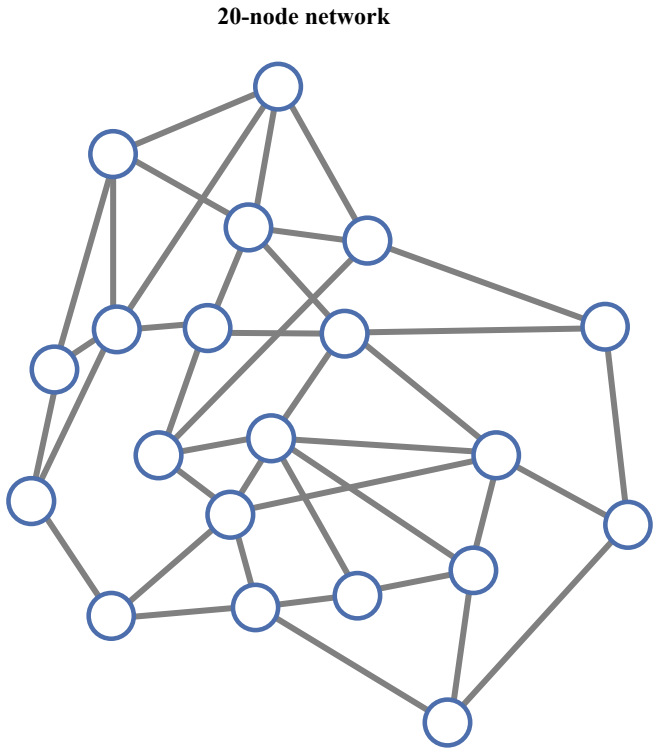


Figure 8-1. 10-node, 15-node, and 25-node test networks used herein

8.4 Results and Discussion

Results of our experiments are shown in Figure 8-2 and Figure 8-4. Each bar represents the optimal capacity cost of the indicated network using the indicated ILP model. Capacity costs were normalized to the benchmark solutions for all test cases. The reason for the normalization was to present the data in a manner that was independent of the scale of the network. For instance, the 10-node network we used happened to have an average span length of approximately 111 (units defined), while the 15-node network had an average span length of approximately 185 and the 20-node network had an average span length of approximately 131. These differences in the average span length effectively result in arbitrary differences in scale between the various networks. What is of importance here is the relative differences between the solutions arising from the benchmark NEPC model and the new ENEPC model, not that the NEPC solution had an objective function value of 75633 and the ENEPC solution had an objective function value of 71313 for the JCA models in the 10-node network, for instance. The data in

Figure 8-2 represents the spare capacity allocation (SCA) variant of both models, where we provided the ILPs with only a single eligible working route (the shortest one). For this variant of the models, the new enhanced ILP model (ENEPC) provides an average of 7.8% reduction in total capacity costs. The 10-node network experienced a 9.3% reduction in capacity, the 15-node network experienced a 9.7% reduction, and the 20-node network experienced a 4.4% reduction. Spare capacity reductions for the SCA variant (shown in Figure 8-3) were actually an average of 12.8% lower in the ENEPC solutions versus the benchmark NEPC solutions (17.3%, 14.2%, and 6.8% lower for the three networks), respectively.

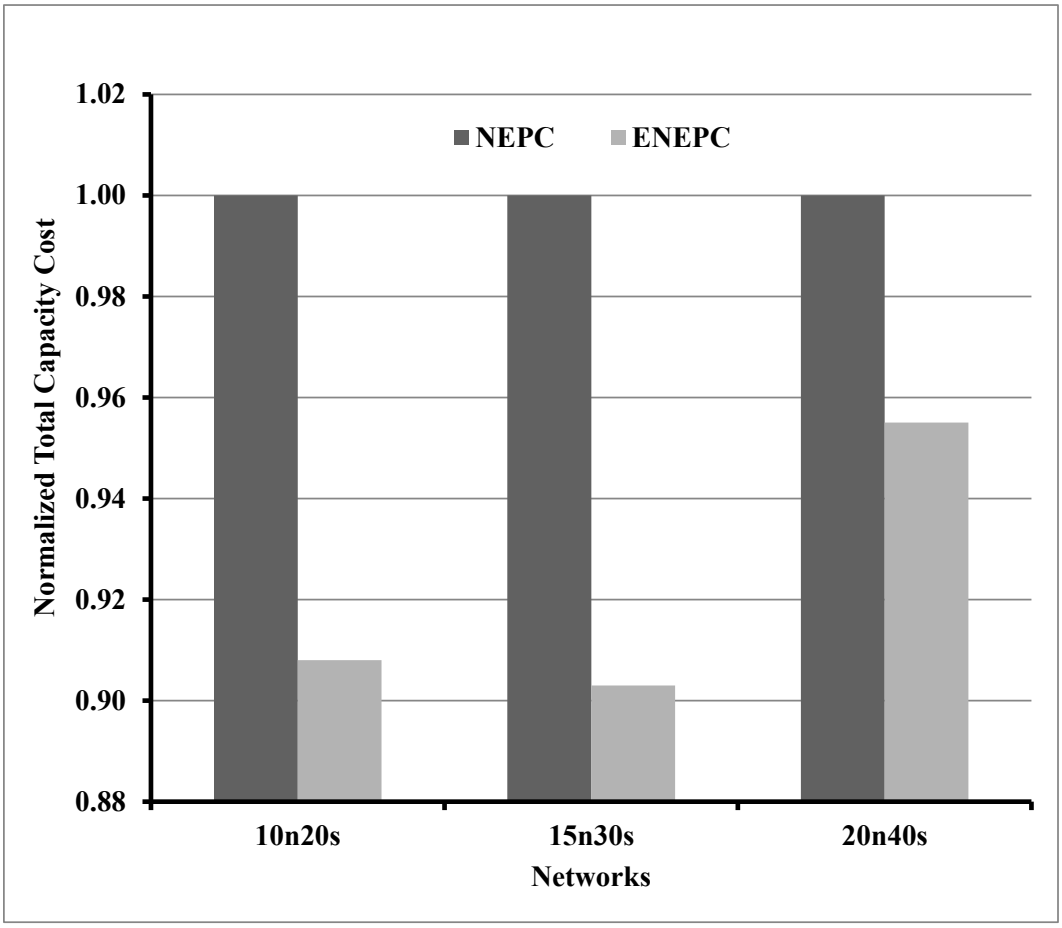


Figure 8-2. Normalized total capacity requirements of the benchmark NEPC and new ENEPC SCA design models

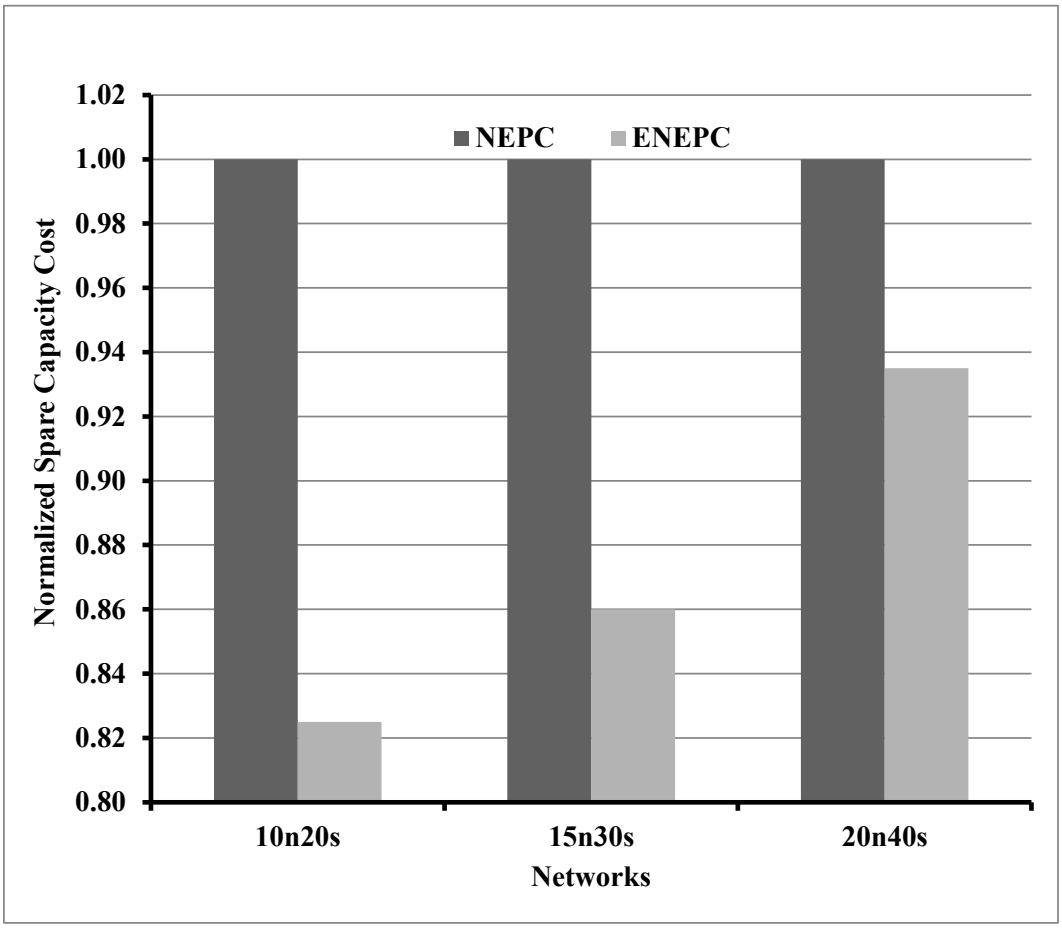


Figure 8-3. Normalized spare capacity requirements of the benchmark NEPC and new ENEPC SCA design models

Using the JCA variants (i.e., as formulated in Section 8.2 and using five eligible working routes as described in a previous section), we saw smaller capacity reductions, but we feel they are still significant enough to report. We can observe in Figure 8-4 that the new ENEPC model provides an average of 2.3% reduction in total capacity relative to the benchmark NEPC model. The 10-node network experienced a 5.7% reduction in total capacity, the 15-node network experienced a 0.44% reduction, and the 20-node network experienced a 0.63% reduction.

While the results for the 15-node and 20-node network are particularly uninspiring, a deeper examination of how the working routing is performed in the JCA variants suggests that the working lightpaths are efficient at finding circuitous routes that alleviate much of the need for additional span-protecting *p*-

cycles. In other words, although there is still a substantial difference between the w_i values (the full working capacity needing span failure protection in the NEPC designs) and the much lower w_i' values (the single-hop working capacity needing span failure protection in the ENEPC designs) in these solutions, the JCA mechanism is able to overcome this by routing working lightpaths in such a way that the w_i values in the NEPC solutions are more easily protected by the node-encircling *p*-cycles needed for node failure protection.

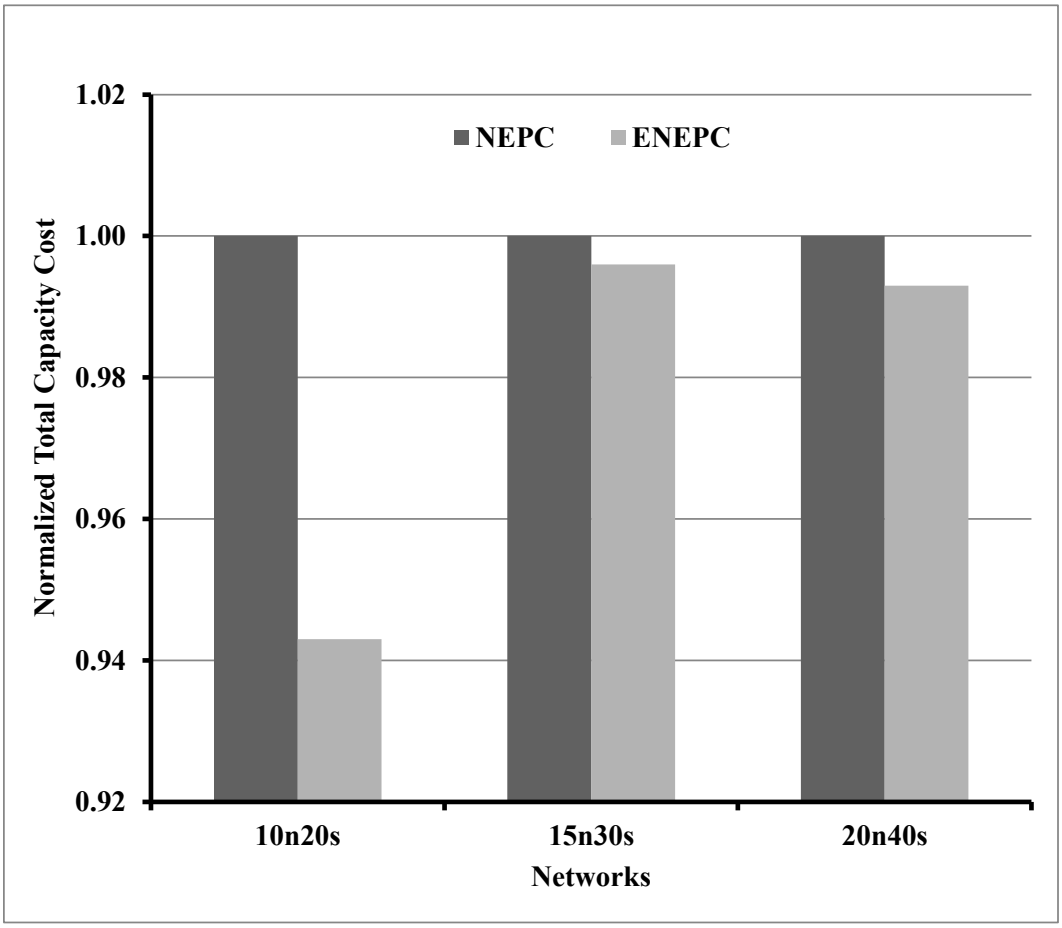
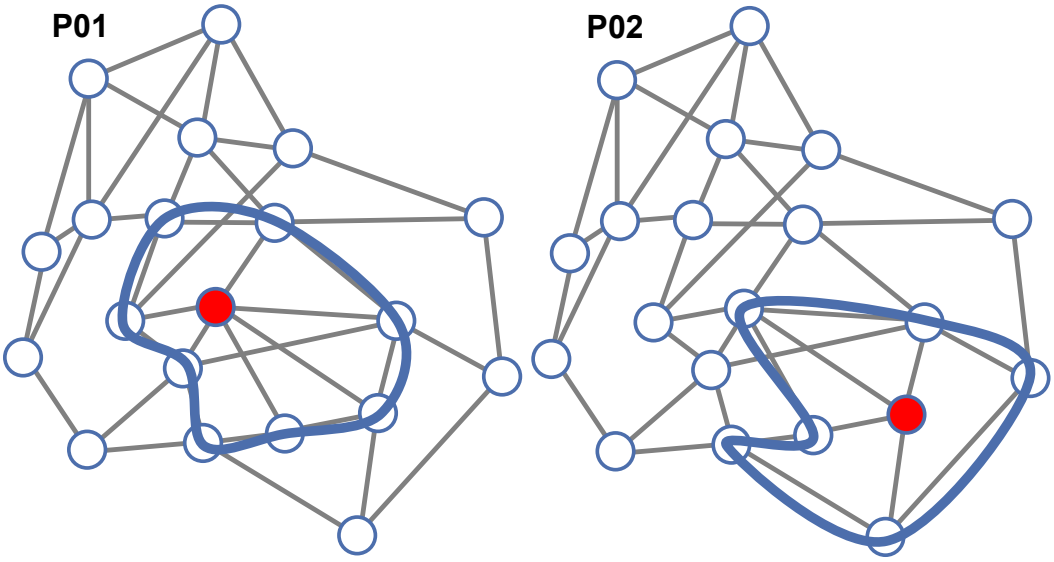


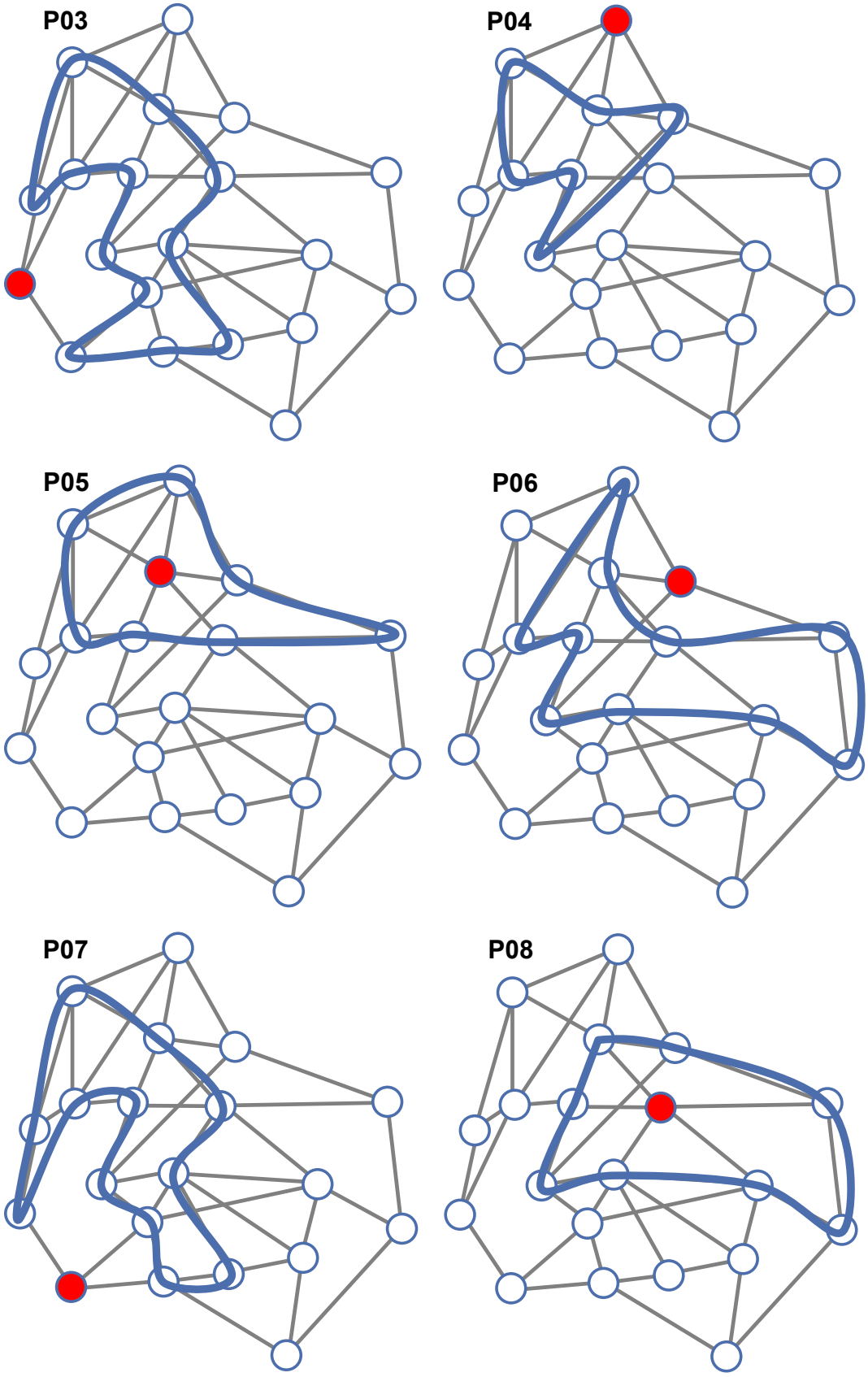
Figure 8-4. Normalized total capacity requirements of the benchmark NEPC and new ENEPC JCA design models

