University of Alberta

PRE-CROSS-CONNECTED PROTECTION ARCHITECTURES FOR TRANSPARENT OPTICAL TRANSPORT NETWORKS

by

Aden Justus Grue

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

Department of Electrical and Computer Engineering

Fall 2009
Edmonton, Alberta

Examining Committee

Dr. Wayne Grover, Electrical and Computer Engineering

Dr. Bruce Cockburn, Electrical and Computer Engineering

Dr. Nelson Amaral, Computing Science

Dr. Ivan Fair, Electrical and Computer Engineering

Dr. Ray DeCorby, Electrical and Computer Engineering

Dr. Nasir Ghani, Electrical and Computer Engineering, University of New Mexico

For my mother, father,
sister and brother.

Abstract

This thesis presents a collection of studies on the topic of survivable transparent optical networks. As backbone networks increase in capacity, the issue of their survivability grows correspondingly in importance. The transparent optical network offers many advantages as the optical backbone network of the future, but also faces several challenges with regards to network protection. The fundamental question addressed by this thesis is therefore "How can we achieve high availability and failure resiliency in transparent optical transport networks?" We cover the design, characterization, and comparison of several protection architectures, many of them novel, that share the property of *pre-cross-connection*, a property that is important for protection of transparent networks. The architectures studied include span p-trees, PXTs, path p-trees, p-cycles, FIPP p-cycles, and UPSR-like p-cycles.

We first present detailed studies of the PXT, span p-tree, and path p-tree architectures. This includes the development of efficient design algorithms and structural analysis of efficient designs. The results indicate a clear hierarchy of efficiency, with cycles being the most efficient, followed by trails, and then trees. However, we discover that architectures with lower average efficiency can be used to complement more efficient structures in rare cases. We also present a new design method for PXTs that is as capacity-efficient as the prior established method, but produces designs with greatly improved structural characteristics.

We then move on to address PXT protection under a collection of real-world design constraints. The results show that PXTs strike a balance between efficiency and flexibility under these constraints. A further study on the problem of failure localization in transparent p-cycle networks demonstrates the possibility of integrating low cost failure localization capabilities into p-cycle network designs.

Finally, we propose UPSR-like p-cycles as a way to combine the simplicity and speed of dedicated protection with the flexibility of mesh-based approaches. The results from our

design experiments show that this architecture is able to take advantage of mesh topologies in a way that traditional ring-based approaches cannot. We also demonstrate methods by which UPSR-like *p*-cycle networks can deliver superior dual failure restorability to a select class of high priority services.

Acknowledgements

I would like to thank the many people who have offered me so much support over the course of my program. This thesis was truly a team effort.

First of all, I am grateful to my supervisor, Dr. Wayne D. Grover, whose mentorship has been invaluable over the years. Even after this many far-ranging conversations in your office I still don't think I appreciate the true depth of your expertise. I am also indebted to all the knowledgeable and talented colleagues who managed to teach me something new on every occasion. In particular, I would like to thank Dr. Matthieu Clouqueur, Dr. Dominic Schupke, Dr. Adil Kodian, Dr. Gangxiang Shen, Dr. Dion Leung, and Anthony Sack. I am also grateful for the financial support offered to me by NSERC, iCore, TRLabs and Nokia Siemens Networks.

I would also like to thank the staff of TRLabs, who always made the office seem both professional and welcoming. In particular, the administrative team of Faith Goller, Linda Richens, and Rhoda Hayes always impressed me with their constant cheerfulness and willingness to help. Also, I will always fondly remember the interesting lunchtime conversations with David Clegg and Dr. Chris Haugen. Additionally, I want to thank my fellow students, who always made TRLabs a fun place to be: Trevor Allen, Brian Forst, Dimitri Baloukov, Diane Onguetou, Brody Todd, and Jude Akpuh.

And of course I must thank Dr. John Doucette, who has variously occupied the roles of boss, colleague, friend, mentor, advisor, and roommate. Thanks, John.

Finally, I would like to express my sincere gratitude to the members of my Ph.D. examining committee: Dr. Bruce Cockburn, Dr. Nelson Amaral, Dr. Ivan Fair, Dr. Ray DeCorby, and Dr. Nasir Ghani. Your incisive comments and questions were instrumental in improving the quality of this thesis.

Contents

List of Tables

List of Figures

Glossary

λ	Wavelength
3R	Regeneration involving re-timing, re-transmission, and pulse re-shaping.
ADM	Add-Drop Multiplexer
AMPL	A Mathematical Programming Language
APS	Automatic Protection Switching
BLSR	Bidirectional Line Switched Ring
CWDM	Coarse Wavelength Division Multiplexing
DCF	Dispersion-Compensating Fibre
DCS	Digital Cross-Connect System
DGE	Dynamic Gain Equalizer
DRS	Disjoint Route Set
DS-N	Digital Signal-Level N
DSP	Demand-wise Shared Protection
DWDM	Dense Wavelength Division Multiplexing
EXC	Electronic Cross-Connect
FIPP	Failure-Independent Path-Protecting (p-Cycles)
Greedy Algorithm	An algorithm that solves an optimization problem by making a series of small, locally optimal choices instead of searching directly for the global optimum.
Green Fields	Design assumption under which the network is built "from scratch", with no initial infrastructure or base level of served traffic.
ILP	Integer Linear Programming
JCP	Joint Capacity Placement
Lightpath	A point-to-point optical circuit with fixed bandwidth and guaranteed transmission characteristics
LP	Linear Programming

Metaheuristic	A heuristic method for solving a general class of problems expressible in a certain form (defined by the particular metaheuristic in question).
MIP	Mixed Integer Programming
MIPGAP	Mixed Integer Programming Gap. A parameter to the MIP solver that determines how close to optimal the current best solution must be before the solver terminates, e.g., a mipgap of 0.005 (or 0.5%) means that the solver must guarantee that the solution at the time of termination is at most 0.5% from optimality.
MPLS	Multiprotocol Label Switching
MPλS	Multiprotocol Wavelength Switching
MTD	Maximum Transmission Distance
Network Family	A set of network topologies consisting of a master network and the set of networks obtained by removing one span at a time from the master while maintaining the property of bi-connection until no more spans can be removed in this way.
Network Topology	A graph, usually undirected, that represents the sites served by the network and the logical connections between them.
NP	Nondeterministic Polynomial (Time)
OADM	Optical Add-Drop Multiplexer
OBS	Optical Burst Switching
OC-N	Optical Carrier Signal - Level N
OEO	Optical-Electronic-Optical
Offline Algorithm	An algorithm pertaining to the design or operation of a network that can be executed off-site, independent from the network nodes, usually to determine some kind of static preplan that is later acted upon by the network during its operation.
Online Algorithm	An algorithm that is executed on-site at a network node(s) to dynamically respond to events in the network while it is operating.
OOO	All optical
Opaque	Describes a channel or process in an optical network that operates on the underlying electronic payload of the optical carrier signal (involving OEO conversion) in a manner that requires the payload to conform to certain specifications. A network in which all paths are generally opaque is called an opaque network.
OPPR	Optical Path Protection Ring
OPS	Optical Packet Switching
OSPR	Optical Shared Protection Ring

OR	Operations Research
OTN	Optical Transport Network
OXC	Optical Cross-Connect
PDH	Plesiochronous Digital Hierarchy
Pre-cross-connected	(also Preconnected) Refers to protection paths that are established as an operational lightpath in advance of failure (as opposed to paths that are only preplanned, which are known in advance of failure but must be established by one or more cross-connect actions at intermediate OXCs at time of failure)
PXT	Pre-Cross-Connected Trail
QoP	Quality of Protection
R_1	Restorability to single failure (span failures usually assumed)
R_2	Restorability to dual failures (span failures usually assumed)
SBPP	Shared Backup Path Protection
SCP	Spare Capacity Placement
SDH	Synchronous Digital Hierarchy
SONET	Synchronous Optical Network
STS-N	Synchronous Transport Signal - Level N
TDM	Time Division Multiplexing
Translucent	Describes a network containing transparent "domains" connected by opaque channels/processes.
Transparent	Describes a channel or process in an optical network that operates purely on the optical signal and is therefore agnostic as to the format of the electronic payload used to modulate the optical carrier. A network in which all paths are transparent is called a transparent network.
UPSR	Unidirectional Path Switched Ring
Waveband	In optical networks, a group of wavelengths that is switched, added, or dropped together as a unit. The wavebands in the network are defined entirely by the capabilities of the switching equipment, i.e., there is no fundamental physical property that groups a certain set of wavelengths into a waveband.
WDM	Wavelength Division Multiplexing
WSS	Wavelength Selective Switch

Chapter 1

Introduction

1.1 Motivation and Objectives

The modern world is highly dependent upon telecommunications networks and the services they provide. The integrity of these services relies directly upon the reliability of the underlying transport (backbone) network. The need for a reliable backbone network is therefore every bit as pressing as it is for any other essential infrastructure of our society, such as water, power, or roads. Modern transport networks invariably use optical fibre as the physical medium, which is by itself highly vulnerable to physical failure (i.e., cable breaks). In fact, the most common cause of network failures is inadvertent cable cuts due to dig-ups [Craw93, Flan90]. And by no means is a cable cut an uncommon event: FCC statistics from 2002 state that metro networks experience 13 cuts per year for every 1000 miles of fibre, and that long haul networks experience 3 cuts per year per 1000 miles of fibre [VePo02]. For long haul networks containing upwards of 30,000 miles of fibre, this corresponds to one cable cut somewhere in the network every four days on average. The importance of network survivability was recently underlined in a new way by the characterization of simple cable cuts as "the ultimate form of denial-of-service attack" [Poul06].

There are many ways to approach the problem of increasing network availability. We could protect the physical fibre itself, but this is a costly and inflexible solution and can only be taken so far. Cables will fail; ensuring high availability means recovering intelligently from failures. We could protect the higher *service* layers of the network directly via redundancy measures in both hardware and software protocols, but one physical layer failure may multiply into hundreds or thousands of service-level connection failures, flooding the

network with retry attempts which can cause further congestion. The reconfigurability and flexibility of the transport layer, combined with the fact that it sits below and therefore supports most other network layers, makes it ideal for implementing restoration techniques. The most popular restorability techniques use switching in the transport layer to reroute damaged paths at failure time, taking advantage of the mesh-like nature of transport networks to utilize alternate paths for the affected flow.

While modern transport networks use optical signals for the point-to-point links, these signals are frequently converted to the electronic domain in order to extract dropped traffic, add new signals, or to perform regeneration, signal monitoring, or other tasks. However, it is highly attractive to consider a network in which signals remain all-optical from origin to destination: a so-called *transparent optical network*. Such a network is agnostic as to the signal format and coding, making it ideal for transporting any number of different signal types from different services simultaneously. However, restoration in transparent networks faces unique challenges. Optical paths in a transparent network must be carefully preengineered, which cannot be achieved if certain paths need to be reconfigured at failure time. We can use dedicated backup paths if we wish to be able to control their optical properties in advance of failures, but this is generally very inefficient in terms of cost.

The aim of this thesis, then, is to investigate methods for achieving cost-efficient restoration in transparent optical networks. We cover many different novel restoration schemes that are appropriate for transparent networks. The focus is on the analysis of capacity efficiency, including comparative studies, through which we gain new knowledge about which architectures are the best suited to certain situations. We also concern ourselves with the structural properties of these architectures, such as complexity and manageability. A significant portion of the thesis is also dedicated to a study on the impact of multiple real-world considerations on the design of a restorable transparent network using various of these architectures.

1.2 Contributions

In brief, the major contributions of this thesis can be summarized as follows:

- Analysis and comparison of novel shared pre-cross-connected protection structures

that establishes a hierarchy of efficiency in the structure space.

- Treatment of real-world design issues in pre-cross-connected architectures.

- Introduction and analysis of a novel dedicated pre-cross-connected protection architecture (UPSR-like *p*-cycles).

1.3 Thesis Organization

Chapter 2 covers fundamental background concepts which are essential to understanding the work presented in this thesis. We cover transport network and restorability concepts, approaches to network design, and present a literature survey on network restorability techniques, focusing on those that are similar or otherwise relevant to the architectures studied in this thesis.

Chapters 3 through 5 provide a thorough analysis of the different structural variations available for preconnected restoration architectures. Chapter 3 begins with span-protecting *p*-trees. The architecture concept is first developed and explained, in terms of the restoration method and switching actions that would be performed at failure time. We then investigate the fundamental efficiency of the span *p*-tree architecture, as deduced from controlled design experiments. We go on to analyze the structural properties of span *p*-trees, to clarify why they are or are not efficient in certain situations. Chapters 4 and 5 repeat this same basic approach for Pre-Cross-Connected Trails (PXTs) and path-protecting *p*-trees, respectively. We also compare multiple architectures in these Chapters, including the architectures studied in this thesis and other well-known schemes (such as *p*-cycles [GrSt98] and FIPP *p*-cycles [KoGr05]). Chapter 4 is unique in that it addresses the characteristics of designs produced using two different methods: first the preexisting PXT design method from the literature and then our own design approach. No such preexisting design methods exist for the *p*-tree architectures.

Chapter 6 changes focus to a more practical treatment of the issues with designing restorable transparent optical networks. The Chapter follows the progression of the design of one specific network, using Pre-Cross-Connected Trails (and to a lesser extent span and path *p*-trees), under an increasing number of design constraints imposed by real-world

considerations.[1] We present the development of our design methods, as well a comparison with other architectures in terms of design cost, flexibility, availability, etc. We also present results for other architectures that were produced by other researchers for comparative purposes. Finally, at the end of this Chapter we also present a small but relevant study into the problem of localizing failures in transparent *p*-cycle networks.

Finally, Chapter 7 investigates UPSR-like *p*-cycles, a new architecture that is conceptually different than those studied earlier in the thesis, due to the way in which it takes advantage of the mesh topology to perform restoration. UPSR-like *p*-cycles use dedicated protection for fast switching, but allow more topological freedom for the working paths than prior ring-based schemes. We begin with an investigation into the best design method for a single UPSR-like *p*-cycle, and then move on to the full network design case, also providing an analysis of characteristic efficiency and a comparison to other similar schemes. Finally, we present ways in which UPSR-like *p*-cycles can be used to implement differentiated QoP classes based on first dual failure restorability levels, and then protection speed classes.

Chapter 8 gives a concluding discussion, summarizing the contributions of the thesis.

[1]This work was performed as part of a joint research project with a team from Nokia Siemens Networks, and was supported by their funding.

Chapter 2

Background

The following Chapter outlines fundamental concepts that will be used throughout the rest of the thesis. We begin with a brief background on transport networking. Next, we give an overview of some terms and language that are frequently used in network theory. We then cover the idea of network restorability and some of the classic approaches that have been studied. Finally, we go over the main techniques for network design used in this thesis, focusing mainly on *Integer Linear Programming* (ILP).

2.1 Transport Networks

2.1.1 Transport Network Concept

From a user's point of view, different types of network services (e.g., telephone service, the Internet, bank machines, etc.) operate independently of one another. For all he knows, there is a separate national telephone network, national data network, national banking network, etc. While this is true in a logical and functional sense, the real view of the underlying network infrastructure is quite a bit more complicated. They may be separated at the local level, but for long distance transport they are all supported by the same high capacity *transport network* (also called a *backbone network*). The fundamental concept is that traffic from multiple different services is packaged together into high rate transport containers that are then routed over a common high-bandwidth infrastructure. The transport network takes advantage of the economies of scale associated with these high rate containers, as well as the statistical stability of large flows, to achieve efficient mass transport of the data produced by a wide variety of services.

Modern transport networks almost exclusively use optical fibre as the transmission medium, so we will restrict ourselves here to discussion of optical transport networks. The current transport networking paradigm is to use reconfigurable optical switching equipment to establish optical circuits between the sites where the optical signals are assembled, disassembled, and switched. This optical switching equipment has a degree of reconfigurability to adapt to changes in or growth of the demand placed on the network, but over the short term optical circuits are relatively fixed. Recently, there have also been many promising developments in the areas of *Optical Packet Switching* (OPS) and *Optical Burst Switching* (OBS), technologies that use rapid setup and teardown of optical circuits to implement a packetized mode of data transfer in optical networks. This represents a move towards the simplified network view of "IP over optics" (with all voice, data, etc. presumably over IP). However, regardless of whether this truly does represent the future of the transport network, circuit-based transport will always have a place, due to the economy of scale associated with the large-scale accumulation of flows.

We will now provide a brief overview of the two most common optical transport networking technologies to facilitate understanding of and motivate the work done in this thesis.

2.1.2 SONET

Conceived in the late 1980s, the *Synchronous Optical Network* (SONET) standard [BlWa02, ToSc02] was developed mainly as a way to enforce standardization of optical signal formats such that network equipment would no longer be proprietary to each operator. In this way, SONET became a widespread standard for *Time Division Multiplexed* (TDM) transport networks. Closely related to SONET is the *Synchronous Digital Hierarchy* (SDH). Both are slight variations on the same concept, with SONET having the highest penetration in the United States and Canada and SDH in the rest of the world. SONET/SDH define a hierarchy of digital signal "packages" that may be multiplexed together via TDM and sent over high capacity optical connections. Table 2.1 (adapted from Chapter 1 of [Grov03]) lists the levels of the SONET signal hierarchy. The capacity of a SONET network is expressed in terms of integer quantities of the signals in this Table. Lower rate signals can be multiplexed together to form higher rate signals, e.g., 3 STS-1 signals multiplexed into a

SONET Signal Level	Optical Signal	Data Rate (Mb/s)
STS-1	OC-1	51.84
STS-3	OC-3	155.25
STS-12	OC-12	622.08
STS-24	OC-24	1244.16
STS-48	OC-48	2488.32
STS-192	OC-192	9953.28

Table 2.1: SONET Signal Hierarchy

STS-3. The SONET level signals are then modulated onto optical carriers for transmission by optical SONET equipment. The equivalent rate optical carrier signals for each SONET signal level are also listed in Table 2.1.

Even though a SONET network is a type of optical network, the concepts discussed in this thesis are not directly applicable to SONET, as the SONET hierarchy rigidly defines the format of the digital signal, so a SONET network is by definition not a transparent optical network. This brief description of SONET is given only to provide some historical background for the concept of *Wavelength Division Multiplexing* (WDM).

2.1.3 WDM

With increasing bandwidth requirements comes the need for higher rate transport signals. Under SONET this means the need to transmit higher and higher data rates on a single wavelength. However, as data rates increase we face challenges with processing this high volume of data in the electronic domain. The alternative is to transmit several optical carriers in parallel over the same fibre, which is the foundation of the WDM concept [Kart00].

While a SONET network manages digital signals by multiplexing in the time domain of a single optical carrier, WDM networks, as the name suggests, multiplex signals on different wavelengths in the same optical fibre. Therefore the fundamental unit of capacity in a basic WDM network model is an entire wavelength. There are two types of WDM in current use: *coarse* WDM (CWDM) and *dense* WDM (DWDM) [Kart00]. In CWDM only two to four lasers operate simultaneously in the low-loss windows of the optical fibre, while in DWDM the wavelengths are more numerous and packed more tightly. Frequencies, power levels, etc., must be much more exactly controlled in DWDM systems as a result.

With the potential to (theoretically) transmit over a thousand wavelengths on a single

fibre, we can see the huge potential transmission capacity offered by DWDM. However, DWDM networks come with their own challenges. It is very difficult to convert the huge quantities of data involved to the electronic domain and process it there, so we require new optical equipment to perform transport layer functions and keep transport signals in the optical layer as much as possible. However, this means that the packaged signals are much more difficult to access because optical technology is currently much less mature than electronic technology. This also means that other transport functions are more difficult, because optical operations tend to be more difficult than electronic ones (e.g., wavelength conversion vs. timeslot management). When we speak of transparent optical transport networks in the remainder of this thesis, we are generally referring to transparent DWDM networks.

2.2 Network Theory

This Section outlines some of the major terms and concepts used to discuss the theory of abstract networks.

2.2.1 Network Topology

We represent the topology of a transport network by its *network topology graph* (or simply *network graph* or *topology*) [Bhan99]. Terms related to the network topology graph are illustrated in Figure 2.1. This graph consists of *nodes* and *spans* (equivalent to *vertices* and *edges* in graph theory). A single span may be made up of a number of *links*. While the spans represent the connectivity of the network as established by the physical layer (cables and ducts), links represent the indivisible units of transmission capacity in the network. Taking an analogy from a different type of traffic engineering: if we think of the network spans as roads then the links are the individual lanes in each road. A link is therefore an abstract concept that will represent different transport containers in different networks. For example, the links in a traditional SONET network may represent the different timeslots available for STS-1 TDM signals, while links in a WDM optical network would represent entire wavelengths. When considering network planning and restoration, individual links are assigned to be either *working* or *spare* links. Working links carry network traffic during

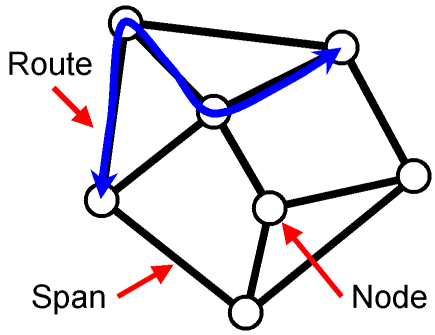


Figure 2.1: A network topology graph and some standard terms

normal network operation, while spare links are only used when they are required to carry rerouted traffic in response to the network entering a failed state.

The nodes in a network topology are an abstract representation of the sites connected by the network. In a transport network these are building sites containing the equipment to switch traffic in the network, and also to groom and add/drop traffic from the smaller metro area or local area networks that are attached to the backbone. Therefore it is important to note that the topology of the transport network alone cannot generally be used to derive the entire end-to-end path from user to user in the network (although each user can be thought to be residing somewhere within one of the nodes in the network).

2.2.2 Demands and Routing

The requirements on the network to carry traffic between nodes are called *demands* or *demand pairs* [Wu92]. In the transport network context, demand represents traffic aggregated from the network *edge* and packaged into higher granularity transport containers. Therefore demand magnitudes (*volume*) are usually given as integer values. A node in the network represents one of the *edge nodes* if it has a nonzero demand between itself and at least one other node, called as such because it sits on the edge of the transport network serving demand from one or more client networks. A node that exchanges zero demand with other nodes therefore represents a *core* node of the network that is used only for the switching of transiting demands. When the set of demand volumes between each pair of nodes is written out as a table, it forms the *demand matrix*. In general we assume that demands in the network are symmetrical (and therefore the *demand matrix* is also symmetrical), unless stated otherwise.

A demand between two nodes is served by being assigned a *route* through the topology from the origin to the destination. A *route* describes a concatenation of one or more network spans over which the demand is transported. The specific concatenation of links that the traffic travels over is called a *path*. A route is therefore only a topological concept, while a path contains also the specifics of which specific capacity "containers" (wavelengths/timeslots/etc.) will be used to carry the traffic. To extend our road analogy, a route describes the sequence of roads needed to travel from point A to point B, whereas a path also includes the details of every lane change taken over the entire trip. Routes and paths created for the service of demand pairs are referred to as *working routes* and *working paths* respectively. In contrast, routes and paths that are used to carry rerouted traffic over spare capacity due to a network failure are called *protection* or *backup* routes/paths.

In the optical transport network context, the network establishes these routes (optical circuits) between nodes to provide client networks with end-to-end connection services. Therefore the topology seen by client networks (e.g., the IP network) is not generally the same topology as the one seen by the transport network. The routes in the transport network effectively form the *virtual topology* used by the client networks. So while two IP routers may see a direct connection between each other in the IP layer, this connection may be equivalent to an optical circuit consisting of many hops in the transport layer. Therefore the degree of connectivity is generally greater in higher network layers because the number of logical circuits established at a node will generally be much larger than the number of direct connections offered to nodes that are directly adjacent in the lower-level topology graph.

2.2.3 Restorable Networks

In general terms, a restorable network is any network that contains some degree of protection from failures of major topology elements (nodes and spans). Of course, any network will contain a certain amount of redundant backup equipment to improve availability, but we usually reserve the term "restorable" for networks that use network intelligence together with topological diversity and routing redundancy to restore the network to a fully functioning state given a certain class of failure scenario (e.g., fibre cut or node failure).

Although we may, in practice, wish to design different networks with varying degrees

of restorability, in this thesis we will focus on networks that are able to completely recover from the failure of any single span. This is referred to as *single span failure restorability*, or simply *span* R_1 [Clou04]. We focus on span failures rather than node failures because of the assumption that span failures are much more likely. Spans represent the physical connective media of the network, i.e., optical fibre, which is fundamentally more vulnerable than the nodes in the network, which represent switching hardware that is kept in secure, disaster-protected buildings and is much more easily maintained, monitored, and guarded in comparison. We focus on single failures because the contribution of single failures to network unavailability is orders of magnitude greater than other failure scenarios that are much less likely (R_2, R_3 restorability, etc.) Therefore full single span failure restorability can be seen as the first order approximation of true "full restorability". Under full span R_1, second order effects (dual span failures or node failures, if considered to be significant) are what define the availability of the network [ClGr02].

To enable restorability we use spare capacity to reroute demands when failure events occur. There are real-world costs associated with the installation, monitoring, and maintenance of this capacity, so we prefer that the amount of spare capacity be kept low. A common efficiency metric for restorable network designs is *spare capacity redundancy* (or simply *redundancy*), defined as the ratio of the cost of the installed spare capacity S to the cost of the installed working capacity W.

$$\text{Redundancy} = \frac{S}{W} \tag{2.1}$$

The redundancy or the raw spare capacity outlay value S are often equated with the concept of network "cost", and in our experiments we usually accordingly attempt to solve for designs with low or (where possible) minimal redundancy. However, it should be recognized that the goal here is not the minimal cost result itself; after all, at this level of abstraction we cannot even be sure of the correspondence between the theoretical redundancy and the cost of a real network. Rather, we use this objective to inform our design methods such that we might generate new knowledge about the best way to design restorable networks: what works and what doesn't, what strategies are most effective, etc. "Low cost" is a guiding principle in our research and not an end in itself. Therefore it is important not to become caught up on incremental differences in design costs. It is instead more im-

portant to ask "what does this result tell us about the scheme(s) in question?" This is the philosophy used when interpreting the results of the experiments throughout this thesis.

2.2.4 Pre-Cross-Connection and Transparent Networks

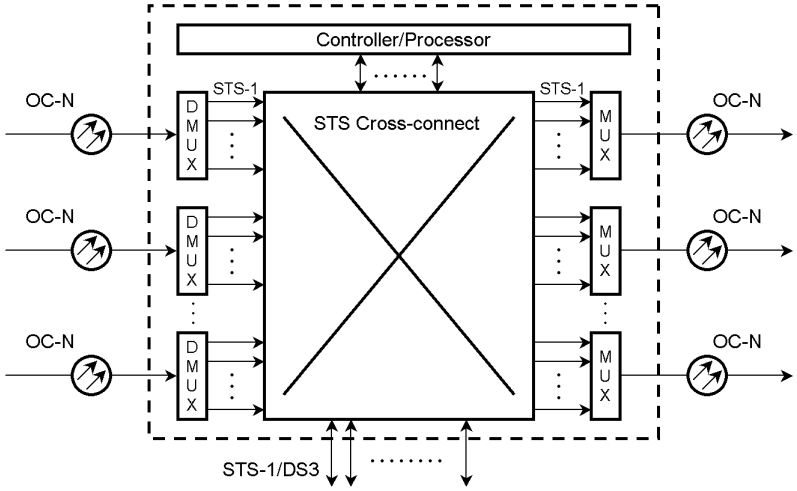
The focus of this thesis is the application of restorable network techniques to the specific case of transparent optical transport networks. In this Section we explain the concept of cross-connection, pre-cross-connected protection, and how this concept is relevant to the problem of restoration in transparent networks.

2.2.4.1 Cross-Connection

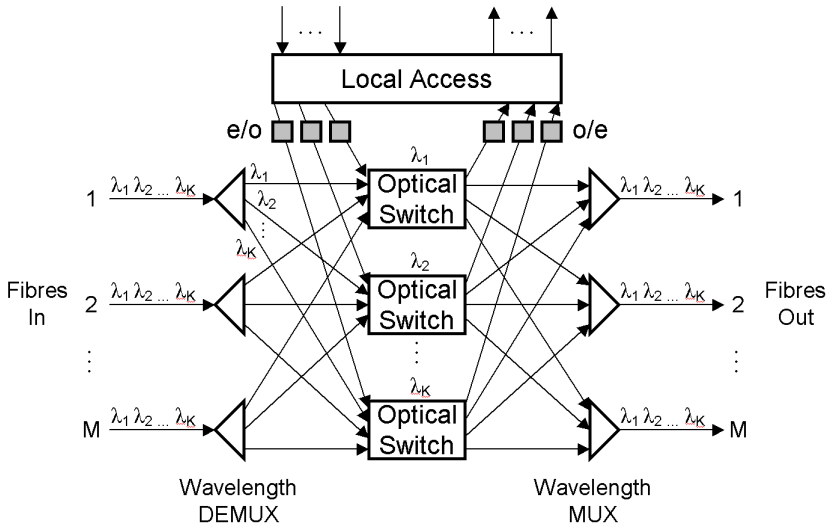
We have already explained how the working paths in a network consist of a concatenation of working links. Likewise, paths that are damaged due to failures are replaced, in whole or in part, by protection paths consisting of a concatenation of spare links. The concatenation process that is used to concatenate the links to form paths in transport networks is called *cross-connection*. The name of the device used to cross-connect the network depends on the type of network being considered. For example, a SONET (or SDH) network uses devices called *Digital Cross-Connect Systems* (DCS), whereas the counterpart for a WDM or DWDM network would be called an *Optical Cross-Connect* (OXC) [ToSc02]. These devices control the forwarding of the basic tributary signals of the network (i.e., STS-1 in SONET, or individual wavelengths in DWDM networks) by mapping connections between the input and output ports on the device. Block diagrams of a SONET DCS and an OXC are given in Figure 2.2 (adapted with permission from [Grov03]). Note that adding/dropping of tributary signals, although notionally a different concept, is in practice often implemented on the same devices that handle cross-connection as a generalized node will both switch transiting traffic as well as handle add/drop from the client network. The diagrams in Figure 2.2 (a) and (b) both contain provisions for add/drop functionality (STS-1/DS3 rate access on the SONET DCS and local access of electronic domain signals for the OXC).