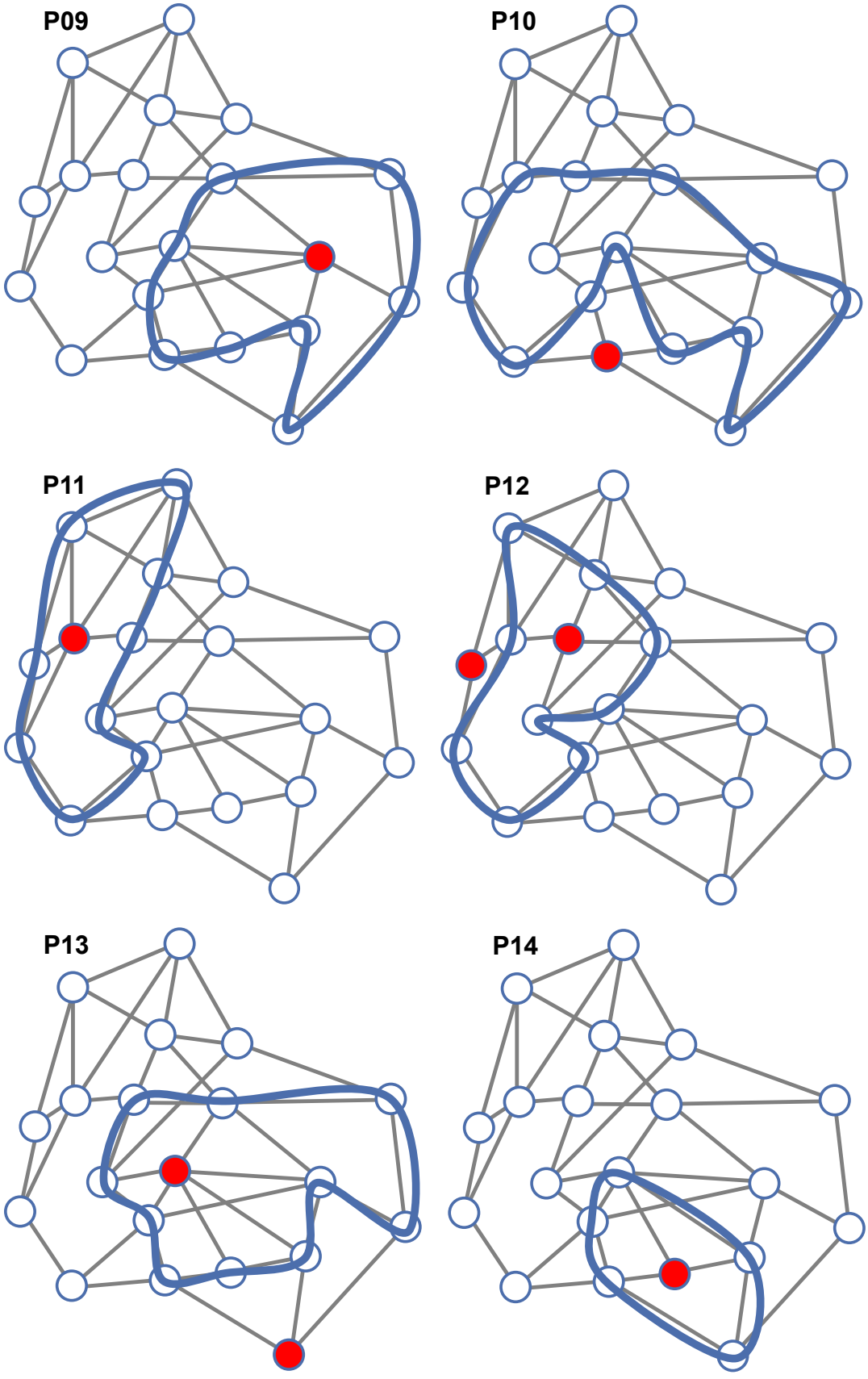
8.4.1 ENEPC Cycle Details

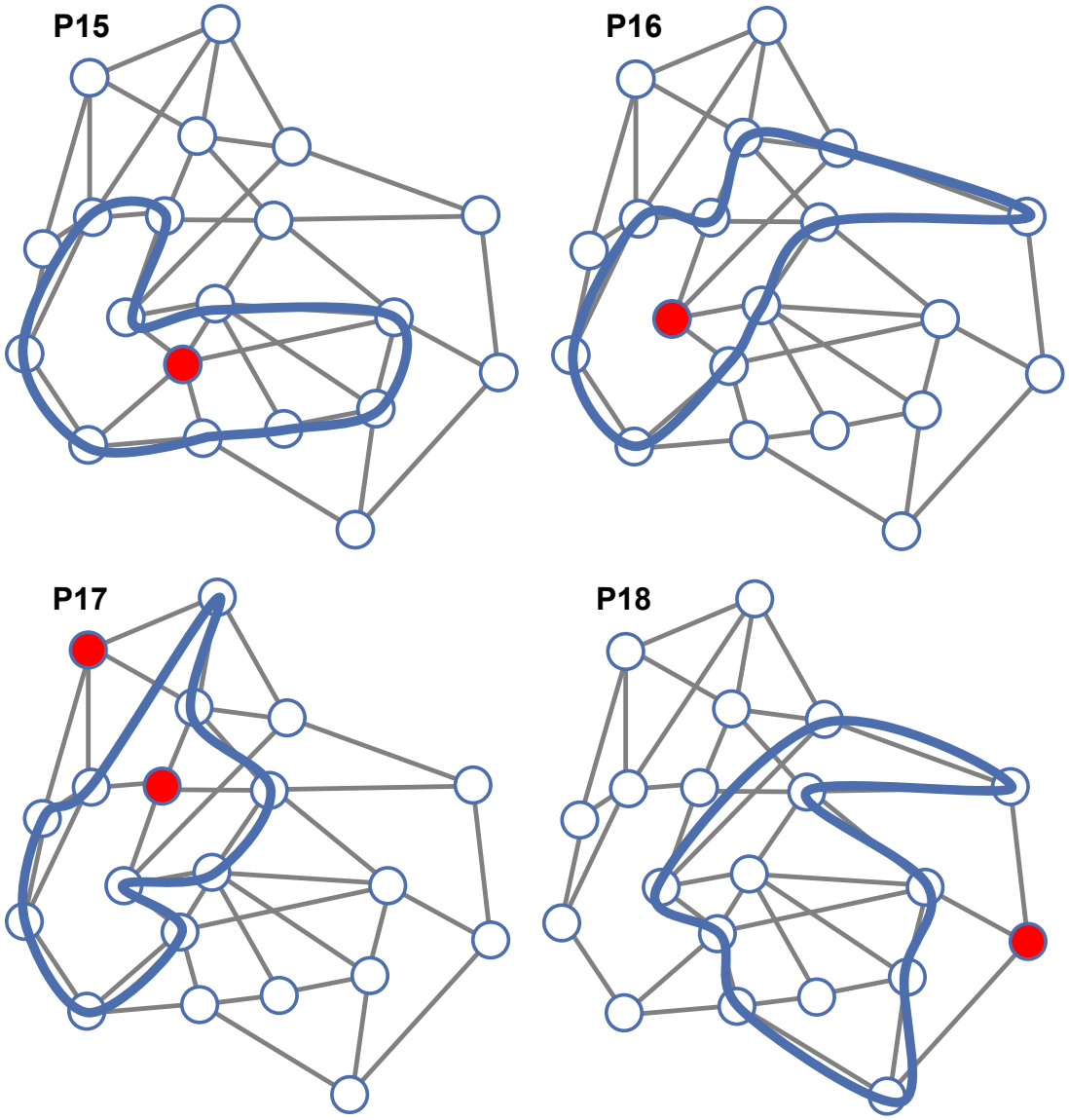
A closer look at the detailed solution provides additional insights. Figure 8-5 illustrates the 19 unique *p*-cycles those were selected for the 20-node network via

either the NEPC model or the ENEPC model (or both), while Table 8-1 indicates the number of copies of each of them, as selected by the two models. Each of The benchmark NEPC and ENEPC solutions consists of 690 total copies of 19 unique *p*-cycles. It is interesting to note that the unique *p*-cycles selected by both of the solutions where all of them NEPCs, which supports our thoughts on how the both of the models will function; selection of lengthy node-encircling *p*-cycles will predominate, as they will capture a greater number of single-hop routed working capacity. This will occur only when you provide the two models with enough NEPCs, on 10-node network we will shed the light on how our enhanced model outperforms the benchmark model when you do not have a complete set of NEPCs that can inherently protect span working capacities.




P03
P04
P05
P06
P07
P08


P09
P10
P11
P12
P13
P14


P15
P16
P17
P18

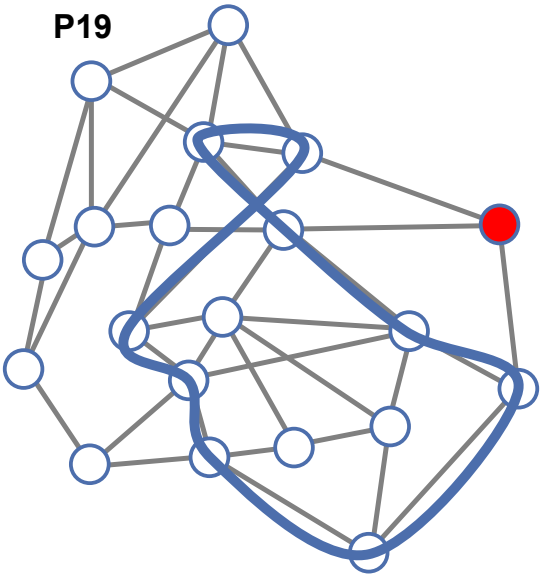
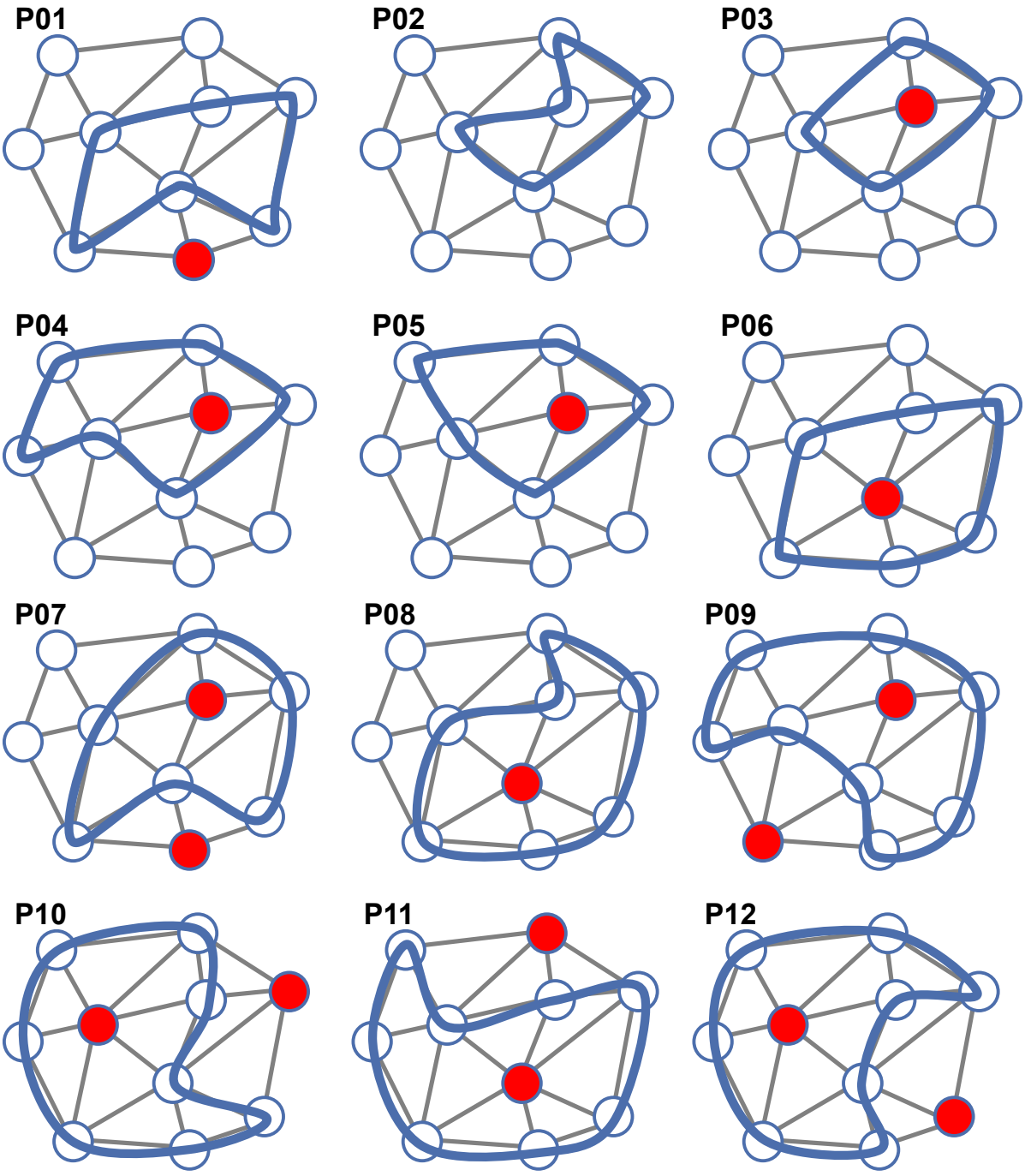


Figure 8-5. All unique cycles used by the benchmark NEPC and new ENPEC models for the 20-node network

Table 8-1. Usage of each *p*-cycle in the NEPC and ENPEC solutions for the 20-node network

***p*-Cycle**	***p*-Cycle Size**	**Copies in NEPC**	**Copies in ENEPC**
P01	8	107	103
P02	6	32	32
P03	11	33	28
P04	6	1	1
P05	7	74	75
P06	10	13	12
P07	12	33	28
P08	8	152	158
P09	9	31	30
P10	11	23	23
P11	9	26	31
P12	9	33	38
P13	10	18	24
P14	5	14	14
P15	9	33	27
P16	10	25	24
P17	10	23	23
P18	9	9	9
P19	9	10	10

Figure 8-6 shows the 14 unique *p*-cycles that were selected for the 10-node network via either the NEPC model or the ENEPC model (or both), while Table 8-2 indicates the number of copies of each of them, as selected by the two models. The benchmark NEPC solution consists of 65 total copies of 11 unique *p*-cycles, while the ENEPC solution consists of 57 total copies of eight unique *p*-cycles. It is interesting to note that the unique *p*-cycles selected by the ENEPC model were predominantly the larger cycles (sizes are simply denoted by the number of hops). Half of the smaller cycles were not selected at all in the ENEPC solution, which supports our thoughts on how the ENEPC model will function; selection of lengthy node-encircling *p*-cycles will predominate, as they will capture a greater number of single-hop routed working capacity. Not surprisingly, the only non-node-encircling *p*-cycle selected by the NEPC model was not selected by the ENEPC model.


P01
P02
P03
P04
P05
P06
P07
P08
P09
P10
P11
P12

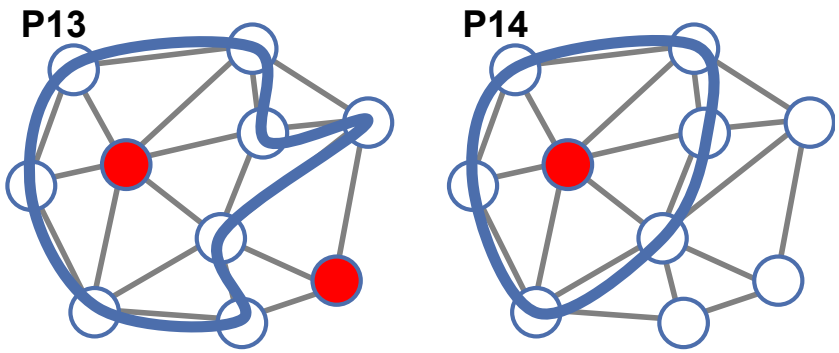


Figure 8-6. All unique cycles used by the benchmark NEPC and new ENPEC models for the 10-node network

We can also take a closer look at the w_i values relative to the total working capacities w_i on each span. On average in the 10-node network, each span has 18.2 units of working capacity (the w_i values) but only 6.15 units of working capacity that needs protection against span failures (the w_i values). The balance (just over 12 units per span on average) will need protection in the NEPC model but not in the ENEPC model.

Table 8-2. Usage of each *p*-cycle in the NEPC and ENPEC solutions for the 10-node network

***p*-Cycle**	***p*-Cycle Size**	**Copies in NEPC**	**Copies in ENEPC**
P01	6	3	0
P02	5	4	0
P03	4	4	9
P04	6	4	0
P05	5	1	0
P06	6	19	21
P07	6	0	3
P08	7	2	0
P09	8	6	3
P10	8	5	4
P11	8	4	4
P12	8	0	2
P13	8	13	0
P14	6	0	11

8.5 Conclusions

The benchmark ILP design model for node-encircling *p*-cycle networks explicitly provide span-failure protection for all working capacity that arises in the network. However, as we have shown, providing node-failure protection of all working lightpaths transiting through each node inherently also protects all working capacity arising from multi-hop working lightpaths. As such, only working capacity arising from single-hop working lightpaths need to be explicitly provided with span-failure protection.