2.2.4.2 Pre-Cross-Connection

To enable restoration in transport networks, the idea is to use the reconfigurability provided by the DCS or OXC hardware to perform switching at failure time to establish new paths



(a) SONET DCS



(b) OXC (no wavelength conversion or regeneration)

Figure 2.2: Functional block diagrams of a SONET DCS and an OXC

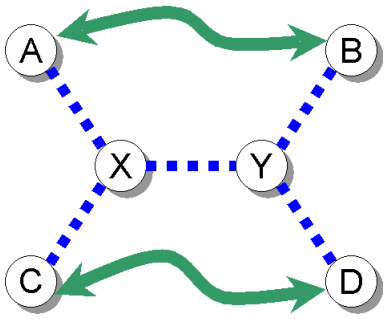


Figure 2.3: How capacity sharing of backup paths can require cross-connect formation at time of failure

that can be used to avoid the failure and prevent a service outage. In the general case, we might imagine a complete reconfiguration of the network at every node in order to optimally reorganize the network for the most efficient use of spare capacity. In reality, however, this is not desirable due to a number of practical considerations.

Forming cross-connections in protection paths at failure time is a fundamental requirement of many protection architectures (though none take it to such an extreme as in our "complete reconfiguration" example). Take as an example a situation that occurs in *Shared Backup Path Protection* (SBPP), shown in Figure 2.3. The solid lines represent working paths and the dashed lines are single units of spare capacity. For now we will not concern ourselves with the details of this architecture; it will be discussed in more detail later. It is enough to know that, in this situation, path AB can be protected by path AXYB through the spare links. Likewise, CD may be protected by CXYD. However, note that depending on whether AB or CD fails, different links become concatenated with link XY to form the protection path. Therefore, even though the protection paths are known ahead of time, the actual cross-connects required to form these paths cannot be preestablished at XY in such a way that they handle both failure scenarios; they must be established at failure time.

The property of *pre-cross-connected* (or *preconnected*) protection was originally proposed as a measure to increase protection switching speed by minimizing the amount of post-failure switching [GrMa94]; if we can configure the cross-connects in the network in such a way as to minimize the average number of changes to this arrangement that must be made over all failure scenarios, failure response speed will increase in the average case. Even now, most authors refer to full pre-cross-connection of protection paths mostly as

a means to decrease switching times [ChCh04, LiHa06, HeSo07]. However, we will see that complete preconnection of protection capacity is also highly relevant when considering the problem of failure recovery in transparent networks. In a completely preconnected restorable network, all cross-connections used in both failure and non-failure states must be established in advance of failure time (except at the source and destination nodes, because there will always have to be some switching here to move between the working and backup path). This may seem to be a very restrictive constraint, but it is in fact a property of many common protection architectures, even some that have been shown to be quite efficient.

2.2.4.3 Transparent Optical Networks

In terms of transparency, optical networks can be classified as fully *transparent*, *translucent*, or *opaque* [Mukh06]. A fully transparent optical network is any network in which a payload may be modulated onto a wavelength carrier and transported to its destination regardless of the formatting of the payload in terms of framing, bit-rate, line-coding, etc. In practice, today, this means that the optical circuit on which a payload is transported is all-optical, i.e., it does not contain any optical to electronic conversions en route which would place requirements on the digital format and bit rate of the payload. In current networks, this also implies that each payload must be carried along an all-optical path that does not change wavelength (i.e., the same WDM wavelength from source to destination), because a viable all-optical wavelength conversion technology is not yet available. This is the type of network that we focus on in this thesis. This explains why we do not consider SONET networks; they consist fundamentally of electronic SONET level signals that are multiplexed/demultiplexed and processed in the electronic domain and then modulated onto optical carriers for transmission by SONET optical equipment. In other words, a SONET network fundamentally standardizes the electronic signal format and performs intermediate electronic-domain signal processing for add/drop and multiplexing, and therefore cannot be transparent.

The other extreme is an optically opaque network in which optical to electronic conversion occurs on every path at every cross-connecting node. From a routing standpoint, such a network provides the greatest flexibility. There is no concern about wavelength matching, and cross-connections can be made at any time without concern about viability of the end-

to-end path because electronic regeneration occurs at the input to every span. In effect, no optical carrier signal travels more than one span in such a network. The term translucent refers to optical networks with a mix of pure-optical and optoelectronic cross-connection functions. A typical translucent architecture involves a number of fully transparent domains that interconnected via o-e-o cross-connecting hubs that provide regeneration, wavelength conversion and signal cross-connection functions. The network is transparent to optical paths that are contained within a single domain, but paths that cross domain boundaries must again conform to the network's signal formatting standards.

Although the simplicity of management and routing is greater in an opaque network, optically transparent cross-connects reportedly cost very much less than o-e-o cross-connection functions. Along the same lines, the complexity and power utilization of optical switching equipment per bit is significantly lower than for corresponding electronic devices. Also, transparency is functionally advantageous to avoid the necessity of electronically adapting the data to match network-specific transmission payload protocols and bit rates. For these reasons there is considerable motivation to consider protection architectures specifically suited to entirely transparent optical networks or within the transparent domains of translucent networks.

From the point of view of survivability, protection paths are not as simply formed in a transparent network as in an opaque network. Any all-optical multi-hop path must be carefully engineered ahead of time to conform to restrictions on optical carrier power, dispersion, noise, attenuation, etc. Preconnecting the protection path prior to failure enables preengineering and testing of the backup path prior to its use. This guarantees that the backup will have sufficient optical path integrity and will be in a known-working condition when required for protection switching. Without this pre-failure guarantee, it is not realistic to expect that a set of optical channels concatenated on the fly will instantly result in a functional (e.g., BER $< 10^{-12}$ say) end-to-end optical path, considering the numerous impairments that must be simultaneously mastered in the engineering of this optical path. Currently, engineering just single-hop lightwave channels in a dense WDM carrier environment at 10 or 40 Gbps rates with multiple shared optical amplifiers in the path (let alone end-to-end lightpaths of 5 or 6 hops) is a largely manual process that involves careful alignment of individual components. Freeman [Free02] (Ch. 6 and 10) describes the nu-

merous impairments that must be compensated for in order to design a single point-to-point optical fibre link operating at transport data rates of 10 or 40 Gbps. At the very least, power level differences from switching optical carriers through multichannel optical amplifiers creates disturbances that take time for adaptive power level schemes to compensate. So any scheme that assumes on-the-fly changes in the configuration of the network's optical cross-connections at the time of failure will suffer from some uncertainty about the optical path engineering of the backup path.

Because cross-connection of protection paths at failure time in a transparent network is not just slow (as it would be in an opaque network), but nearly impossible, we see that preconnection is practically a requirement of this type of network. This is the motivation behind investigating preconnected network protection architectures in this thesis.

2.2.5 Established Methods for Protection and Restoration

Many different strategies for network restoration have been developed in order to face the growing need for reliable transport networks. This Section contains an overview of some of these approaches, proceeding roughly in the order of their discovery, with particular attention paid to whether or not they are preconnected and their suitability for transparent networks.

2.2.5.1 Automatic Protection Switching

The simplest form of protection, *Automatic Protection Switching* (APS) involves maintaining dedicated, disjoint backup paths for each working path in the network [Bhan99]. In the event of the failure of some network element, the end nodes of any affected working paths detect the failure and switch their transmission onto the pre-determined backup path. This method can be used to protect against node failures as well as span failures if the protection paths are made node-disjoint from their corresponding backup paths (although obviously demands with the failed node as the source or destination cannot be restored from within the transport layer).

This approach certainly has the advantage of simplicity as the failure response requires no knowledge about the nature of the failure or its location. The only intelligence required is that nodes are able to detect failures in transmission and coordinate the switch onto the

pre-defined backup path. Otherwise there is no communication between different sets of node pairs and the possibility of spare capacity contention is nonexistent. The trade-off for this simplicity is that this approach is very costly in terms of the amount of spare capacity used. Because each working path requires its own dedicated backup path, the APS scheme is generally at least 100% redundant.

Because backup paths are dedicated, it is also possible to transmit the signal payload along both the working and backup paths simultaneously. This type of arrangement is called 1+1 APS. Under this configuration, neither path can really be identified as primary or secondary; the receiver simply selects continuously between the signal of higher quality. This allows switching to be near-instantaneous, as the "backup" path is already active the instant a failure occurs. The alternative is 1:1 APS, where the backup path is not active until the failure occurs [Bhan99]. A related approach is 1:N protection, in which N working channels share a single backup path. As long as the working paths are all disjoint from each other, 100% restorability is assured in all single span failure scenarios.

APS is preconnected, and therefore suitable for transparent networks. In fact, this scheme is common in modern optical networks even though it is capacity inefficient, due to its simplicity.

2.2.5.2 Demand-Wise Shared Protection

Demand-Wise Shared Protection (DSP), while very similar to APS, was only proposed very recently. It was introduced in [KoZy03, KoZy05] as a method to "combine the advantages of both dedicated (APS) and shared (SBPP) protection" [KoZy05]. It is functionally similar to M:N APS, in that working capacity is diversified across a set of M topologically disjoint working routes that then are able to share N protection channels laid down on a single protection route. This path is therefore dedicated to the protection of that particular set of working routes, but shared between them (hence "demand-wise shared"). DSP requires a highly connected topology to achieve significant savings over APS because it becomes more efficient as it is able to diversify the working capacity over more disjoint paths (increasing the number of paths that share protection). If it is impossible or inefficient to discover more than 2 disjoint paths between the end nodes, DSP essentially reduces to 1:1 APS.

2.2.5.3 Unidirectional Path-Switched Rings

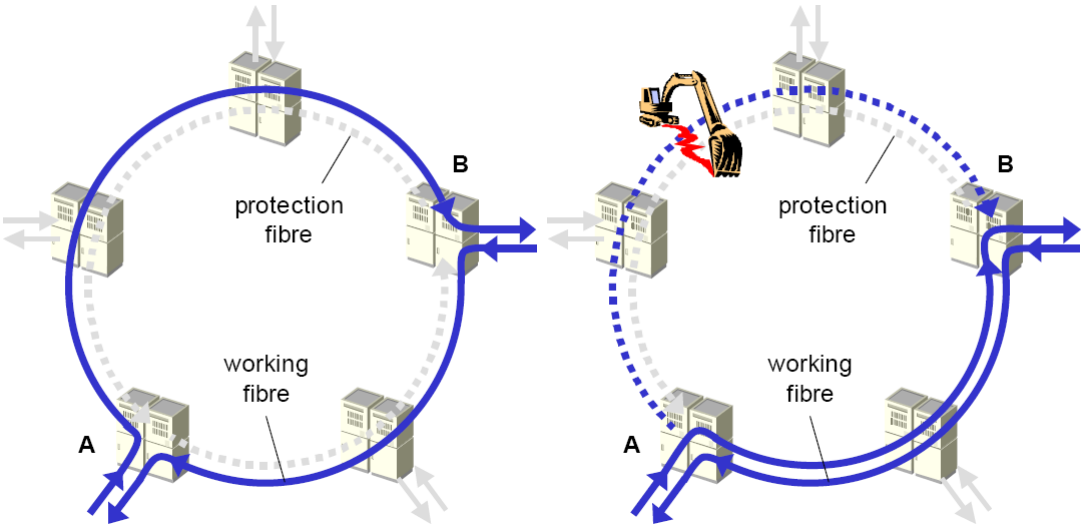
Unidirectional Path-Switched Rings (UPSR) are an evolution of APS systems that attempt to integrate the fast switching and simplicity of APS into a more structured approach to protection [Grov03]. In a UPSR network, spare and working capacity is arranged in cycles, called *rings*. A single ring consists of a cycle of working capacity coupled with a cycle of an equal amount of spare capacity. The working capacity of the ring is used to route demands between the nodes on the ring during normal network operation. In the event of a failure on the ring, the cycle of spare capacity is used to reroute the demand around the failure. UPSR rings can be used to protect entire paths end-to-end if both end nodes lie on the ring, but paths can also traverse multiple rings (using "matched node" arrangements to prevent single points of failure at the transition nodes), with each ring protecting its own segment of the path.

Figure 2.4 shows the restoration action of a UPSR. Note that before failure, the bidirectional traffic exchanged between A and B is split such that the two directions take routes around opposite sides of the ring, both in a clockwise direction. In the event of a failure on one of these paths, the traffic is switched onto the protection fibre in the counter-clockwise direction such that the new route avoids the damaged section of the ring. The technical term "UPSR" is, strictly speaking, specific to SONET and its standards for self-healing ring topologies, but the concept illustrated in Figure 2.4 can be equally applied to a DWDM network if we simply assume that each fibre transports a set of wavelengths from the DWDM grid, as opposed to a single optical carrier modulated with a SONET signal. The WDM equivalent of UPSR is the *Optical Path Protection Ring* (OPPR) [Maed98].

UPSR rings retain most of the benefits and drawbacks of APS systems. They are pre-connected, and switching is still fast and simple, but the necessity of coupling the working and protection rings means that rings are again 100% redundant at best.

2.2.5.4 Bidirectional Line-Switched Rings

Bidirectional Line-Switched Rings (BLSR) are a self-healing ring topology similar to UPSR, except that they use a slightly different protection mechanism that allows the spare capacity on the ring to be shared between failures [Grov03]. As the name suggests, BLSR rings are line-switched (i.e., span-switched, using our terminology), meaning that rerouting is done



(a) UPSR before failure (b) UPSR after failure

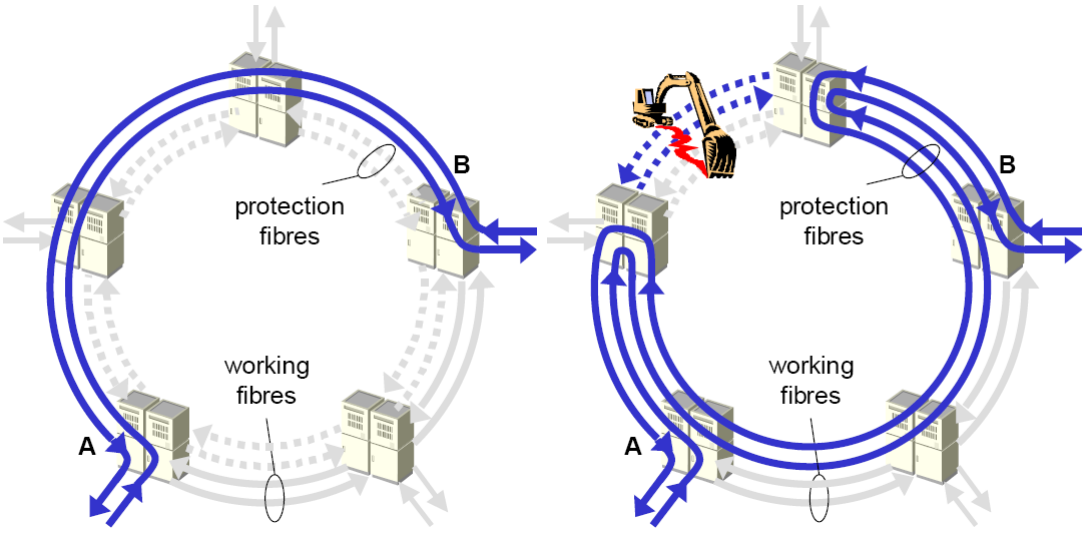
Figure 2.4: Operation of a UPSR (taken with permission from [Douc04])

between the end-nodes of the failed span, not the entry and exit points of the failed path on the ring as with UPSR.

Figure 2.5 contains a diagram of the operation of a BLSR. Unlike a UPSR, in a BLSR the two opposing traffic directions are routed along the same side of the cycle. When a span fails, the end-nodes of that span perform a loopback into the protection fibres, restoring the failed path. Because a BLSR uses line switching, protection involves a certain degree of *backhaul*, where the signal must travel to an end-node of the failed span before it reverses direction to be transported along the protection fibres on the opposite side of the cycle. BLSR, like UPSR, has fast switching and is also preconnected, although BLSR is able to achieve smaller ring sizes and be more capacity-efficient due to the sharing of the protection fibres. The WDM equivalent of BLSR is the *Optical Shared Protection Ring* (OSPR) [Maed98].

2.2.5.5 Shared Backup Path Protection

Shared Backup Path Protection (SBPP) is one of a set of techniques known as mesh-based techniques, called so because they take advantage of the highly interconnected, mesh-like nature that most real networks exhibit to achieve efficient restoration, instead of forcing working and protection routing into artificial structures like rings. SBPP is presented



(a) BLSR before failure (b) BLSR after failure

Figure 2.5: Operation of a BLSR (taken with permission from [Douc04])

amongst a survey of network restoration techniques in [RaMu99]. SBPP is similar to APS in that each working path is assigned a single predetermined backup path. However, it differs in that the backup paths are not dedicated to their respective working paths. Instead, the spare capacity that makes up the backup paths can be shared for the protection of multiple working paths. This is possible if the working paths that share protection capacity are disjoint from each other. If this is the case, then no single failure can affect any one of the working paths at the same time, and thus it is ensured that no two working paths will ever contend for the same spare capacity.

When multiple failures occur, outage can be produced either due to the simultaneous failure of both a working and backup path (true of any restoration scheme), or because of the failure of multiple working paths that share the same spare capacity. Therefore availability using SBPP will generally be lower than for APS. However, this is made up for by a large spare capacity savings due to spare capacity sharing.

We have seen an illustration of two working paths sharing portions of their backup paths under SBPP in Figure 2.3. SBPP is not preconnected, and therefore not suitable for transparent networks, as we have discussed.

2.2.5.6 Span Restoration/Protection

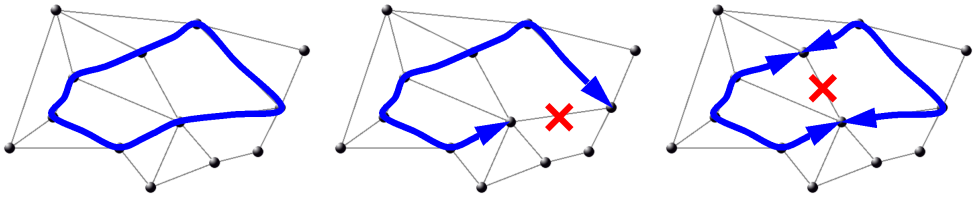
Span restoration (or *span protection*) is another mesh-based approach to survivability. Distributed automatic span restoration was first proposed in [Grov87, Grov89]. There is actually an important conceptual difference between the terms "protection" and "restoration" here, but we will treat them as the same for the sake of our simple explanation of the basic survivability mechanism.

Unlike SBPP, span protection acts locally, protecting the network's working capacity on a span-by-span basis (as with BLSR) instead of protecting whole working paths end-to-end. It works by designating a fixed protection path for each working link on each span in the network. This designation is *pre-planned*, i.e., it is predetermined and not calculated as a real-time response to span failures. Then, if a span fails, the end-nodes of the span switch transmission from the failed working links to these predetermined backups. Backup paths are also allowed to share spare capacity. Therefore span protection can be thought of as a span-protecting version of SBPP. In both SBPP and span protection, the spare capacity in the network can be considered as a single "pool" of capacity that is available for the formation of pre-planned protection paths at failure time.

Span protection, since it allows sharing of capacity between failure scenarios, is a very efficient scheme. However, it suffers the same drawback as SBPP: even though protection paths are pre-planned, they cannot in general be established by cross-connect switching actions until the time of failure (unless special design considerations are made with this constraint in mind). Therefore generalized span protection it is also not suitable for transparent networks. However, there is some degree of ambiguity of terminology here as "span protection" has also over time come to refer to any protection scheme that protects on a span-by-span basis. Some of these schemes (such as *p*-cycles) are preconnected and thus are more suitable for transparent networks. The above description of span protection refers to the most general case in which backup paths for each span are not chosen with any particular regard towards preconnection.

2.2.5.7 *p*-Cycles

The p-*cycle* concept arose from a desire to combine the fast switching capabilities of rings with the high efficiencies of mesh-based restoration. The concept was first brought to



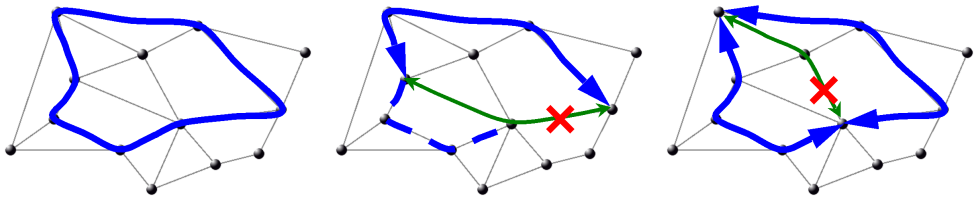
(a) A *p*-cycle structure (b) Failure of an on-cycle span (c) Failure of a straddling span

Figure 2.6: A *p*-cycle and its operation

light in the seminal paper [GrSt98], which demonstrated that such a "best of both worlds" approach was possible. Since then, *p*-cycles have attracted a great deal of attention, and are currently very well-studied [ScGr02, StGr00, ZhYa02, LiWa06, LiRu04].

A *p*-cycle is a cycle of capacity, similar to a UPSR or BLSR ring, except that a *p*-cycle is a spare capacity structure only, and does not need to be co-routed with the paths it protects. A *p*-cycle "sits above" the working layer and simply provides protection relationships to the working capacity on the spans that it is able to protect. The protection capabilities of a *p*-cycle are illustrated in Figure 2.6. When there is a failure on one of the spans that is on the *p*-cycle itself, it is able to perform a loopback operation to reroute the failed span around the rest of the *p*-cycle (similar to BLSR). However, if there is a failure on a span that is not on the *p*-cycle but has both end-nodes on the *p*-cycle (a *straddling span* or *straddler*), the end-nodes of the failed span can "break in" to the cycle to provide up to *two* protection paths around the two halves of the cycle bisected by the failed span. In general, a restorable network is protected by a whole set of cycles that is together able to provide at least one unit of protection to every unit of working capacity on every span.

It has been shown that *p*-cycles are a very efficient protection architecture, able in some cases to closely approach the theoretical redundancy lower bound for span protection [Grov94]. This is thought to be mainly due to the 2:1 protection relationship offered to straddling spans. At the same time, we can see that the *p*-cycle structures themselves have the preconnection property because we can form any of the available protection paths for the protected spans simply by performing loopback or break-in, where appropriate. Therefore *p*-cycles seem to be an interesting choice for transparent networks. There is one caveat, however. Because *p*-cycles are span-protecting structures, a failure effectively "breaks" the working path at the end-nodes of the failed span, where a substitute protection path from



(a) A FIPP p-cycle structure (b) Failure of an on-cycle path (c) Failure of a straddling path

Figure 2.7: A FIPP p-cycle and its operation

the p-cycle is reinserted. Therefore we still have the problem of the optical integrity of the restored-state path being unknown before the failure occurs. This is surely better than, say, span protection, where we do not even establish the protection paths until failure time, but still poses a problem for transparent networks. This is true of any span-protecting preconnected structure, including others studied in this thesis.

2.2.5.8 FIPP p-Cycles

Failure Independent Path-Protecting (FIPP) p-cycles are essentially a path-protecting variant of the p-cycle concept [KoGr05]. It is not the only way to marry path protection with p-cycles, but it is the most direct and the most relevant to the other architectures studied in this thesis, so we will describe it briefly here. Like p-cycles, FIPP p-cycles can provide both on-cycle and straddling protection, except that FIPP p-cycles protect paths end-to-end instead of individual spans. A demonstration of FIPP p-cycle protection is given in Figure 2.7. Similar to p-cycles, when path that is (partially or fully) on the cycle fails, the cycle can provide one protection path; when a straddling path fails, two protection paths can be provided. FIPP p-cycles are preconnected, like p-cycles, with the added advantage that protection switching does not break up the existing working path because it is replaced wholesale with the preconnected protection path. We discuss FIPP p-cycles in greater detail later in Section 2.4 as part of the literature survey of established pre-cross-connected protection architectures.

2.3 Restorable Network Design Methods

Up to this point we have discussed the basic concepts behind network protection/restoration, as well as several approaches for achieving high network availability. In this Section we

discuss the main experimental and theoretical methods that are used to study these protection approaches, paying particular mind to the methods that are used in this thesis. Later Chapters will refer occasionally back to this Section for more complete explanations of certain techniques.

2.3.1 Goals and Assumptions of Network Design

Before undertaking comparative studies of these and other protection architectures, we must first define the parameters and objectives of such studies. For example, as mentioned in Section 2.2.3, it is often assumed that a "restorable network design" is a design in which there is sufficient protection to provide 100% recovery of damaged flow in the case of any single span failure. While other design objectives are also used frequently in the literature, the 100% span R_1 assumption is usually assumed unless stated otherwise, as it is in this thesis. Not all 100% span R_1 designs will be exactly equivalent in terms of *availability* (which will be determined by susceptibility to dual failures), but 100% span R_1 is generally assumed to provide a minimum standard of availability in most cases.

Although the network design approach used in this thesis is appropriate for the theoretical nature of our comparisons between different architectures, it is also very different to how one would expect the planning of real networks to be done in the field. The approach taken for most of this thesis is the *green fields* approach, which refers to network design "from the ground up", assuming zero installed network infrastructure (except the right-of-way agreements and building locations that define the network topology) and a fixed demand matrix that represents exactly known quantities with zero uncertainty. This is opposed to real network planning, which is almost always an incremental approach that considers how to best serve incremental demand growth while at the same time leaving options open for predicted future growth, taking into account a certain degree of uncertainty in the predictions. We use the green fields approach most often in this thesis because it enables a pure comparison of ideal designs, allowing us to more easily obtain new knowledge about properties of the architectures we study. However it is important to remember that achieving this ideal in real networks is not realistically possible.

2.3.2 Evaluation Criteria

Given the constraints on the design problem (e.g., 100% span R_1) and a given design method, the one remaining question is how we are to evaluate the quality of the design. This comes into play especially when performing comparative studies, when we wish to know how various architectures, that are otherwise identical in terms of restorability, compare in terms of other metrics. Again, there are many possible choices, but by far the most common is to evaluate them in terms of a cost metric. Because true system cost is a highly complex value to calculate (besides also being very system-specific and therefore not suited to more general studies), we generally use a simplified cost surrogate. The most common cost metric is simply the amount of capacity used by the design. A design that uses less capacity to achieve the same 100% R_1 target is more efficient and therefore more desirable.

There are a number of variants of this metric; depending on the study, we might use either spare capacity or total capacity. Redundancy metrics (e.g., Equation 2.1) are also used. When measuring amounts of capacity, we generally either measure absolute capacity units (e.g., total wavelength channels in a WDM network), or a total distance-channel count (e.g., channel-km for WDM). The prior case treats the cost of any unit of capacity on a span the same, regardless of how long the span is, whereas the second case gives proportional weight to capacity used on longer spans. Which model is used will generally depend on the type of network being considered and the economics of the span costs involved. In long-haul networks, span lengths can be great and cause distance-related costs (e.g., shielding/regeneration/etc.) to dominate (meriting the distance-channel model), whereas in metro scale networks span costs tend to be dominated by a constant factor per-span (suggesting the absolute channel count).

The cost of the design is often not the only important property, however. Secondary evaluation criteria may include availability (or span R_2, which is closely related), length of paths (working or protection), or metrics related to the complexity of the design. In fact, we will often investigate structural complexity of the 100% span R_1, low capacity designs that we generate throughout the architecture studies in this thesis. Structural complexity is different from these other criteria in that it is not easily quantified into metrics. We will often use various metrics to support discussion of complexity, but this discussion will always be coloured by the context in which complexity is important. For the most part,

we will be discussing either complexity concerns related to operational complexity (that is, operational issues that need to be understood by a human designer that would be hindered by a high degree of complexity in the protection layer), or complexity concerns that have implementation issues for transparent optical networks specifically, as per the focus of this thesis.