We have developed an enhanced version of the ILP design model for node-encircling *p*-cycle networks that does not place excess spare capacity unless explicitly needed for span-failure protection. The resulting optimally designed networks are more capacity efficient than those resulting from the benchmark ILP model, while remaining 100% restorable in the event of any single node or span failure. Capacity requirements were reduced an average of 7.8% in our three test-case networks using the SCA variants of the new ENEPC and benchmark NEPC models, and an average of 2.3% using the JCA variants.

The poorer performance in the JCA models is certainly a weakness in the new ENEPC model. However, in scenarios where working lightpaths are already routed or they need to be routed via specific paths without regard for subsequent *p*-cycle requirements, the significant reductions in capacity requirements in the SCA models would still be realized. For instance, in the event that a carrier wishes to implement span-protecting *p*-cycle and node-encircling *p*-cycle restoration in an existing network, then the SCA variant of the ENEPC would apply, not the JCA variant.

Chapter 9. Conclusions and Recommendations for Future Work

9.1 Conclusions

The influence of backbone telecommunication networks failure is rising significantly as more and more aspects of modern life rely on them, even for elementary services. Companies looking to consolidate their data centers using cloud computing technologies, healthcare providers trying to increase their efficiency by moving to electronic health records, and even retail stores with electronic payment methods. All of these services entail a communication system with high reliability, where even a few minutes of failure can result in a major monetary and social impact. As shown in Figure 9-1, this thesis has provided four main research studies on telecommunication network survivability schemes. The first two studies are related to the incremental topology optimization problem, the first project used the meta-mesh span restoration scheme and the second one utilized the *p*-cycle scheme. The last two studies are related to the NEPC network design, the first is concerned with algorithms for enumerating NEPC's and efficiently solving this kind of design and the second is interested in improving the NEPC conventional ILP model.

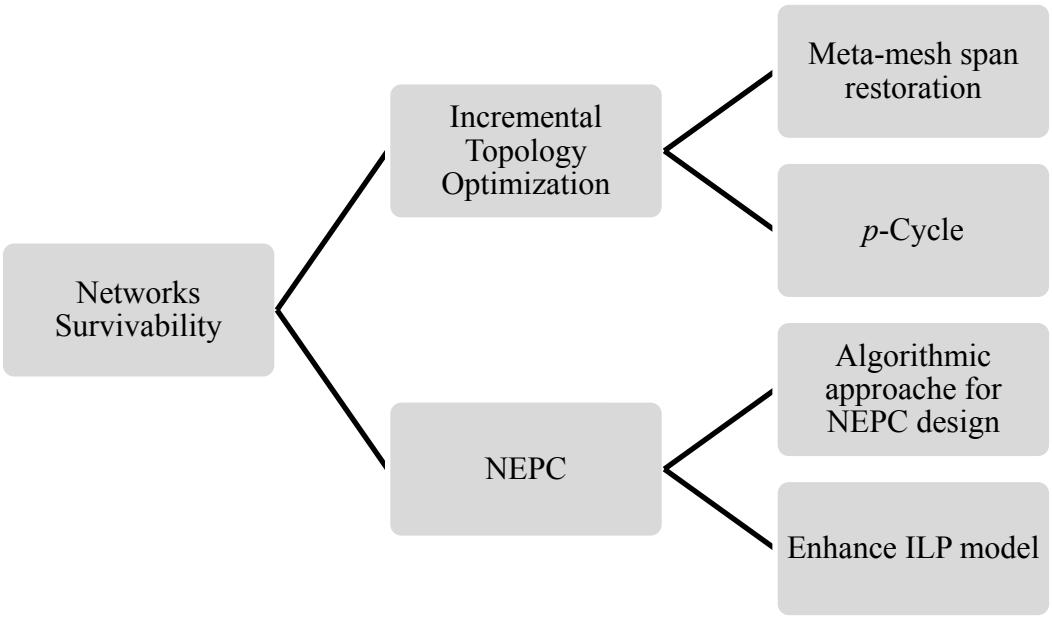


Figure 9-1. Thesis projects summary

The main outcomes of the *Incremental Network Topology Optimization Using Meta-Mesh Span Restoration* are:

- The conventional meta-mesh model uses an arc-path approach. We developed node-arc (i.e., transshipment) ILP model for meta-mesh restoration, so we can more easily extend it to accommodate changes in the network topology.
- We implement incremental topology optimization ILP model (i.e., optimal span additions) by adding constraints to control the structure of meta-mesh chain bypass spans when adding new physical spans.
- Results show that even where topology is flexible, thereby allowing a span-restorable network to use a higher-connectivity topology, meta-mesh restoration can outperform span restoration in terms of capacity and number of spans required.

The main results of *Incremental Network Topology Optimization Using p-Cycle Technique* are:

- Develop a new ILP model for incremental topology optimization using a *p*-cycle network design.
- Develop a relaxation-based decomposition heuristic that significantly reduces runtime of the ILP in large networks.
- While the ILP model proves to be relatively easy to solve for small test case network instances, it is computationally complex to solve for larger networks.
- A relaxation-based decomposition heuristic can significantly reduce runtime of the ILP model in our large test networks, while having minimal impact on optimality. In the most computationally complex instance, the ILP runtime of over 184,000 seconds (more than two days) was reduced to less than 2,300 seconds (less than an hour), while the objective function value remained within the optimality gap. In fact, the heuristic solution was slightly better than the full ILP (though again, we note that it was not provably better since the difference was smaller than the optimality gap).

The main results of *Efficient Algorithms for Node-Encircling p-Cycle Network Design* Project are:

- Typical NEPC network design could be done on two steps, (1), enumerate a potential set of cycles, (2) select the least cost combination of cycles that will fully protect the network working capacities. The number of cycles in a network grows exponentially with the size of the network while using a depth-first search algorithm. To solve this complexity problem, we developed a new algorithm to enumerate NEPC *p*-cycles called Node-Disjoint Path Partitioning algorithm (NDPP)
- We developed a new NEPC network design algorithm, called NEPC Capacitated Iterative Design Algorithm (NEPC-CIDA).

- The NDPP outperforms the full-network and level partitioning method in terms of the time complexity and the spare capacity cost in all of our network test cases.
- NDPP surpasses the LCMA and NCMA methods regarding the spare capacities cost in our experiments.
- GA shows that to get better results in our test cases, the "transiting flow factor" should be around double the "straddling factor".

The main findings of *An Enhanced ILP Design Model for Node-Encircling p-Cycle Networks* is:

- We have developed a new enhanced ILP design model that optimally designs a node-encircling *p*-cycle network. The new model takes advantage of the observation that NEPCs assigned solely for node-failure protection will inherently protect all two-hop segments of every multi-hop working lightpath. As a result, only single-hop working lightpaths need explicit span-failure protection in the conventional manner.
- The new ILP model shows a significant reduction in capacity requirements. The resulting optimally designed networks are more capacity efficient than those resulting from the benchmark ILP. Capacity costs were reduced an average of 14% in our three test-case networks.

9.2 Recommendations for Future Work

In this section, we will propose some future directions for the current work.

In all of our ILP models in this thesis, we have calculated the cost function as a function of single span failure. All of the spans in this work were treated equally, but in the real networks, each span can have its own risk value that defines the importance of this span. One possibility for future work is to classify the spans into different categories from the risk point of view, (*e.g.*, high, intermediate, and low risk level). Each one of these groups can be weighted in the cost function differently, in that, the high risk group gets the highest weight, and the

intermediate risk group gets lower weight and the low risk group gets the lowest weight.

In *Incremental Network Topology Optimization Using p-Cycle Technique*, we can investigate the use of *Column Generation* technique (CG) to solve this problem instead of a relaxation-based decomposition heuristic. CG could be a suitable technique for this problem because the number of zero-decision variables is much more than the basic variables in all of the experiments that we have conducted.

9.3 Contribution of Thesis Research

These four studies are presented in several publications as follows:

Refereed Journal Publications (Available in Archived Literature):

- Md. Noor-E-Allam, A. Kasem, J. Doucette "*ILP Model and Relaxation-Based Decomposition Approach for Incremental Topology Optimization in p-Cycle Networks*," Journal of Computer Networks and Communications, Vol. 2012, pp. 1-10, 2012.

Refereed Peer Reviewed Conference Publications (Available in Archived Literature):

- A. Kasem, J. Doucette, "*Algorithmic Approaches for Efficient Enumeration of Candidate Node-Encircling p-Cycles*", (INFORMS Telecommunication) Conference, Montreal, Quebec, 5 – 7 May 2010.
- A. Kasem, J. Doucette, "*Incremental Optical Network Topology Optimization Using Meta-Mesh Span Restoration*", Design of Reliable Communication Networks (DRCN 2011), Krakow, Poland, 10-12 October 2011.
- A. Kasem, R. Gallardo, J. Doucette, "*An Enhanced ILP Design Model for Node-Encircling p-Cycle Networks*", Design of Reliable Communication Networks (DRCN 2014), Ghent, Belgium, 1-3 April 2014.

Other Publications and Presentations:

- A. Kasem, J. Doucette, "*Node-Encircling p-Cycle Enumeration and Algorithmic Network Design*", MITACS/CORS Annual Conference, Edmonton, AB, 25-28 May 2010.
- R. Gallardo, A. Kasem, J. Doucette, "*Node-Encircling p-Cycle Design with Enhanced Span-Failure Protection*", MITACS/CORS Annual Conference Edmonton, AB, 25-28 May 2010.

References

[1] B. Xiang *et al.*, "A traffic grooming algorithm based on shared protection in WDM mesh networks," *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies* (*PDCAT'2003),* China*, 2003*.

[2] Capital Ethopia, "FRIDAY Telecom network failure felt acutely," 2013, [Online]. Available: http://www.capitalethiopia.com/index.php?option=com_content&view=article&id=2503:friday-telecom-network-failure-felt-acutely&catid=35:capital&Itemid=27 "Retrieved: February 8, 2015".

[3] Bloomberg, "France Telecom plans compensation after network failure," 2012, [Online]. Available: http://www.bloomberg.com/news/2012-07-07/france-telecom-looks-to-compensate-clients-after-network-failure.html "Retrieved: February 8, 2015".

[4] CBC News, "Backup systems failed in St. John's phone outage," 2006, [Online]. Available: http://www.cbc.ca/news/canada/backup-systems-failed-in-st-john-s-phone-outage-1.572471 "Retrieved: February 8, 2015".

[5] CNN, "Third undersea Internet cable cut in Mideast," 2008, [Online]. Available: http://www.cnn.com/2008/WORLD/meast/02/01/internet.outage/index.html "Retrieved: February 8, 2015".

[6] Data Center Knowledge, "Cable cut cited in Silicon Valley outage," 2009, [Online]. Available: http://www.datacenterknowledge.com/archives/2009/04/09/cable-cut-cited-in-

silicon-valley-outage/ “Retrieved: February 8, 2015”.

[7] The Seattle Times, “Vandals cut fiber optic cables to Selah,” 2010, [Online]. Available: http://seattletimes.com/html/localnews/2010903931_apwafiberopticvandals.html “Retrieved: February 8, 2015”.

[8] B. Todd, “The use of demand-wise shared protection in creating topology optimized high availability networks,” M.S. thesis, University of Alberta, Edmonton, Canada, 2009.

[9] Continuity Forum, “Manchester telecommunications failure report,” 2004, [Online]. Available: http://www.continuityforum.org/content/news/2005/11/manchester-telecommunications-failure-report “Retrieved: February 8, 2015”.

[10] EVOLVEN, “Downtime, outages, and failures – understading their true cost,” 2013, [Online]. Available: http://www.evolven.com/blog/downtime-outages-and-failures-understanding-their-true-costs.html “Retrieved: February 8, 2015”.

[11] W. Grover, *Mesh-Based Survivable Networks: Option and Strategies for Optical, MPLS, SONET, and ATM Networking*, Upper Saddle River, NJ: Prentice Hall, 2003.

[12] D. Crawford, “Fiber optic cable dig-ups, network reliability: a report to the nation,” *A Compendium of Technical Papers, National Engineering Consortium*, Chicago, IL, 1993.

[13] J. Doucette, “Network restoration and high-availability network design,” *Presentation at City of Edmonton IT Conference*, Edmonton, Canada, 2008.

[14] J. Manchester *et al.*, “The evolution of transport network survivability,” *IEEE Communications Magazine,* vol. 37, no. 8, pp. 44-51, 1999.

[15] D. Marinescu, *Cloud Computing: Theory and Practice*, MA: Morgan

Kaufmann, 2013.

[16] Group Branham Inc., *eHealth in Canada: Current Trends and Future Challenges*, Canada: Information and Communications Technology Council, 2009.

[17] K. Oestreich, "Converged infrastructure," 2010, [Online]. Available: http://www.cioandleader.com/cioleaders/features/7505/converged-infrastructure "Retrieved: February 8, 2015".

[18] B. Mukherjee, *Optical WDM Networks*, New York: Springer Science+Business Media Inc., 2006.

[19] O. Gerstel, "Optical networking: a practical perspective," *Tutorial at IEEE Hot Interconnects*, 2000.

[20] C. Lam, *Passive Optical Networks: Principles and Practices*. CA: Elsevier Inc., 2007.

[21] Cisco Networking Academy, C*CNA Exploration: Accessing the WAN*. Indianapolis, IN: Cisco Press, 2009.

[22] F. Shepherd and A. Vetta, "Lighting fibers in a dark network," *IEEE Journal on Selected Areas in Communications,* vol. 22, no. 9, pp. 1583-1588, 2004.

[23] H. Zheng, *Optical WDM Networks: Concepts and Design Principles*. Hoboken, NJ: Wiley, 2004.

[24] ITU Recommendation G.694.1, *Spectral Grids for WDM Applications,* 2002, [Online]. Available: http://www.itu.int/rec/T-REC-G.694.1-201202-I/en "Retrieved: February 8, 2015".

[25] L. Zeyu, "On Routing and wavelength assignment in WDM optical networks," Ph.D. dissertation, North Carolina State University, Raleigh, 2012.

[26] H. Zang *et al*, "Review of routing and wavelength assignment approches for

wavelength routed optical WDM networks," *Optical Networks Magazine,* vol. 1, no. 1, pp. 47-60, 2000.

[27] A. Bondy, *Graph Theory*. New York: Springer, 2010.

[28] C. Pinter, *Set Theory*. Mineola, NY: Dover Publications, 2013.

[29] H. Cormen and E. Leiserson, *Introduction to Algorithms*, 2nd ed. New York: MIT Press and McGraw–Hill, 2001.

[30] R. Ahuja *et al.*, *Network Flows: Theory, Algorithms, and Applications,* Upper Saddle River, NJ: Prentice Hall, 1993.

[31] D. Johnson, "Find all elementary circuits of a directed graph," *SIAM Journal on Computing,* vol. 4, pp. 77-84, 1975.

[32] D. Morely, "Analysis and Design of Ring-Based Transport Networks," Ph.D. dissertation, University of Alberta, Edmonton, Canada 2001.

[33] P. Matie and N. Deo, "On algorithms for enumerating all circuits of a graph," *SIAM Journal on Computing,* vol. 5, no. 1, pp. 90-99, 1976.

[34] E. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik,* vol. 1, no. 1, pp. 269–271, 1959.

[35] R. Bhandari, *Survivable Networks: Algorithms for Diverse Routing,* New York: Kluwer Academic Publishers, 1999.

[36] L. Ford and D. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathmatics,* pp. 399–404, 1956.

[37] J. Doucette, "Advances on design and Analysis of mesh-restorable networks," Ph.D. dissertation, University of Alberta, Edmonton, Canada, 2004.

[38] C. Ma *et al.,* "Pre-configured ball (p-ball) protection method with minimum back up links for dual-linkfailure in optical mesh networks," *IEEE Communications Letters*, no. 99, 2015.

[39] C. Ma *et al.,* "Pre-configured polyhedron (p-poly) with optimal protection efficiency for dual-link failure in optical mesh networks," *Reliable Network Design and Modeling (RNDM),*Barcelona, Span, 2014.

[40] CISCO White Paper, "Visual cisco networking index: forecast and methodology," 2014.

[41] W. Grover and D. Stamatelakis, "Cycle-oriented distributed pre-configuration: ring-like speed with mesh-like capacity for self-planning network restoration," *IEEE International Conference on Communications (ICC 1998)*, Atlanta, GA, 1998.

[42] R. Ramaswami and K. Sivarajan, *Optical Networks: A Practical Perspective*, 2nd ed., San Francisco, CA: Morgan Kaufmann Publishers, 2002.

[43] R. Bhandari, *Survivable Networks: Algorithms for Diverse Routing*. New York: Kluwer Academic Publishers, 1999.

[44] R. Iraschko and W. Grover, "A highly efficient path-restoration protocol for management of optical network transport integrity," *IEEE Journal on Selected Areas on Communications,* vol. 18, no. 5, pp. 779-793, 2000.

[45] J. Doucette *et al*., "On the availability and capacity requirements of shared backup path-protected mesh networks," *Optical Networks Magazine,* vol. 4, no. 6, pp. 29-44, 2003.

[46] G. Shen *et al.*, "Optimal design for shared backup path protected elastic optical networks under signle-failure," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 6, no. 7, pp. 649-659, 2014.

[47] D. Stamatelakis and W. Grover, "Theoretical underpinnings for the efficiency of restorable networks using pre-configured cycles ("*p*-cycles")," *IEEE Transactions on Communications,* vol. 48, no. 8, pp. 1262-1265, 2000.

[48] Y. Wei *et al.*, "Applying *p*-cycle technique to elastic optical networks,"

International Conference on Optical Network Design and Modelling, Sweden, 2014.

[49] R. Iraschko *et al.*, "Optimal capacity placement for path restoration in STM or ATM mesh-survivable networks," *IEEE/ACM Transctions on Networking*, vol 6, no. 3, pp 325 – 336, 1998.

[50] S. Sengupta and R. Ramamurthy, "Capacity efficient distributed routing of mesh-restored lightpaths in optical networks," *IEEE Global Communications Conference (GlobeCom 2001)*, San Antonio, TX, 2001.

[51] W. Grover, "Selfhealing networks - a distributed algorithm for k-shortest link-disjoint paths in a multi-graph with application in realtime network restoration," Ph.D. dissertation, University of Alberta, Edmonton, Canada, 1989.

[52] W. Grover, "The selfhealing network: a fast distributed restoration technique for networks using digital cross-connect machines," *IEEE Global Communications Conference (GlobeCom 1987)*,, Tokyo, Japan, 1987.

[53] W. Grover and J. Doucette, "Increasing the efficiency of span-restorable mesh networks on low-connectivity graphs," *Design of Reliable Communication Networks (DRCN 2001)*, Budapest, Hungary, 2001.