2.3.3 Integer Linear Programming Methods for Network Design

Linear programming (LP) [Dant63] provides a mathematical framework in which we may implement the constraints and objectives for network design that we have expressed above in qualitative terms. It provides a simple way of framing and solving a variety of optimization problems, including network design problems for the above-mentioned architectures and others. The resulting mathematical "program" is given to an optimization engine that then processes the problem until obtaining an optimal solution (or one within a predefined margin of optimality). The assurance of the optimality of the results allows us to perform quantitative comparisons of different architectures with confidence. *Integer linear programming* (ILP), a subset of linear programming in which all quantities to be computed are integers, is the primary method used for obtaining the results and performing the analysis in the rest of this thesis.

2.3.3.1 Overview

Mathematical programming refers to a specific way of expressing an optimization problem mathematically. Under this framework, the problem is expressed in terms of a series of constraints (expressed as inequalities or equalities) on a set of *variables*, as well as a single objective function expressed in terms of those same variables that must be minimized (or maximized). Linear programming problems are mathematical programming problems in which both the constraints and objective function are linear expressions. Common and efficient algorithms exist to solve linear programming problems, and there are many commercial tools that can parse such problems from standardized notation and solve them automatically. The linear program itself is simply a standard form that presents the problem in a form that the solver can understand. Once the solver has found an optimal set of values for the problem variables, it is up to the user to assign significance to the values in terms of

the original problem.

Take as an example a classical diet problem. As the problem goes, a person is on a special diet, during which they must take in a certain minimal amount of each of N types of nutrients. There are M foods, each unit of food type i containing an amount n_{ij} of nutrient j. Food i has a cost of c_i per unit. The problem is to minimize the total cost of the diet while still meeting the nutrient requirements. One can see immediately that the nutrient requirements will need to be expressed as constraints, while the cost minimization goal will be represented in the objective function. The variables will be the amount of each type of food that is consumed. For this problem, the nutrient requirements can be expressed as:

$$\sum_{i=1}^{M} n_{ij} \cdot x_i \geq R_j \qquad j = 1..N \tag{2.2}$$

Note that this expression represents not just a single constraint (inequality), but a set of N constraints (one for each nutrient). This constraint set conveys the requirement that, for each nutrient, the amount of that nutrient obtained from all foods combined must exceed R_j (which we use to represent the minimum requirement). Note that we have used x_i to represent the amount of food i that is consumed during the diet. Then all that is left is to specify the quantity to minimize:

$$\text{Minimize} \quad \sum_{i=1}^{M} c_i \cdot x_i \tag{2.3}$$

This minimizes the total cost of all foods consumed. We can see that this mathematical program is linear because all expressions are a sum of terms that consist of a constant multiplied by a variable (one of the x_i values). Note that even though the n_{ij} and c_i values are represented by letter symbols, and thus are "variables" in the mathematical sense, they are constants in the context of an instance of this mathematical program. In other words, they are part of the input to the problem and are not decided by the solver (unlike the values x_i which are true variables in the linear programming sense). We refer to terms such as these as *parameters* because they are varied only outside the program by the user in order to change the details of the particular problem instance under study.

An integer linear programming problem is simply a linear programming problem with the additional constraint that all of the variables involved are constrained to only take on

integer values instead of continuous real values. For example, if the previous diet problem was modified to consider a schedule of pills that had to be taken whole, the number of units of each food would then cease to be continuous and become a strictly integer quantity. Even though this integrality constraint is itself not linear, fast algorithms also exist (based on traditional linear programming algorithms) to solve integer linear programming problems. In theory, the solution of ILP problems is an *NP-complete* activity, meaning that there is currently no known polynomial time solution algorithm [GaJo79]. In practice, however, problems of the size encountered in the field of real-world network design can be solved quickly using these methods, making them a useful addition to the toolset of a network designer.

2.3.3.2 Application to Network Design

We have seen how a simple example problem can be expressed using (integer) linear programming notation. However, over time, the field of network design has accrued its own customs and notations for network-specific ILP problems. These customs are also used in this thesis and thus bear some explanation here.

In addition to variables, parameters, constraints and an objective function, network-related problems often make use of *sets* of objects. These objects do not have any direct representation in the mathematical expressions of the problem but represent higher level entities in the design problem that have various parameters and variables as attributes. For example, in the diet problem mentioned above we might define the set **F** of foods (which has a cardinality of M). Then we would say that x_i and c_i are the amount of food i eaten and the cost of food i, respectively, for each i in **F** (instead of for i from 1 to M). Converting integer indices to actual sets of objects in this way can often make the representation of the problem more clear, especially for large problems.

In network problems, sets are generally used for each of the major types of network element. For example, a set of spans is usually defined. Then variables for the amounts of spare and working capacity on each of these spans can be defined based on these sets. In addition, sets of working paths and protection paths are also sometimes defined, depending on the architecture in question, that can then have variables for working/restoration flow quantities assigned to them. The use of these sets actually represents a significant advance

in the use of ILPs for network design. Initially, work in this area tended to use the so-called arc-flow approach, in which working or protection flow was assigned on a per-span basis, with various constraints used so that the conservation of flow from one node to another along a route was maintained. Eventually, however, Herzberg et al. [Herz94] proposed that these routes be enumerated explicitly in the problem statement, instead of arising from the solution process itself. While this approach may seem unwieldy in that the set of possible routes between each node pair may become quite large, it has the advantage of allowing the user to limit the set of routes according to his preference. This allows, for example, elimination of routes longer than some predefined hop limit, which is very difficult using arc-flow approaches.

Finally, the approaches such as *p*-cycles that use the concept of "protection structures" will have a set of such structures, with their costs and protection capabilities defined by parameters. As with the Herzberg route-enumeration approach, this allows the designer to groom the set of allowed structures according to his preference and the capabilities of the network.

2.3.3.3 Tools

The linear programming solver used in the following thesis work is a commercial program called CPLEX [ILOG09], from ILOG. The software used to interpret the symbolic linear programming language into problem tableaus solvable by CPLEX is called AMPL [FoGa03]. For this thesis, CPLEX and AMPL software versions 9 and 10 were both used (the precise version number used is not relevant to the quality of the solutions, but different versions may have optimization differences that affect solution times). Problems were solved on one of two machines: either on a desktop PC running Windows 2000 with a 2.8GHz Pentium 4 processor and 1 GB of RAM, or, when large amounts of memory or CPU power were required, a shared server running SunOS 5.8 with four 900 MHz SPARC-V9 processors and 16 GB of RAM. These specifications are not necessarily enough to exactly recreate the conditions under which our ILP problems were run, but rather are given here to provide a reference for the amount of computing power required to solve these problems to the degrees of accuracy and within the timeframes mentioned in the following pages.

2.3.3.4 Terminology

In discussion of the results of the ILP models in this thesis, we will sometimes refer to the *mipgap* of the solution. The mipgap (gap from optimality of the mixed integer program) is a parameter that controls how close to optimality the solution must be before the solver terminates. The solver is able to make use of information gained in the branch-and-bound ILP optimization algorithm to place a bound on the cost of the optimal solution, and will terminate when the cost of the best discovered solution approaches this bound to within the mipgap.

Ideally we want truly optimal solutions, but often it can be productive to set the mipgap to a very small but nonzero value to prevent the solver from examining countless nearly-identical solutions to find the "true" optimum. This gives us solutions that are "close enough" to true optimality in potentially much less time. In the experiments that follow, unless a specific mipgap value is mentioned, it should be assumed that the default mipgap value of 0.0001 was used. A mipgap value of 0.0001 implies that the solution is guaranteed to be within 0.01% of optimal.

2.3.4 Common Techniques

There are other techniques aside from ILP that are used in the field of restorable network research that have become widely adopted due to their usefulness, generality, or other factors. This Section provides an overview of any such techniques that are used in this thesis.

2.3.4.1 Network Families

One of the major challenges of studying the characteristics of protection in mesh networks is that we can only rarely make definitive, generalized statements on the properties of certain architectures or algorithms over the set of all possible networks. More commonly, we need to perform a number of experiments under a wide variety of conditions to produce a general picture of the property under investigation. This raises questions regarding which test cases are appropriate for which experiments, whether or not we have considered networks with a wide enough variety of properties in each case, etc.

One technique that has been introduced to address this problem is the practice of generating network designs across entire "families" of networks. A network family is based

on a single "master network" topology, which is the most highly connected member of the family. In this thesis, all our master networks are of average nodal degree $\overline{d} = 4$. We then proceed by removing one span from this network at random to obtain the next network in the family. This random removal is controlled so that the new network retains the property of *bi-connection* (for every pair of nodes we can find at least two span-disjoint routes between them). This process continues, removing one span at a time until we can no longer remove a span without violating the bi-connection property. All of the networks generated in this way make up the network family.

Performing tests on a set of networks from a network family as opposed to a set of randomly generated networks has the advantage of being able to slowly vary the connectivity of a fixed set of nodes [ToDo08]. In a set of randomly generated networks, topologies of degree 3.0 and 3.01 may differ wildly from each other, for example, but we know that "adjacent" networks in a family will vary by only a single span. Also, because the node pattern remains the same, we can keep the same demand pattern over all members of the family in a meaningful way (we can keep the same demand pattern over any set of random networks with the same number of nodes, but its meaning will be totally altered if the node positions and connectivity are completely changed). This method has some pitfalls too, of course: by limiting ourselves to a family based on a single network, we may be blinding ourselves to effects that will only occur or behave differently on a network with a different arrangement of nodes and overall connectivity. To address this issue, we can solve our design problems over a number of different network families.

2.3.4.2 Enumeration and Limitation of Candidate Structure Sets for ILP

The flexibility of the mathematical language used to express ILP problems gives us many ways to represent the same network protection problem. In general, however, we tend to use an approach in which much of the problem data is pre-generated in an initial data preparation (occasionally "datprep" for short) stage. This data is then loaded into the AMPL program to fill in the values for the parameters of the problem. The most important function of this step is to generate the set of *candidate structures* that the problem is allowed to consider as protection options for the network. For example, in a *p*-cycle network, the set of candidate structures would be a set of cycles. It is also possible to represent the *p*-cycle

problem without explicit cycle enumeration [WuYe07a, WuYe07b, WuYe08], but we use explicit generation because it is both less complex and more flexible. It is less complex in that, even though a large set of data needs to be generated, the generation of cycles can be done with a straightforward search algorithm. The explicit enumeration approach is also more flexible because it allows us to control precisely which structures are included in the set, making it simple to control the characteristics of the structures used in the design. For example, if the length of protection paths is of particular concern, we can restrict the candidate set to only structures below a certain size.

Throughout this thesis there will be many cases where explicit enumeration of the entire space of structures for a particular problem is infeasible because of the large size of this space. In these cases, we use this method to restrict the size of the candidate set out of necessity, so that we can obtain results in a reasonable amount of time. The criteria used to limit the candidate set will be mentioned where applicable, but otherwise it should be assumed that we use the entire set of structures available in the network.

2.4 General Literature Survey

This Section contains an overview of the literature that is directly relevant to work in this thesis. Literature that pertains more specifically to the topics of the subsequent Chapters will be introduced and/or discussed in more detail at the beginning of those Chapters.

2.4.1 Preconnection to Enhance Restoration Speed of Span-Protected Networks

Before the emphasis on preconnection for transparent networks, preconnection was generally seen as the means to increase the speed of protection actions only. For quite a while, the field of network protection was plagued with the false dichotomy that one could either have fast protection in a very inefficient design (generally using 1+1 APS or SONET rings), or designs with high efficiency that were slower to respond to failure (such as span restoration). The search for improved restoration speed under the capacity efficiency of mesh-based protection led Grover and MacGregor [GrMa94] to investigate the possibility of optimizing the cross-connections in preexisting mesh-based restorable designs so as to minimize the number of changes to the cross-connect state of the network over all possible

span failures. This work outlines a method to establish a partially preconnected design that represents a sort of middle-ground between totally dynamic schemes (such as span restoration) and schemes with static cross-connections (such as a ring-based network). This line of work eventually lead to the discovery of *p*-cycles [GrSt98].

2.4.2 Novel Preconnected Schemes

Many of the well-established protection architectures we covered in Section 2.2.5 qualify as preconnected, even though they may not have been recognized as such at the time of their discovery. Table 2.2 gives a summary of these schemes, classified by protection mechanism and structure type. Since the realization of the importance of preconnection in transparent networks has become more widespread, there has been a small explosion of research interest in different types of preconnected protection architectures. Recent work has explored many variations on the concept of a preconnected structure, including different types of structure topology (i.e., linear, cyclical, branching) and protection approaches (generally span-protecting vs. path-protecting). The following Section outlines the work that has been done on the architectures that will be investigated and discussed in this thesis.

Scheme	**Structure**	**Protection Type**
APS	Linear	Path-protecting
UPSR Rings	Cyclical	Path-protecting
BLSR Rings	Cyclical	Span-protecting
p-Cycles	Cyclical	Span-protecting

Table 2.2: Classification of traditional or well-studied preconnected protection schemes

2.4.2.1 PXTs: Linear Path-Protecting Structures

Pre-Cross-Connected Trails (PXTs) were originally proposed in [ChCh04]. PXTs, as the name suggests, are preconnected segments of spare capacity that form a trail through the network. Such trails can be used to both protect against the failure of individual spans or the failure of entire paths end-to-end, but as defined in this work PXTs are considered to be path-protecting structures only. In addition, [ChCh04] explicitly allows PXTs to be looping and self-intersecting structures, meaning that they can cross the same node or span multiple times. This is different from most architectures studied previously, such as *p*-

cycles, under which it is assumed, for simplicity's sake, that *p*-cycles are not allowed to intersect themselves on either spans or nodes. This topological freedom means that PXTs can be potentially very complicated, making them both more difficult to research and also to implement in real networks. Note that by the definition in [ChCh04], PXTs can also be closed structures (i.e., cycles), in which case they behave effectively like path-protecting *p*-cycles.

In [ChCh04], the PXT concept was developed in tandem with the development of a greedy online heuristic algorithm for PXT-based network design with dynamic demand arrivals. In this investigative framework, more demand will periodically be requested by node pairs, and the network must have the capability to both serve the demand under normal operating conditions as well as assure its continued operation in the event of any single span failure. The algorithm described in [ChCh04] is adapted specifically to this type of network operation as it dynamically grows PXTs in response to demand arrivals. That is, the PXT configuration of the network is never rearranged such that existing preconnections are broken: either an existing PXT is extended or a new PXT is created altogether. The algorithm is greedy in the sense that it takes the action (extending one of the existing PXTs or creating a new one) that minimizes the incremental increase in cost of the protection capacity in the network due to the protection of the new demand. This approach has the advantage of being very fast, but makes a sacrifice in terms of capacity efficiency because as demands accumulate the design produced by many incrementally efficient choices may drift further from an optimal design that could be achieved by a complete recomputation of the PXT configuration (e.g., using ILP methods).

Even though [ChCh04] makes reference to the possibility of shrinking or eliminating PXTs if demands disappear from the network, no provision is made in their algorithmic description for this mechanism, and their results focus on networks designed purely based on the accumulation of demand with no departures. The only results given in [ChCh04] are the capacity costs of the final network designs of 6 different networks over 3 different demand patterns (18 designs overall). Since they only give the costs of the final designs and provide no information about the intermediate incremental cost increases, the results from [ChCh04] essentially characterize only the performance of the algorithm in the offline, green fields design case, despite the fact that it is intended for online operation.

2.4.2.2 Tree-Based Preconnected Protection

A *tree* is defined as a set of connected nodes and spans in a network such that each node has exactly one path to every other node in the tree through the spans of the tree (i.e., a connected, acyclic graph) [Brua92]. A tree used as a protection structure can use these paths as protection paths whenever the transmission between a pair of nodes in the tree has experienced a failure. A tree may protect against the failure of single-span or multi-span working paths between tree nodes in this way. We call a tree that does the former a *span-protecting* p-*tree*, while a tree that protects an end-to-end working path we call a *path-protecting* p-*tree*. In general, a *p*-tree design will consist of several copies of many topologically different trees, the combination of which is able to protect all of the network's spans or demands against any single span failure. In this sense the concept is very similar to that of *p*-cycles.

The development of tree-based preconnected protection is difficult to trace, particularly because the property of preconnection has not been emphasized in studies of trees. In addition, some fluctuation of terminology has contributed to the confusion of some of the discussion of tree-based protection. We will cover a broad range of the literature on trees here in order to clarify exactly what work has contributed to the preconnected tree concept.

Span *p*-Trees

The work that has most clearly stated its intent for the examination of trees as preconnected structures is [Stam97]. It investigates trees as span-protecting structures only, in two different contexts. First it studies a greedy heuristic algorithm that generates preconnected (or pre-configured, to use its own terminology) trees that are generated in the protection layer of a network with working and spare capacities already installed. The intent is for the resulting trees to maximize the network's single span failure restorability, but the algorithm does not guarantee that the result will be 100% restorable. In addition, the resulting designs are analyzed with two different restoration responses in mind: the first uses only preconnected paths from the trees, but the second also allows preconnected paths to be broken to form additional protection paths on demand if the preconnected paths cannot completely restore a span failure. Evidently, this is quite a reversal from the traditional *p*-cycle design exercise: instead of starting from low capacity designs and proceeding to investigate their

characteristics, the intent here was to form trees in an existing network using best guesses about their desirable characteristics and then analyze the effectiveness of the result.

The second way in which [Stam97] addresses tree-based protection is an exercise that uses a genetic algorithm to build preconnected span-protecting structures. Again, this method starts with a capacitated design and generates structures by creating preconnections. The idea of creating tree-based designs from scratch and analyzing them based on confidence in their near-optimality is therefore not used in this work. However, this approach still allows the author to compare trees with other types of preconnected structures and come to the conclusion that trees are not very efficient structures as compared to cycles in particular.

The next closest publications to touch on the idea of preconnected trees for span protection is the series of papers [ShYa01, ShYa04, YaZh02] produced by O. Yang et al. These works develop the idea of *hierarchical trees*, or *hierarchical p-trees* as branching, preplanned protection structures. However, despite the authors' use of the term "p-trees" (the "p-" prefix originally intended to include the concepts of both pre-planning and preconnection), the concept of preconnection is never directly addressed.

The concept of the hierarchical tree is first presented in [ShYa01], and a distributed network protocol for the discovery of such a tree is later proposed in [YaZh02]. A hierarchical tree is defined here as a logical spanning-tree arrangement of network spans such that the amount of capacity on a span is smaller than the capacity of its parent span. The idea is that there will be more protection paths going through the higher-level spans of the tree, and thus they will need more capacity. Note that [ShYa01] only seems to consider network designs that use a single hierarchical tree. Organizing spare capacity into multiple distinct structures is not considered. Also, it is noted in [ShYa04] that using only one such structure by itself cannot guarantee full single failure restorability because spans on the tree itself are unprotected. Therefore [ShYa04] defines a special restorability action for these spans that uses a pre-designated "secondary parent" node for the child node of all on-tree spans. In the event that an on-tree span fails, its child node will connect to its secondary parent, which will then use a backup path through the tree as if a span between the child and secondary parent had failed. Therefore this scheme is not a purely tree-based scheme, but rather a hybrid. Finally, [ShYa04] forms hierarchical trees in networks with pre-installed capacity,

and does not consider green fields *p*-tree design. The intent here is again to generate a best-efforts tree from an existing design and determine its performance.

Path *p*-Trees

A group of related publications [LaSt02, GrCo03a, GrCo03b] investigates the use of tree structures, called *backup trees*, used for the protection of entire end-to-end paths. Backup trees are developed in these works as a method for reducing spare capacity requirements in restorable *Multiprotocol Lambda Switching* (MPλS, a type of label switching) networks. These trees offer protection to unidirectional paths that converge at the root node. This configuration has the advantage that protection flows never split, which makes it conceptually simpler than a configuration with truly branching protection. Therefore rather than providing true tree-like protection, these trees instead provide a mechanism to share a single channel between different flows, allowing capacity savings.

This is a direct result of the label-directed nature of the MPλS networks that this scheme was designed to protect. Because all protection flows that transit a node are assigned the same label, their ultimate destination must be the same; the network has no way of distinguishing them. Therefore the type of trees studied in [LaSt02, GrCo03a, GrCo03b] are uniquely suited to the protection of MPλS networks, and are not necessarily suited to the context of transparent DWDM networks that we assume in this thesis.

Another set of publications on the subject of tree-based path protection focuses on the use of a single pair of complementary trees to protect the network against failure [MeFi99, ZhXu08, XuCh02, XuCh03]. These trees (termed the "red" and "blue" trees) are not protection structures per se, but rather logical tree routings that are arranged such that in the case of any single link failure, every pair of nodes will still be connected through at least one of the trees. Therefore this is similar to the backup trees concept in that although the protection routings may form trees, no true branching protection is being performed. Red/blue trees are tools for the operation of pre-planned protection, but they do not define actual capacitated structures.

2.4.2.3 Failure Independent Path-Protecting Cycles

We have already mentioned FIPP *p*-cycles as an example of an established protection architecture in Section 2.2.5. We now go into more detail regarding their operation and prior literature. The *p*-cycle concept was first fully generalized to the path-protecting case in [KoGr05, KoGr05a]. FIPP *p*-cycles provide protection to working paths in the same way that *p*-cycles provide protection to spans. If a working path shares any spans in common with the cycle itself, it is considered to have an on-cycle relationship with that cycle. In this case, the cycle only provides one protection path for the working path if it fails, because if the failure occurs on one of the spans that the working path and the cycle share then that half of the cycle will obviously be failed and unable to provide a protection path.

Note that it is technically possible for a cycle to provide two protection paths to the working path if the failure in question occurs on a span that is not on the cycle. However, this introduces failure dependence into the FIPP *p*-cycle restoration mechanism because the nodes in question must have knowledge about the location of the failure. While this is feasible with intelligent enough network equipment, it goes against the failure independent nature of FIPP *p*-cycles. If a working path does not share any spans in common with the cycle, the working path has a straddling relationship with the cycle, which may then provide two protection paths in the same manner that a regular *p*-cycle does to a straddling span.

There is one special case of FIPP *p*-cycle protection that bears further explanation, even though it also falls under the heading of on-cycle protection. This special case occurs when a working path has an on-cycle relationship to the cycle on both of the "halves" of the cycle as divided by the end nodes of the working path. The simplest possible such case, called the *Z-case* due to the Z-like appearance of the working path, is illustrated in Figure 2.8. The Figure shows that we cannot find a single protection path between the end-nodes of the working path on the cycle that can be used in all failure scenarios because both potential protection paths intersect with the working path on one span. This means that the protection path that is used when the working path fails will necessarily be different between certain failure scenarios, in order to avoid overlapping with the failed section of the cycle.

At first this may seem to be a violation of the failure independence property of FIPP *p*-cycles. However, [KoGr05] clarifies that in this sense failure independence refers only to the knowledge required to perform restoration. If restoration can be performed using

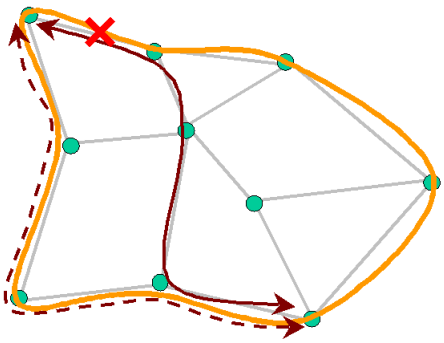


Figure 2.8: The "Z-case" of protection for FIPP *p*-cycles

the exact same actions without any more specific knowledge than the simple fact that the working path has failed then it is still considered failure independent. Therefore [KoGr05] is able to justify this special case as failure independent using the following explanation: upon detection of a working path failure at the end-nodes, the nodes attempt to form a protection path on a predefined, arbitrary half of the cycle. During this process they check whether or not a failure has occurred on this half of the cycle. If it has, they simply switch to using the other half. Because this does not use any network capability besides the detection of a failure somewhere on a path, and because this action works properly regardless of the location of the failure, it is still failure independent; no special-case signaling is required.

There is a complication introduced into the FIPP *p*-cycle concept as a consequence of the ability of a single FIPP *p*-cycle to protect multiple different end-to-end working paths. Because FIPP *p*-cycles protect entire paths end-to-end instead of single spans, there is the potential for multiple protected paths to fail simultaneously if they are non-disjoint. This is not a problem for regular *p*-cycles because the spans that a cycle protects are disjoint by definition and cannot fail simultaneously under the single span failure assumption. But in the case of FIPP *p*-cycles, contention for spare capacity between simultaneously failed paths may occur if they are both protected by the same FIPP *p*-cycle. Therefore the additional constraint has been added to the FIPP architecture that all of the paths protected by a cycle must belong to a *disjoint route set* (DRS). As the name suggests, this refers to a set of routes that are all span-disjoint from each other.

The first FIPP *p*-cycle design method was introduced as a spare capacity placement formulation in [KoGr05a], and then a joint capacity placement method was outlined in [BaGr07]. These methods are optimization-based approaches that use an ILP model to

find a minimum cost FIPP *p*-cycle arrangement such that all demands in the network are protected by at least one cycle. The formulations are similar to the standard *p*-cycle network design model (i.e., from [GrSt98] or elsewhere). However, because of the aforementioned DRS restriction, new parameters must be introduced to encode the DRSs in the network, and the constraints are altered such that cycles are assigned to protect DRSs instead of individual demands explicitly. Unfortunately, in any network of realistic size, the number of possible DRSs will be far too large for the complete model to solve in a reasonable amount of time. Therefore the method used in [KoGr05a] and [BaGr07] is to only include a randomized selection of possible DRSs in the model, under the assumption that this selection contains enough variety to produce a solution that is close to optimal.

This solution, though practical, is unsatisfactory in the sense that it makes it difficult to draw definite answers to questions of network science regarding FIPP *p*-cycles from the solutions of the model. Therefore efforts continue to find methods to solve the optimal FIPP *p*-cycle problem efficiently. The approach taken in [JaRo07] is to use an Operations Research (OR) technique called *column generation* to reduce the size of the ILP problem. This technique works by dynamically generating the "columns" of the problem (in the case of FIPP *p*-cycles, a column corresponds more or less to a DRS) as they are needed, instead of including them all from the beginning. This technique can be used to solve problems to optimality without including all of the possible columns for the problem. However, even with the column generation method, the runtimes for FIPP *p*-cycle problems of realistic size remain high.

Although we will not be studying FIPP *p*-cycles directly in this thesis, we will often use FIPP *p*-cycle designs for reference and comparison purposes because of the well-known efficiency of cycles as protection structures. The DRS method for FIPP *p*-cycles also serves as a basis for the network design methods that we propose for other path-protecting structures (PXTs and path *p*-trees). Therefore FIPP *p*-cycles serve as an important conceptual precursor for much of the work in this thesis.

2.4.2.4 Summary

Table 2.3 gives a summary classification of the novel schemes discussed above, in the same way as Table 2.2 did for the more well-established architectures.

Scheme	Structure	Function
PXTs	Linear	Path-protecting
FIPP p-Cycles	Cyclical	Path-protecting
Span p-Trees	Branching	Span-protecting
Path p-Trees	Branching	Path-protecting

Table 2.3: Classification of newly proposed preconnected protection schemes

2.4.3 Generalized Pre-Cross-Connected Frameworks

Occasionally, the subject of pre-cross-connected protection has been approached using methods that attempt to take into account the entire set of pre-cross-connected structure simultaneously (as opposed to dividing this set into different architectures, e.g., FIPP p-cycles, trees, PXTs, etc.) These approaches are theoretically powerful because they can compare many types of structure simultaneously, but run into trouble because the size of this problem becomes large very quickly as the network's size increases. Therefore they have to resort to measures such as using metaheuristics, using only very small "toy" networks as test cases, or otherwise restricting the parameters of the simulation so as to reduce the problem size, which in turn reduces the strength of the conclusions that can be drawn from the results.

We have already mentioned one example [Stam97] that used a genetic algorithm to build pre-cross-connected structures. Another more recent work [HeSo07] uses an ILP model to impose the pre-cross-connection constraint on optimal SBPP designs. Because SBPP is the most general case of shared path protection, the results for pre-cross-connected SBPP could theoretically tell us the best way of using pre-cross-connected path protection. However, closer inspection shows that the proposed ILP method by its nature must operate given a fixed number of available wavelengths. The number of wavelengths can be increased arbitrarily to produce the corresponding green fields designs, but this comes with a corresponding increase in complexity. Furthermore, the results are greatly limited by the authors' choice of parameters (only 2 alternate protection routes are considered for each demand pair). Finally, the authors do not investigate the structural properties of the solutions, focusing only on capacity utilization. Therefore, despite the vast theoretical generality of the ILP model, this work in practice does not have a direct impact on knowledge in the field concerning the relative merits of different types of pre-cross-connected protection.