[54] A. Kodian and W. Grover, "Failure-independent path-protecting *p*-cycles: efficient and simple fully pre-connected optical-path protection," *Journal of Lightwave Technology,* vol. 23, no. 10, pp. 3241-3259, 2005.

[55] G. Shen and G. Grover, "Extending the *p*-cycle concept to path segment protection for span and node failure recovery," *IEEE Journal on Selected Areas in Communications,* vol. 21, no. 8, pp. 1306-1319, 2003.

[56] D. Onguetou and W. Grover, "A new approach to node-failure protection with span-protecting *p*-cycles," *International Conference on Transparent Optical Networks (ICTON 09),* 2009.

[57] D. Stamatelakis and W. Grover, "IP layer restoration and network planning based on virtual protection cycles," *IEEE Journal on Selected Areas in Communications,* vol. 18, no. 10, pp. 1938-1949, October 2000.

[58] H. Sakauchi *et al.*, "Spare-channel design schemes for self-healing networks," *IEICE Trans. Commun*, 1992.

[59] B. Venables *et al.*, "Two strategies for spare capacity placement (SCP) in mesh restorable networks," *IEEE International Conference on Communications (ICC 93)*, Geneva,1993.

[60] C. Wynants, *Network Synthesis Problems (Combinatorial Optimization Series),* New York: Kluwer Academic Publishers, 2001.

[61] Y. Xiong and L. Mason, "Restoration strategies and spare capacity requirements in self-healing ATM networks," *IEEE/ACM Transaction on Networking,* vol. 7, no. 1, pp. 98 -110, 1999.

[62] J. Doucette and W. Grover, "Influence of modularity and economy of scale effects on design of mesh-restorable DWDM networks," *IEEE Journal on Selected Areas in Communications,* vol. 18, no. 10, pp. 1912-1923, 2000.

[63] W. Winston, *Operations Research Applicat. and Algorithms*, 4 ed., Belmont, CA: Duxbury Press, 2004.

[64] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers and Operation Research*, vol 13, no. 5, pp.533-549, 1986.

[65] Metaheuristics for Optimization - ParadisEO - [Online]. Available on: http://paradiseo.gforge.inria.fr/addon/paradiseo-eo/concepts/paradiseo-eo-design.pdf "Retrieved: February 8, 2014".

[66] E. Talb, *Metaheuristics from Design to Implementation*. Hoboken, NJ: John Wiley & Sons, 2009.

[67] K. Le and T. Huynh, “Genetic algorithm for solving survivable network design problem with extending-cycle-based protected working capacity envelope,” *IEEE Conference on Nature and Biologically Inspired Computing*, Porto, Portugal, 2014.

[68] M. Fisher, “The Lagrangian relaxation method for solving integer programming problems,” *INFORMS,* vol. 50, no. 12, pp. 1861-1871, 2004.

[69] M. Maeda, “Management and control of transparent optical networks,” *IEEE Journal on Selected Areas in Communications,*vol. 16, no. 7, pp. 1005-1023, 1998.

[70] W. Grover and J. Doucette, “Design of a meta-mesh of chain sub-networks: enhancing the attractiveness of mesh-restorable WDM networking on low connectivity graphs,” *IEEE Journal on Selected Areas in Communications,* vol. 20, no. 1, pp. 47-61, 2002.

[71] J. Doucette and W. Grover, “Comparison of mesh protection and restoration schemes and the dependency on graph connectivity,” *Design of Reliable Communication Networks Conference (DRCN 2001)*, Budapest, Hungary, 2001.

[72] R. Doverspike and B. Wilson, “Comparison of capacity efficiency of DCS network restoration routing techniques,” *JNSM,* vol. 2, no. 2, pp. 95-123, 1994.

[73] A. Kershenbaum *et al.*, “MENTOR: an algorithm for mesh network topological optimization and routing,” *IEEE Transactions on Commununications,* vol. 39, no. 04, pp. 503-513, 1991.

[74] W. Grover and J. Doucette, “Topological design of span-restorable mesh transport networks,” *Journal of Operations Research,* vol. 106, pp. 79-125, 2001.

[75] M. Herzberg *et al.*, “The hop-limit approach for spare-capacity assignment in survivable networks,” *IEEE/ACM Transactions on Networking*, vol. 3, no. 6, pp.

775-784, 1995

[76] J. Akpuh and J. Doucette, "Sizing eligible route sets for restorable network design and optimization," *IEEE International Conference on Communications (ICC 2008)*, Beijing, China, 2008.

[77] B. Todd and J. Doucette, "Use of network families in survivable network design and optimization," *IEEE International Conference on Communications (ICC 2008)*, Beijing, China, 2008.

[78] R. Boorstyn and H. Frank, "Large-scale network topological optimization," *IEEE Transactions on Communications.,* vol. 25, no. 1, pp. 29-47, 1977.

[79] A. Kershenbaum, *Telecommunications Network Design Algorithms*. New York: McGraw-Hill, 1993.

[80] A. Kasem and J. Doucette, "Incremental optical network topology optimization using meta-mesh span restoration," *Design of Reliable Communication Networks Conference (DRCN 2011)*, Krakow, Poland, 2011.

[81] H. Dirilten and R. Donaldson, "Topological design of distributed data commun. networks using linear regression clustering," *IEEE Transactions on Communnications,* vol. 25, no. 10, pp. 1083-1092, 1977.

[82] R. Cahn, *Wide Area Network Design: Concepts and Tools for Optimization.* San Francisco, CA: Morgan Kaufman Publishers, 1998.

[83] D. Schupke, "An ILP for optimal *p*-cycle selection without cycle enumeration," *Optical Network Design and Modelling (ONDM 2004)*, Ghent, Belgium, 2004.

[84] H. Li *et al.*, "Scalable design of *p*-cycles for node protection wothout candidate pre-enumeration," *International Conference on Multimedaia Technology (ICMT 2010)*, Ningbo, China, 2010.

[85] S. Rajagopalan *et al.*, "A Lagrangian relaxation approach to solving the integrated pick-up/drop-off point and AGV flow path design problem," *Applied*

Mathematical Modelling, vol. 28, no. 8, pp. 735-750, 2004.

[86] R. Fourer *et al.*, *AMPL: A Modeling Language for Mathematical Programming*, Belmont, CA: Duxbury Press, 2002.

[87] ILOG, *ILOG CPLEX 11.0 User's Manual*, ILOG Inc., 2007.

[88] B. Gendron *et al.*, *Multicommodity Capacitated Network Design*, New York: Kluwer Academic Publishers, 1999.

[89] H. Kim and J. Hooker, "Solving fixed-charge network flow problems with a hybrid optimization and constraint programming approach," *Journal of Operations Research,* vol. 115, pp. 95-124, 2002.

[90] M. Fisher, "The Lagrangian relaxation method for solving integer problems," *Management Science,* vol. 27, no. 1, pp. 1-18, 1981.

[91] J. Doucette *et al.*, "Combined Node and Span Protection Strategies with Node-Encircling *p*-Cycles," *Design of Reliable Communication Networks Conference (DRCN 2005),* Ischia (Naples), Italy, 2005.

[92] T. Zhao *et al.*, "Finding good candidate node-encircling pre-configuration cycles in survivable WDM mesh networks," *Ineternal Conference on Communications, Circuits, and Systems*, Guilin, China, 2006.

[93] T. Zhao *et al.*, "A novel algorithm for node-encircling and link candidate *p*-cycles design in WDM mesh network," *Journal of Chinese Institute of Engineers,* vol. 29, no. 7, pp. 1227-1233, 2006.

[94] J. Doucette *et al.*, "Algorithmic approaches for efficient enumeration of candidate *p*-cycles and capacitated *p*-cycle network design," *Design of Reliable Communication Networks Conference (DRCN2003)*, Banff, Alberta, 2003.

[95] H. Zhang and O. Yang, "Finding protection cycles in DWDM networks," *IEEE International Conference on Communications (ICC 2002)*, New York, 2002.

[96] B. WU *et al.*, "ILP Formulations for *p*-Cycle Design Without Candidate Cycle Enumeration," *IEEE/ACM Transactions on Networking,* vol. 18, no. 1, pp. 284-295 , 2010.

[97] J. Chu and P. Beasly, "A Genetic Algorithm for the Set Covering Problem," *European Journal of Operation Research,* vol. 94, pp. 392-404, 1996.

[98] M. Luchian and H. Ionita, "Two Problem Independent Method for Generating Initial Solution," *IEEE congress on evolutionary computation,* vol. 2, 2005.

[99] O. Peyran and W. Zhunag, "Educating initial solution for genetic algorithms: a chip planning optimization example," *Asia-Pacific Conference on Simulated Evolution and Learning,* Singapore, 2002.

[100] K. Sastry and D. Goldberg, *Genetic Algorithms, Search methodologies,* New York: Springer, 2005.

[101] S. Liu, "Approximate Alorgrithms for the Global Planning Problem of UTMS Networks," M.S. thesis, Carleton University, Ontario, Canada, 2009.

[102] D. Goldberg *et al.*, "Genetic algorithms, noise, and the sizing of populations," *Complex Systems,* vol. 6, pp. 333-362, 1992.

[103] B. Goldberg and L. Miller, "Genetic algorithms, tournament selection, and the effects of noises," *Complex Systems,* vol. 9, pp. 193-212, 1995.

[104] B. Zhang and J. Kim, "Comparison of selection method for evolutionary optimization," *An International Journal on Internet,* vol. 2, no. 1, pp. 55-70, 2000.

[105] M. Chakrabotry and U. Chakraborty, "An analysis of linear ranking and binary tournament selection in genetic algorithms," *International Conference on Information, Communications and Signal Processing (ICICS),* vol. 1, 1997.

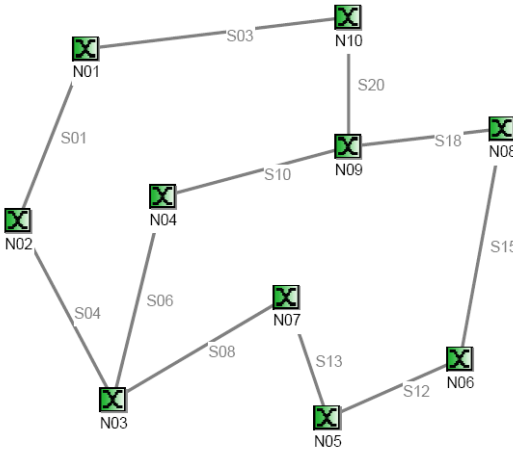
[106] C. Reeves and J. Rowe, *Genetic Algorithms Principles and Perspectives–A*

Guide to GA Theory, New York: Kluwer Academic Publisher, 2003.

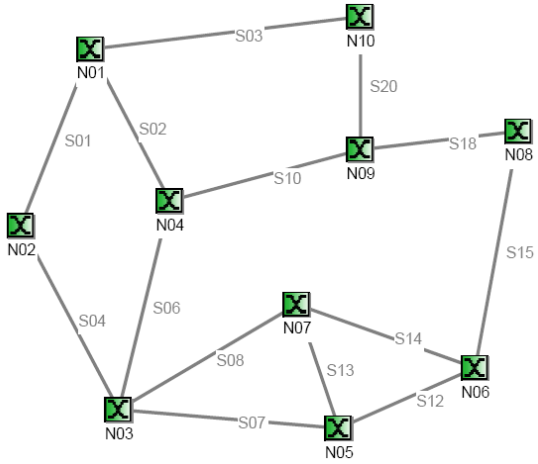
[107] L. Davis, *Handbook of Genetic Algorithms,* ST: Van Nostrand Reinhold, 1991.