Chapter 3

Span-Protecting *p*-Trees

3.1 Introduction

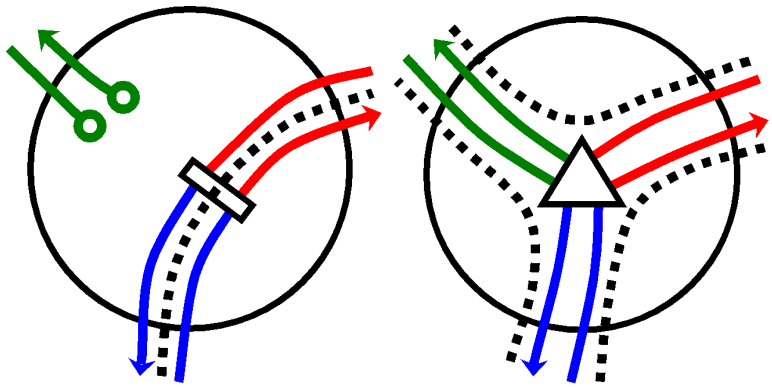
This Chapter describes our investigations into the first truly preconnected, tree-based protection architecture for transparent networks: span-protecting *p*-trees. First we provide an explanation of the span-protecting *p*-tree concept, addressing the theoretical interest as well as concerns about practicality. We then survey the literature related to span-protecting *p*-trees and explain why the concept introduced in this thesis is distinct from that presented in most prior literature on tree-based protection and why the work performed here is different from what has come before. We then go through our in-depth investigation into the characteristic efficiency of span *p*-trees and the structural properties of efficient span *p*-tree designs.[1]

3.1.1 Background

3.1.1.1 Degree-N Cross-Connections

Traditionally, preconnected network protection structures are thought of implicitly as *degree-2* structures, meaning that cross-connections are always between two and only two units of spare capacity at a network node. This reflects the capabilities of the physical cross-connect hardware used in real networks. A high-level view of a path formed through a traditional cross-connection action at a degree-3 node is shown in Figure 3.1 (a), while a more detailed illustration of the internal switching actions at the node is given in Figure

[1] Some of the work in this Chapter has been published in Photonic Network Communications:
A. Grue, W. D. Grover, "Comparison of *p*-Cycles and *p*-Trees in a Unified Mathematical Framework," *Photonic Network Communications*, vol. 14, no. 2, October 2007, pp. 123-134.



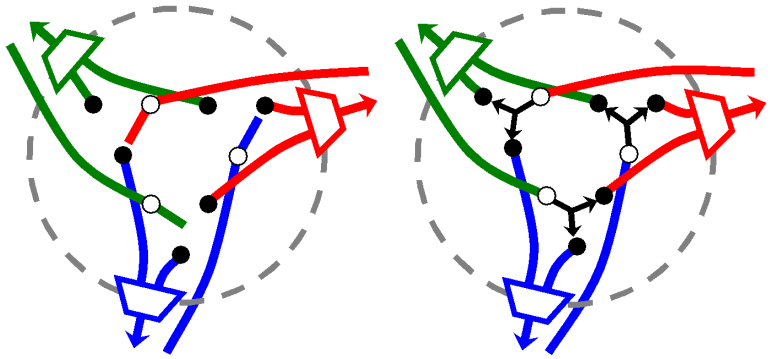
(a) Degree-2 (b) Degree-3

Figure 3.1: High-level logical views of paths formed through degree-2 and (hypothetical) degree-3 cross-connections

3.2 (a).[2] Purely within the realm of theory, however, there is no reason that we cannot consider cross-connections between three or more units of spare capacity. In a theoretical degree-3 cross-connection, one can consider any one of the three units involved to be cross-connected to both of the other units, meaning that a traffic-serving path may be established in any one of three ways through the cross-connect without the need for any switching actions. The possible path traversals of such an arrangement are illustrated in Figure 3.1 (b). Figure 3.2 (b) shows the equivalent switching-level detail required to implement such a cross-connection, with equipment for signal splitting and merging taking the place of the switches. This arrangement assumes that each pair of "merged" incoming signals for outgoing transport on the third connection contains at most one active signal at any one time; otherwise, some contention resolution must occur to decide which signal gets to use the outgoing channel.

Allowing this type of cross-connect (and equivalent arrangements for a degree of 4, 5, etc.) opens up the possibility for branching preconnected structure, i.e., trees. In fact, the presence of degree-N cross-connections allows the realization of any arbitrary preconnected structure, containing any combination of adjacent network spans and any number of cycles and branches. Using degree-N cross-connections, we can even conceive of a process in which the assignment of unit spare capacity links produced by an optimal span-

[2]The colours in Figures 3.1 and 3.2 are purely to distinguish the inputs and outputs from each of the 3 adjacent spans, and should not necessarily be taken as indicators of optical wavelength; these diagrams apply to any generalized switching transport network, optical or otherwise.



(a) Degree-2 (b) Degree-3

Figure 3.2: Internal switching views of paths formed through degree-2 and degree-3 cross-connections

restorable mesh or SBPP design algorithm is consolidated into a single preconnected structure, incorporating both theoretically maximum efficiency and full preconnection into the same design. Therefore the concept of the degree-N cross-connect seems at first to trivialize the restorable network design problem. However, as we will see next, some conceptual difficulties that arise when considering the implementation of more complicated degree-N preconnected structures suggest that we should restrict ourselves to the set of branching structures that do not contain cycles, i.e., trees only.

3.1.1.2 Concept Restrictions: The *p*-Tree Definition

It is understandable to be initially skeptical as to whether the concept of cross-connections with degree higher than 2 are meaningful in a real-world sense. After all, the concept of cross-connection seems to imply linking the flows in two capacity units together. To use a water analogy for network flow, the idea is similar to that of connecting two pipes together so that water may flow from one into the other. However, there is no reason to reject the possibility outright if one thinks of the multiple degree cross-connect as a type of flow splitter. Only instead of splitting the flow in two, as in the water analogy, the data or signal coming into the node from any one span is copied onto each of the spans in the outgoing direction.

It is productive at this point to stop and determine whether considerations such as this, that arise when trying to reconcile the theoretical construct of degree-N preconnection with

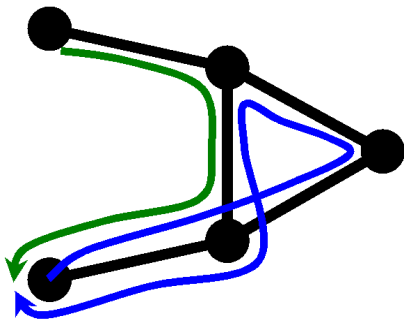


Figure 3.3: Conceptual difficulty with the operation of a cyclical, branching preconnected protection structure due to signal collisions

the realities of current technology, can be used to provide some reasonable restrictions on the complexity of degree-N preconnected structures in order to focus the concept enough for an initial experimental examination into its implications. Right away, the characterization of degree-N preconnection as fundamentally a signal-splitting operation raises a concern about the feasibility of branching structures that also contain cycles. Figure 3.3 illustrates one possible problem with such a structure. This Figure shows a preconnected structure configuration in a 5-node network. The diagram shows a subset of the possible propagation directions for flow on the tree between the two leftmost nodes (assuming some span or path between these nodes requires restoration). Upon failure of the span, the backup signals will propagate into the structure from both ends of the span. Only some of the branches of the flow are illustrated, for the sake of simplicity. The path indicated by the solid arrows represents one path that the restoration flows would take given the splitting model of degree-3 preconnections. The diagram shows that, due to the cycle in the structure, the flow actually loops back on itself and re-enters the end-node that dispatched the flow originally, overlapping with the path that the restoration of the failed downward flow would also take. In this circumstance, the network would have no way to determine which flow to transmit over this and other spans.

Another way to phrase this problem is that the propagation of restoration flow in a branching structure with cycles violates our stipulation that a merging action at a degree-3 cross-connection must never have two signals present simultaneously on the two incoming channels. At the very least, the operation of such a structure would therefore need to incorporate significant node-to-node signaling to control the propagation of restoration flow to

prevent situations like this one. Thus the first obvious restriction to the degree-N preconnected structure concept is that such a structure must sacrifice the possibility of cyclicity in order to make sense as a restoration concept, even under some very modest assumptions. Therefore our degree-N structures should be strictly trees. In a tree, this flow conflict problem cannot occur; restoration flow simply spreads outward from the node that introduces it, until it reaches the leaf nodes of the tree or is intercepted by the other end-node of the node pair that is effecting restoration.

Note that this restriction rules out the scenario we described earlier, in which the capacity plan produced by an arbitrary network design method is simply preconnected at all possible junction points, creating an ideal minimum cost preconnected capacity plan. Generally, preconnecting an arbitrary plan in this way will at some point create a cycle in the preconnected structure, making its implementation infeasible by the justification we have given above. However, it may be that some types of designs under certain architectures can either be preconnected this way without producing cycles, or can be easily modified slightly such that this would become possible. This may be an interesting approach to tree-based preconnected network design, but remains for future work, as it will not be considered in this thesis.

From now on we will refer to non-cyclical, purely tree-like pre-cross-connected protection structures as *p*-trees. Depending on whether the trees provide protection between the end-nodes of a single span or protect entire paths end-to-end, we will refer to them as span-protecting *p*-trees and path-protecting *p*-trees, respectively. Another restriction to the *p*-tree concept that we can make out of hand is that *p*-trees must be *simple* structures, i.e., they cannot cross the same span or node more than once (as with *p*-cycles). This is practically a requirement for the experimental study of *p*-tree architectures because, without it, the size of the set of possible protection structures is simply too large. This has practical implications for ILP-based design models, as they generally rely on the ability to enumerate sets of candidate structures for the design explicitly. Consider the fact that a non-simple tree could, in theory, contain multiple spare links on the same span cross-connected at one of the span's end nodes. This means that even the set of potential protection structures that cross only two particular adjacent spans would be infinite, because this set consists of structures containing N spare links on one span cross-connected to M spare links on the

other span, with no upper bound on the values of N or M. Restricting the set of *p*-trees to be simple, on the other hand, sets a strict upper bound on the number of possible structures in a network. As an additional benefit, it facilitates comparison with other architectures that traditionally operate under similar restrictions, such as *p*-cycles.

Thus we define *p*-tree architectures as architectures in which the protection structures consist of simple tree structures, not containing any cycles, that are used to protect failed flow between pairs of end-nodes by forming protection paths within their preconnected spare capacity. We will begin our investigation into tree-based protection by focusing first on span-protecting *p*-trees.

3.1.1.3 Previous Span-Protecting Tree Literature

At this point, distinctions should be drawn between the *p*-tree architecture as discussed in this thesis and previous work on tree-based protection schemes, most of which has been performed using significantly different definitions and assumptions. [Stam97] is the only source to examine true tree-based pre-cross-connected protection. However, it did so in the context of a pre-capacitated network, and not with an eye towards characterization of trees as a standalone architecture in terms of efficiency and complexity, as we do here. Despite this, the results can still be used to draw conclusions about the fundamental efficiency of *p*-trees: namely, that trees compare poorly in efficiency to cycles. This is a conclusion that we will address later in our own results as well.

Reference [ShYa01] and its extensions [ShYa04] and [YaZh02] are the next most relevant to this discussion, as they also address the use of tree-based protection arrangements for span protection. Reference [ShYa04], in fact, introduces the term "p-tree" to describe the concept. This is unfortunate, however, because their conception of tree-based protection is not at all like the tree-based extension of *p*-cycles that is suggested by the name and that is treated in this thesis. To help underline the differences between their hierarchical trees and true *p*-trees, the basic concepts of [ShYa01, ShYa04, YaZh02] will be outlined here.

Reference [ShYa01] describes the concept of a hierarchical tree that seems to be motivated by the idea of extracting a natural tree hierarchy from a network in which each node in the network is connected to each other node through some path in the (spanning) tree, and

each link in the tree contains more capacity than all of its child links. The idea is that such a hierarchy can be used as a natural protection arrangement for the entire network. Spans that are not part of the tree are protected by paths that travel from the span's end-nodes up through parent links in the tree until they meet at a single parent node, possibly (but not necessarily) the root. In this way, hierarchical tree protection is somewhat similar to how a single spanning tree, used as a span-protecting *p*-tree and capacitated many times over, could protect all of the spans in a network at once. Reference [ShYa01] outlines the basics of a distributed algorithm to form a hierarchical protection tree in a network given a single root node, while [ShYa04] goes on to develop in much more detail the specifics of both an optimization model and a heuristic for finding such a tree. Reference [YaZh02] further develops a distributed protocol for discovering such a tree during network operation.

While this concept is tree-based, it is similar to true *p*-trees only superficially; they differ in numerous other ways. First of all, hierarchical tree protection fundamentally only uses a single spanning tree to protect the entire network. Therefore the tree itself cannot protect its own spans, so an alternate method must be used for their protection. The possibility suggested in [ShYa01] is that nodes that become disconnected from their parent due to the failure of a tree span be assigned a backup parent elsewhere on the tree. Protection then occurs by routing the first part of the protection route through to the backup parent. Therefore this method is not a pure tree-based method like *p*-trees, but requires hybridization with other methods to even provide 100% single failure restorability. Also, hierarchical trees are not single-unit preconnected structures, but instead consist of different amounts of spare capacity on each span. Indeed, the concept of preconnection is not addressed in either [ShYa01] or [ShYa04], raising the question of why this scheme is called "*p*-trees" at all, given that the "*p*-" prefix was originally intended to represent the idea that *p*-cycles were preconnected protection structures. The authors do point out that protection paths in hierarchical tree protection are pre-determined (i.e., pre-planned), but this is a separate concept from preconnection. Preconnected paths are by definition pre-planned, but the reverse is not generally true.

We can actually impose the idea of preconnection upon the hierarchical *p*-tree concept in retrospect by considering a single hierarchical tree to actually be a combination of *p*-trees layered on top of each other, each *p*-tree containing a subset of the spans of the spanning

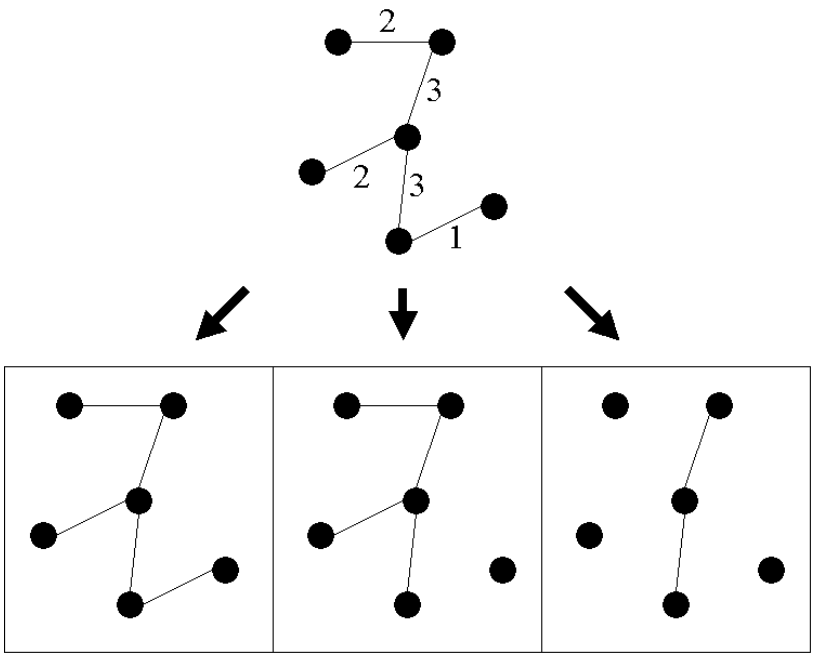


Figure 3.4: Splitting a "hierarchical *p*-tree" into single-unit copies of three different true span-protecting *p*-trees

hierarchical tree. This idea is illustrated in Figure 3.4. The master tree, with spare capacity amounts on spans as indicated, is split into three different single-unit *p*-trees. The reader can verify that this split maintains all the protection paths provided by the original tree. Therefore the hierarchical tree concept can be seen in hindsight as a very specialized case of multi-tree *p*-tree protection, and even then only for the off-tree spans. Of course, when it comes to the on-tree spans, the hybrid "secondary parent" approach is still taken, as discussed above. Therefore, even under this supplementary framework, the scheme still cannot be fully preconnected.

Overall, then, hierarchical tree protection is only very tenuously related to the idea of true span-protecting *p*-trees. While *p*-trees are truly discrete, self-contained tree-based protection structures, a hierarchical tree only represents a tree-based organization of nodes that regulates the sharing of spare capacity between protection paths for failed spans. Preconnection can be imposed on the scheme, but this idea was not considered in the original work. In fact, the hierarchical tree concept need not use the terminology of structure-based protection at all. Instead, it can simply be viewed as an additional constraint imposed on the standard span restoration architecture, with protection paths constrained to lie within the

structure of the hierarchical tree (along with the use of a backup parent for on-tree spans). Therefore it should not be assumed that any work within this thesis on the subject of span-protecting p-trees is in any way a duplication of the efforts of [ShYa01, ShYa04, YaZh02], because their use of the "p-tree" label for their work is not consistent with our use of the term or with the spirit of the use of similar terminology (i.e., p-cycles).

One other publication on the topic of span-protecting trees [LiYa03] by the same authors as [ShYa01, ShYa04, YaZh02] investigates a different concept in which the network is protected by two pre-configured spanning trees. This is similar to the red/blue tree concept [MeFi99, ZhXu08, XuCh02, XuCh03], except that red/blue trees are path-protecting, not span-protecting (see the corresponding discussion of the literature in Chapter 5 on path-protecting p-trees). As with red/blue trees, these trees are not physical structures but rather *logical trees* that define pre-configured (but not necessarily preconnected) protection paths. Finally, this concept is presented as a method for protecting the physical mesh network, not the transport network, which leads to different assumptions about protection requirements (e.g., that a single link failure corresponds to the failure of only a single unit of capacity (fibre) on each span). Therefore this concept is again distinct from that of p-trees as proposed in this thesis.

[WuYe08] addresses a concept that the authors call "PXTs", but closer reading reveals that the structures they discuss are span-protecting segments, unlike PXTs, which are path-protecting. Therefore they are more properly seen as a special case of span-protecting trees in which the tree nodes are all at most degree-2 (which we will come to call p-segments later in this Chapter), and thus are relevant to our work on p-trees here. [WuYe08] extends their ILP framework for non-simple p-cycles without explicit cycle enumeration from [WuYe07a, WuYe07b] to p-segments, and uses the ILP model to establish the difference of capacity usage in designs using simple vs. non-simple structures. However, their approach has several limitations. Despite their claims that explicit structure enumeration has "huge computational complexity", their implicit structure generation constraints increase the complexity of the model itself, meaning they can only use it on very small test networks. Furthermore, their model involves a parameter J that limits the maximum number of protection structure sets that can be used, and all their tests are run with $J = 2$, quite a severe restriction. Finally, they are only concerned with capacity efficiency, and do not

focus at all on structural properties. Therefore their analysis is both quite different and significantly more limited in scope than the one we provide for *p*-trees and *p*-segments in the following pages.

3.1.2 Goals and Objectives

With the span-protecting *p*-tree concept defined and distinguished from previous work on tree-based span protection, we were able to make initial forays into the problem of designing networks that use this architecture. We wanted first of all to make basic observations about the architecture's own characteristic spare capacity efficiency, as well as make some comparisons between *p*-trees and *p*-cycles, currently the most prominent span-protecting preconnected architecture in the literature. The secondary goal was to draw conclusions about the structural characteristics of efficient trees, both as a stand-alone architecture and when combined with *p*-cycles.

3.2 Experimental Tools

3.2.1 ILP Design Model

In order to begin investigating span-protecting *p*-trees, we first had to define a standard method for producing *p*-tree based designs. Rather than producing an entirely new design algorithm from scratch, however, we made the observation that the standard ILP *p*-cycle design model (introduced in [GrSt98]) could also be used, with no modifications, to obtain minimum spare capacity cost *p*-tree designs. This model is as follows:

ILP Model

Sets:

S The set of spans in the network, indexed by i for a failure span, and j for surviving spans.

P The set of (simple) cycles of the graph eligible for formation of *p*-cycles, indexed by k.

Input Parameters:

w_i The number of working channels (or capacity units) on span i that require protection.

x_i^k The number of protection relationships provided to span i by a unit-sized copy of p-cycle k. $x_i^k = 2$ if span i straddles cycle k, $x_i^k = 1$ if span i is on cycle k, and $x_i^k = 0$ in all other cases.

δ_j^k Encodes the spans on a p-cycle. $\delta_j^k = 1$ if cycle k includes span j (i.e., if $x_i^k = 1$), and $\delta_j^k = 0$ if it does not (i.e., $x_i^k = 0$ or $x_i^k = 2$).

C_j The cost of a unit of capacity placed on span j.

Decision Variables:

s_j The integer number of spare channels assigned to span j in the design.

n_k The integer number of unit-capacity p-cycle copies of cycle k in the design.

Objective Function:

Minimize

$$\sum_{j \in S} C_j \cdot s_j \tag{3.1}$$

Constraints:

$$w_i \leq \sum_{k \in P} x_i^k \cdot n_k \quad \forall i \in S \tag{3.2}$$

$$s_j = \sum_{k \in P} \delta_j^k \cdot n_k \quad \forall j \in S \tag{3.3}$$

Constraint 3.2 ensures that every span has all of its working capacity protected by p-cycles and constraint 3.3 ensures that there is enough spare capacity to support the placement of the p-cycles chosen for the solution. Objective function 3.1 minimizes the total cost of spare capacity in the network.

Note that despite the definition of P above as a set of cycles specifically, there is nothing in this model restricting the set of protection structures to be cycles. If the parameters x_i^k and δ_j^k are set to the appropriate values, they can be used to represent any preconnected,

span-protecting structure. If, for some k in P, δ_j^k is set to 1 for the spans on a specific tree structure in the network, and x_i^k is set to 1 for spans with end-nodes on this tree but that are not on the tree itself, then the structure represented by member k of the set P is actually a span-protecting p-tree. If the set P is then populated entirely with these tree structures instead of cycles, the model can be used without changes to compute a lowest cost set of p-trees to protect the network from single span failures.

Furthermore, the ability of the model to accommodate both p-cycles and p-trees also allows it to solve for designs that use a hybrid of the p-cycle and p-tree architectures. This is possible by populating the set P with both candidate cycles and candidate trees. The solver is indifferent to the distinctions we make conceptually between the two protection architectures and will simply use the best span-protecting structures out of the set it is given to protect the network with minimum spare capacity cost. This high degree of generality allows us to make some very fair and objective comparisons between the p-tree and p-cycle architectures by first allowing the solver to find hybrid designs, and then examining the result to determine the relative values of the tree and cycle contributions to the solution.

3.2.2 Tree Generation

Of course, even though the ILP model remains the same for the transition from p-cycle to p-tree design, the data preparation stage must be changed in order to generate sets of parameters that describe trees instead of cycles. Because the p-tree architecture is relatively new, models for designing p-tree networks have not seen much use or explanation in the literature. Therefore the method that we have used to generate the candidate trees for the model is described here in detail. This method can be considered a highly modified breadth-first search technique, designed to produce the set of all simple trees within certain size restrictions and nodal degree restrictions, with no duplicates. Note that we are only interested in the topology of the trees we generate, and therefore we do not define a root node or a hierarchy of nodes in our trees; they are *unrooted* trees.

The fundamental concept behind this algorithm is the idea of a tree "template" that describes a set of trees with certain spans in common. A template consists of both a tree "skeleton", the set of spans that a tree must contain, and a set of "forbidden" spans, the spans that a tree must not contain. In other words, a tree template describes a set of trees that

contain the spans in the skeleton and do not contain forbidden spans, optionally containing the spans that are in the network but neither in the skeleton nor forbidden. The algorithm then works based on the observation that the set of trees described by a template T is exactly equal to the sum of the set of trees described by two modified templates, $T+S$ and $T-S$, where we use the notation $T+S$ to mean T with some span S added to the skeleton and $T-S$ to represent T with some span S added to the forbidden set. Evidently $T+S$ and $T-S$ cannot contain any trees in common, as trees in $T+S$ must contain S and trees in $T-S$ cannot contain S. Therefore $T+S$ and $T-S$ represent the division of the trees represented by T into the set of trees that contain S and the set of trees that do not. Furthermore, $T+S$and $T-S$ together represent all of the trees described by T with no overlap.

This observation was used to implement a recursive tree generation function that takes as input a template T and returns the trees described by that template. The function finds a span S not currently in the skeleton or forbidden set of T but that would also result in a valid tree when combined with the existing skeleton, and then calls itself recursively twice, first with $T+S$ as the argument and with $T-S$ as the argument. This template splitting operation is illustrated in Figure 3.5. The sets of trees returned by these two function calls are combined and then returned as set of trees described by T. Because, as mentioned, the sets provided by $T+S$ and $T-S$ cannot overlap, no expensive duplicate checking is required. The recursion is stopped when no span can be found to add to the tree that is not in the skeleton or forbidden set of T; in this case, T describes simply a single tree, because all spans are mandated as to whether or not they may be in the tree, and this single tree (the skeleton) is returned as the only element in the set. Using this function, the set of all trees in the network can be generated by simply calling this function with the argument being a template with an empty skeleton and forbidden set.

Limitations on tree size and nodal degree were also easily implemented in this algorithm. To implement tree size limit, the algorithm needed only to be modified to terminate recursion when adding a span S to the skeleton would cause the tree to become too large. In this case all trees within the size limit and specified by T consists only of the single tree described by the skeleton, since the tree cannot be expanded any more, so this tree is returned by itself. Similarly, nodal degree was limited by not considering choices of span

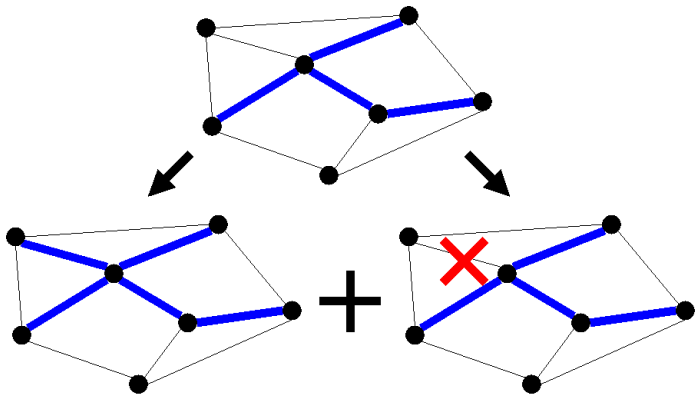


Figure 3.5: A template is split into two complementary templates that together represent all the trees of the original

S such that addition of *S* to the skeleton would result in a node of degree that exceeds the limit. The same effect can also be accomplished performing a pass at the beginning of each recursive call that automatically adds all such spans to the set of forbidden spans.

3.3 Pure *p*-Tree and *p*-Cycle Design Comparison

3.3.1 Test Cases

This model was first used to generate *p*-tree designs for two different network families to determine the efficiency and other properties of the architecture in networks with a wide range of nodal degrees. For an explanation of the network family concept, see Section 2.3.4.1.

The two families used were created from a master network with 15 nodes and 30 spans, and a master network with 20 nodes and 40 spans, respectively (see Appendix A.1.1 and Appendix A.1.2 respectively for topology details). The set of candidate trees for each test case was produced by finding all trees in the test network containing 7 or fewer spans with a maximum tree nodal degree of 3. These size restrictions may seem severe, but were necessary in order to limit the set to a size that would allow a solution to be found in a reasonable amount of time. For the purposes of comparison, corresponding *p*-cycle designs were also calculated for each test case. The set of candidate cycles for these tests contained the entire set of cycles in the network. Therefore the *p*-cycle designs are truly optimal designs for their corresponding topologies. For all tests, both *p*-tree and *p*-cycle,

working capacity was assigned by routing all demands along a single shortest path between the nodes that exchange the demand.

3.3.2 Results

Figure 3.6 shows the spare capacity costs for these *p*-tree and *p*-cycle designs. Note that data for *p*-trees is not available for the sparser (lower nodal degree) members of the network families. This is because of a serious difficulty with the design of pure *p*-tree networks that was revealed by these tests: the solver was unable to find any feasible solution that satisfied 100% single span failure restorability for the sparser networks in question. This is a consequence of the size limitation on trees that is specified above. In the sparser networks the smallest possible protection path for some spans may exceed the length of the largest tree in the candidate set (in this case, 7 hops), meaning that there are no candidate trees that can provide restoration for that span. Of course, the same situation could occur in a *p*-cycle design if *p*-cycle length were restricted similarly, but this is not necessary in practice because the set of network cycles will always be very much smaller than the set of trees.

Therefore span-protecting *p*-trees already pose a significant design challenge, in that using as the candidate set a complete set of trees under a size limit that is high enough to allow a feasible design to exist means using a set that is also too large to be practical. Of course, it is always possible to supplement a complete set of small trees with a small set of large trees to ensure the existence of a feasible solution. But that approach would negatively impact our ability to make judgments about the pure *p*-tree architecture, as we could not be certain that we were not simply measuring the efficiency of the largest trees in the network as opposed to the entire set. Therefore we have avoided taking this approach for the purposes of this initial investigation.

Setting aside this concern for the moment, we also see that, for the networks for which a design does exist, the cost of the designs is exceedingly high, double or even triple that of the *p*-cycle designs. Given these results and the above problem, it is hard to see *p*-trees as practical network protection structures. It is possible that not allowing the solver to consider the larger trees in the network is a great hindrance on its ability to produce efficient design, but attempting to include these trees makes the design problem unacceptably difficult, as the size of the set of trees grows very fast as the limit on tree length is increased.

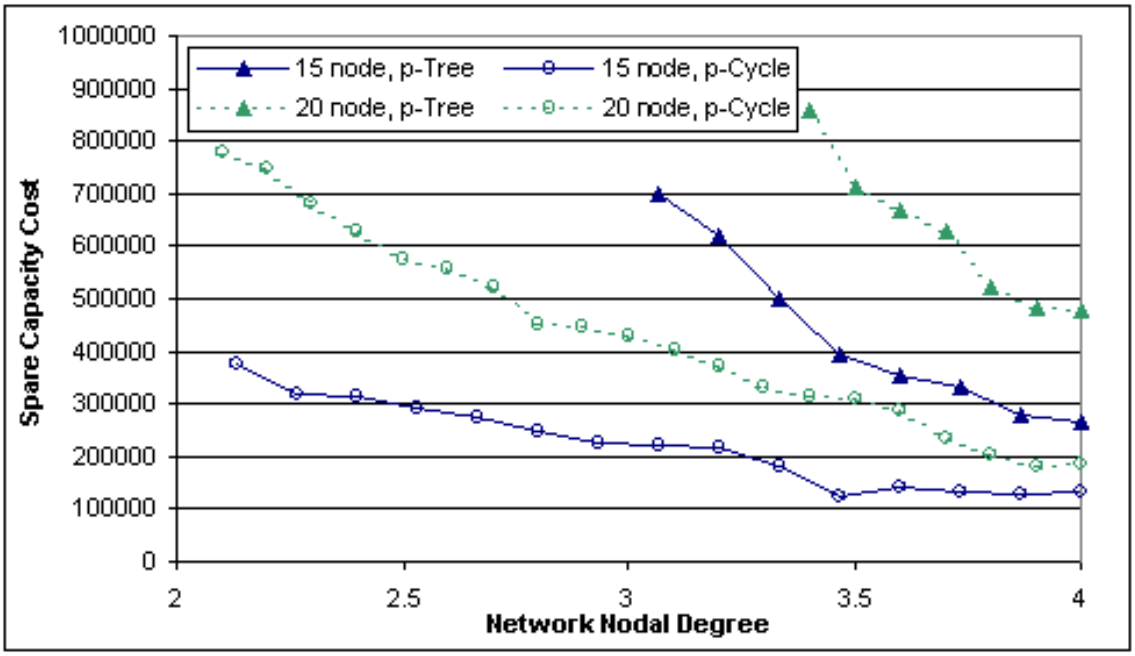


Figure 3.6: Comparison of spare capacity costs of *p*-tree and *p*-cycle designs for networks from the 15 node and 20 node network families

Even though these designs have impractically high capacity costs, analyzing the individual trees that they contain provides some insights into the characteristics of span-protecting *p*-trees. For example, diagramming the solutions by drawing their trees as overlays on top of a diagram of the network topology reveals that a significant fraction of the trees in the network do not actually use any degree-3 junctions. In other words, these "trees" are simply degree-2 linear segments, like PXTs except that they are span-protecting instead of end-to-end path-protecting. This observation led us to analyze the composition of the network designs in terms of the proportion of "true" degree-3 trees and degree-2 segments. Figure 3.7 breaks down each of the designs in terms of the number of unique structures of both types that they contain. In other words, multiple capacitated copies of the same tree or segment are considered as only a single structure for the purposes of this Figure. The Figure reveals that, as nodal degree increases, the diversity of the set of true tree structures chosen by the solver tends to increase, while the number of different types of segments chosen tends to slowly decrease. This trend is present in networks from both the 15 node and 20 node families but is stronger in the 20 node case. The reason for this trend can be understood intuitively; as the network increases in connectivity, the opportunities for

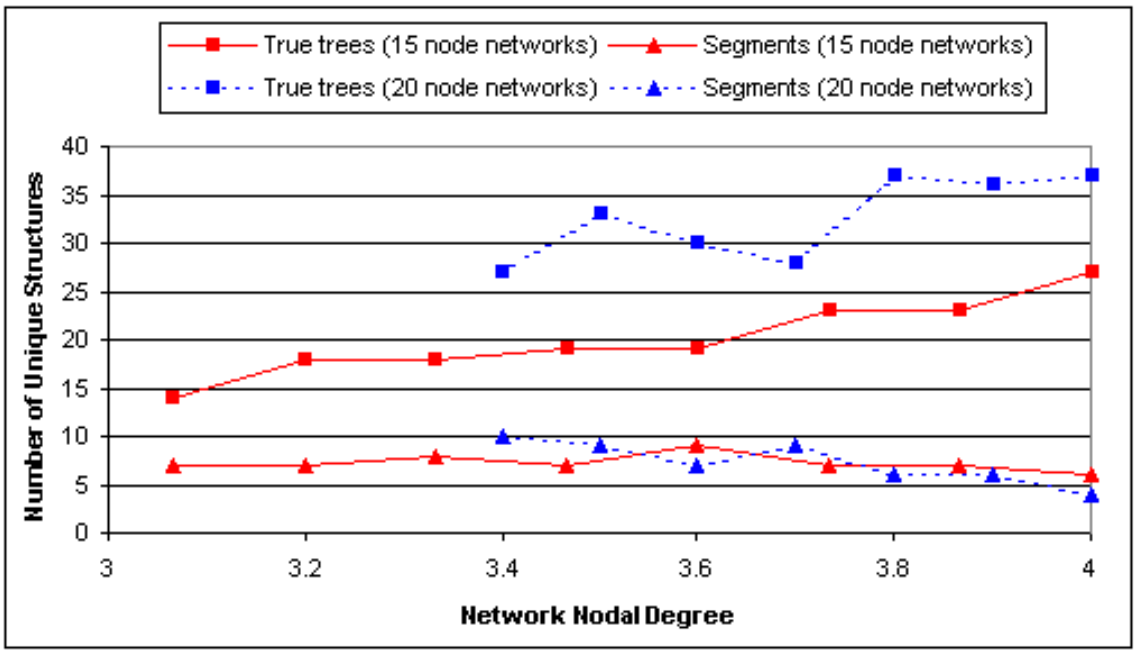


Figure 3.7: The usage of "true" degree-3 trees vs. degree-2 linear segments in the pure *p*-tree designs, represented by the number of unique structures used

tree-like connectivity will increase, and the number of degree-3 trees will increase faster than the number of segments. Therefore, all other things being equal, by sheer probability the solver will be more likely to use trees than segments. In fact, given how quickly the set of degree-3 structures increases, it is actually remarkable that the levels of segments remain as high as they do.

Normalizing the data from 3.7 to the total number of candidate degree-3 and degree-2 structures given to the solver tells us what fraction of the candidate structures of each type are used in the optimal designs. The plot of this data is given in Figure 3.8. According to this Figure, the set of candidate trees and the set of candidate segments both see about the same proportion of usage across all of the test cases. This reinforces our initial observation that segments are an important part of tree-based design. Furthermore, the fact that segments match trees in terms of their proportional usage in the designs suggests that it may be possible to simplify the design problem by discarding trees completely and only using segments. If the trees can be replaced with segments that are able to do the job of efficient protection almost as well, it may be possible to do so without incurring much of a cost penalty. We will return to this idea later.

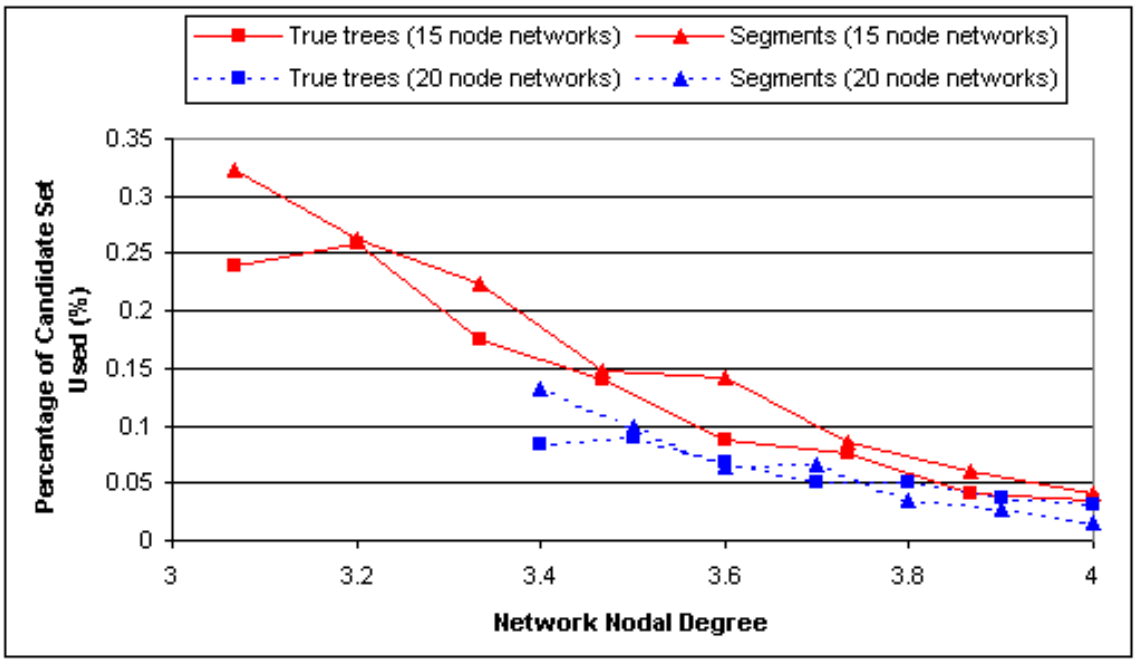


Figure 3.8: The percentage of the candidate structure sets that are actually used in the network designs

Figure 3.9 gives another comparison of tree usage versus segment usage, similar to that given by Figure 3.7, except that this Figure is a plot of the actual number of capacitated structure copies of each type. This differs from the plot of unique structures because a single unique structure represents a topological template that may be instantiated as multiple unit capacity structures in the spare capacity layer of the network. Figure 3.7 represents the topological templates while Figure 3.9 represents the capacitated structures.

This Figure shows a similar trend as the one seen in Figure 3.7, only the decreasing importance of segments in more highly connected networks is even more exaggerated. This data can be thought of as more closely representing the importance of each type of structure to the network design, because it takes into account the actual capacity allocated to each type rather than just the diversity of different topological structures within the tree and segment categories. From this point of view, segments and trees are of roughly equal importance in the least dense networks, but in the most densely connected networks segments are reduced to merely a supplement to degree-3 trees, taking up only 10 to 15% of the total number of capacitated structures. Therefore we can expect that attempting to eliminate trees in the more highly connected designs may incur a higher cost penalty.

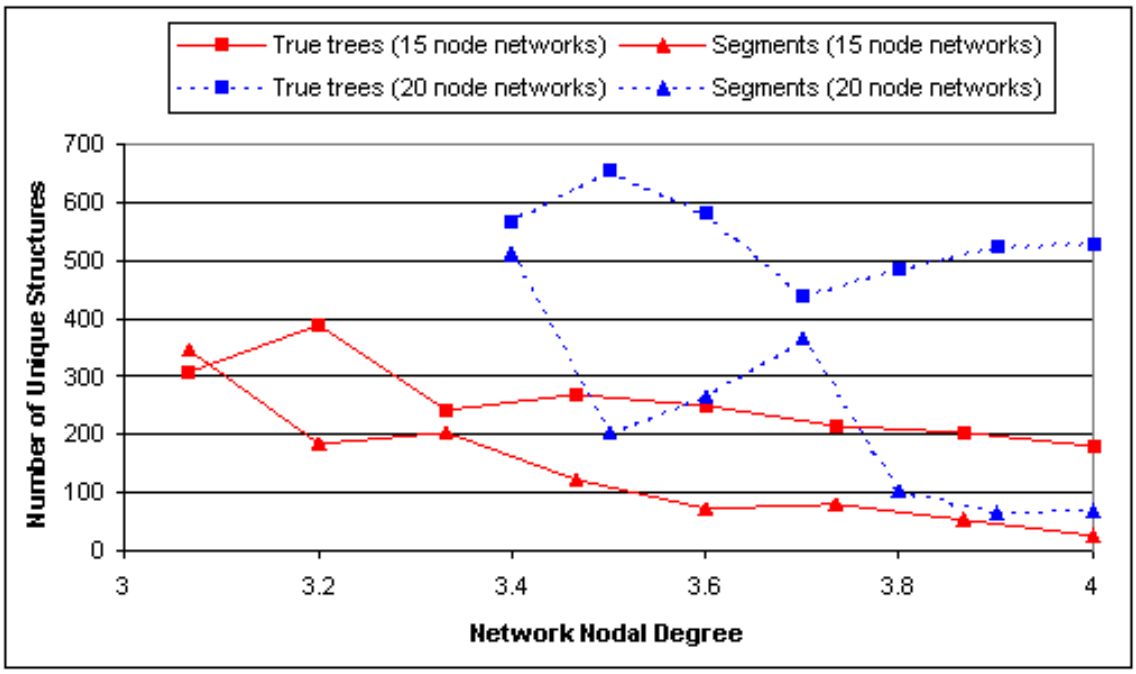


Figure 3.9: The usage of "true" degree-3 trees vs. degree-2 linear segments in the pure *p*-tree designs, represented by the number of structure copies used

3.4 Hybrid *p*-Tree and *p*-Cycle Designs

3.4.1 Motivation

We have already seen that it is difficult to design pure *p*-tree networks using "complete" sets of trees as candidates (i.e., the entire set of trees under certain size and nodal degree limitations), because the size of the set of trees that is required even to obtain a feasible design is in some cases impractically large. Furthermore, even in cases where designs can be obtained, these designs are impractically inefficient. A solution to both of these problems is to hybridize *p*-trees with an architecture that uses both manageably-sized and efficient sets of candidate structures, such as *p*-cycles. If a feasible *p*-cycle design exists for a certain set of candidate cycles, then a design with equal or lesser cost is also guaranteed to exist when the solver is presented with the option of improving on the cycle-only design by using trees for protection as well. In this way, the hybrid tree/cycle designs can be viewed as a method that uses *p*-trees as a supplement to *p*-cycles. Considering the problems with using pure *p*-trees encountered above, this may be the best way to view tree-based architectures going forward.

The other benefit of solving for hybrid tree/cycle designs is that it provides a fair, objective way of making comparisons between the two architectures. The exercise is completely free from human intervention; the ILP solver itself is the one that discriminates between trees and cycles by simply determining how many of each belong in an optimal network design. After the solver determines this, analysis can be performed on the composition of the resulting design to draw conclusions about the strengths and weaknesses of each architecture. It is, in essence, a way of having the two types of structures "compete" with each other on an even playing field in order to determine how they each contribute to efficient span protection.

3.4.2 Method and Test Cases

To reiterate, as mentioned above, the design method involves using the same model as before (from Section 3.2.1), and populating the set P with both candidate cycles and candidate trees. The solver then uses these structures to come up with a solution including some combination of both p-cycles and p-trees. Using this method, hybrid designs were found for all of the test cases used for the pure span p-tree architecture designs. As mentioned in Section 3.2.1, this was accomplished by simply including both trees and cycles in the set of candidate structures for the standard p-cycle SCP ILP model. The candidate sets given to the model for each of the tests were formed by combining the tree and cycle candidate sets from both of the corresponding pure architecture tests. In other words, all network cycles were used, as well as all trees containing 7 or fewer spans and with a nodal degree limit of 3.

3.4.3 Results and Analysis

None of the hybrid designs were able to surpass the efficiency of the pure p-cycle designs by more than 1.6%. Therefore p-trees seem to be totally outclassed by p-cycles for span protection. Despite this fact, some of the optimal designs did contain significant numbers of trees (up to 33%, 73 tree copies out of a total of 200 capacitated structures, in the 20 node, 36 span network). This means that even though trees are not efficient in general, it is possible for select p-trees in select problem instances to be used in a way that is no worse than the use of p-cycles for the same problem instance. There is some interest in

inspecting these trees to see if they have any common characteristics that identify them as the best examples of p-trees. Before doing so, however, we will describe some experiments that were performed to ensure that the presence of trees in these designs was necessary to achieve optimality and not a result of the solver choosing among equivalent-cost optimal designs. Performing this check first lets us analyze the chosen trees with certainty that their selection for inclusion in the network design over p-cycles represents a cost-reducing, rather than arbitrary, decision by the solver.

3.4.4 Further Experiments and Results

We have seen that some optimal hybrid designs were able to use a large number of p-trees, though they did not significantly improve on pure p-cycle designs. This leads us to suspect the possibility that, due to the large variety of choices provided to the solver because of the large size of the set of candidate trees, many designs of equivalent minimum cost may actually exist, some of which use a significantly larger number of p-trees than others. To test this possibility, the previous tests were run again using a model with a slightly modified objective function:

Minimize

$$\sum_{j \in S} C_j \cdot s_j - \alpha \cdot \sum_{k \in T} n_k \tag{3.4}$$

Where T is the subset of P that consists of all trees in P. This transforms the model into a *bicriteria* formulation that minimizes the network cost while also trying to maximize the total number of trees used in the design. The value of α sets the trade-off between network cost and total tree count. In our tests the value of α was set to 1. Because the costs of the network designs involved were all significantly greater than 1, this tells the solver to find designs with essentially the same costs as the designs found in the previous experiment, only using the greatest number of trees possible. Therefore this experiment tests for the existence of the postulated equivalent-cost designs with more trees by trying to solve directly for them. If, however, the resulting designs have the same cost with the same number of trees and cycles, we can be sure that the designs are unique.

The designs produced by this experiment, however, were almost exactly equivalent to those produced by the unmodified model. *Only one* design used more trees than the original, and even then it used only *one more tree* with the same number of cycles. Therefore

there are no designs that are both cost equivalent to the original designs and that contain a significantly larger proportion of trees.

The same tests were all performed again one final time using another modified objective function:

Minimize

$$\sum_{j \in S} C_j \cdot s_j + \alpha \cdot \sum_{k \in T} n_k \tag{3.5}$$

This is identical to the last objective function except that the tree-weighting term is given a positive sign instead of a negative sign. This means that when α is set to some small positive value (again, 1 in our tests), the formulation now tries to minimize the total number of trees instead of maximizing it (while still finding an optimal cost solution). The results of these tests tell us whether or not p-trees can be partially or completely expunged from optimal hybrid solutions.

However, the results of this test were again almost identical to the original results. The greatest difference was that in one design (the 20 node, 36 span network) a solution was found using 71 trees and 129 cycles instead of 73 trees and 127 cycles. Also, one design (the 20 node, 31 span network) actually used one more tree than the design found with the original model. The explanation for this anomaly is that the default mipgap of the solver, although very small, is nonzero. Thus the solver was able to come within this very small mipgap in this anomalous case while still using more trees than in the original solution, and terminated. However, all other cases solved to complete optimality (0% mipgap at termination) and thus this one trial is the only anomalous case.

This one anomaly does not affect our conclusion that there do not exist cost-equivalent designs that contain fewer trees than the original hybrid designs. Combined with the results of the last test, we may conclude that the minimum cost hybrid designs for these networks are unique and represent the true optimal proportion of trees to cycles to attain networks of absolute minimum cost.

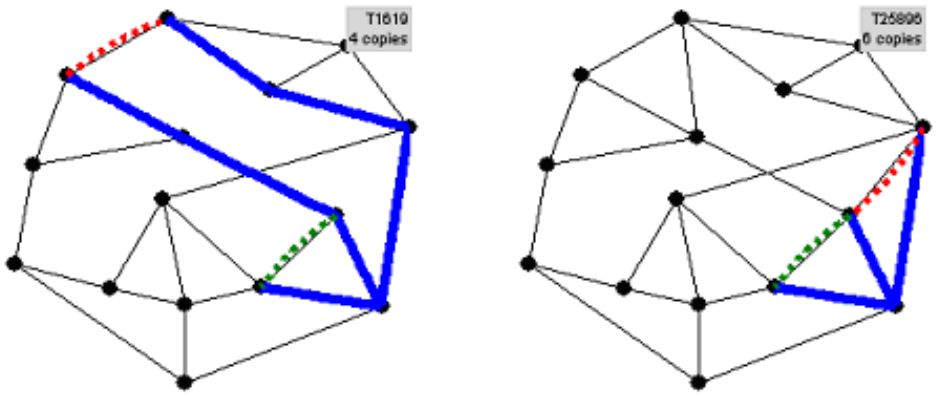
3.4.5 Visualization of Efficient p-Trees

Because the uniqueness of the optimal hybrid designs has been established, we can conclude that the p-trees that are present in these designs are in some sense "as efficient" as

p-cycles, even if it is only in the context of these specific solutions. Therefore the structure of these trees merits investigation to determine if any traits of "good" trees can be abstracted from their commonalities.

The trees from the hybrid designs were extracted and drawn on top of the network topology diagram using an automated process. In these (and other structure diagrams included in this thesis), the structures themselves (i.e., *p*-trees or *p*-cycles) are indicated by the solid blue lines, while the spans they protect are drawn using dashed lines of various colours. These diagrams are included in Appendix C. The most obvious feature of this set of diagrams is that in all but two of the cases the "trees" are actually only degree-2 segments. This is quite different from the pure *p*-tree design case, in which the proportion of true trees was normally quite large, with segments being in the minority. Therefore it seems that adding *p*-cycles to the mix of structures has the effect of changing the relative efficiency of segments versus true trees. In the presence of the generally superior *p*-cycles, segments must have characteristics that enable them to remain attractive for occasional use more often than the degree-3 trees. We will return to this idea later when we investigate segments in more detail.

In addition, all of the segments used in the hybrid designs are used for the protection of only a single span. In other words, they function in effect as dedicated 1:1 APS arrangements, only used for span protection instead of end-to-end path protection. Even though they could be closed with only the addition of a single span to transform them into *p*-cycles, the details of the design problems must be such that in these specific cases the added benefits of *p*-cycle protection would not make up for the addition of even this small amount of spare capacity. This idea is supported by the observation that none of these degree-2 segments, if closed into cycles by adding a single span, would have any straddling spans. Straddling spans are generally thought to be essential to the high efficiency of *p*-cycles [Stam97]. Therefore, these 1:1 APS segments represent cases in which the transition from *p*-tree protection to *p*-cycles is the least attractive. This is evidence that APS-type dedicated protection arrangements can in fact be an important part of mesh-based efficient preconnected network designs. These results are supported by those found prior in [WuYe08]; even though their experiments are less thorough and more limited in scope than ours, their diagrams clearly show the same pattern of protection provided mostly by cycles, with var-



(a) Tree from the 15 node, 25 span network (b) Tree from the 15 node, 27 span network

Figure 3.10: The two "true" trees from the hybrid tree/cycle designs

ious dedicated APS-like segments for a small number of spans.

We now turn to an analysis of the two "true" trees (trees containing junctions of degree-3 or higher) in the network designs, in order to determine why these and only these trees are used in the optimal designs. These trees are found in the designs of the 25 span and 27 span networks from the 15 node network family. Diagrams of these trees can be found in Appendix C with the rest, but are also given in Figure 3.10 (a) and (b) respectively. Looking first at tree (a), we see that it is used to protect two spans. One of these spans uses a protection path in the tree consisting of only two hops, the smallest protection path that a span can ever use. The other span, however, uses a long path that traverses the diameter of the network twice over. We can see that this span must take such a long path because no shorter paths exist between its end-nodes once the span has failed. Now, consider if this span was protected by a segment composed only of this protection path, instead of the tree in Figure 3.10 (a). In other words, consider if the extra span that is attached at the degree-3 junction at the bottom of the tree were removed. In this case, adding the protected span itself to the segment would create a p-cycle that could protect not only the original span but also all of the six newly on-cycle spans with only a small increase in cost.

In contrast, consider the choice actually made by the solver, in which the segment is not closed but instead has an extra span added to make it a tree. This span only allows for the protection of one more span, as opposed to the 6 on-cycle protection relationships that

would be gained if the segment were closed. So we can see that both our hypothetical cycle and the tree in Figure 3.10 (a) have the same approximate cost, yet theoretically the cycle can protect more capacity. It is simply the peculiarities of this network design that make this a priori inefficient decision into an efficient one in this specific context. However, we can understand why the tree protection is more likely to be used in this case by making a few observations. First of all, notice that, as with the 1:1 APS segments discussed before, our theoretical *p*-cycle would not have any straddling spans and therefore would not be able to achieve the characteristic *p*-cycle 2-to-1 protection relationship for any span. Also notice that the upper protected span cannot be protected as a straddler of any cycle. No matter what structure protects it, it will do so in a 1-to-1 manner. Therefore we can explain the use of this tree not by the fact that it is so intrinsically efficient, but because the conditions for *p*-cycle use in this situation are so poor that the a priori estimated efficiency of a tree becomes comparable.

Moving to Figure 3.10 (b), we can see that the presence of 2 additional spans in this network means that many shorter protection paths are introduced for the upper protected span from Figure 3.10 (a). This explains the fact that the same near-segment type of protection is no longer used for this span, and that there is some *p*-cycle that protects it instead. But the tree that is present instead is a smaller tree that still makes up a subset of the spans in tree (a). This tree, like tree (a), also protects 2 spans: the lower span protected by (a) as well as a span that is present in the 27 span network but not the 25 span network. Interestingly, these two spans are the first and fourth most highly capacitated (with working capacity) spans in the network design. Therefore it makes sense to use a very efficient structure to protect them. By inspection, we can easily find many *p*-cycles that could protect these two spans. However, in order to protect either of them as straddling spans, the cycles would have to be large, due to the nature of the topology, negating the advantage of 2-to-1 protection. It is possible to protect them as on-cycle spans with smaller *p*-cycles, but to do so would still be less efficient than the pictured tree. For example, the smallest cycle containing both spans is still 4 spans long, while the pictured tree contains only 3. Therefore *p*-cycle use is again made difficult in this case, this time due to a combination of topological and working path routing factors.

So in both cases, as with the degree-2 segments, we can explain the use of degree-3 trees

over cycles as a result of situations in which the usual advantages of *p*-cycle protection are negated. In these exceptional situations, the solver is occasionally able to make a gain by reaching outside the set of *p*-cycles to consider structures that normally would not be very efficient. However, we have seen that *p*-cycles are sufficient for cost-optimal span protection in the vast majority of cases.

3.5 Pure *p*-Segment and *p*-Tree Design Comparison

3.5.1 Motivation

We saw initially that a significant proportion of the structures in a pure *p*-tree design can actually be degree-2 linear segments, especially in sparser network topologies. Furthermore, the experiments in the previous Section have shown that the majority of "trees" chosen for inclusion in hybrid *p*-cycle/*p*-tree designs are in fact only degree-2 structures, like span-protecting PXTs. To distinguish these structures from PXTs (which have been established in the literature as path-protecting structures only) and to extend terminology from existing literature, we shall call these span-protecting segments p-*segments*.

The fact that trees of degree higher than 2 are seldom used in the hybrids raises the possibility that the addition of more complex trees on top of simple *p*-segments might not provide much of a cost improvement. In other words, most of the efficiency of *p*-trees may in fact be achievable using only the significantly smaller set of *p*-segments, just as we have seen that the maximum possible efficiency of the overall preconnected span protection concept is encapsulated almost entirely in the relatively small set of *p*-cycles. To investigate this possibility, pure *p*-segment designs were compared with *p*-tree designs.

3.5.2 Experimental Method

The pure *p*-tree network design experiments were repeated with *p*-segments in place of *p*-trees. This was easily accomplished by using the same tree generation algorithm as with the *p*-tree tests, only with the tree degree limit set to 2. The size limit was again set to 7 for all tests, the same as in the pure *p*-tree designs, in order to reproduce the previous test set with the all degree-3 structures removed. These tests let us determine the difference created by the presence or absence of the set of tree structures containing degree-3 nodes.

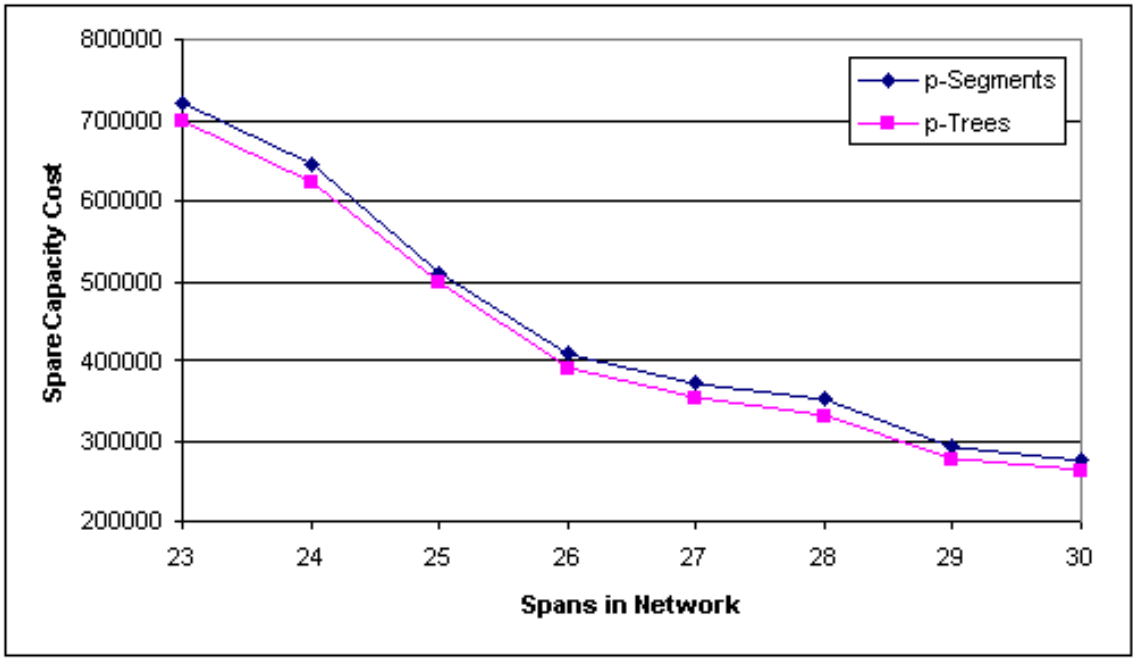


Figure 3.11: Spare capacity costs of *p*-tree and *p*-segment designs for the 15 node network family

After this, another experiment was performed in which the size limit for segments was increased in order to give the *p*-segment design problems a number of candidate segments roughly equal to the number of trees in the *p*-tree design problems. To accomplish this, a size limit of 9 spans was used for all of the 15 node test networks and a limit of 10 spans was used for the 20 node test networks. The results of these tests were used to compare the overall efficiency of segments versus full trees when given comparable numbers of candidate structures.