Appendix 1 Network Families

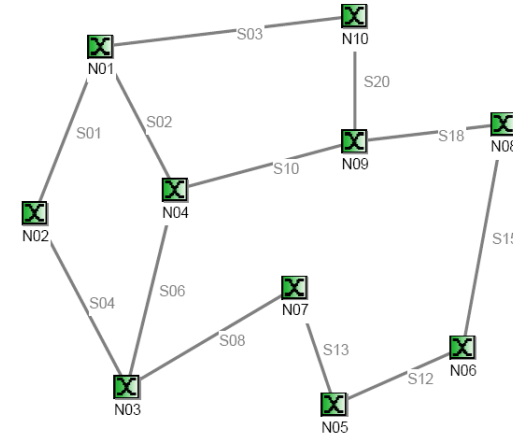
1.1 10 Node Network Family



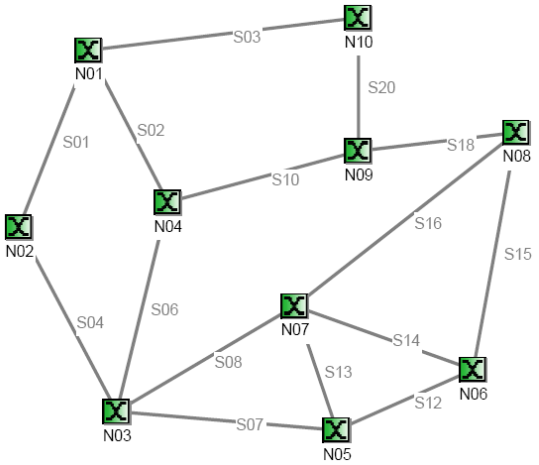
11 s



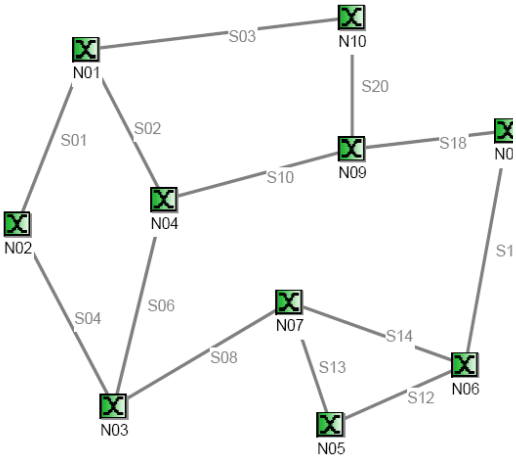
14 s



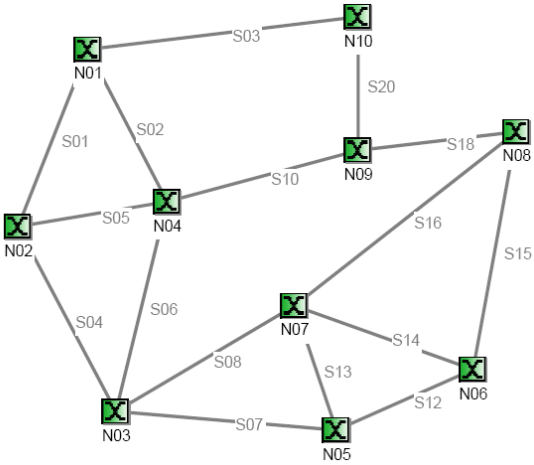
12 s



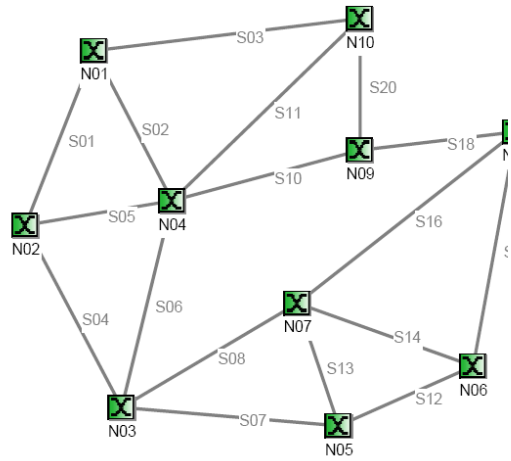
15s

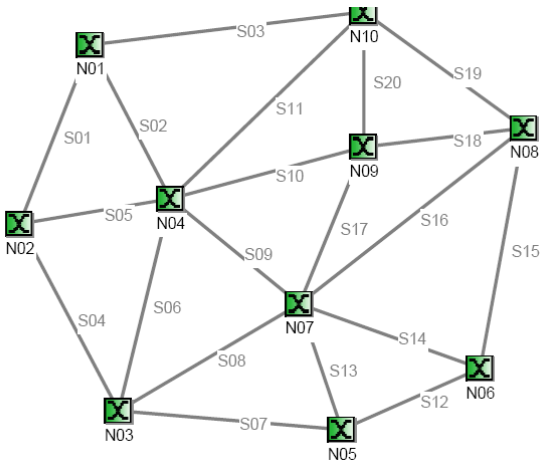


13 s



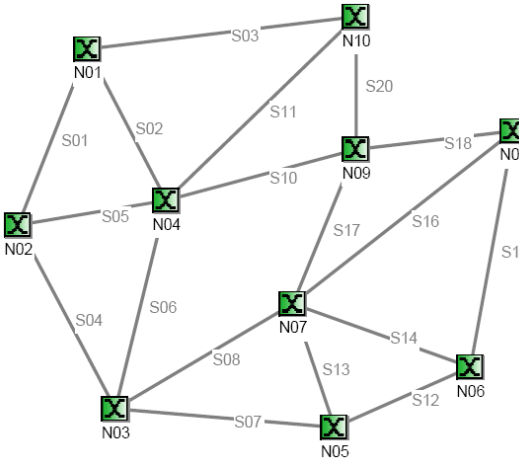
16 s



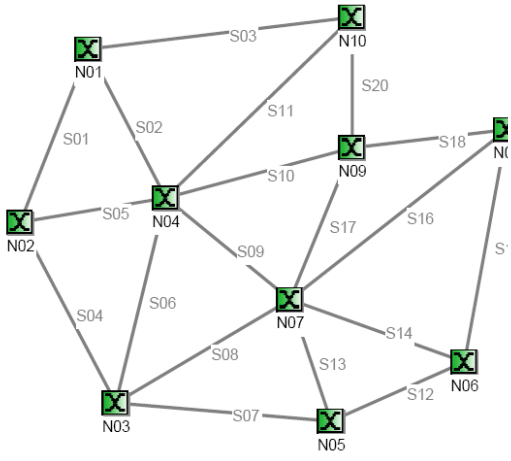


20s

17s

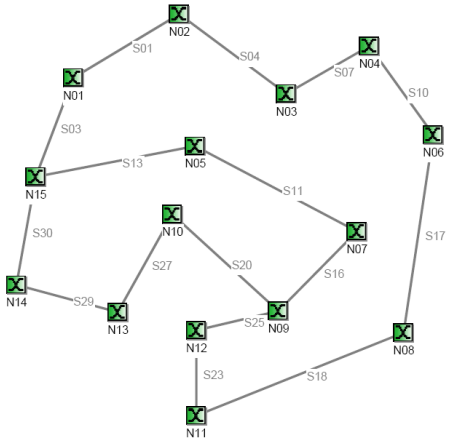


18 s

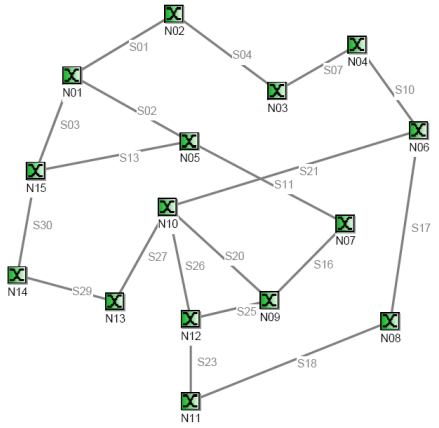


19 s

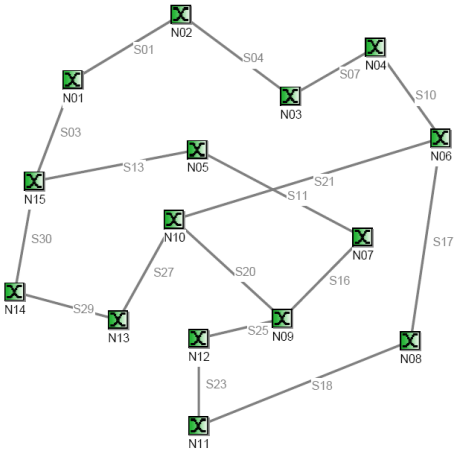
1.2 15 Node Network Family



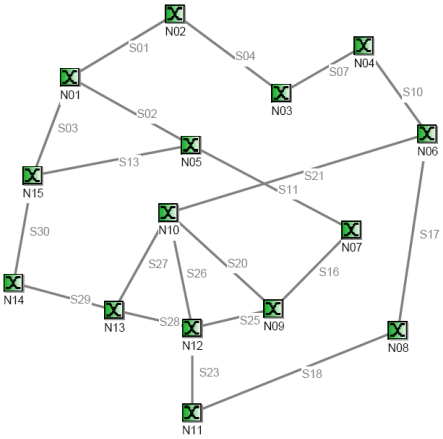
16 s



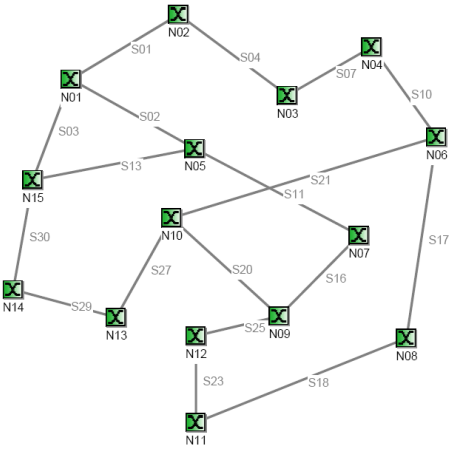
19 s



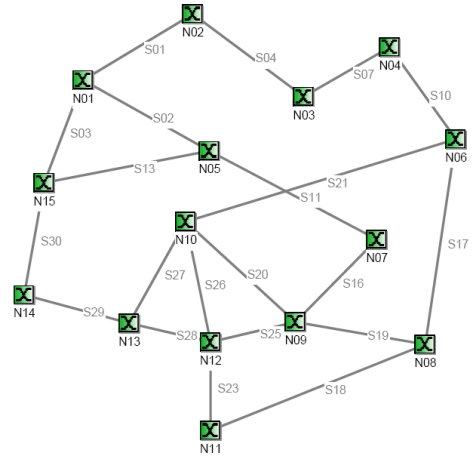
17 s



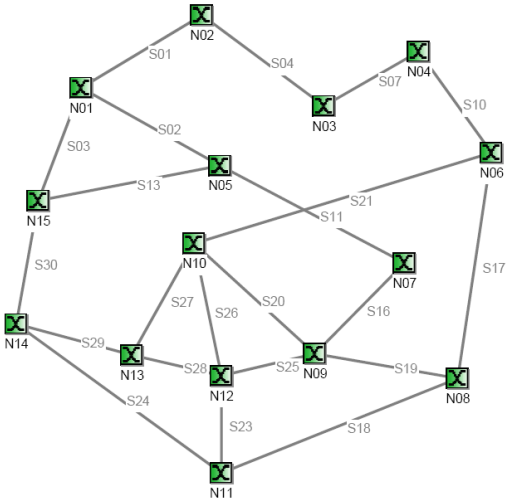
20 s



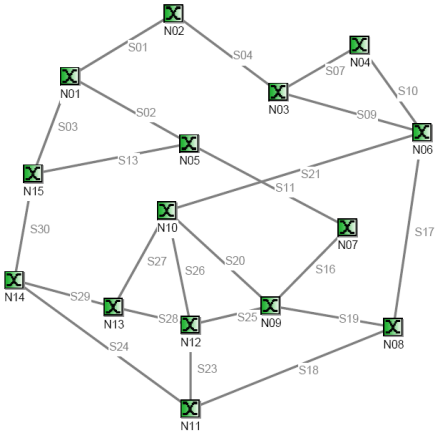
18 s



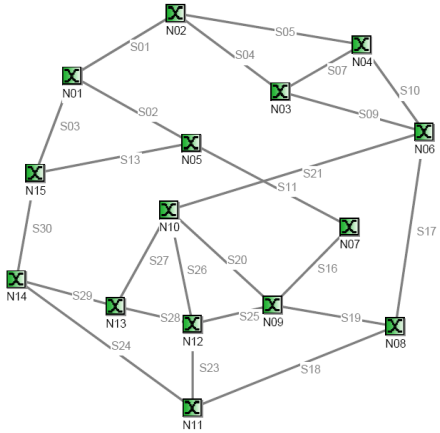
21 s



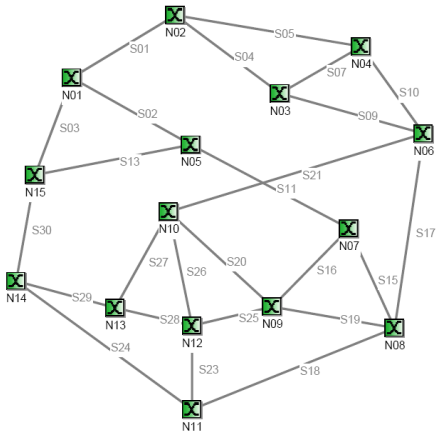
22 s



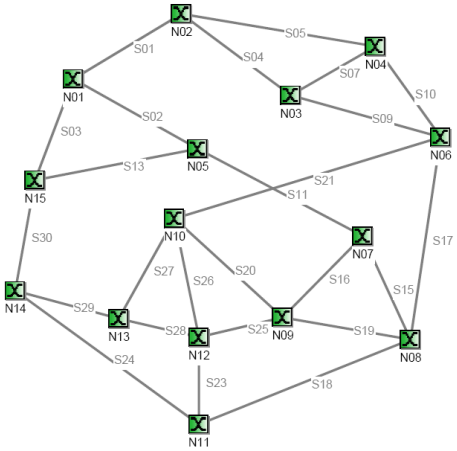
23 s



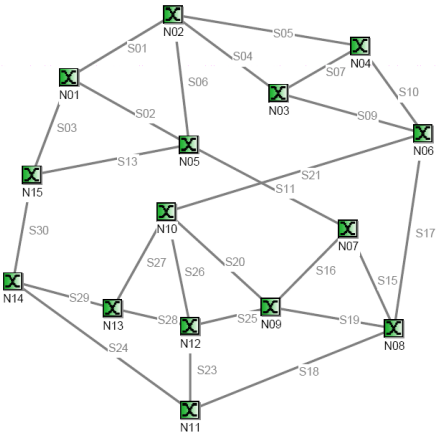
24 s



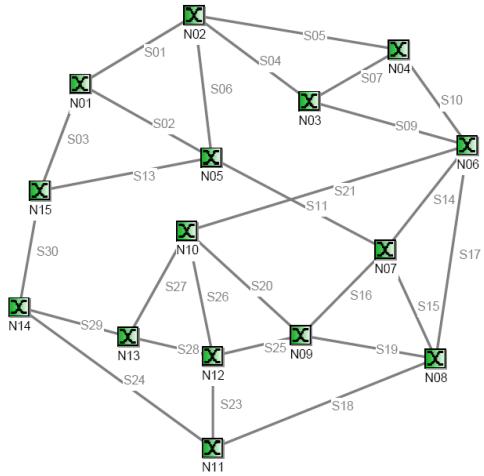
25 s



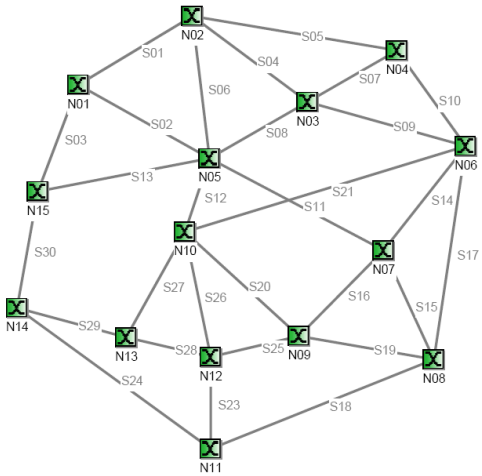
26 s



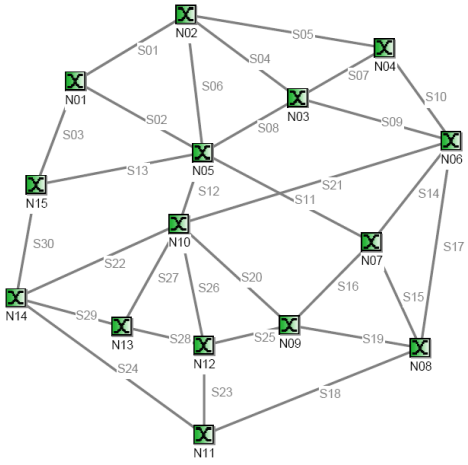
27 s


N02
S05
N04
S01
S04
S07
N01
S06
N03
S10
S02
S09
S03
N06
N05
S13
S21
S11
S14
N15
N10
N07
S17
S30
S27
S20
S26
S16
S15
N14
S29
N13
S28
S25
N09
S19
N12
N08
S24
S23
S18
N11

28 s


N02
S05
N04
S01
S04
S07
N01
S06
N03
S10
S02
S08
S09
S03
N06
N05
S13
S12
S21
S11
S14
N15
N10
N07
S17
S30
S27
S20
S26
S16
S15
N14
S29
N13
S28
S25
N09
S19
N12
N08
S24
S23
S18
N11

29 s


N02
S05
N04
S01
S04
S07
N01
S06
N03
S10
S02
S08
S09
S03
N06
N05
S13
S12
S21
S11
S14
N15
N10
N07
S17
S30
S22
S27
S20
S26
S16
S15
N14
S29
N13
S28
S25
N09
S19
N12
N08
S24
S23
S18
N11

30 s

Appendix 2 AMPL ILP Models

2.1 MetaMesh Span Restoration

```
# Joint capacity placement - Joint Fixed Charge plus Work Routing and Sparing
# using flow variables
# 13-May-2010 - John Doucette and Ahmed Kasem

# TOPOLOGY DEFINITION
set Node;
      # Set of all logical nodes.

set Demand;
      # Set of all demands.

set DirectSpan default {};
      # Set of all direct spans.

set ChainSpan;
      # Set of all chain spans.

set OptionalSpan;
      # Set of all optional spans.

set BypassSpan;
      # Set of all possible bypass spans.

set ChainOfBypass{BypassSpan};
      # Chain spans for each bypass span.

set DirectOptional:={DirectSpan union OptionalSpan};
      # Set of all possible optional spans and direct spans.

set Span:={DirectSpan union ChainSpan union OptionalSpan union BypassSpan};
      # Set of all logical spans.

set ExistSpan:={DirectSpan union ChainSpan};
      # Set of all existing spans.

set NewSpan:={OptionalSpan union BypassSpan};
      # Set of all possible new spans.

set BypassOfChain{ChainSpan};
      # Bypass spans for each chain span.

set Ncnb;
      # set of all nodes that belong to a chain excluding the anchor nodes.

set BypassOfNode{Ncnb};
      # Bypass spans for each node that belongs to a chain excluding the
```

```
      # anchor nodes.

param Incidence{j in Span, n in Node} default 0;
      # Equal to 1 if span j is incident on node n, 0 otherwise.

param IncidenceA{j in Span, n in Node} default 0;
      # Equal to 1 if span j starts at node n, 0 otherwise (abitrary whether
      # starts or ends on node).

param IncidenceB{j in Span, n in Node} default 0;
      # Equal to 1 if span j ends at node n, 0 otherwise.

# WORKING DEMANDS

param Origin{r in Demand} symbolic in Node;
      # Origin node of span demand r.

param Destination{r in Demand} symbolic in Node;
      # Destination node of demand r.

param DemandUnits{r in Demand} default 0;
      # Size of demand r.

# COST DATA

param UnitCost{j in Span};
      # The cost of placing a unit of capacity on span j.

param FixedCost{j in OptionalSpan};
      # The fixed cost of optiona span j.


# VARIABLES

var work_flow_from{n in Node, j in Span, r in Demand: Incidence[j,n]=1} >=0,
<=DemandUnits[r];

var work_flow_into{n in Node, j in Span, r in Demand: Incidence[j,n]=1} >=0,
<=DemandUnits[r];
      # Total directed working traffic flows from and into node n on span j
      # for demand r.

var work{j in Span} >=0, <=10000 integer;
      # Total amount of working capacity (i.e. number of wavelengths) placed
      # on span j.

var spare_flow_from{n in Node, j in Span, i in Span: Incidence[j,n]=1 and
i<>j} >=0, <=10000;

var spare_flow_into{n in Node, j in Span, i in Span: Incidence[j,n]=1 and
i<>j} >=0, <=10000;
      # Total directed restoration flows from and into node n on span j for
      # span failure i.

var spare{j in Span} >=0, <=10000 integer;
      # Total amount of spare capacity (i.e. number of wavelengths) placed on
      # span j.
```

```
var Delta{j in NewSpan} >=0, <=1 integer;
      # Equal to 1 if span j is used, equal to 0 otherwise.

var y{n in Ncnb} >=0, <=1 integer;
      # Equal to 1 if no optional span is chosen for any node in the Ncnb
      # set, equal to 0 otherwise.

# OBJECTIVE FUNCTION
minimize total_cost:
sum{j in Span} (spare[j] + work[j]) * UnitCost[j] + sum{k in OptionalSpan}
FixedCost[k]* Delta[k];

# WORK-RELATED CONSTRAINTS

subject to source_work_flows{r in Demand, n in Node: n =  Origin[r]}:
sum {j in Span: Incidence[j,n]=1} work_flow_from[n,j,r] = DemandUnits[r];
      # Working flows for demand r flowing FROM the origin node is equal to
      # the total demand on that demand pair.

subject to no_work_flow_into_origin{r in Demand, n in Node, j in Span: n =
Origin[r] and Incidence[j,n]=1}:
work_flow_into[n,j,r] = 0;
      # Working flows into the origin of a demand are zero.

subject to sink_work_flows{r in Demand, n in Node: n = Destination[r]}:
sum {j in Span: Incidence[j,n]=1} work_flow_into[n,j,r] = DemandUnits[r];
      # Working flows for demand r flowing INTO the destination node is equal
      # to the total demand on that demand pair.

subject to no_work_flow_from_destination{r in Demand, n in Node, j in Span: n
=  Destination[r] and Incidence[j,n]=1}:
work_flow_from[n,j,r] = 0;
      # Working flows from the destination of a demand are zero.

subject to work_flow_conservation{r in Demand, n in Node: n <> Origin[r] and
n<>Destination[r]}:
sum {j in Span: Incidence[j,n]=1} work_flow_from[n,j,r] = sum {j in Span:
Incidence[j,n]=1} work_flow_into[n,j,r];
      # Flow coming out of a node equals flow going into the node.

subject to anti_symettry_w1{j in Span, r in Demand}:
sum {n in Node: IncidenceA[j,n]=1} work_flow_from[n,j,r] = sum {n in Node:
IncidenceB[j,n]=1} work_flow_into[n,j,r];
      # Flow from one node on a span must be going into the other node on
      # that same span.

subject to anti_symettry_w2{j in Span, r in Demand}:
sum {n in Node: IncidenceB[j,n]=1} work_flow_from[n,j,r] = sum {n in Node:
IncidenceA[j,n]=1} work_flow_into[n,j,r];
      # Flow from one node on a span must be going into the other node on
      # that same span.


subject to working_capacity_placement{j in Span}:
work[j] >= sum{n in Node, r in Demand: Incidence[j,n]=1}
work_flow_from[n,j,r];
```

```
      # Sufficient working capacity must be placed on span j to accommodate
      # all flows routed over it.

subject to NoOptionalSpan_NoBypassSpan{n in Ncnb}:
2 * sum{j in OptionalSpan : Incidence[j,n]=1} Delta[j] >= sum{k in BypassSpan
: Incidence[k,n]=1} Delta[k];
      # Make sure that there is no bypass span for any nodes which does not
      # connected to any chosen optional span.

subject to bypassSpans_nodeonchain_1{n in Ncnb}:
1 >= sum{j in BypassOfNode[n]} Delta[j];
      # Make sure that only one bypass span or less is chosen for any node on
      # a chain.

subject to optionalspan_nodeonchain_2_1{n in Ncnb}:
sum{j in OptionalSpan : Incidence[j,n]=1} Delta[j]<=
(card(OptionalSpan)+1)*(1-y[n]);

subject to optionalspan_nodeonchain_2_2{n in Ncnb, r in Demand, b in
BypassOfNode[n],g in Node: Incidence[b,g]=1}:
work_flow_from[g,b,r] <= 25000*y[n];
      # Make sure that no bypass spans over a chosen  optional span.


# SPARE-RELATED CONSTRAINTS

subject to bypass_spare_flows{i in Span,j in BypassSpan, n in Node:
Incidence[j,n]=1 and i<>j}:
spare_flow_from[n,j,i] = 0;
      # Restoration flows over the bypass spans are zero.

subject to source_spare_flows{i in Span, n in Node: IncidenceA[i,n]=1}:
sum {j in Span: Incidence[j,n]=1 and i<>j} spare_flow_from[n,j,i] = work[i];
      # Restoration flows for failure of span i flowing FROM its origin node
      # is equal to the total working capacity on that failed span.

subject to no_spare_flow_into_origin{i in Span, n in Node, j in Span:
IncidenceA[i,n]=1 and Incidence[j,n]=1 and i<>j}:
spare_flow_into[n,j,i] = 0;
      # Restoration flows into the origin of a failed span are zero.

subject to sink_spare_flows{i in Span, n in Node: IncidenceB[i,n]=1}:
sum {j in Span: Incidence[j,n]=1 and i<>j} spare_flow_into[n,j,i] = work[i];
      # Restoration flows for failure of span i flowing INTO the destination
      # node is equal to the total working capacity on that failed span.

subject to no_spare_flow_from_destination{i in Span, n in Node, j in Span:
IncidenceB[i,n]=1 and Incidence[j,n]=1 and i<>j}:
spare_flow_from[n,j,i] = 0;
      # Restoration flows from the destination of a failed span are zero.

subject to spare_flow_conservation{i in Span, n in Node: Incidence[i,n]<>1}:
sum {j in Span: Incidence[j,n]=1 and i<>j} spare_flow_from[n,j,i] = sum {j in
Span: Incidence[j,n]=1 and i<>j} spare_flow_into[n,j,i];
      # Flow coming out of a node equals flow going into the node.

subject to anti_symettry_s1{j in Span, i in Span: i<>j}:
```

```
sum {n in Node: IncidenceA[j,n]=1} spare_flow_from[n,j,i] = sum {n in Node:
IncidenceB[j,n]=1} spare_flow_into[n,j,i];
      # Flow from one node on a span must be going into the other node on
      # that same span.

subject to anti_symettry_s2{j in Span, i in Span: i<>j}:
sum {n in Node: IncidenceB[j,n]=1} spare_flow_from[n,j,i] = sum {n in Node:
IncidenceA[j,n]=1} spare_flow_into[n,j,i];
      # Flow from one node on a span must be going into the other node on
      # that same span.

subject to no_spare_flow_forBypass_OverItsChain{i in BypassSpan, n in Node, j
in ChainOfBypass[i]: Incidence[i,n]=1 and Incidence[j,n]=1 and i<>j}:
spare_flow_from[n,j,i] = 0;
      # Restoration flows for bypass span over its chain spans are zero.

subject to spare_capacity_placement_1{j in Span, i in DirectOptional, n in
Node: IncidenceA[j,n]=1 and i<>j}:
spare[j] >= spare_flow_from[n,j,i] + spare_flow_into[n,j,i];
      # Sufficient spare capacity must be placed on span j to accomodate all
      # flows routed over it for each span failure i belong to direct spans
      # or optional spans.

subject to spare_capacity_placement_2{j in Span, i in ChainSpan, b in
BypassOfChain[i], n in Node: IncidenceA[j,n]=1 and i<>j and j<>b}:
spare[j] >= spare_flow_from[n,j,i] + spare_flow_into[n,j,i] +
spare_flow_from[n,j,b] + spare_flow_into[n,j,b];
      # Sufficient spare capacity must be placed on span j to accomodate all
      # flows routed over it for each span failure i belong to chain spans
      # with its corresponding bypass spans.

# SPAN-USE CONSTRAINT

subject to span_use_1{j in NewSpan}:
work[j] + spare[j] <= Delta[j]*25000;
      # If there is any capacity placed on span j, then psi[j] must be forced
      # to equal one (i.e. span j is used).

subject to span_use_2{j in NewSpan}:
work[j] + spare[j] >= Delta[j];
      # This condition to keep the delta at zero in case that the bypass span
      # did not be used.
```

2.2 Incremental Topology Optimization using *p*-Cycle method

```
#Written by Ahmed Zaky, Md Noor E Noor, and John Doucette

#***************************
# SETS
#***************************

set Node;
# Set of all logical nodes.

set Demand;
# Set of all demands.

set Existing_SPANS;
# Set of all existing spans.

set Future_SPANS;
# Set of all future spans.

set Span:=Existing_SPANS union Future_SPANS;
# Set of all spans.It is a union of existing span and future span we may add.

set PCYCLES;
# Set of all p-cycles.

param Incidence{j in Span, n in Node} default 0;
      # Equal to 1 if span j is incident on node n, 0 otherwise.

param IncidenceA{j in Span, n in Node} default 0;
      # Equal to 1 if span j starts at node n, 0 otherwise (abitrary whether
      # starts or ends on node).

param IncidenceB{j in Span, n in Node} default 0;
      # Equal to 1 if span j ends at node n, 0 otherwise.

#param factor;

# WORKING DEMANDS

param Origin{r in Demand} symbolic in Node;
# Origin node of span demand r.

param Destination{r in Demand} symbolic in Node;
# Destination node of demand r.

param DemandUnits{r in Demand} default 0;
# Size of demand r.

param Existing_Cost{j in Existing_SPANS};
# Cost of each unit of capacity on existing span j.

param Future_Cost{i in Future_SPANS};
# Cost of each unit of capacity on future span i.

param Xpi{p in PCYCLES, i in Span} default 0;
# Number of paths a single copy of p-cycle p provides for restoration of
```

```
      # failure of span i (2 if straddling span, 1 if on-cycle span, 0
      # otherwise).

param pCrossesj{p in PCYCLES, j in Span} := sum{i in Span: i = j and Xpi[p,j]
= 1} 1;
      # Equal to 1 if p-cycle p passes over span j, 0 otherwise. i.e. if
      # Xpi[p,j] = 1, then p-cycle p crosses span j.

param M :=10000;

param factor:=10;

#***************************
# VARIABLES
#***************************

var p_cycle_usage{p in PCYCLES} >=0 integer, <= 10000;
# Number of copies of p-cycle p used.

var Selection_variable{i in Future_SPANS} integer >=0,<=1 ;
# if we select any future span , this will have a value 1, otherwise 0

var d1{i in Future_SPANS} integer >=0,<=1 ;

var d2{i in Future_SPANS} integer >=0,<=1 ;

var spare{i in Span}>=0 integer, <= 10000;

var work_flow_from{n in Node, j in Span, r in Demand: Incidence[j,n]=1} >=0,
<=DemandUnits[r];

var work_flow_into{n in Node, j in Span, r in Demand: Incidence[j,n]=1} >=0,
<=DemandUnits[r];
      # Total directed working traffic flows from and into node n on span j
      # for demand r.

var Work{j in Span} >=0, <=10000 integer;
      # Total amount of working capacity (i.e. number of wavelengths) placed
      # on span j.

#***************************
# OBJECTIVE FUNCTION
#***************************
minimize totalcost: sum{j in Existing_SPANS} (Existing_Cost[j] * spare[j])+
sum{j in Existing_SPANS} (Existing_Cost[j] * Work[j])
+sum{i in Future_SPANS}(factor*
Future_Cost[i]*Selection_variable[i]+Future_Cost[i] * spare[i]+Future_Cost[i]
* Work[i]);

#***************************
# CONSTRAINTS
#***************************

# WORK-RELATED CONSTRAINTS

subject to source_work_flows{r in Demand, n in Node: n =  Origin[r]}:
sum {j in Span: Incidence[j,n]=1} work_flow_from[n,j,r] = DemandUnits[r];
```

```
      # Working flows for demand r flowing FROM the origin node is equal to
      # the total demand on that demand pair.

subject to no_work_flow_into_origin{r in Demand, n in Node, j in Span: n =
Origin[r] and Incidence[j,n]=1}:
work_flow_into[n,j,r] = 0;
      # Working flows into the origin of a demand are zero.

subject to sink_work_flows{r in Demand, n in Node: n = Destination[r]}:
sum {j in Span: Incidence[j,n]=1} work_flow_into[n,j,r] = DemandUnits[r];
      # Working flows for demand r flowing INTO the destination node is equal
      # to the total demand on that demand pair.

subject to no_work_flow_from_destination{r in Demand, n in Node, j in Span: n
=  Destination[r] and Incidence[j,n]=1}:
work_flow_from[n,j,r] = 0;
      # Working flows from the destination of a demand are zero.

subject to work_flow_conservation{r in Demand, n in Node: n <> Origin[r] and
n<>Destination[r]}:
sum {j in Span: Incidence[j,n]=1} work_flow_from[n,j,r] = sum {j in Span:
Incidence[j,n]=1} work_flow_into[n,j,r];
      # Flow coming out of a node equals flow going into the node.

subject to anti_symettry_w1{j in Span, r in Demand}:
sum {n in Node: IncidenceA[j,n]=1} work_flow_from[n,j,r] = sum {n in Node:
IncidenceB[j,n]=1} work_flow_into[n,j,r];
      # Flow from one node on a span must be going into the other node on
      # that same span.

subject to anti_symettry_w2{j in Span, r in Demand}:
sum {n in Node: IncidenceB[j,n]=1} work_flow_from[n,j,r] = sum {n in Node:
IncidenceA[j,n]=1} work_flow_into[n,j,r];
      # Flow from one node on a span must be going into the other node on
      # that same span.


subject to working_capacity_placement{j in Span}:
Work[j] >= sum{n in Node, r in Demand: Incidence[j,n]=1}
work_flow_from[n,j,r];
      # Sufficient working capacity must be placed on span j to accomodate
      # all flows routed over it.

subject to full_span_restoration{i in Span}:
Work[i] <= sum{p in PCYCLES} Xpi[p,i] * p_cycle_usage[p];
# All span failures must be restorable.

subject to spare_capacity_placement{j in Span}:
spare[j] >= sum{p in PCYCLES} pCrossesj[p,j] * p_cycle_usage[p];
      # Sufficient spare capacity must be placed on each span to
      # simultaneously accommodate all p-cycles used.

subject to decision1  {i in Future_SPANS}: -Selection_variable[i]+1<=M*d1[i];
subject to decision2 {i in Future_SPANS} :  Work[i]+spare[i]<=M*(1-d1[i]);
subject to decision3  {i in Future_SPANS}: Selection_variable[i]-1<=M*d2[i];
subject to decision4 {i in Future_SPANS} :   Work[i]+spare[i]<=M*(1-d2[i]);
```

2.3 Enhanced NEPC ILP Model

```
# Enhanced Node-Encircling p-Cycle JCP IP Model for AMPL - Version 1.0
# March 2010 by Ahmed Kasem and John Doucette
# Copyright (C) 2010 TRLabs, Inc. All Rights Reserved.

# *******************************************************

# TRLabs
# 7th Floor
# 9107 116 Street NW
# Edmonton, Alberta, Canada
# T6G 2V4

# +1 780 441-3800
# www.trlabs.ca

# *******************************************************

# This model, including any data and algorithms contained herein, is the
# exclusive property of TRLabs, held on behalf of its sponsors. Except
# as specifically authorized in writing by TRLabs, the recipient of this
# model shall keep it confidential and shall protect it in whole or
# in part from disclosure and dissemenation to all third parties, and the
# associated readme file must accompany any such disclosure or dissemenation.

# If any part of this model, including any data and algorithms contained
# herein, is used in any derivative works or publications, TRLabs shall be
# duly cited as a reference.Recommended citation is as follows:

# J. Doucette, "ENEPC.mod: Node-Encircling p-Cycle JCP IP Model for
# AMPL - Version 1.0," TRLabs proprietary AMPL ILP model, Edmonton, AB,
# March 2005.

# TRLabs makes no representation or warranties about the suitability of
# this model, either express or implied, including but not limited to
# implied warranties of merchantability, fitness for a particular purpose,
# or non-infringement. TRLabs shall not be liable for any damages suffered
# as a result of using, modifying or distributing this model or its
# derivatives.

#***************************

# This is an AMPL model for determining the minimum-cost NEPC network design.
# This model optimizes NEPC and span-protecting p-cycles so as to fully
protect
# the network from all possible single-link and single-node failures.
# Working capacities are simultaneously optimized.

#***************************

#***************************
# SETS
#***************************

set NODES;
# Set of all nodes.
```

```
set SPANS;
# Set of all spans.

set PCYCLES;
# Set of all p-cycles.

set DEMANDS;
# Set of all O-D demand pairs.

set WORK_ROUTES{r in DEMANDS};
# Set of all working routes for each demand pair r.


#***************************
# PARAMETERS
#***************************

param Cost{j in SPANS};
# Cost of each unit of capacity on span j.

param DemandUnits{r in DEMANDS} default 0;
# Number of demand units between node pair r.

param DemEndNodes{r in DEMANDS, n in NODES} default 0;
# Equal to 1 if node n is an end node of demand r, 0 otherwise.

param ZetaWorkRoute{j in SPANS, r in DEMANDS, q in WORK_ROUTES[r]} default 0;
# Equal to 1 if qth working route for demand between node pair r uses span j
# and 0 otherwise.

param WorkRouteByNode{n in NODES, r in DEMANDS, q in WORK_ROUTES[r]} default
0;
# Equal to 1 if qth working route for demand between node pair r crosses node
# n and 0 otherwise.

param Xpi{p in PCYCLES, i in SPANS} default 0;
# Number of paths a single copy of p-cycle p provides for restoration of
# failure of span i (2 if straddling span, 1 if on-cycle span, 0 otherwise).

param pCrossesj{p in PCYCLES, j in SPANS} := sum{i in SPANS: i = j and
Xpi[p,j] = 1} 1;
# Equal to 1 if p-cycle p passes over span j, 0 otherwise.
# i.e. if Xpi[p,j] = 1, then p-cycle p crosses span j.

param Xpn{p in PCYCLES, n in NODES} default 0;
# Equal to 1 if p-cycle p can act as an NEPC for node n, 0 otherwise.

param MaxCapacity := sum {r in DEMANDS} DemandUnits[r];
# Used for upper bounds on variables.


#***************************
# VARIABLES
#***************************

var workflow{r in DEMANDS, q in WORK_ROUTES[r]} >=0, <= MaxCapacity;
```

```
# Working capacity required by qth working route for demand between node pair
r.

var work{j in SPANS} >=0 integer, <= MaxCapacity;
# Number of working wavelengths placed on span j.

var workToProtect{j in SPANS} >=0 integer, <= MaxCapacity;
# Number of working wavelengths to be protect by p-cycles on span j.

var transitingflow{n in NODES} >=0 integer, <= MaxCapacity;
# Amount of transiting flow through each node.

var p_cycle_usage{p in PCYCLES} >=0 integer, <= MaxCapacity;
# Number of copies of p-cycle p used.

var spare{j in SPANS} >=0 integer, <= MaxCapacity;
# Number of spare links placed on span j.

var total_cost_spare >=0, <=1000000000000;
# Total cost of spare capacity.

var total_cost_work >=0, <=1000000000000;
# Total cost of working capacity.

#*****************************
# OBJECTIVE FUNCTION
#*****************************

minimize TotalCost: total_cost_spare + total_cost_work;
# Minimize the total cost of capacity.
# Total costs of working and spare are calculated individually below as
# variables.
# We do it this way so that we can simply look at the values of those two
# variables to determine the separate costs of working and spare (instead of
# needing to set up a spreadsheet with individual span capacities to
# calculate them).


#*****************************
# CONSTRAINTS
#*****************************

subject to demands_met{r in DEMANDS}:
sum{q in WORK_ROUTES[r]} workflow[r,q] = DemandUnits[r];
# All demands must be fully routed.

subject to working_capacity_assignment{j in SPANS}:
work[j] = sum{r in DEMANDS, q in WORK_ROUTES[r]: ZetaWorkRoute[j,r,q]=1}
workflow[r,q];
# There must be enough working capacity on span j to accomodate all working
# flows simultaneously routed over it by all demand pairs.

subject to working_Capacities_protected{j in SPANS}:
workToProtect[j] = sum{r in DEMANDS, q in WORK_ROUTES[r]:
ZetaWorkRoute[j,r,q]=1 and  sum{n in NODES} WorkRouteByNode[n,r,q]==2}
workflow[r,q];
# Working capacities that will be protected by the p-cycles.
```

```
subject to transiting_flow_calculation{n in NODES}:
transitingflow[n] =
sum{r in DEMANDS, q in WORK_ROUTES[r]: WorkRouteByNode[n,r,q]=1 and
DemEndNodes[r,n] = 0}
workflow[r,q];

subject to full_span_restoration{i in SPANS}:
workToProtect[i] <= sum{p in PCYCLES} Xpi[p,i] * p_cycle_usage[p];
# All span failures must be restorable.

subject to full_node_restoration{n in NODES}:
transitingflow[n] <= sum{p in PCYCLES} Xpn[p,n] * 2 * p_cycle_usage[p];
# All node failures must also be restorable.
# The 2x multiplier is because Xpn is a 1/0 parameter and if
# Xpn[p,n] = 1, then it actually provides two restoration routes.

subject to spare_capacity_placement{j in SPANS}:
spare[j] >= sum{p in PCYCLES} pCrossesj[p,j] * p_cycle_usage[p];
# Sufficient spare capacity must be placed on each span to
# simultaneously accommodate all p-cycles used.



subject to calulate_spare_cost:
total_cost_spare = sum{j in SPANS} Cost[j] * spare[j];
# The total cost of spare capacity is the sum of the costs of spare on each
# span.

subject to calulate_work_cost:
total_cost_work = sum{j in SPANS} Cost[j] * work[j];
# The total cost of working capacity is the sum of the costs of working on
# each span.
```