3.5.3 Results and Analysis

3.5.3.1 *p*-Segments vs. *p*-Trees with the Same Size Limit

Plots of the costs of the *p*-segment and *p*-tree designs for the 15 node and 20 node networks are shown in Figure 3.11 and Figure 3.12, respectively. Because the *p*-segment designs only use a degree-2 subset of the structures used in the *p*-tree designs, each *p*-segment design must always be either the same cost or more expensive than the *p*-tree design for the same network.

However, the results show that *p*-segments are not very much worse at all. The 15 node

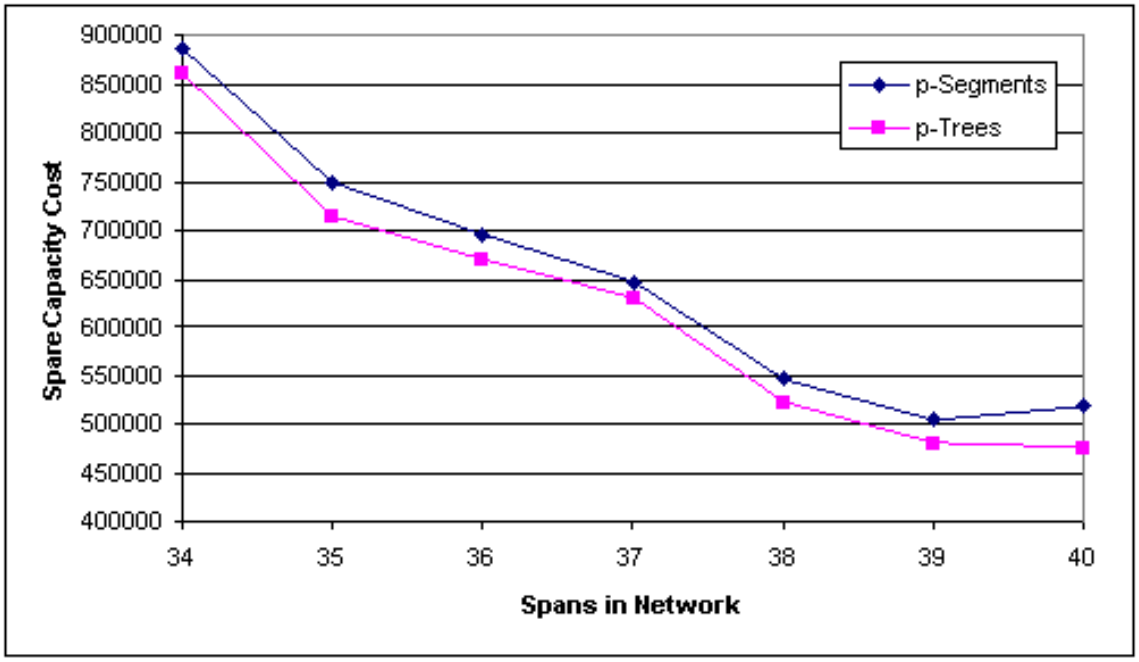


Figure 3.12: Spare capacity costs of *p*-tree and *p*-segment designs for the 20 node network family

p-segment designs are at most only 6% more costly than the corresponding *p*-tree designs. The corresponding figure for the 20 node designs is only 8.6%. Note that this is true even for the highly connected networks, even though it was found earlier that segments make up only 10 to 15% of the total number of capacitated structures in the *p*-tree designs for these networks. Therefore, even though tree-based designs are not efficient in general (for example, as compared to *p*-cycles, as we have seen), most of the efficiency they do have seems to be able to be carried by the degree-2 structures. This is even more significant when one considers that degree-2 structures consist of only about 15% to 20% of the candidate structures provided to the design problems for the most highly connected test networks, where high nodal degree causes the size of the set of purely degree-3 structures to increase the most dramatically.

3.5.3.2 *p*-Segments vs. *p*-Trees with Similar Candidate Structure Set Sizes

Figure 3.13 and Figure 3.14 show the results of the second experiment in which the length limit on the *p*-segments was increased such that the numbers of candidate segments were approximately equal to the number of candidate trees in the *p*-tree designs. First of all, the

p-segment curves extend further than the *p*-tree curves because feasible solutions exist for more of the networks. This is because of the higher size limit on candidate structures. As we have stated, limiting the size of trees to 7 spans means that, in the sparser networks, there are no trees that are able to protect some of the network's spans. When the size limit for the segments is extended to 9 (in the 20 node networks) or 10 (in the 15 node networks), far more spans can be protected in the sparser networks. Therefore in the *p*-segment case, the only networks containing unprotectable spans are the 15 node network with 16 spans and the 20 node networks with 21 and 22 spans. This is another advantage of *p*-segments over *p*-trees; the smaller size of the structure set means that longer structures can be included more easily.

Whereas the first test evaluated the contribution of *p*-trees on top of *p*-segments, this second test is more of a competition between the two architectures, due to the equalized size of the structure sets. The *p*-segment designs are not allowed to use degree-3 structures, but they make up for this by using more degree-2 structures. Therefore it is not guaranteed that one or the other will be more efficient in any particular case. However, in the networks where comparable *p*-tree designs exist, we see that the *p*-segment designs usually have a lower cost than the corresponding *p*-tree designs. In the set of 15 node networks, the *p*-segment designs are between 3.6% and 11.4% less costly than their corresponding *p*-tree designs. In the set of 20 node networks, the *p*-segment designs are up to 11% less costly. However, it must be noted that one of the *p*-segment designs (for the most dense 40 span network) is actually 4.6% more costly.

As mentioned above, the method used to produce appropriately sized candidate *p*-segment sets (experimentally varying the structure size limit) was only approximate. Therefore the numbers of candidate structures used in each pair of *p*-tree and *p*-segment tests for any given network were never exactly the same. Figure 3.15 and Figure 3.16 show the numbers of such structures used on a logarithmic scale for the 15 node and 20 node designs respectively. The sizes of the two sets are quite close over all of the 15 node networks, but differ more for the 20 node networks. It is more difficult to attain a close match in the 20 node case because the curves have significantly different shapes; the curve for trees "bulges" up more in the middle, so that the sizes of the sets are more similar at the extremes of nodal degree and differ more in the intermediately sized networks. However, in no case

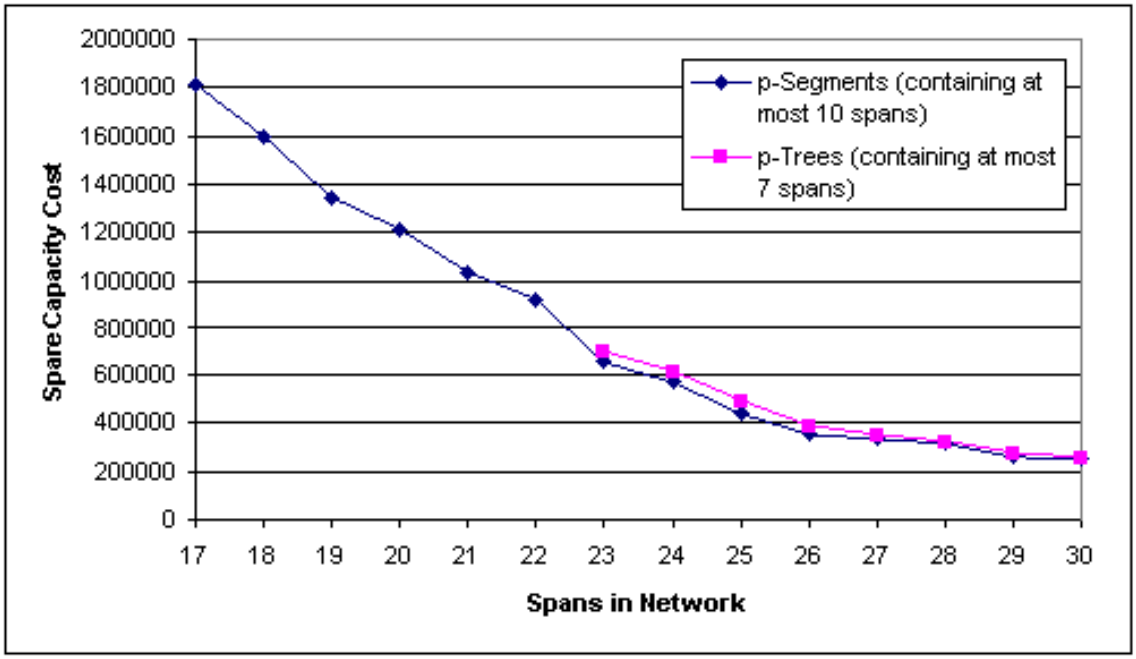


Figure 3.13: Comparison of *p*-segment and p-tree designs (for the 15 node network family) that use approximately the same number of candidate structures

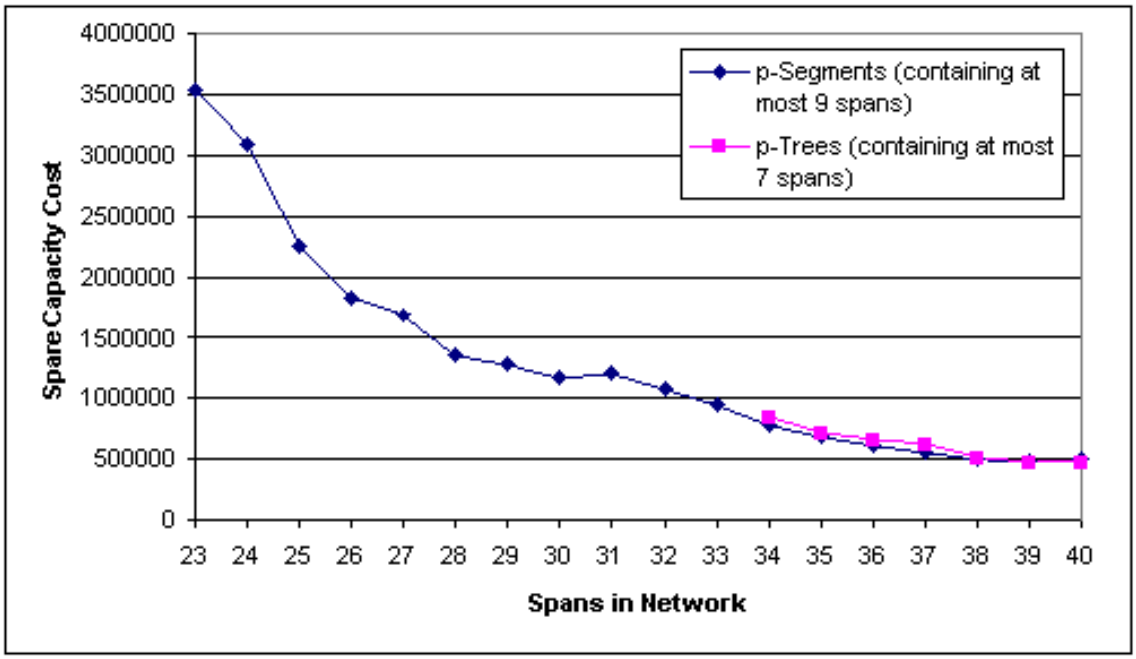


Figure 3.14: Comparison of *p*-segment and *p*-tree designs (for the 20 node network family) that use approximately the same number of candidate structures

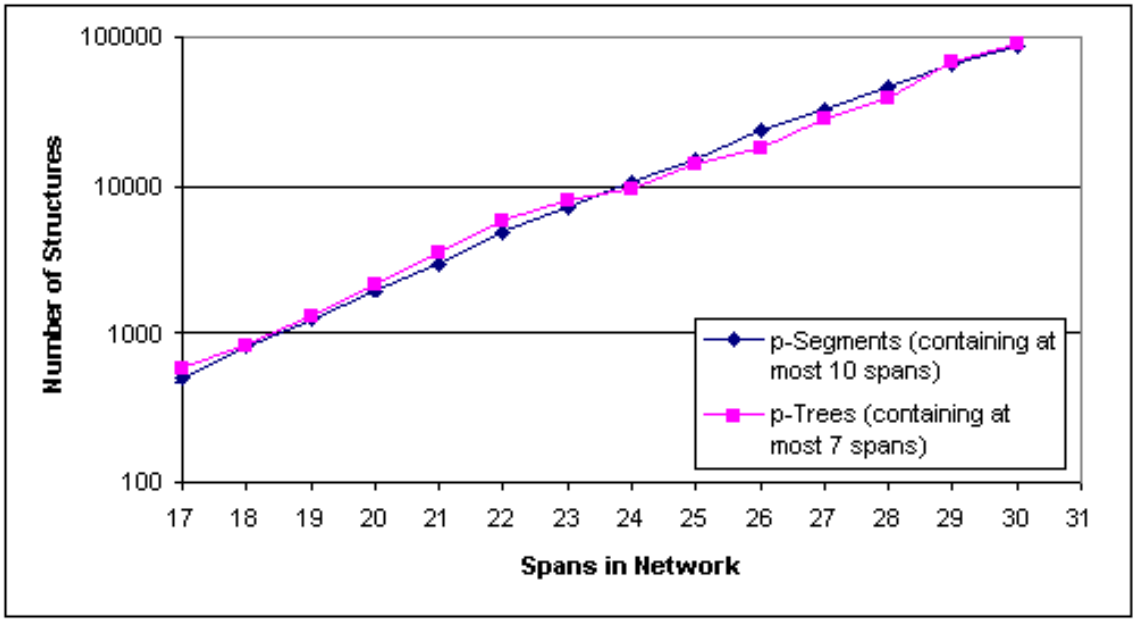


Figure 3.15: The sizes of the candidate structure sets used by the *p*-segment and *p*-tree designs with equalized candidate set sizes for the 15 node network family (Figure 3.13)

is the set of segments for the 20 degree networks larger than the set of trees. This makes it even more remarkable that *p*-segments are usually able to outperform *p*-trees in these networks.

These two exercises have demonstrated that using true *p*-trees instead of just degree-2 *p*-segments represents a type of diminishing returns. Increasing the degree limit on the structures significantly increases the size of the possible structure set, but only allows a small (5% to 10%) decrease in cost. Furthermore, replacing the degree-3 structures with roughly the same number of larger degree-2 structures decreases cost in almost all cases. This is fairly conclusive evidence that structures that utilize degree-3 preconnections are not generally efficient when used for span protection.

3.6 Hybrid *p*-Segment and *p*-Cycle Designs

3.6.1 Motivation

Now that we have examined the performance of *p*-segments, both as a pure architecture and when included as a subset of the *p*-tree architecture, we would like to investigate the performance of *p*-segments when hybridized with *p*-cycles, both to compare the *p*-segment

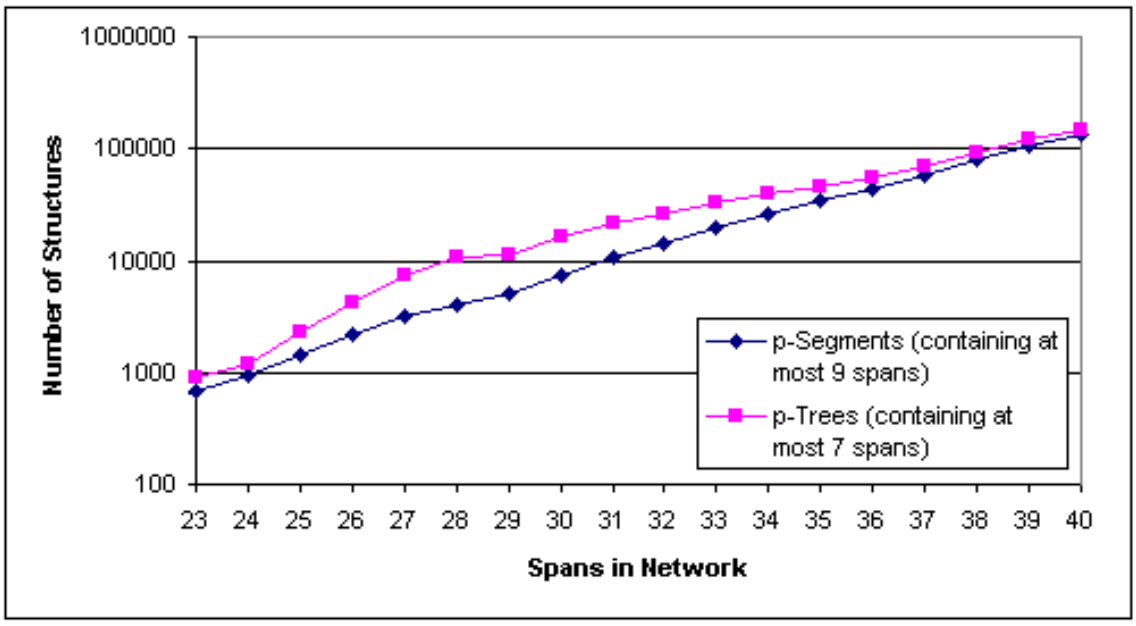


Figure 3.16: The sizes of the candidate structure sets used by the *p*-segment and *p*-tree designs with equalized candidate set sizes for the 20 node network family (Figure 3.14)

and *p*-cycle architectures and to make comparisons with the previous *p*-tree/*p*-cycle hybrid designs.

3.6.2 Test Cases

The test cases for this experiment consisted of all of the *p*-segment design problems from the previous Section, with the set of all network cycles added as candidate structures in every case. In other words, hybrid designs were computed for all of the 15 node networks using *p*-segment size limits of both 7 and 9 spans, and for all of the 20 node networks using *p*-segment size limits of both 7 and 10 spans.

3.6.3 Results and Analysis

3.6.3.1 *p*-Segment vs. *p*-Tree Hybrids with the Same Size Limit

First we will perform a comparison between the *p*-segment/*p*-cycle and *p*-tree/*p*-cycle hybrid designs that both use size limits of 7 for their segments/trees respectively. It turns out that the differences between the design costs of these two sets of results are very small, and in fact zero in most cases. First of all, any *p*-tree hybrid designs that did not actually use any degree-3 trees are not affected by the shift to *p*-segment-only hybridization, as all of

the original structures used are either segments or cycles, and thus the same designs remain optimal in the p-segment-restricted problem specification. Therefore the only networks for which differences should exist are the ones that used degree-3 trees in the original hybrids, i.e., the 25 and 27 span networks in the 15 node network family. Looking at the results, there is indeed a difference between the p-tree and p-segment hybrid design costs for both networks. However, the difference in the 25 span case is so minuscule that it would not be noticeable were the differences in most of the other networks not identically zero; the difference in this case is only approximately 1.2 out of a total cost of about 183000. The difference in the 27 span case is similarly small: about 100 out of 141000.

Both pairs of designs merit in-depth investigation to determine what rearrangements of structures were performed to find designs of such similar cost. To do this, we again used an automated process to draw diagrams of all the structures (both cycles and trees/segments) used in the designs. The structure diagrams are given in Appendix D. We have already discussed the p-cycle/p-tree designs and included their tree diagrams in Appendix C, but Appendix D gives complete diagrams for these two networks in particular, including all of their structures, trees and cycles alike. For the 25 span network, even though the two types of hybrid designs differ by less than a thousandth of a percent in cost, they use quite different sets of p-cycles, sharing only 4 unique p-cycles in common (out of 11 and 10 total unique cycles for the p-segment and p-tree design respectively). As for trees, the p-tree hybrid uses only a single degree-3 tree and a single segment, which are topologically similar; they differ only by a single additional span. The p-segment hybrid uses this same segment, meaning that the it remained efficient despite significant changes to the p-cycle configuration. Additionally, together the tree and segment from the p-tree design contribute a total of 15 protection units to the protected span that they share in common (due to their topological similarity). Then, when the tree is made unavailable in the p-segment design, the number of copies of the segment is increased to 15, meaning that 15 units of non-cycle protection is used in both designs to protect the same span. In retrospect, this is what we would expect, as the poor efficiency of trees and segments in general means that very few different possibilities for efficient tree-based protection will exist in a given topology. Therefore we would expect "tree-like" protection to occur to a similar degree in differently constrained designs for the same topology.

In the 27 span case, the *p*-tree design contained copies of a single degree-3 tree (the same tree already depicted in Figure 3.10 (b)). The *p*-segment hybrid design, however, is composed entirely of cycles. We might have expected that the solver would be able to find a segment very similar in structure to the tree and use that instead, as we saw in the 25 node *p*-tree and *p*-segment hybrid designs. But this is not the case, so evidently we cannot always expect that the presence of a tree in a design indicates some fundamental deficiency of cycles that requires a non-cyclical structure to correct. The lack of a transition from a tree to a similar segment in this case likely cannot be given a simplified, intuitive explanation; the use of other structures to make up for the lack of trees depends entirely on the complex interactions of the protection capabilities of both the entire set of segments and the set of cycles.

3.6.3.2 *p*-Segment vs. *p*-Tree Hybrids with Similar Candidate Structure Set Sizes

As noted in the previous Section, all *p*-segment hybrids were identical to their corresponding *p*-tree hybrid where the *p*-tree hybrid did not actually contain any true *p*-trees. We have seen that the 25 span and 27 span *p*-tree hybrids from the 15 node family do contain true *p*-trees. However, the *p*-segment hybrids for these two networks do not change even when the size of the candidate segment set is increased such that the *p*-tree and *p*-segment hybrid problems are given roughly the same number of structures. In other words, the added number of larger segments provides no benefit to the hybrid design problems in these cases. Therefore no new comparisons may be made; all the same observations made in the previous Sections apply, as the designs under consideration are identical.

3.6.3.3 *p*-Segment Hybrid Designs Using Different Numbers of Candidate Segments

We can compare the two types of segment designs to each other in order to gain some insight into how increasing the number of segments provided to the hybrid problem can increase the efficiency of the designs obtained. It was mentioned previously that no improvement was obtained in the two specific cases of the 25 and 27 span networks in the 15 node network family; this is also the case for 28 out of the 35 total test cases. Even in cases where an improvement in cost is found, it never exceeds 0.14%. This is unsurprising, considering that the improvement that the addition of segments/trees provided to basic

p-cycles in the first place was already very small (at most 1.6% as mentioned earlier). The fact that this experiment shows diminishing returns as the segment size limit is increased suggests that there is no significantly higher level of efficiency that is attainable with larger segments that would make them competitive with *p*-cycles in terms of efficiency. If this is in fact true in certain cases, it must occur at a higher segment length limit.

We have tested this possibility explicitly in 4 out of the 35 test cases: the 20 span and 26 span networks in the 15 node family, and the 25 span and 36 span networks in the 20 node family. This set includes both a sparsely connected and richly connected network from both families, so as to allow us to test the effect of the *p*-segment length limit under different conditions without having to repeat the test on every one of the 35 different networks. For each of the test networks, the *p*-segment/*p*-cycle hybrid design problem was solved multiple times, each time with a different size limit for the *p*-segments. In the 15 node/20 span network, the 15 node/26 span network, and the 20 node/25 span network, the problem was solved for *p*-segment length limits from 5 to 14, 5 to 14, and 5 to 19 respectively. These ranges extend all the way up to the maximum *p*-segment length in the respective networks; in other words, increasing the limit further would not admit any more *p*-segments to the design problem, as segments that long cannot exist in these networks. In the 20 node/36 span network, tests were run using length limits from 5 to only 10, as this network is so highly connected that increasing the limit past this point results in a problem so large that it is unsolvable.

The results given in Table 3.1 show that long *p*-segments are not useful in *p*-cycle-hybridized designs. In the two sparse networks, any segments longer than only 6 hops are not useful at all for increasing the efficiency of the design. When the networks are more connected, the maximum useful length limit increases, demonstrating that segments are slightly more useful in more densely connected networks. This is likely because a trail's efficiency relative to that of a cycle increases with connectivity; the relative incremental gain of adding an extra straddling span on a trail is higher than for a cycle because the cycle protects all of its on-cycle spans as well

Regardless, in none of the tests were any but the shortest of segments useful for improving design cost. In contrast, both small and large *p*-cycles were useful in all of the designs. We have seen already that tree- or segment-like protection supplements *p*-cycles

	Network			
	15 node, 20 span	**15 node, 26 span**	**20 node, 25 span**	**20 node, 36 span**
Maximum improvement from increasing length limit	0.055%	0.030%	0.32%	0.30%
Maximum length limit that provides cost improvement	6	7	6	9
Number of cycles in design problem	43	985	32	9800
Shortest p-cycle used in lowest cost solution	7 hops	8 hops	4 hops	9 hops
Largest p-cycle used in lowest cost solution	12 hops	15 hops	18 hops	19 hops

Table 3.1: The effect of the p-segment length limit on the costs of p-segment/p-cycle hybrid designs

by performing protection in very specialized cases where the usual efficiency of p-cycles does not apply. This new information tells us that, more specifically, p-segments tend to outperform p-cycles in these specialized cases only when the p-segments are short. We might have suspected this from the fact that long p-segments suffer (in comparison to p-cycles) from the lack of on-segment protection relationships, but this investigation provides experimental confirmation of our intuition.

This result is especially significant given that the same effect does not occur in the pure p-segment designs. As we found earlier, in the pure designs increasing the length limit on the set of segments from 7 to even just 9 or 10 decreases the cost in most networks to the point that the segment designs are better than p-tree designs found using equally sized candidate structure sets. In the hybrid designs, however, increasing the length limit has almost no effect, as Table 3.1 shows. Therefore this is another instance in which the presence of p-cycles changes the properties of other structures in a relative sense. We have seen previously that the addition of p-cycles to the p-tree design problem increases the usefulness of the simpler p-segments over true degree-3 p-trees, and now we see that adding p-cycles to p-segments alone has the effect of making them shorter, simplifying the segments even further.

3.7 Summary and Future Work

In this Chapter we have created and examined minimum cost designs for span-protecting *p*-trees and *p*-segments, both with and without hybridization with *p*-cycles. The results have shown that degree-3 trees by themselves are extremely inefficient in comparison with *p*-cycles. Furthermore, even when trees are hybridized with *p*-cycles, the resulting designs show only an incremental improvement over designs that use *p*-cycles alone. This, combined with the fact that the set of trees is orders of magnitude larger than the set of cycles, and the fact that the implementation of preconnected degree-3 branches is a questionable concept in the first place, means that span-protecting *p*-trees are not an attractive standalone protection architecture. Their implementation should only really be considered as a supplement to more efficient or simple schemes in cases where efficiency is of such paramount importance that the incremental cost savings would outweigh all of the difficulties.

However, trees remain interesting from a theoretical perspective because they are able to improve on *p*-cycle protection in select instances. We have found that *p*-trees are used instead of *p*-cycles only in peculiar situations where it can be seen that the normal advantages of *p*-cycles (namely on-cycle protection and the 2:1 straddling span protection relationship) do not apply. Furthermore, we have found that when hybridized with *p*-cycles, the type of trees that are used in the solutions is for the most part limited to the subset of degree-2 segments. Further experiments have shown that these segments tend to be both short and dedicated to the protection of a single span. Therefore the main purpose of "tree" protection in *p*-cycle hybrid designs is to provide small patches of dedicated APS-like protection in cases where efficient *p*-cycle protection for those spans cannot be found.

Perhaps the most interesting higher-level result we have obtained from these experiments is the observation that the hybridization of *p*-tree structures with *p*-cycles fundamentally changes the relative attractiveness of certain tree structures over others. In the degree-3 *p*-tree case, adding *p*-cycles results in the dominance of segments over degree-3 structures in the solution, whereas in pure *p*-tree designs degree-3 structures are in the majority. Then, even when the design is limited to using degree-2 segments, the presence of *p*-cycles prevents all but the shortest of segments from being useful, whereas segments of all lengths are present in pure designs. These observations can be tied together by the

stipulation that *p*-cycles are such good protection structures by themselves that any type of supplementation with other structures requires only the simplest and smallest of non-cycle-based additions or changes. Interestingly, this observation agrees with statements in [LiYa03] concerning the efficiency of trees, namely that the authors found that it was best to construct their spanning trees with fewer branches, rather than more, by using a depth-first search. Furthermore, in Figure 1 of [LiYa03], which shows two example spanning trees, the two trees are both examples of segments that would form (Hamiltonian) *p*-cycles, save for the absence of a single span in both cases. Therefore our observed hierarchy of cycles over segments over trees is not without precedent in the literature, though in reference to a concept quite different from our own *p*-trees.