Appendix 3 Data Files Samples

```
# Enhanced NEPC AMPL Data File.

# Mon Apr 19 21:25:18 2012
# for use with NEPC-INC-TOP.mod AMPL model.
# Generated by ENEPC.exe, written by Ahmed Kasem, and John Doucette, April
2012.

#************************************************

# Contact john.doucette@trlabs.ca or Ahmed Kasem for more information.
# Copyright (C) 2012 TRLabs, Inc. All Rights Reserved.

#    TRLabs
#    Edmonton, AB, Canada
#    +1 780 441-3800
#    www.trlabs.ca

#************************************************
# This software, including any data and algorithms contained within,
# is the exclusive property of TRLabs, held on behalf of its sponsors.
# Except as specifically authorized in writing by TRLabs, the recipient
# of this software shall keep it confidential and shall protect it in
# whole or in part from disclosure and dissemenation to all third
# parties, and the associated readme file must accompany any such
# approved disclosure or dissemenation.
# If any part of this software, including any data and algorithms
# contained herein, is used in any derivative works or publications,
# TRLabs shall be duly cited as a reference. The recommended citation
# is as follows:
# A.Kasem J. Doucette, 'ENEPC-DatPrepV2.exe:
# AMPL Data File Preparation Software Version 1.0,' TRLabs proprietary
# software, Edmonton, AB, Canada, April 2012.

# TRLabs makes no representation or warranties about the suitability
# of this software, either express or implied, including but not
# limited to implied warranties of merchantability, fitness for a
# particular purpose, or non-infringement. TRLabs shall not be liable
# for any damages suffered as a result of using, modifying or
# distributing this software or its derivatives.

#************************************************

# Command Line Used:
# c:\src\ENEPC\Release\Enhanced NEPC.exe
# c:\temp\10n20s1.top c:\temp\10n20s1.dem c:\temp\10n20s1.dat

set NODES :=
N01
N02
N03
N04
N05
N06
N07
```

```
N08
N09
N10
;

set SPANS :=
S01
S02
S03
S04
S05
S06
S07
S08
S09
S10
S11
S12
S13
S14
S15
S16
S17
S18
S19
S20
;

param UnitCost :=
S01   107.703
S02   97.062
S03   154.175
S04   119.000
S05   86.145
S06   122.483
S07   125.399
S08   117.478
S09   93.086
S10   111.826
S11   150.629
S12   84.172
S13   74.000
S14   107.224
S15   137.295
S16   160.240
S17   95.079
S18   90.670
S19   111.018
S20   76.000
;

param TransitingFlow :=
N01   0
N02   0
N03   6
N04   34
N05   5
```

```
N06   4
N07   49
N08   7
N09   29
N10   8
;

param Work :=
S01   7
S02   25
S03   15
S04   12
S05   18
S06   17
S07   12
S08   24
S09   16
S10   15
S11   15
S12   14
S13   28
S14   20
S15   19
S16   12
S17   33
S18   19
S19   22
S20   23
;

set PCYCLES := P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16 P17
P18 P19 P20 P21 P22 P23 P24 P25 P26 P27 P28 P29 P30 P31 P32 P33 P34 P35 P36
P37 P38 P39 P40 P41 P42 P43 P44 P45 P46 P47 P48 P49 P50 P51 P52 P53 P54 P55
P56 P57 P58 P59 P60 P61 P62 P63 P64 P65 P66 P67 P68 P69 P70 P71 P72 P73 P74
P75 P76 P77 P78 P79 P80 P81 P82 P83 P84 P85 P86 P87 P88 P89 P90 P91 P92 P93
P94 P95 P96 P97 P98 P99 P100 P101 P102 P103 P104 P105 P106 P107 P108 P109
P110 P111 P112 P113 P114 P115 P116 P117 P118 P119 P120 P121 P122 P123 P124
P125 P126 P127 P128 P129 P130 P131 P132 P133 P134 P135 P136 P137 P138 P139
P140 P141 P142 P143 P144 P145 P146 P147 P148 P149 P150 P151 P152 P153 P154
P155 P156 P157 P158 P159 P160 P161 P162 P163 P164 P165 P166 P167 P168 P169
P170 P171 P172 P173 P174 P175 P176 P177 P178 P179 P180 P181 P182 P183 P184
P185 P186 P187 P188 P189 P190 P191 P192 P193 P194 P195 P196 P197 P198 P199
P200 P201 P202 P203 P204 P205 P206 P207 P208 P209 P210 P211 P212 P213 P214
P215 P216 P217 P218 P219 P220 P221 P222 P223 P224 P225 P226 P227 P228 P229
P230 P231 P232 P233 P234 P235 P236 P237 P238 P239 P240 P241 P242 P243 P244
P245 P246 P247 P248 P249 P250 P251 P252 P253 P254 P255 P256 P257 P258 P259
P260 P261 P262 P263 P264 P265 P266 P267 P268 P269 P270 P271 P272 P273 P274
P275 P276 P277 P278 P279 P280 P281 P282 P283 P284 P285 P286 P287 P288 P289
P290 P291 P292 P293 P294 P295 P296 P297 P298 P299 P300 P301 P302 P303 P304
P305 P306 P307 P308 P309 P310 P311 P312 P313 P314 P315 P316 P317 P318 P319
P320 P321 P322 P323 P324 P325 P326 P327 P328 P329 P330 P331 P332
;

param Xpi :
[P0, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0 S12
1 S13 1 S14 1 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
```

```
[P1, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0 S12
0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 1 S19 1 S20 1
[P2, *] S01 1 S02 1 S03 0 S04 0 S05 1 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0 S12
0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P3, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 1 S11 0 S12
0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 0
[P4, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 1 S08 1 S09 0 S10 0 S11 0 S12
0 S13 1 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P5, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 1 S07 0 S08 0 S09 0 S10 0 S11 0 S12
0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P6, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 1 S10 0 S11 0 S12
0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P7, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 1 S11 1 S12
0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 1
[P8, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0 S12
0 S13 0 S14 0 S15 0 S16 1 S17 1 S18 1 S19 0 S20 0
[P9, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 1 S12
0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P10, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0
S12 0 S13 0 S14 1 S15 1 S16 1 S17 0 S18 0 S19 0 S20 0
[P11, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P12, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 1 S10 0 S11 0
S12 0 S13 1 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P13, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 1 S10 0 S11 0
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P14, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0
S12 0 S13 0 S14 1 S15 1 S16 2 S17 1 S18 1 S19 0 S20 0
[P15, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 1 S08 1 S09 0 S10 0 S11 0
S12 1 S13 2 S14 1 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P16, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 1
[P17, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0
S12 0 S13 0 S14 0 S15 0 S16 1 S17 1 S18 2 S19 1 S20 1
[P18, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 1 S07 0 S08 0 S09 0 S10 0 S11 0
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P19, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 1 S11 0
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 0
[P20, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0
S12 1 S13 1 S14 2 S15 1 S16 1 S17 0 S18 0 S19 0 S20 0
[P21, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 1 S11 0
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 0 S20 0
[P22, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 1 S11 1
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 1 S19 1 S20 2
[P23, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0
S12 1 S13 1 S14 2 S15 1 S16 2 S17 1 S18 1 S19 0 S20 0
[P24, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 1 S10 0 S11 0
S12 0 S13 1 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P25, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 0 S10 0 S11 1
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P26, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 0 S11 1
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P27, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P28, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0
S12 0 S13 0 S14 1 S15 1 S16 2 S17 1 S18 2 S19 1 S20 1
```

[P29, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 1 S11 0
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 0
[P30, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 1 S11 0
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 0
[P31, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 1 S10 0 S11 0
S12 1 S13 2 S14 1 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P32, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 0 S08 1 S09 1 S10 0 S11 0
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P33, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 0 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 1
[P34, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 1 S11 0
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P35, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P36, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 2 S19 1 S20 1
[P37, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P38, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 1 S19 1 S20 2
[P39, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P40, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 0 S10 0 S11 0
S12 1 S13 1 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 1
[P41, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 1 S11 0
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P42, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 0 S11 1
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P43, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 1 S11 0
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 0 S20 0
[P44, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 0
[P45, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 1 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P46, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 1 S10 0 S11 0
S12 1 S13 2 S14 1 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P47, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 0 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P48, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 0 S11 0
S12 0 S13 1 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P49, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 1 S08 1 S09 0 S10 0 S11 0
S12 1 S13 2 S14 2 S15 1 S16 1 S17 0 S18 0 S19 0 S20 0
[P50, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 2 S10 1 S11 1
S12 0 S13 0 S14 0 S15 0 S16 1 S17 1 S18 2 S19 1 S20 2
[P51, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 1 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P52, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P53, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 1
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P54, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P55, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 1 S11 0
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 0
[P56, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 1 S11 0
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 0

[P57, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 1 S08 1 S09 0 S10 0 S11 0
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 1 S19 0 S20 0
[P58, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 0 S11 1
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P59, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 0 S09 0 S10 0 S11 1
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P60, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 2 S11 1
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P61, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 0 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 1 S19 1 S20 2
[P62, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 0 S11 1
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P63, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P64, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 0 S08 1 S09 0 S10 0 S11 0
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P65, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P66, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 0 S09 0 S10 1 S11 0
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 0 S20 0
[P67, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 1 S11 0
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 0 S20 0
[P68, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 1 S11 0
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 0 S20 0
[P69, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 1 S11 0
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P70, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P71, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 1 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P72, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 0 S09 0 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 1
[P73, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 2 S19 1 S20 1
[P74, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 0 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P75, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 2 S11 1
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P76, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 1 S10 0 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P77, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 2 S10 1 S11 1
S12 0 S13 0 S14 1 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P78, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P79, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 1 S10 0 S11 0
S12 1 S13 2 S14 2 S15 1 S16 1 S17 0 S18 0 S19 0 S20 0
[P80, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P81, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 0
[P82, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 2 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 1 S18 2 S19 1 S20 2
[P83, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 0
[P84, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 0 S09 0 S10 0 S11 1
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 0 S19 1 S20 0

[P85, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 0 S11 0
S12 1 S13 2 S14 1 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P86, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 1 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P87, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 2 S08 1 S09 2 S10 1 S11 0
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P88, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 0 S08 1 S09 1 S10 0 S11 1
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P89, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 0 S11 1
S12 0 S13 1 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P90, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P91, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 0 S11 1
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P92, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P93, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 0 S11 1
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P94, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 1 S08 1 S09 0 S10 0 S11 0
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 1
[P95, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 1 S10 2 S11 0
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 1 S19 0 S20 0
[P96, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 0 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P97, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P98, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 0 S09 0 S10 1 S11 0
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 0 S20 0
[P99, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 2 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P100, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P101, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 2 S07 1 S08 1 S09 1 S10 1 S11 0
S12 1 S13 2 S14 2 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P102, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P103, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 0 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P104, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P105, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 0 S07 0 S08 0 S09 2 S10 1 S11 1
S12 1 S13 1 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P106, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 0 S20 0
[P107, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 2 S09 1 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P108, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 1 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P109, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 0 S09 0 S10 1 S11 2
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 2 S19 1 S20 1
[P110, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 1 S11 0
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P111, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 1
S12 0 S13 1 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P112, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 0 S08 1 S09 0 S10 0 S11 0
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0

[P113, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 1 S11 0
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P114, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P115, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 0 S08 1 S09 1 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 2 S18 0 S19 0 S20 1
[P116, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 1 S11 2
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 2 S19 1 S20 1
[P117, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 2 S19 1 S20 1
[P118, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 1 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P119, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 0 S09 0 S10 2 S11 1
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 2 S20 1
[P120, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P121, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 0 S08 1 S09 0 S10 0 S11 0
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P122, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 1 S10 0 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P123, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 2 S08 1 S09 2 S10 0 S11 1
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P124, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 1
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P125, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P126, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P127, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 1 S11 0
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 0 S20 0
[P128, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 1 S11 0
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 0 S20 0
[P129, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 1 S10 0 S11 0
S12 1 S13 2 S14 2 S15 1 S16 1 S17 0 S18 0 S19 0 S20 0
[P130, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 1 S10 2 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P131, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 0 S09 0 S10 0 S11 1
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 0 S19 1 S20 0
[P132, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 2 S10 1 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P133, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 0 S09 0 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 0 S17 0 S18 1 S19 1 S20 2
[P134, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P135, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 2 S07 1 S08 1 S09 1 S10 0 S11 1
S12 1 S13 2 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P136, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 2 S08 1 S09 2 S10 1 S11 0
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P137, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 0 S11 1
S12 0 S13 1 S14 0 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P138, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 0 S08 1 S09 0 S10 0 S11 0
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P139, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 2 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 1 S18 2 S19 1 S20 2
[P140, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 0 S11 1
S12 0 S13 1 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0

[P141, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 2
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P142, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 0 S11 1
S12 2 S13 1 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P143, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 0 S11 1
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P144, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 1 S10 2 S11 0
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 1 S19 0 S20 0
[P145, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 0 S09 0 S10 0 S11 2
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 0 S19 1 S20 0
[P146, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 2 S08 1 S09 2 S10 1 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P147, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 1 S11 0
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 0 S20 0
[P148, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 0 S09 0 S10 0 S11 0
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 0 S19 1 S20 0
[P149, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P150, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P151, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 0 S11 2
S12 0 S13 1 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P152, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 1 S10 2 S11 2
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 1
[P153, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P154, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 0 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P155, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 0 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P156, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 0
[P157, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 1 S10 0 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P158, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 0 S09 0 S10 1 S11 2
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 2 S19 1 S20 1
[P159, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 0 S13 1 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P160, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P161, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 0 S13 1 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P162, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 1 S10 2 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P163, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 2 S08 1 S09 2 S10 2 S11 1
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P164, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 0 S08 1 S09 0 S10 0 S11 0
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P165, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 1 S11 1
S12 0 S13 0 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 2
[P166, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 2 S07 1 S08 1 S09 1 S10 1 S11 2
S12 1 S13 2 S14 2 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P167, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 0 S08 1 S09 2 S10 0 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P168, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 0 S11 1
S12 1 S13 2 S14 1 S15 2 S16 1 S17 0 S18 0 S19 1 S20 0

[P169, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 1 S11 2
S12 0 S13 1 S14 0 S15 0 S16 0 S17 2 S18 0 S19 0 S20 1
[P170, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P171, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 0 S11 1
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P172, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 1 S11 2
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 2 S19 1 S20 1
[P173, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 0 S07 0 S08 0 S09 2 S10 1 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P174, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 1 S11 2
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P175, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 1 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P176, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 2 S09 1 S10 0 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P177, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P178, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 0 S09 0 S10 2 S11 1
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 2 S20 1
[P179, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 0 S13 1 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P180, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P181, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 0 S09 0 S10 1 S11 0
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 0 S20 0
[P182, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 2 S07 1 S08 1 S09 1 S10 2 S11 1
S12 1 S13 2 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P183, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 0 S13 1 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P184, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 2 S08 1 S09 2 S10 0 S11 1
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P185, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P186, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 1
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P187, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P188, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 0 S20 0
[P189, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P190, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 0 S20 0
[P191, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 1 S15 2 S16 2 S17 1 S18 1 S19 1 S20 2
[P192, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 0 S09 0 S10 2 S11 2
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 2 S20 1
[P193, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 1 S11 0
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P194, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P195, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 2 S09 1 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P196, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 0 S09 0 S10 0 S11 0
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 2 S20 1

[P197, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 2 S08 1 S09 2 S10 0 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P198, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 1 S11 2
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 2 S19 1 S20 1
[P199, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P200, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 2 S19 1 S20 1
[P201, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P202, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P203, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 1 S10 2 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P204, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 2 S08 1 S09 0 S10 0 S11 0
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P205, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 0 S08 1 S09 1 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 2 S17 2 S18 1 S19 1 S20 2
[P206, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P207, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 0 S08 1 S09 0 S10 0 S11 0
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P208, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 2 S10 1 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P209, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 0 S11 1
S12 2 S13 1 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P210, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 0 S09 0 S10 0 S11 2
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 0 S19 1 S20 0
[P211, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P212, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 2 S20 1
[P213, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 2 S08 1 S09 2 S10 1 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P214, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P215, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 0 S20 0
[P216, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 2 S07 1 S08 1 S09 1 S10 0 S11 2
S12 1 S13 2 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P217, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 2 S09 1 S10 2 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P218, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 0 S11 0
S12 1 S13 2 S14 2 S15 1 S16 1 S17 0 S18 0 S19 0 S20 0
[P219, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 0 S11 2
S12 0 S13 1 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P220, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 1 S10 2 S11 2
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 1
[P221, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 0 S09 0 S10 0 S11 2
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 0 S19 1 S20 0
[P222, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 0 S11 2
S12 2 S13 1 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P223, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 0 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P224, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 0 S09 0 S10 0 S11 1
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 0 S19 1 S20 0

[P225, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 2 S07 1 S08 1 S09 2 S10 1 S11 1
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P226, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 2 S08 1 S09 2 S10 2 S11 1
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P227, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 2 S13 1 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P228, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 1
S12 0 S13 1 S14 0 S15 0 S16 0 S17 1 S18 0 S19 0 S20 2
[P229, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 2 S08 1 S09 2 S10 1 S11 0
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P230, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 0 S11 2
S12 0 S13 1 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P231, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 0 S11 1
S12 1 S13 2 S14 1 S15 0 S16 0 S17 0 S18 0 S19 0 S20 0
[P232, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 0 S08 1 S09 2 S10 0 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P233, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 0 S11 1
S12 1 S13 2 S14 1 S15 2 S16 1 S17 0 S18 0 S19 1 S20 0
[P234, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P235, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 0 S11 1
S12 0 S13 1 S14 0 S15 0 S16 1 S17 0 S18 0 S19 1 S20 0
[P236, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 0 S11 1
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P237, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 1 S11 2
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P238, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 2 S11 0
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 1 S19 0 S20 0
[P239, *] S01 0 S02 0 S03 0 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 2 S20 1
[P240, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 2 S09 1 S10 0 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P241, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 2
S12 0 S13 1 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P242, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 2 S08 1 S09 2 S10 2 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P243, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P244, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 2 S08 1 S09 0 S10 0 S11 0
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P245, *] S01 1 S02 2 S03 1 S04 0 S05 1 S06 0 S07 0 S08 0 S09 2 S10 1 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P246, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 0 S11 2
S12 1 S13 2 S14 1 S15 2 S16 1 S17 0 S18 0 S19 1 S20 0
[P247, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 1
[P248, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P249, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 0 S13 1 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P250, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 1 S13 2 S14 1 S15 2 S16 1 S17 0 S18 0 S19 1 S20 0
[P251, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 1 S15 2 S16 2 S17 1 S18 1 S19 1 S20 2
[P252, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 0 S09 0 S10 1 S11 2
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 2 S19 1 S20 1

[P253, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 0 S09 0 S10 2 S11 2
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 2 S20 1
[P254, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 0 S20 0
[P255, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 0 S13 1 S14 0 S15 0 S16 2 S17 1 S18 1 S19 1 S20 2
[P256, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 2 S07 1 S08 1 S09 1 S10 2 S11 2
S12 1 S13 2 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P257, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 1 S11 2
S12 1 S13 2 S14 1 S15 0 S16 0 S17 2 S18 0 S19 0 S20 1
[P258, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 2 S08 1 S09 2 S10 0 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P259, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 1 S11 2
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 2 S19 1 S20 1
[P260, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 2 S09 2 S10 1 S11 2
S12 0 S13 0 S14 0 S15 0 S16 1 S17 1 S18 2 S19 1 S20 2
[P261, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 2
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 2 S19 1 S20 1
[P262, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 2
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P263, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 0 S09 0 S10 2 S11 2
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 2 S20 1
[P264, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P265, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 1 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P266, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P267, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 1 S11 2
S12 0 S13 1 S14 0 S15 0 S16 2 S17 2 S18 1 S19 1 S20 2
[P268, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 1 S15 2 S16 2 S17 1 S18 1 S19 1 S20 2
[P269, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 0 S09 0 S10 2 S11 1
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 2 S20 1
[P270, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P271, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 1 S13 2 S14 1 S15 2 S16 2 S17 1 S18 1 S19 1 S20 2
[P272, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 2 S08 1 S09 2 S10 0 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P273, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P274, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 2 S08 1 S09 2 S10 0 S11 1
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P275, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 0 S08 1 S09 2 S10 2 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P276, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 2 S20 1
[P277, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 1
[P278, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 2 S08 2 S09 1 S10 0 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P279, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 2 S11 1
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P280, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 0 S20 0