The next step in the research of span-protecting *p*-trees would be to expand the network design problem to consider joint placement of working and spare capacity (i.e., JCP as opposed to SCP design). While the results obtained so far all indicate that trees are very inefficient, it is possible that they have more significant benefits when working routing is allowed to deviate from the shortest-path routing that is assumed in the above SCP formulations. The other logical direction in which to proceed (and which this thesis will take in Chapter 5) is to consider trees for the protection of end-to-end paths instead of just individual spans. Previous work, along with the work presented in the following Chapter, has shown that linear segment-based path protection using PXTs can actually be a relatively cost-effective approach, even when compared to efficient schemes like *p*-cycles. Therefore tree-based protection may also become more viable when applied to path protection.

Chapter 4

Characterization of Pre-Cross-Connected Trails

4.1 Introduction

This Chapter describes our investigations into the concept of *pre-cross-connected trails*, or PXTs. First we present the PXT concept. Unlike *p*-trees, PXTs stood as a well-defined, preexisting architecture before our investigations, so we begin by explaining the concept as it stood prior to our research. We then continue our investigations using the established greedy heuristic design method to more fully determine its implications. We then present our own design approach based on ILP and demonstrate how it is able to produce designs with superior properties. Finally we introduce some variations on the ILP approach and investigate their implications on the efficiency and other properties of PXT designs.[1]

4.1.1 Background

4.1.1.1 PXT Definition

PXTs are linear (degree-2) preconnected path-protecting structures. A PXT consists of a series of spare capacity units (spare channels) that are cross-connected to each other forming a *trail* through the network. A trail is formally defined in graph theory as an alternating sequence of connected nodes and edges through a graph such that all edges

[1] Some of the work in this Chapter has been published in the Journal of Optical Networking:

A. Grue, W. D. Grover, "Characterization of pre-cross-connected trails for optical mesh network protection," Journal of Optical Networking, vol. 5, no. 6, June 2006, pp. 493-508.

A. Grue, W. D. Grover, "Improved method for survivable network design based on pre-cross-connected trails," Journal of Optical Networking, vol. 6, no. 2, February 2007, pp. 200-216.

are distinct. In the optical network context, a single span may contain multiple channels (or edges) so while a PXT may not contain a single channel more than once, it may cross a single span several times, as long as each traversal of this span is through a different channel within that span. This is unlike *p*-cycles, which are generally restricted to being simple cycles (i.e., cycles that do not cross the same node or span more than once). Their topological freedom allows PXTs to become quite complex.

A PXT protects a failed working path by providing an end-to-end protection path that is made out of a segment of the PXT's preconnected capacity. Under the assumption of single span failures only, a PXT is allowed to protect multiple working paths as long as any two working paths that can be affected by the same span failure do not use the same protection path on the PXT (i.e., as long as contention for spare capacity does not occur). Generally, a restorable PXT-based network design consists of many PXTs that provide complementary coverage to the network's working paths such that the network is fully restorable in the case of the failure of any single network span. PXTs may also be used to protect against node failures, but the work in this Chapter is focused exclusively on span failures.

4.1.1.2 Previous PXT Literature

The PXT definition given above was first introduced into the literature in [ChCh04]. The same work also outlined the description of an algorithm to perform PXT-based network design. This algorithm is a spare capacity placement algorithm only, meaning that it produces a spare capacity assignment and preconnection plan based on a fixed description of working routing given in the input. The specification of the algorithm given in [ChCh04] states that the input working routing is done as shortest-path routing, which is the usual assumption for spare capacity assignment methods. However, in theory the same PXT-building algorithm could be used for any arbitrary working routing assignment. In fact, we will see later that it is necessary in some situations to deviate from shortest-path routing in order to allow basic 100% single span failure restorability.

The proposed algorithm is also developed as an *online* algorithm, meaning that it is meant to be an algorithm that runs during the operation of the network to handle the protection of connection requests as they arrive dynamically. This is notably different from the common "green-fields" assumption used in much *p*-cycle design literature, for exam-

ple, under which entire networks are designed at once for the protection of a static list of demands. The algorithm from [ChCh04] instead performs a type of incremental design, extending existing PXTs or creating new ones to protect newly arrived demands in a low cost way. It does not, however, perform extensive revisions of the PXT arrangement in order to use the absolute minimum possible capacity at all times. Therefore its use does not generally result in minimum cost PXT designs. Instead, it is a *greedy algorithm* that attempts to produce a design with relatively low cost by following some simple heuristics.

In addition to providing an extensive description of the greedy heuristic PXT design algorithm, [ChCh04] gives some initial results for the amount of spare capacity used by designs produced by this algorithm for a small collection of test networks and demand patterns, and compares them against comparable results for 1+1 APS and so-called "Path Protection". However, the authors do not go into much depth in this section, devoting most of their space towards an explanation of the algorithm. Therefore many questions about the details of the PXT designs go unanswered: the simple capacity numbers given do not reveal any information about the underlying PXT structures used by the designs. Subsequently, the authors of [MaHa07] challenged the methodology of [ChCh04], providing a comparison between the PXT heuristic and other protection approaches that was both more thorough and more fair, but they did not investigate the properties of the PXT designs or any new approaches to designing PXT networks.

Shortly after the publication of [ChCh04], [KiLu04] was published, introducing a concept identical to PXTs under the name "Streams", seemingly independent of and unknown to the authors of [ChCh04]. Even more interestingly, [KiLu04] also introduces a design algorithm that mirrors that from [ChCh04] almost exactly. Therefore the same concept seems to have been co-discovered by these two independent groups almost simultaneously. However, [ChCh04] was published earlier, so we will continue to use the terminology and assumptions from this work going forward.

There is other work that deals with the concept of linear cross-connected segments while pre-dating the concept of PXTs as proposed in [ChCh04]. Specifically, [GrMa98, GrMa94, MaGr97] were written on the topic of performing partial preconnection of spare capacity in pre-computed restorable network designs in order to maximize the average speed of any restoration response. Although these works experiment with forming lin-

ear segment of preconnected capacity, they cannot be called PXTs, as dynamic cross-connections may be made between segments, or broken within segments, in response to failure in order to form protection paths. The idea in this case is to use knowledge about failure responses ahead of time to form preconnections where they will likely be needed in order to speed up the restoration response of the network. The idea of total preconnection as required for transparent networks was not yet introduced at this point.

The contributions to PXT research since the original paper have been sparse. [LiHa06] investigates using PXT protection to enable QoP classes with differentiated protection speeds. [WuYe08] addresses a concept that the authors call "PXTs", but which are actually span-protecting segments, which we would call "*p*-segments" in the context of this thesis. Therefore this work is discussed in the literature review of Chapter 3.

4.1.2 Goals and Objectives

The over-arching goal of our PXT investigations was to understand the PXT architecture more thoroughly to the degree that we would be able to make comparisons to other architecture, both novel (*p*-trees) and well-known (*p*-cycles). More specific goals included discovering the fundamental, characteristic capacity efficiency of PXTs (as opposed to the efficiency achieved by specific algorithms), determining the structural properties of efficient PXT designs, and developing a better PXT design method.

4.2 Heuristic Approach to PXT Design

4.2.1 Goals and Objectives

The impetus for our study of PXTs was the publication of [ChCh04]. As mentioned, it presented a PXT design algorithm without delving very deeply into the fundamental properties of PXTs. Therefore our first objective was to re-implement the PXT design algorithm described in [ChCh04] and inspect the designs generated by this method in more detail. A secondary objective was to test modifications to the algorithm to determine if it could be improved (either in terms of producing more efficient designs or designs with other desirable properties).

4.2.2 Heuristic PXT Algorithm Overview

This Section contains a description of our implementation of the greedy heuristic PXT design algorithm. The overview that follows is simply our own reinterpretation of the explanation given in [ChCh04], and is included both to demonstrate our understanding of the concept and to clarify our implementation in the case that there are any subtle differences between our interpretation and the implementation used by the original authors.

This heuristic is based on a greedy iterative approach that protects the unit demands of the network one-by-one, attempting to minimize the increase in spare capacity at each step. Each step consists of protecting an as-yet unprotected demand, considered in the context of already developed PXTs for prior demands. This may involve using only existing PXTs as they stand, extending certain PXTs, or even creating entirely new PXTs. This step-by-step approach is suited to the protection of demands that are provisioned dynamically as they arrive throughout the operational life of the network. This is opposed to the static "green fields" design approach, in which all demands are known (or projections of them are made) at design time, and are used to provision the initial network state (which may then be upgraded if sufficient demand growth is experienced in the future). Reference [ChCh04] refers to its algorithm as an "online" algorithm because it is run continuously during network operation (the green fields approach would usually be taken by "offline" algorithms only).

The PXT algorithm proceeds as follows. Initially the network design contains no PXTs and zero spare capacity. To protect the first demand, an initial PXT is created to serve as its dedicated backup path (the working path and its PXT together form the logical equivalent of a dedicated 1:1 APS arrangement). In subsequent iterations, the algorithm attempts to use existing PXTs in the network to protect other demands so as to minimize the incremental capacity required for each demand, extending the existing PXTs with additional spare channels if necessary. When extending existing PXTs is not the minimum cost option to protect the current demand, a new dedicated APS-equivalent PXT is generated for that demand and the design is iteratively developed from there. When all demands have been protected in this way, the result is a set of PXTs that defines an amount of spare capacity on each span and the pre-cross-connections that must be made such that PXTs are formed which are able to completely protect against all combinations of working path failures that

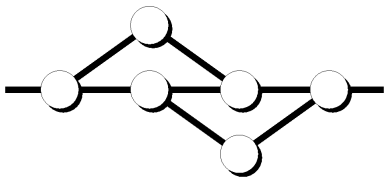
result from the failure of a single span.

The output of the algorithm is deterministic and does not contain randomization at any point (the choice of equal-cost protection decisions at each step might be arbitrary, but can still be implemented deterministically). However, the final output will depend on the order in which the demands are protected. It will produce the same final result each time it is run as long as the same demands are considered for protection in the same order. Because this ordering for the green fields problem is arbitrary but may affect the solution quality, the algorithm can be run multiple times, each with a different randomized demand order, when used in the offline sense, in order to avoid any artifacts that may be associated with a particular demand sequence.

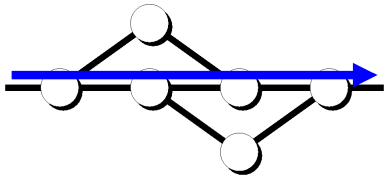
4.2.2.1 Working Routing Algorithm

Some words should be given to a brief description of the algorithm that we use to generate the working routes for demands before the protection algorithm described above can even begin. The algorithm is a form of minimum cost routing, modified slightly to reflect the requirements of the PXT architecture. Specifically, the PXT concept assumes that a demand has a single predefined protection path for its working path, embedded within the PXT that protects it. For this to be possible there must always exist at least one route between the end-nodes of a demand that is disjoint from its working route. But if working routing is done via a naïve shortest-path approach, this may not be possible. Our working routing algorithm is modified slightly to use shortest working routes except in those cases where this would lead to what is called the "trap," where taking the shortest path would cause there to be no other disjoint routes between the end-nodes to use as a protection path. In these cases, Bhandari's algorithm [Bhan99] (also described in [Grov03], pp. 210-211) is used to find the shortest combination of a working route and one other disjoint route that could act as a backup.

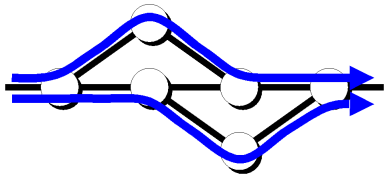
An example of the "trap" topology is illustrated in Figure 4.1. Assume span costs are proportional to distance, so the straight-through path in this topology is strictly the shortest path. Figure 4.1 (b) shows what would happen using naïve shortest-path routing. We can see here that a shortest-path working route eliminates any other route between the input and output. In Figure 4.1 (c), we can make protection possible by choosing a slightly



(a) The trap topology



(b) Falling into the trap



(c) Routing using the shortest cycle

Figure 4.1: Working routing for PXTs when a trap topology exists

"suboptimal" working path. We also see that the two paths here form the shortest cycle (here the only cycle) containing the input and output nodes. Another way to view the "trap" problem in the context of this particular design algorithm is that in each step the greedy heuristic PXT design algorithm removes the current working path from the graph to search for the lowest cost disjoint protection path for that working path. In trap situations, this would result in disconnecting the graph between the endpoints of the demand if we used naïve shortest-path routing.

4.2.3 Spare Capacity Efficiency of Greedy Heuristic PXT Designs

4.2.3.1 Experimental Method

Our first activity upon completing and testing the program was to use it to produce PXT designs for a variety of network topologies and demand patterns in order to characterize its capacity efficiency. We first attempted to reproduce the test cases used in [ChCh04] as a check to ensure that our implementation matched that of the original authors as closely as possible, before moving on to other test suites. Our interest at this point is only in the overall costs of the final designs produced by this algorithm for the protection of a fixed,

predetermined set of demands; the intermediate designs produced with each single step of the algorithm will not be considered here.

Reproduction of Original Test Suite

Reference [ChCh04] gives results in the form of network spare capacity costs for the final network designs produced by the algorithm for a set of 6 test networks, each under 3 different demand patterns. Unfortunately, these tests are not completely reproducible using the given information. The first problem is that one of the demand pattern called "unbalanced" is a randomized demand pattern, and precise numbers are not given. Therefore we have only computed results using the first 2 types of demand pattern, "uniform" and "neighbor". The "uniform" pattern consists of a constant amount of 5 units of demand between every node pair. The "neighbor" demand pattern consists of 10 units of demand between each pair of adjacent nodes only. Network designs were calculated using each of these 2 demand patterns for each of the 6 test network topologies given.

Another factor that hinders reproducibility is the fact that the tests given in [ChCh04] present their demands to the algorithm in an unpublished order. Therefore, the best we can do to attempt to reproduce these tests is to perform a variety of randomized trials and then compare the overall average result to the values given in [ChCh04]. While this will not totally confirm whether or not our implementation is correct, a close agreement between the values should give us some confidence in our interpretation of the given algorithm. Therefore we solved for 10 network designs per network topology and demand pattern combination, and we have reported both the average network spare capacity cost and the range of capacity costs (minimum and maximum) for this set of 10 designs.

Additional Test Suite

The set of test cases used in [ChCh04], in addition to being small, is also not very representative of real networks. Real topologies and demand patterns are much more irregular than those used for most of the tests. Therefore, in order to gain more of an idea about how the algorithm would perform in real networks, we performed tests using the algorithm on a suite of realistic topologies and demand patterns.

This test suite consisted of each of the networks from 3 network families, containing

15, 20, and 25 nodes respectively (see Appendix A.1.1, Appendix A.1.2, and Appendix A.1.3 respectively for topology details). Each network family was given its own demand pattern that was used for all networks within that family. To create these 3 demand patterns, a uniformly randomized value of demand between 1 and 10 was assigned to each node pair. 5 randomized demand orders were calculated for each of these three demand patterns. Each of these 5 demand orders was then used by the heuristic to solve for a PXT network design for each of the 61 test networks.

4.2.3.2 Results and Analysis

Reproduction of Original Test Suite

Table 4.1 gives our results for each of the 12 test cases. It also includes the values reported in [ChCh04] for reference. Although in most test cases the average capacity cost value of our 10 test runs is remarkably close to the original values, it does differ markedly (by almost 70%) in one case, marked with italics (K6,6). The reason for the discrepancy in this case is unknown. Given the striking agreement in all other cases, even the case that uses the same K6,6 topology with a different demand pattern, it is possible that there was simply a typographical error in the original paper. This is supported by the fact that the two values differ by almost exactly 100; a single digit may have mistyped. On the other hand, it is possible that there is some detail of the specific demand pattern and network topology combination in this case that is unusually sensitive to the possible differences in implementation details between the two programs (e.g., the way they handle the arbitrary choice between equal-cost decisions). Whatever the case, the overall good agreement for the rest of our tests gives us confidence in the accuracy of this implementation.

Additional Test Suite

Figure 4.2 is a plot of the spare capacity costs of the designs calculated for the networks from the 15 node, 20 node, and 25 node network families. Each PXT data point actually represents an average cost over the 5 different randomized demand orders for that test case. The spare capacity costs of corresponding optimal *p*-cycle designs for the same networks are also given for comparison. From the plot, we can see that the PXT designs are at least comparable to the *p*-cycle designs, and are in most cases actually more efficient. The cost

	Results Presented in [ChCh04]		***Results Obtained by Our Implementation (10 randomized trials)***		
Network/Demand Pattern	**Working Capacity**	**Spare Capacity**	**Working Capacity**	**Average Spare Capacity**	**Spare Capacity Range**
UNIFORM Demand Pattern					
12-cycle + 3 edges	840	894	840	884.1	874 to 894
3 x 4 grid	770	587	770	590.7	564 to 611
Tietze's graph	645	362	645	366.9	350 to 384
Murakami & Kim	600	533	600	510.2	505 to 517
Icosahedron	540	178	540	183.7	174 to 196
K6,6	480	***139***	480	***236.2***	230 to 241
NEIGHBOR Demand Pattern					
12-cycle + 3 edges	150	189	150	196.7	187 to 209
3 x 4 grid	170	236	170	233.1	225 to 242
Tietze's graph	180	206	180	201.1	170 to 225
Murakami & Kim	240	233	240	234.4	225 to 243
Icosahedron	300	205	300	199.5	174 to 224
K6,6	360	188	360	171.6	161 to 189

Table 4.1: Our greedy heuristic test results compared with those from [ChCh04]

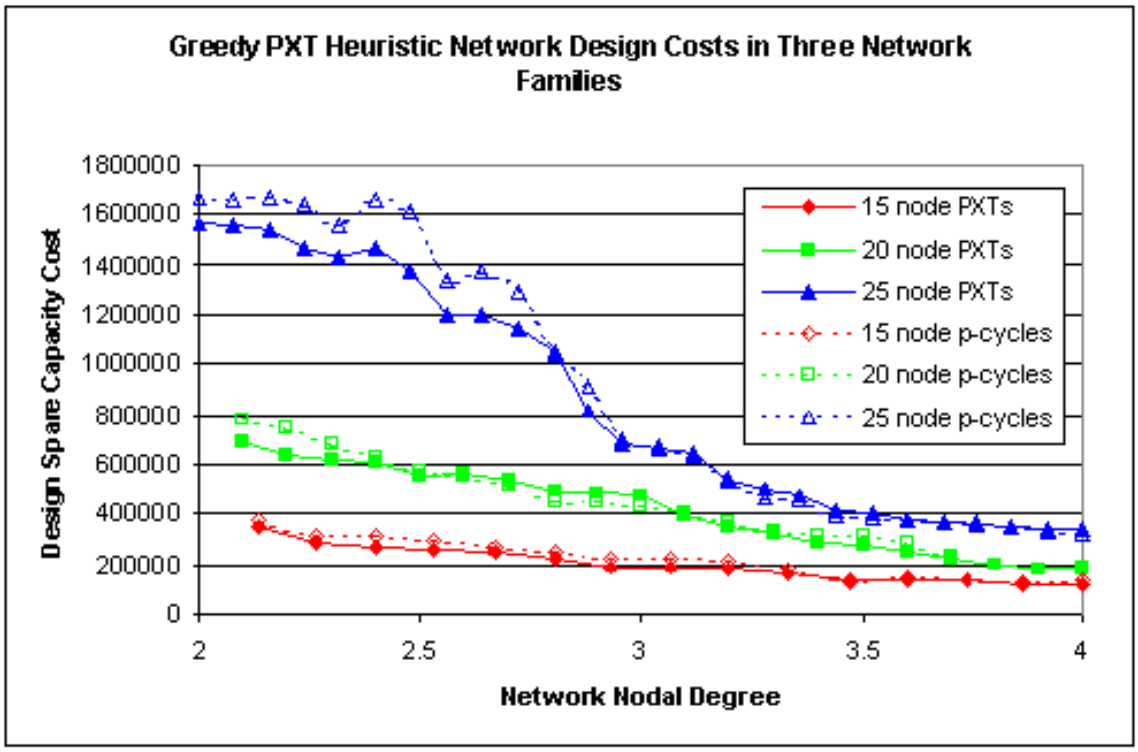


Figure 4.2: Costs of greedy heuristic PXT designs compared with optimal *p*-cycle designs

differences between the two are more visible in Figure 4.3, which plots the cost ratio of PXT to *p*-cycle designs.

A feature not shown by the graph is that the costs for each network design hardly vary at all over the 5 different demand orders. Over all test cases, the greatest discrepancy between a single one of the 5 trials and the average is only 3.3%. On average, the worst of the 5 trials only differs from the average by 1.1%. Therefore it seems that even though the designs are order-dependent, the costs of these designs are not very sensitive to variations in demand order at all. This result explains why we did not see much of a difference between our results and those from [ChCh04] in the previous Section, even though we could not know their exact demand orderings.

The similarity in cost between our random trials intuitively suggests that there may be a deeper similarity in the details of the PXT configurations themselves. To investigate this possibility, we examined the correlation between the protection routing of the demands in each of the 5 different designs for each network. For each of the 10 unique combinations of 2 designs chosen from the set of 5, we calculated the number of demand units that use

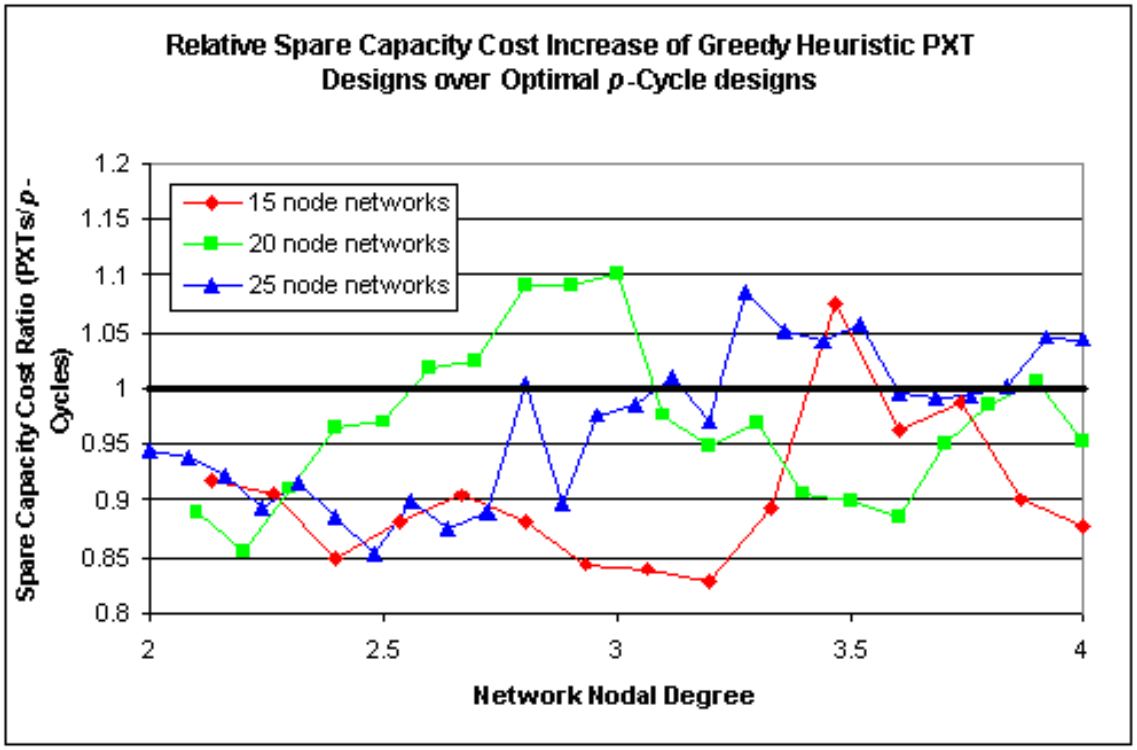


Figure 4.3: Ratio of PXT to p-Cycle spare capacity costs for data from Figure 4.2

the same protection paths in both designs. This value is averaged over the 10 two-design combinations and then divided by the total number of demand units in the demand pattern in order to obtain the values for the data points in Figure 4.4. These data points represent an approximate metric for the average correlation between protection routing of all pairs of the 5 randomized trials for each test case. The metric is approximate because it does not take into account the possible differences between the sharing of protection paths between different demands, meaning that it reports a value that we can consider to be higher than the "true" similarity between designs. For example, the 25 span networks in the 25 node family all report 100% similarity in protection routing (the network is simply a ring, so the protection possibilities are so limited that the heuristic chooses the same paths every time). However, the network designs have slightly different costs because of the aforementioned differences in PXT sharing.

Despite this limitation, we can use this metric to show that the heuristic protects the demand set in remarkably similar ways over a variety of randomized runs. Figure 4.4 shows a relatively consistent pattern of demand routing correlation over the entire range

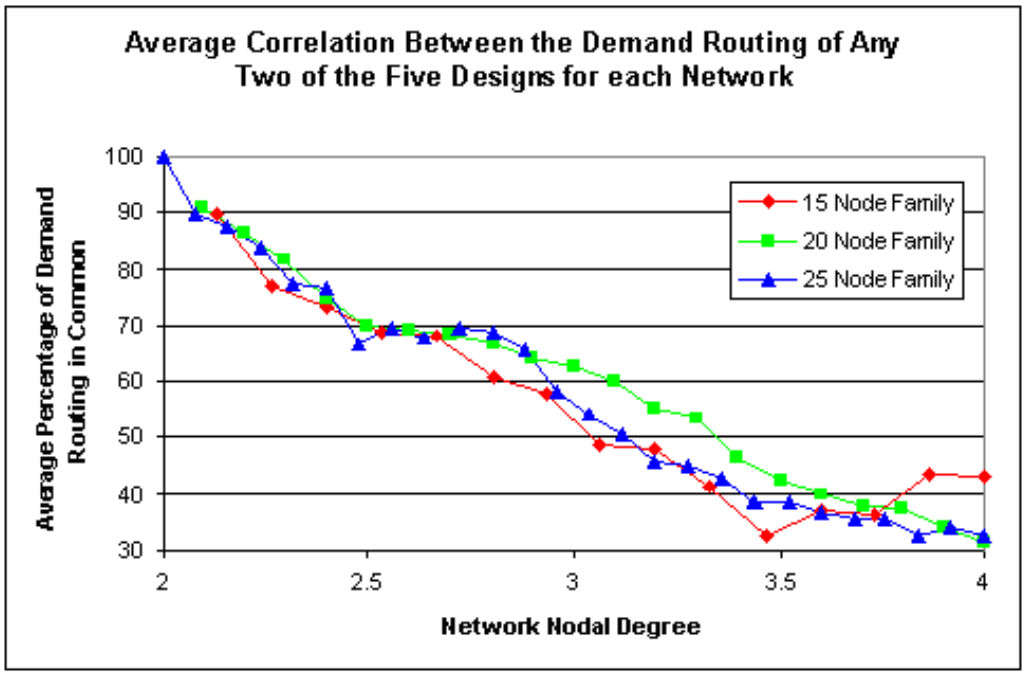


Figure 4.4: Correlation between the protection routing of pairs of designs from 5 randomized trials of the greedy PXT heuristic

of nodal degrees for each of the 3 network families. Even in the most dense networks, the correlation does not drop below 30%, meaning that even in the worst case 1/3 of the demands in the network still share the same protection routing between any two of the five designs. This observation is reinforced by Figure 4.5, which shows the degree of correlation across all five random trials (i.e., the percentage of protection paths that remain constant across all five designs). Even here, the amount of similarity is striking. The most obvious explanation for this is the fact that, under this algorithm, there are certain demands that will usually generate new 1:1 APS-equivalent PXTs as their protection paths instead of sharing protection with existing PXTs.[2] Because these paths are found via shortest-path search, they will always be the same for the same demand in different designs (even if these 1:1 APS PXTs are extended later by the algorithm, the original protection path will remain the same). Therefore, if the number of demands that generate their own 1:1 APS PXTs is large enough, any two given designs should have a large number of these demands (and thus

[2]Note that just because a demand generates a 1:1 APS PXT for its protection, this does not mean that this PXT will remain as-is in the final network design. As subsequent demands are protected, this PXT may be extended or merged with other PXTs such that it shares its protection capacity between the protection of multiple different demands.