[P281, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 2 S09 1 S10 2 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P282, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 0 S11 2
S12 2 S13 1 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P283, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 1 S11 1
S12 0 S13 0 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 2
[P284, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 2 S07 1 S08 1 S09 1 S10 0 S11 2
S12 1 S13 2 S14 2 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P285, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 2 S20 1
[P286, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 2 S20 1
[P287, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 0 S11 2
S12 2 S13 1 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P288, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 0 S11 1
S12 2 S13 1 S14 1 S15 1 S16 2 S17 0 S18 0 S19 1 S20 0
[P289, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 2 S07 1 S08 1 S09 2 S10 1 S11 2
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P290, *] S01 0 S02 0 S03 0 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 2 S20 1
[P291, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 2 S08 1 S09 2 S10 1 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P292, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 2 S08 1 S09 2 S10 2 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P293, *] S01 1 S02 1 S03 0 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 0
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 0 S20 0
[P294, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 0 S11 2
S12 1 S13 2 S14 1 S15 2 S16 1 S17 0 S18 0 S19 1 S20 0
[P295, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 2 S11 2
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 1
[P296, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 2 S08 1 S09 2 S10 2 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P297, *] S01 0 S02 1 S03 1 S04 0 S05 0 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 2 S20 1
[P298, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 2 S08 1 S09 2 S10 2 S11 1
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P299, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 1
S12 1 S13 2 S14 1 S15 0 S16 0 S17 1 S18 0 S19 0 S20 2
[P300, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 0 S11 2
S12 1 S13 2 S14 1 S15 2 S16 1 S17 0 S18 0 S19 1 S20 0
[P301, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 2 S08 2 S09 1 S10 2 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P302, *] S01 1 S02 0 S03 1 S04 1 S05 0 S06 0 S07 1 S08 2 S09 0 S10 0 S11 0
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 2 S20 1
[P303, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 0 S11 1
S12 1 S13 2 S14 1 S15 2 S16 1 S17 0 S18 0 S19 1 S20 0
[P304, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 0 S08 2 S09 2 S10 1 S11 2
S12 0 S13 0 S14 1 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P305, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 2
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 2 S19 1 S20 1
[P306, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 2
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P307, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 1 S15 2 S16 2 S17 1 S18 1 S19 1 S20 2
[P308, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 2 S07 1 S08 1 S09 1 S10 2 S11 2
S12 1 S13 2 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1

```
[P309, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 0 S09 0 S10 1 S11 1
S12 1 S13 0 S14 0 S15 1 S16 0 S17 0 S18 1 S19 2 S20 2
[P310, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P311, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 1 S15 2 S16 2 S17 1 S18 1 S19 1 S20 2
[P312, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 1
[P313, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 1 S15 2 S16 2 S17 1 S18 1 S19 1 S20 2
[P314, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 1
S12 0 S13 1 S14 0 S15 0 S16 1 S17 2 S18 1 S19 2 S20 2
[P315, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 0 S08 1 S09 2 S10 1 S11 1
S12 0 S13 0 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 2
[P316, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 2
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 2 S19 1 S20 1
[P317, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 2 S20 1
[P318, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 1 S11 2
S12 1 S13 2 S14 1 S15 2 S16 2 S17 2 S18 1 S19 1 S20 2
[P319, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 2 S20 1
[P320, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 1 S07 2 S08 2 S09 2 S10 1 S11 2
S12 1 S13 1 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P321, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 2 S20 1
[P322, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 2 S07 1 S08 1 S09 2 S10 1 S11 2
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 2 S19 1 S20 2
[P323, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 0 S11 1
S12 1 S13 2 S14 2 S15 1 S16 1 S17 0 S18 0 S19 2 S20 0
[P324, *] S01 2 S02 1 S03 1 S04 1 S05 1 S06 2 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 2 S20 1
[P325, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 2 S08 1 S09 2 S10 1 S11 1
S12 1 S13 1 S14 2 S15 1 S16 2 S17 2 S18 1 S19 2 S20 2
[P326, *] S01 1 S02 2 S03 1 S04 2 S05 1 S06 1 S07 1 S08 2 S09 2 S10 2 S11 2
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 2 S20 1
[P327, *] S01 1 S02 1 S03 2 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 2 S11 1
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 2 S20 1
[P328, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 2 S11 1
S12 1 S13 2 S14 2 S15 1 S16 2 S17 1 S18 1 S19 2 S20 2
[P329, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 1 S10 1 S11 2
S12 1 S13 2 S14 2 S15 1 S16 1 S17 2 S18 2 S19 2 S20 1
[P330, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 1
S12 2 S13 1 S14 1 S15 1 S16 2 S17 2 S18 1 S19 2 S20 2
[P331, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 1
S12 1 S13 2 S14 1 S15 2 S16 1 S17 2 S18 1 S19 2 S20 2
[P332, *] S01 1 S02 2 S03 1 S04 1 S05 2 S06 2 S07 1 S08 2 S09 2 S10 1 S11 1
S12 1 S13 2 S14 2 S15 1 S16 1 S17 1 S18 2 S19 2 S20 2
;

param Xpn :=
[P0, *] N05 1 N06 1 N07 1
[P1, *] N08 1 N09 1 N10 1
[P2, *] N01 1 N02 1 N04 1
[P3, *] N04 1 N07 1 N09 1
[P4, *] N03 1 N05 1 N07 1
[P5, *] N02 1 N03 1 N04 1
```

```
[P6, *] N03 1 N04 1 N07 1
[P7, *] N04 1 N09 1 N10 1
[P8, *] N07 1 N08 1 N09 1
[P9, *] N01 1 N04 1 N10 1
[P10, *] N06 1 N07 1 N08 1
[P11, *] N04 1 N07 1 N09 1 N10 1
[P12, *] N03 1 N04 1 N05 1 N07 1
[P13, *] N02 1 N03 1 N04 1 N07 1
[P14, *] N06 1 N07 1 N08 1 N09 1
[P15, *] N03 1 N05 1 N06 1 N07 1
[P16, *] N01 1 N04 1 N09 1 N10 1
[P17, *] N07 1 N08 1 N09 1 N10 1
[P18, *] N01 1 N02 1 N03 1 N04 1
[P19, *] N03 1 N04 1 N07 1 N09 1
[P20, *] N05 1 N06 1 N07 1 N08 1
[P21, *] N04 1 N07 1 N08 1 N09 1
[P22, *] N04 1 N08 1 N09 1 N10 1
[P23, *] N05 1 N06 1 N07 1 N08 1 N09 1
[P24, *] N02 1 N03 1 N04 1 N05 1 N07 1
[P25, *] N01 1 N02 1 N04 1 N10 1
[P26, *] N04 1 N07 1 N08 1 N09 2 N10 1
[P27, *] N01 1 N04 1 N07 1 N09 1 N10 1
[P28, *] N06 1 N07 1 N08 1 N09 1 N10 1
[P29, *] N03 1 N04 1 N05 1 N07 1 N09 1
[P30, *] N02 1 N03 1 N04 1 N07 1 N09 1
[P31, *] N03 1 N04 1 N05 1 N06 1 N07 1
[P32, *] N01 1 N02 1 N03 1 N04 1 N07 1
[P33, *] N01 1 N02 1 N04 1 N09 1 N10 1
[P34, *] N04 1 N06 1 N07 1 N08 1 N09 1
[P35, *] N04 1 N07 1 N08 1 N09 1 N10 1
[P36, *] N04 1 N07 1 N08 1 N09 1 N10 1
[P37, *] N03 1 N04 1 N07 1 N09 1 N10 1
[P38, *] N01 1 N04 1 N08 1 N09 1 N10 1
[P39, *] N04 1 N07 1 N08 1 N09 1 N10 1
[P40, *] N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P41, *] N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P42, *] N04 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P43, *] N03 1 N04 1 N07 1 N08 1 N09 1
[P44, *] N02 1 N03 1 N04 1 N05 1 N07 1 N09 1
[P45, *] N01 1 N02 1 N04 1 N07 1 N09 1 N10 1
[P46, *] N02 1 N03 1 N04 1 N05 1 N06 1 N07 1
[P47, *] N01 1 N04 1 N07 1 N08 1 N09 2 N10 1
[P48, *] N01 1 N02 1 N03 1 N04 1 N05 1 N07 1
[P49, *] N03 1 N05 1 N06 1 N07 1 N08 1
[P50, *] N04 1 N07 1 N08 1 N09 1 N10 1
[P51, *] N04 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P52, *] N01 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P53, *] N03 1 N04 1 N05 1 N07 1 N09 1 N10 1
[P54, *] N01 2 N02 1 N03 1 N04 1 N07 1 N09 1 N10 1
[P55, *] N03 1 N04 1 N05 1 N06 1 N07 1 N09 1
[P56, *] N01 1 N02 1 N03 1 N04 1 N07 1 N09 1
[P57, *] N03 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P58, *] N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P59, *] N01 1 N02 1 N03 1 N04 1 N10 1
[P60, *] N04 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P61, *] N01 1 N02 1 N04 1 N08 1 N09 1 N10 1
[P62, *] N03 1 N04 1 N07 1 N08 1 N09 2 N10 1
```

[P63, *] N01 1 N02 2 N03 1 N04 1 N07 1 N09 1 N10 1
[P64, *] N01 1 N02 1 N03 1 N04 2 N07 1 N09 1 N10 1
[P65, *] N01 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P66, *] N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1
[P67, *] N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1
[P68, *] N02 1 N03 1 N04 1 N07 1 N08 1 N09 1
[P69, *] N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1
[P70, *] N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P71, *] N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P72, *] N01 1 N02 1 N03 1 N04 1 N09 1 N10 1
[P73, *] N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P74, *] N01 1 N04 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P75, *] N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P76, *] N01 1 N02 1 N04 1 N07 1 N08 1 N09 2 N10 1
[P77, *] N04 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P78, *] N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P79, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1
[P80, *] N01 2 N02 1 N03 1 N04 1 N05 1 N07 1 N09 1 N10 1
[P81, *] N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N09 1
[P82, *] N01 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P83, *] N01 1 N02 1 N03 1 N04 1 N05 1 N07 1 N09 1
[P84, *] N03 1 N04 1 N05 1 N06 1 N08 1 N10 1
[P85, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1
[P86, *] N01 1 N02 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P87, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P88, *] N01 1 N02 1 N03 1 N04 1 N07 1 N10 1
[P89, *] N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 2 N10 1
[P90, *] N01 1 N02 2 N03 1 N04 1 N05 1 N07 1 N09 1 N10 1
[P91, *] N01 2 N02 1 N03 1 N04 1 N07 1 N08 1 N09 2 N10 1
[P92, *] N01 1 N02 1 N03 1 N04 1 N07 1 N09 1 N10 1
[P93, *] N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 2 N10 1
[P94, *] N03 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P95, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P96, *] N01 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P97, *] N01 1 N02 1 N03 1 N04 2 N05 1 N07 1 N09 1 N10 1
[P98, *] N02 1 N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1
[P99, *] N01 1 N04 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P100, *] N01 1 N02 1 N03 1 N04 1 N07 1 N09 1 N10 1
[P101, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P102, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 2 N09 1 N10 1
[P103, *] N01 1 N02 2 N03 1 N04 1 N07 1 N08 1 N09 2 N10 1
[P104, *] N01 1 N02 1 N03 1 N04 1 N07 1 N09 1 N10 1
[P105, *] N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P106, *] N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1
[P107, *] N01 1 N02 1 N03 1 N04 1 N07 1 N09 1 N10 1
[P108, *] N01 1 N02 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P109, *] N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 1
[P110, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P111, *] N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P112, *] N01 1 N02 1 N03 1 N07 1 N08 1 N10 1
[P113, *] N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1
[P114, *] N01 2 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P115, *] N01 1 N02 1 N03 1 N04 1 N07 1 N09 1 N10 1
[P116, *] N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P117, *] N01 2 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P118, *] N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P119, *] N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 1

[P120, *] N01 1 N02 2 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P121, *] N01 1 N02 1 N03 1 N04 2 N07 1 N08 1 N09 1 N10 1
[P122, *] N01 1 N02 1 N04 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P123, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P124, *] N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P125, *] N01 2 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P126, *] N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P127, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P128, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 2
[P129, *] N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1
[P130, *] N01 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P131, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N08 1 N10 1
[P132, *] N01 1 N04 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P133, *] N01 1 N02 1 N03 1 N04 1 N08 1 N09 1 N10 1
[P134, *] N01 1 N02 2 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P135, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P136, *] N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P137, *] N01 1 N02 1 N03 1 N04 1 N05 1 N07 1 N10 1
[P138, *] N01 1 N02 1 N03 1 N04 2 N07 1 N08 1 N09 1 N10 1
[P139, *] N01 1 N02 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P140, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 2 N10 1
[P141, *] N01 1 N02 1 N03 1 N04 1 N05 1 N07 1 N09 1 N10 1
[P142, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P143, *] N01 2 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 2 N10 1
[P144, *] N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P145, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N08 1 N10 1
[P146, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P147, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P148, *] N01 1 N02 1 N03 1 N05 1 N06 1 N08 1 N10 1
[P149, *] N01 1 N02 1 N03 1 N04 1 N05 1 N07 1 N09 1 N10 1
[P150, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 2 N09 1 N10 1
[P151, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 2 N10 1
[P152, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P153, *] N01 1 N02 1 N03 1 N04 1 N05 1 N07 1 N09 1 N10 1
[P154, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 2 N10 1
[P155, *] N01 1 N02 2 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 2 N10 1
[P156, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N09 1
[P157, *] N01 1 N02 1 N03 2 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P158, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 1
[P159, *] N01 1 N02 1 N03 1 N05 1 N06 2 N07 1 N08 1 N10 1
[P160, *] N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P161, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P162, *] N01 1 N02 1 N04 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P163, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P164, *] N01 1 N02 1 N03 1 N05 2 N06 1 N07 1 N08 1 N10 1
[P165, *] N01 1 N02 1 N03 1 N04 1 N07 1 N09 1 N10 1
[P166, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P167, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 2 N10 1
[P168, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P169, *] N01 1 N02 1 N03 1 N04 1 N05 1 N07 1 N09 1 N10 1
[P170, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 2 N09 1 N10 1
[P171, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 2 N10 1
[P172, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P173, *] N01 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P174, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P175, *] N01 2 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P176, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 2 N10 1

[P177, *] N01 1 N02 1 N03 1 N04 2 N05 1 N06 1 N07 1 N08 2 N09 1 N10 1
[P178, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 1
[P179, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P180, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P181, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 2
[P182, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P183, *] N01 1 N02 1 N03 1 N04 2 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P184, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P185, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P186, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P187, *] N01 2 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P188, *] N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P189, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P190, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 2
[P191, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P192, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 1
[P193, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 2
[P194, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P195, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P196, *] N01 1 N02 1 N03 1 N05 1 N06 1 N08 1 N09 1 N10 1
[P197, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P198, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P199, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P200, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P201, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P202, *] N01 1 N02 2 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P203, *] N01 1 N02 1 N03 2 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P204, *] N01 1 N02 1 N03 1 N05 1 N06 1 N07 1 N08 1 N10 1
[P205, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P206, *] N01 1 N02 1 N03 1 N04 2 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P207, *] N01 1 N02 1 N03 1 N04 2 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P208, *] N01 1 N02 1 N04 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P209, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P210, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N08 1 N10 1
[P211, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P212, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P213, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P214, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P215, *] N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1
[P216, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P217, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P218, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1
[P219, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 2 N10 1
[P220, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P221, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N08 1 N10 1
[P222, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P223, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 2 N10 1
[P224, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N08 1 N10 1
[P225, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P226, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P227, *] N01 1 N02 1 N03 1 N05 1 N06 1 N07 1 N08 1 N10 1
[P228, *] N01 1 N02 1 N03 1 N04 1 N05 1 N07 1 N09 1 N10 1
[P229, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 2
[P230, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 2 N10 1
[P231, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N10 1
[P232, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 2 N10 1
[P233, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1

[P234, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 2 N09 1 N10 1
[P235, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 2 N10 1
[P236, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 2 N10 1
[P237, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P238, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 2
[P239, *] N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P240, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 2 N10 1
[P241, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P242, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P243, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 2 N09 1 N10 1
[P244, *] N01 1 N02 1 N03 1 N04 2 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P245, *] N01 1 N02 1 N03 2 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P246, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P247, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 2 N09 1 N10 1
[P248, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P249, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P250, *] N01 1 N02 1 N03 1 N05 1 N06 1 N07 1 N08 1 N10 1
[P251, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P252, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 1
[P253, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 1
[P254, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 2
[P255, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P256, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P257, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 2 N09 1 N10 1
[P258, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P259, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P260, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P261, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P262, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P263, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 1
[P264, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P265, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P266, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P267, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P268, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P269, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 1
[P270, *] N01 1 N02 1 N03 1 N04 2 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P271, *] N01 1 N02 1 N03 1 N04 2 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P272, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P273, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P274, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P275, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P276, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P277, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P278, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P279, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P280, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 2
[P281, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P282, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P283, *] N01 1 N02 1 N03 1 N04 1 N07 1 N08 1 N09 1 N10 1
[P284, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P285, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P286, *] N01 1 N02 1 N03 1 N04 2 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P287, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P288, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P289, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P290, *] N01 2 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1

```
[P291, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P292, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P293, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 2
[P294, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P295, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P296, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P297, *] N01 1 N02 2 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P298, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P299, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 2 N09 1 N10 1
[P300, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P301, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P302, *] N01 1 N02 1 N03 1 N04 2 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P303, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P304, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P305, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P306, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P307, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P308, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P309, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 2 N08 1 N09 1 N10 1
[P310, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P311, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P312, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P313, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P314, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 2 N07 1 N08 1 N09 1 N10 1
[P315, *] N01 1 N02 1 N03 1 N04 1 N05 2 N06 1 N07 1 N08 1 N09 1 N10 1
[P316, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P317, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P318, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P319, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P320, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P321, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P322, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P323, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 2 N10 1
[P324, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P325, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P326, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P327, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P328, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P329, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P330, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P331, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
[P332, *] N01 1 N02 1 N03 1 N04 1 N05 1 N06 1 N07 1 N08 1 N09 1 N10 1
 ;
```