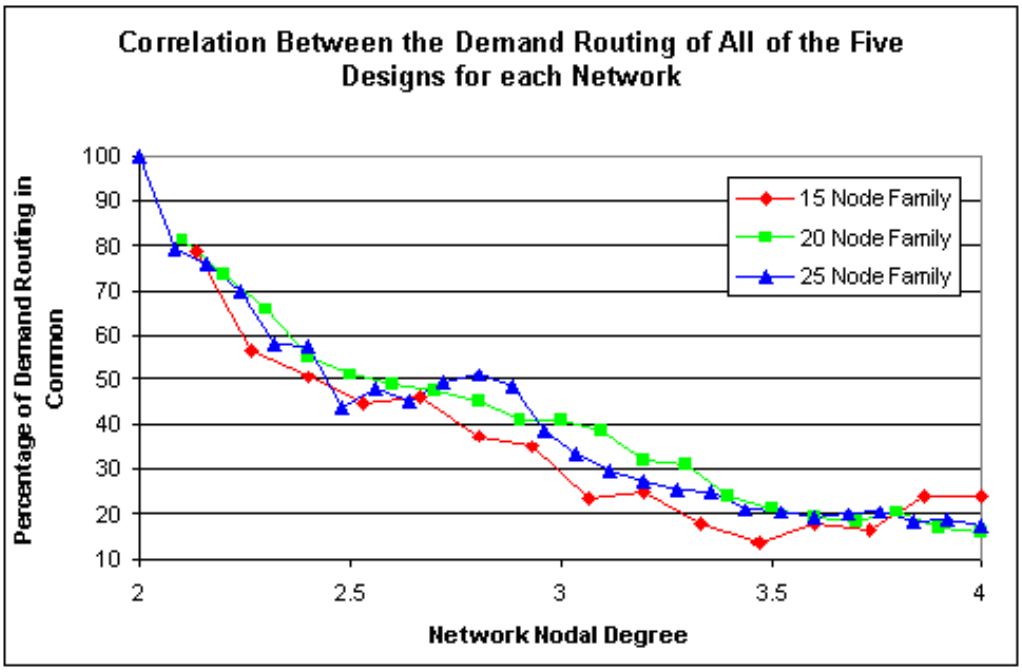


Figure 4.5: Correlation between the protection routing of all 5 randomized trials of the greedy PXT heuristic

protection paths) in common.

Further investigation shows that the number of demands for which a new 1:1 APS PXT is generated for protection is indeed significant. The average figure over each of the 5 tests cases for all of the 15 node networks is 19.7% (fraction of demand volume for which a new APS-equivalent PXT is generated out of total demand volume), 21.8% for the 20 node networks, and 26.4% for the 25 node networks. Furthermore, the protection routing similarity between 1:1 APS PXT-creating demands makes up a significant proportion of the total protection similarity seen between each of the test cases. For example, when considering similarities between each of the 10 combinations of 2 test runs for each network, the similarity between APS-creating demands makes up between 15% and 45% of the total similarity in the set of 15 node networks, between 18% and 39% in the 20 node networks, and between 24% and 56% in the 25 node networks. In most cases the percentage of similarity contributed by APS-creating demands exceeds the proportion of these demands out of the entire demand set volume, indicating that they are more likely to be similar to each other than other demands, as we predicted. Furthermore, the percentage of similarity contributed by APS-creating demands tends to increase as we generate and compare more test

runs for a particular network, indicating that the consistency of routing for APS-creating demands is higher in general than for other demands.

To clarify with an example, Figure 4.6 gives the salient results for just the 20 node, 40 span test network. This Figure shows the average number of similarly routed demands when the comparison is made between sets of 2, 3, 4, and 5 of the 5 different designs calculated for this network. One curve gives these values for the case where all demands are considered, while the other gives values for only APS-creating demands. The data points for a comparison set size of "1" express self-similarity of the designs, and therefore simply represent the total number of demands in the network and average number of APS-creating demands across the 5 test cases, respectively. This plot shows the increasing proportion of similarly routed APS-creating demands as larger sets of designs are compared together. APS-creating demands make up an average of 17% of the total demand volume over the 5 different test runs, but when comparing these test runs in pairs, we find that on average 38% of the similarity between pairs is caused by APS-creating demands. When comparing triplets of designs, this figure increases to 45%, and comparing sets of 4 and 5 designs raises this figure to 50% and 54%, respectively. We can see from the plot that this effect is caused by the fact that the similarity in protection routing decreases drastically, but the number of similar APS-creating demands remains roughly the same. For this amount of similarity to exist, most of the same demands must be choosing to create new 1:1 APS PXTs in each of the test cases. This Figure only shows the results for the 20 node, 40 span test network, but the same pattern is seen across all of the test cases. This is quite significant because demand order is randomized between every trial, meaning that for some demands the creation of a 1:1 APS PXT is generally the most efficient incremental protection solution, no matter what the current state of the PXT configuration of the network is or what demands have already been protected. This idea will be returned to in a later Section, in which we investigate the PXT configuration of a design produced by the greedy heuristic algorithm in detail.

4.2.3.3 Conclusions

This exercise has shown us what we can expect, in terms of both spare capacity efficiency as well as consistency of the results, from designs produced from the greedy heuristic. Many test cases over three separate network families have shown that the greedy PXT

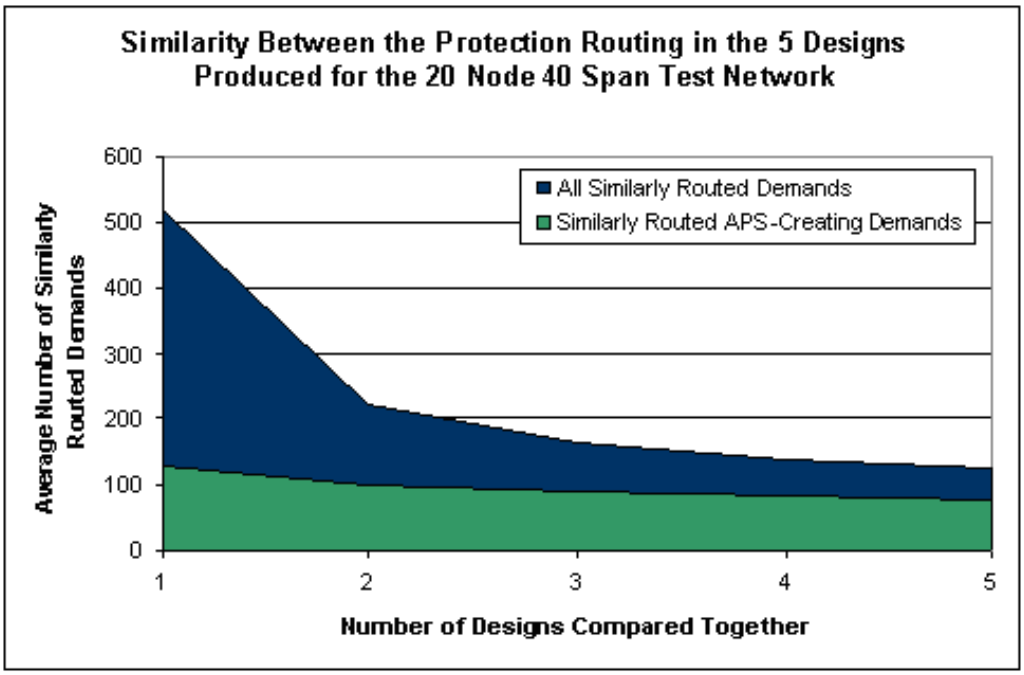


Figure 4.6: Proportion of protection routing correlation contributed by APS-generating demands in the 20 node 40 span network

heuristic algorithm produces designs with efficiencies on par with those of *p*-cycle designs, and sometimes significantly better in cases where the network is both large and sparsely connected. As for consistency, we have seen that consistency in the protection routing of demands, particularly demands that produce new 1:1 APS PXTs for their protection in the course of the algorithm, means that the efficiency of the designs does not vary wildly even when the algorithm is assigned to protect the same set of demands in different, randomized orders. This result suggests a regularity to the greedy heuristic design algorithm that is unexpected in the face of its order-dependence.

4.2.4 Improving the Results of the Greedy Heuristic Using Simulated Annealing

4.2.4.1 Motivation

At this point we have seen that using the greedy heuristic algorithm (as presented in Section 4.2.2) to design PXT-based networks results in designs with capacity efficiencies comparable to or better than those of *p*-cycle designs for the same networks. Despite this, the algorithm is fundamentally suboptimal, relying on a simplification of the total design prob-

lem into a series of incremental protection subproblems. Therefore we suspect that there will in general exist a method to further improve the cost of these solutions. Unfortunately, the fully optimal problem will likely always elude our grasp due to its size, which means that we will never be able to know with certainty how far the designs are from optimal. However, we can try to use some standard heuristic optimization methods (*metaheuristics*) to improve our sub-optimal designs, both for the practical benefits and also to gain further insight into how close to optimal we can expect to come using the original greedy heuristic alone.

For these purposes we applied the metaheuristic called *simulated annealing*. Simulated annealing is a process that performs a succession of small random modifications to a starting solution in an attempt to find improvements to a given objective function. Random changes are either accepted, in which case the process continues using the new, changed solution, or rejected, in which case the new solution is discarded and another randomized modification is attempted. Changes that worsen the value of the objective function may be accepted, but the process is biased such that changes that result in a local improvement are more likely to be accepted. Furthermore, the chance of accepting locally negative changes decreases as the process goes on, until the process only accepts local improvements and eventually stagnates in a local minimum/maximum when it can find none. The hope is that this local minimum/maximum is close to the global minimum/maximum. The algorithm gets its name from the process of annealing metal, in which metal is heated and then gradually cooled. This process increases the strength of the metal by allowing the crystal structure to slowly shift into a regular pattern. In much the same way, simulated annealing first attempts to escape local minima by allowing a more random search but then slowly "cools down" its search to more closely approximate a standard gradient search as time passes, ending with a final "crystallized" solution that is hopefully an improvement over the initial solution.

The following Section describes our experiments with adding a simulated annealing process to the end of the greedy heuristic, such that the result of the original algorithm is the starting point for the annealing process. We examine two different approaches to annealing and present some results that show how much simulated annealing can be expected to improve on the original solution.

4.2.4.2 Implementation

In order to implement simulated annealing for a specific problem, a number of components of the metaheuristic must be defined in relation to the elements of that problem. These components are the *objective function*, the *random change algorithm*, and the *annealing schedule*.

The *objective function* is the same concept as in the integer linear program; it is simply the cost of the spare capacity used in the PXT design, which we want to minimize.

The *random change algorithm* is the algorithm that is used to make the incremental random changes to the current solution to the problem (in our case, a PXT arrangement for the network). This is the most complex part of the implementation of simulated annealing for the problem of PXT design. This is because this process must both change the design significantly enough that the search can make progress through the search space, but at the same time the result of each change must remain a valid PXT design that provides 100% single span failure restorability to the network. Therefore we cannot simply add, remove, or change the spans on PXTs at random because doing so would almost certainly result in a design where certain demands were not protected. We could still follow this approach if we followed up each random change with a repair step in which the design was repaired back to full 100% restorability, but that approach would likely result in a random change step that is too coarse to produce good results on our problem. Instead, the current functionality of the greedy heuristic itself can be altered slightly to obtain the type of change function we are looking for.

The fundamental operation of the greedy heuristic is the incremental protection step, in which a single-unit unprotected demand is protected using a least-cost combination of existing PXTs and newly allocated spare capacity. We can modify this operation to create a suitable random change function by first running this process in reverse on a number of demands to "unprotect" them, and then running the incremental protection step again for each of the unprotected demands in a different, randomized order. In other words, first the selected demands are removed from their respective PXTs and any spare capacity units on sections of these PXTs that become unused as a result are deallocated. This may result in fracturing certain PXTs into smaller parts, or even the elimination of some PXTs entirely. Then the demands are protected again, one by one, by using the existing PXTs or

by allocating new spare capacity, as in the original greedy heuristic algorithm. It is possible that this process may result in a design equivalent to the original, but in general we can expect that the PXT configuration will be at least slightly different because the environment of existing PXTs that each demand is protected under will likely be different than when that demand was incrementally protected in the original run of the greedy heuristic.

The degree of change produced by this process will depend on the number of demands on which we perform this re-protection operation. In the most extreme case, where it is allowed to re-protect every demand in the network, a single step of this randomization procedure actually represents an entirely new execution of the original greedy heuristic. Such an extreme case would violate the purpose of the random change algorithm, which is to introduce minor changes to the design that explore the search space of the problem, but we will experiment with less extreme variations on the magnitude of this parameter in the experiments that follow.

Finally, the *annealing schedule* of a simulated annealing implementation refers to the way in which the "cooling" process is handled. This refers both to the change in the acceptance rate of negative (i.e., cost-increasing) random changes over time, as well as the method used to decide when to terminate the annealing process. As with most aspects of a metaheuristic, there is no de facto correct choice for the annealing schedule, but there are some standard methods that have been found to work well for a variety of problems. One of the simplest methods, and the method we use in the following experiments, is to use a threshold variable T that represents the largest possible change in the objective function for which a random change will be accepted. In other words, all negative (cost increasing) changes that produce a change in the objective function lower than this value will be accepted. Changes that improve the objective function are always accepted. The "cooling" of the process is implemented by establishing a schedule by which T is decreased over time, reducing the degree to which the simulation will explore negative changes in order to avoid local minima. The simulation is then terminated when T reaches a predefined value T_0. For our experiments, we follow a simple schedule of reducing T by a predefined amount ΔT every n steps. The parameters T, T_0, ΔT, and n can be varied to observe their effects on the improvements found by the annealing simulation. This approach has the advantage that termination will occur in a constant, pre-determined number of steps $\left\lceil \frac{T-T_0}{\Delta T} \right\rceil \cdot n$.

4.2.4.3 Initial Tests: a Degenerate Case

Before performing a full investigation into the effects of variations in the parameters on the simulated annealing process, we ran some simple tests using a degenerate case of the simulated annealing algorithm in which the number of demands re-protected was limited to one. This is a degenerative case because, under this restriction, the random change function will never produce a design with a higher cost than the old design. Because the demand protection function from the greedy heuristic algorithm finds the protection path with absolute minimum cost, there is no way for it to find a more expensive way to protect the demand than was used originally. At the very least it can always use the same path that was used originally, resulting in an unchanged design. This guarantee breaks down in the multiple demand re-protection case because the demands are re-protected in a new, randomized order that has the possibility to create a worse design than the original.

The ability to avoid local minima in the search space by making locally bad decisions is a fundamental part of a simulated annealing implementation, so we cannot use the term "simulated annealing" to describe a process that uses this degenerative form of the change function. Instead, it is simply a brute force search that successively polls random demands and determines if their protection paths can be reduced in cost by re-routing them individually. This is similar to how a human being might inspect a design manually to determine if any obvious improvements could be made to the protection routing. We ran this search on a large variety of test networks and demand patterns in order to determine how much progress we might typically expect to make by using this straightforward method. This set of tests included all of the topology and demand pattern combinations used in [ChCh04] and previously repeated in Section 4.2.3.1. Also used for the tests were the most and least highly connected networks from each of the 15 node, 20 node, and 25 node network families (i.e., the 16 span and 30 span networks from the 15 node family, the 21 span and 40 span networks from the 20 node family, and the 25 span and 50 span networks from the 25 node family). These networks were chosen because they are more realistic than most of the test networks from [ChCh04], and therefore their results will likely represent more closely the benefits that could be expected from using this algorithm in practice. For all of the chosen test cases, the search was performed on all of the designs found for these test cases in Section 4.2.3.2, i.e., the process was run on each of the 10 randomized demand order

Network Topology	Demand Pattern	Average % Improvement	Average Time for Search as % of Heuristic Runtime
12-cycle + 3 edges	Uniform	0.83	56
3 x 4 grid	Uniform	1.22	50
Tietze's graph	Uniform	1.01	47
Murakami & Kim	Uniform	0.45	50
Icosahedron	Uniform	1.42	32
K6,6	Uniform	0.64	29
12-cycle + 3 edges	Neighbor	0.25	61
3 x 4 grid	Neighbor	0.04	94
Tietze's graph	Neighbor	1.29	59
Murakami & Kim	Neighbor	1.19	52
Icosahedron	Neighbor	2.26	35
K6,6	Neighbor	0.23	20
15 node 16 span	Random	0.30	44
15 node 30 span	Random	0.25	29
20 node 21 span	Random	0.11	21
20 node 40 span	Random	0.34	16
25 node 25 span	Random	0.08	14
25 node 50 span	Random	0.19	12

Table 4.2: Design improvements obtained with simulated annealing degenerate case (1 demand random change function)

designs produced for the 12 test cases from [ChCh04] and on each of the 5 randomized demand order designs produced for the networks from the network families.

The results of the degenerate annealing search algorithm are given in Table 4.2. The values that are reported are averages over all of the designs that were used for each test case. The average amount of time required by the search procedure is also given in the Table as a percentage of the time required to generate the original design, in order to give an idea of the time cost required to obtain these improvements. Overall, the performance of this search is not impressive. In the best case, it improves a design on average by a little over 2%, at the cost of an additional 35% of runtime.

Test	d	T	T_0	$\triangle T$	n
1	2	10000	1	1000	10
2	10	10000	1	1000	10
3	100	10000	1	1000	10
4	100	1000	1	100	10
5	100	100	1	10	10

Table 4.3: Parameter values for simulated annealing tests

4.2.4.4 Full Tests

The initial tests do not give us much confidence in the ability of this approach to find improvements to the PXT design. However, it is possible that limiting the algorithm to making such small incremental changes causes it to become trapped in local minima of the search space. We conducted further tests using the full range of the capabilities of the simulated annealing algorithm to determine if some combination of parameters would allow the algorithm to make some significant improvements. Unfortunately, as with most metaheuristic search algorithms, there is no way to analytically determine the best parameter settings for simulated annealing ahead of time. Furthermore, the large number of degrees of freedom in the parameters of the algorithm (e.g., T, T_0, $\triangle T$, and n as discussed above as well as d, the number of demands re-protected in each step) means that making an exhaustive scan of the parameter space is impractical. So the experiments that follow take a trial-and-error approach, covering a wide range of parameter combinations while at the same time making some educated guesses as to which combinations are more likely to work well than others, so as to limit the number of experiments to a practical level.

As part of the effort to limit the number of test cases, we decided to stop using the 6 original test networks from [ChCh04] and limit ourselves to only the networks from our network families. The networks from [ChCh04] are, after all, not representative of real networks, and so results on these networks would not be as valuable in terms of extrapolating the results to real network design scenarios. We limit ourselves to the maximally and minimally connected networks from our three network families. The combinations of the values of the d, T, T_0, $\triangle T$, n parameters that we used for the tests are given in Table 4.3.

The results for these test cases are given in Table 4.4. Each "% improvement" result is an average over 5 test cases. Tests 1, 2, and 3 investigate the effect of increasing the d

Network	Average % Improvement				
	Test 1	Test 2	Test 3	Test 4	Test 5
15 nodes, 16 spans	0.56	1.16	2.59	2.77	2.90
15 nodes, 30 spans	0.32	0.53	0.99	1.60	1.76
20 nodes, 21 spans	0.13	0.28	1.20	1.20	0.98
20 nodes, 40 spans	0.31	0.83	2.43	2.54	2.32
25 nodes, 25 spans	0.11	0.30	0.71	1.56	1.48
25 nodes, 50 spans	0.22	0.66	2.28	2.33	1.99

Table 4.4: Design improvements obtained with full simulated annealing under various annealing parameter combinations

parameter. We can see that there is a uniform improvement between test 1 and 2, and again between 2 and 3, where the only difference is an increased number of re-protected demands in each step. However, the time cost for achieving these improvements is extreme. Figure 4.7 shows the cost improvements for tests 1, 2, and 3 from Table 4.4 plotted against the corresponding runtime for the annealing process, given as a percentage of the time required for the initial greedy heuristic (e.g., a value of 100% corresponds to a doubling of the original heuristic runtime to achieve the new result). Runtime is plotted on a logarithmic axis to correspond to the exponential increase in d over our 3 tests. The plot shows that this corresponds to roughly an exponential increase in runtime; the data points are clustered horizontally around approximately 20%, 100%, and 1000% of the original algorithm runtime for d=2, 10, and 100 respectively. This is something we would expect assuming that the re-protection time for a single working path were a constant time operation. Sustaining this exponential increase in d was beyond our capability, as the runtimes were already almost impractically long at this point (nearly a day for the longest test). Therefore we did not investigate increasing d any further.

Note that at $d = 100$ we are already protecting a large fraction of the network in each step. The 15 node family has a demand pattern with only 518 working paths, which means we are effectively re-doing 20% of the original algorithm for each step, over the course of 100 steps. For the 25 node family this is proportionally less, 100 paths being only about 6% of the 1615 working paths. However, this still means that it is highly likely that each demand is re-protected several times through the course of the algorithm (a 6% chance for each demand to be chosen in each of 100 steps means that the chance it has *not* been

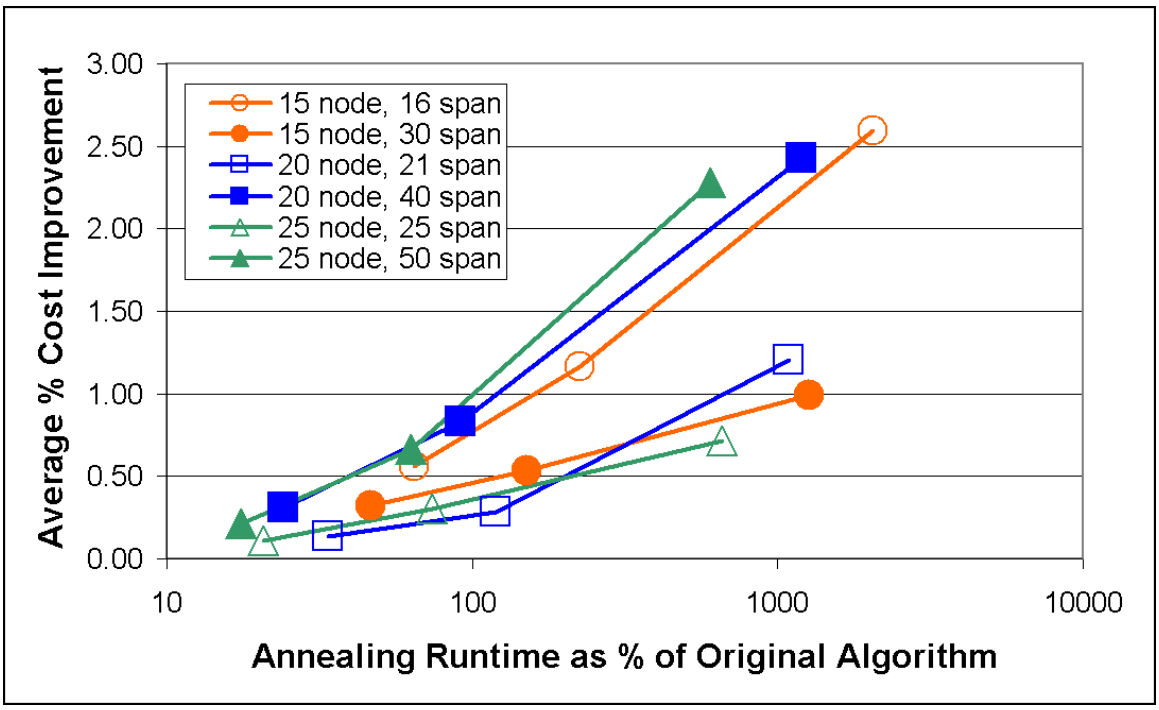


Figure 4.7: Extra runtime required to achieve improvements to heuristic PXT designs through simulated annealing by varying the number of re-protected paths per step (parameter d)

re-protected by the end of the algorithm is less than 0.2%). This gives us confidence that simulated annealing is exploring the solution space in a significant way at these parameter values.

As increasing the number of steps (by increasing the ratio of T to $\triangle T$) would result in a further increase in runtime, our fourth and fifth tests investigate the effect of varying the overall average temperature. Here, we scale down T and$\triangle T$ such that the temperature is smaller in each step, but the same total number of steps are executed. Test 4 scales down the values by a factor of 10, and test 5 scales them down by a factor of 100. Test 4 shows a further minor improvement over all networks, indicating that more tightly controlling the less advantageous changes has a minor benefit. With test 5, however, the results are uncertain, with some networks showing a small gain and others showing a small loss.

4.2.4.5 Conclusions

We have seen that simulated annealing can achieve incremental improvements on the initial designs from the greedy heuristic PXT design algorithm. However, scaling up these

improvements requires a significant computational investment. In particular, on our test machine and for this class of networks, achieving anything greater than a 0.5% decrease in cost generally requires at least a doubling of the runtime of the initial algorithm. Up to a 2.5% decrease is possible, but at runtimes exceeding 10 times that of the original algorithm. A degenerate case of simulated annealing in which only positive changes are accepted will complete very quickly but can achieve only at most about 0.3% improvement on realistic test networks. Overall, it seems that either we require more computational power or a different algorithmic approach altogether to improve upon these PXT designs, or the designs produced by the heuristic are in fact near-optimal.

4.2.5 Characterization of PXTs Produced by the PXT Heuristic

4.2.5.1 Motivation

Up until this point we have concerned ourselves only with the overall capacity costs of the designs produced by the PXT heuristic, without taking into account any other properties of interest. This echoes the superficial treatment given in [ChCh04], and is suitable for an initial characterization of the general attractiveness of the design algorithm. However, to understand PXTs more fully, we must answer questions that remain unanswered in the original work [ChCh04]. For example: How long are the PXTs produced by this algorithm? How many different PXT structures are required? How often are the PXTs self-looping in nature? How practical would a network operator find such structures? With PXTs, can a network operator clearly visualize and control the restored-state routing and path length of failed working paths? These are unaddressed issues of concern about the practical complexities and operational implications of the PXT proposal in [ChCh04], and they compel us to perform a more detailed characterization study. The following work was undertaken to gain insight into the characteristics of the PXT concept; to be able to visualize what the set of PXTs arising from the design algorithm in [ChCh04] would look like, and appreciate how they would be structured so as to be used in different ways under different failure scenarios, and so on.

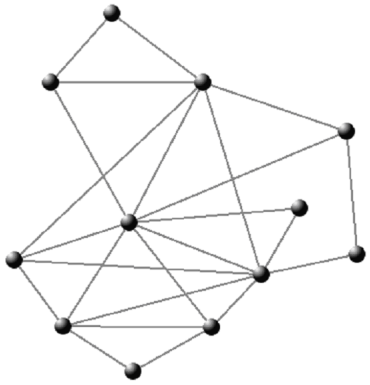


Figure 4.8: The Murakami & Kim network topology (12 nodes, 24 spans)

4.2.5.2 Experimental Method

This experimental study focuses on the in-depth characterization of the PXTs in a single network design. For the purposes of this study, the most "realistic" network topology from [ChCh04] was chosen (Murakami & Kim). The network is shown in Figure 4.8. In order to limit the number of PXTs produced by the algorithm to a manageable level for our analysis, the demand pattern contained only 3 units of demand between each node pair. The heuristic algorithm was used to produce a design for this network using a randomized ordering of the network's demands. In addition, the program was run several more times with different randomized demand orders in order to determine if the design under study was typical in terms of cost redundancy and therefore roughly representative of the problem, as opposed to an artifact of one specific demand ordering.

Once the design was obtained, the program's output was inspected and used to produce a list of PXTs, along with their structural details (spans and nodes crossed) and lists of the demands protected by each PXT. This information was used to produce the diagrams in Appendix E that show the PXTs and the demands they protect overlaid on the topology of the network. Each PXT has its own diagram, in which the thick blue arrowed line represents the PXT itself. Where possible, the working paths of the demands the PXT protects are also drawn as the various coloured lines tipped with circular knobs. This cannot be done for all PXTs as some of them simply protect too many demand to be illustrated on a single diagram.

4.2.5.3 Results

Overview

For later reference, the total cost of working routing for the network was 59,696. In these and other cost statements, the units are distance-channel counts, i.e., if the lengths of spans are taken in km then costs are in units of channel-kilometers. The cost of spare capacity for the heuristic design was 53,591, giving it a (distance-weighted) capacity redundancy of 89.8%. When compared to 5 other randomized trials, the design under study was found to be quite typical in terms of cost, with the costs of the other designs falling within 10% of the case selected for detailed inspection. Therefore it is reasonable to study this design as a representative to draw general conclusions about the nature of PXTs produced by the algorithm.

For comparative reference purposes, *p*-cycle and span-restorable mesh designs were also produced for the same topology and demand pattern. These were perfectly optimal reference solutions produced with now-standard ILP methods and documented in [Grov03] or elsewhere (e.g., the *p*-cycle model is also given in Section 3.2.1). The *p*-cycle model was given the complete set of network cycles as candidate *p*-cycles and the span-restorable mesh model was given the complete set of restoration routes as candidates, so we can be confident that these solutions are completely optimal for their respective architectures. The spare capacity costs were 51,748 (*p*-cycle) and 46,681 (span-restorable mesh), corresponding to redundancies of 86.7% and 78.2% respectively. Therefore the PXT heuristic algorithm was able to approach the efficiency of an optimal *p*-cycle design within 3% while also providing the additional desirable property of end-to-end working path protection.

Typical Illustrative Results

As we can see from the detailed portrayal of PXT solutions in Appendix E, some individual PXTs can be surprisingly complex. An immediate intuitive concern is that these structures would be quite impractical in the view of most network operators. As [ChCh04] itself alludes, protection structures should be compact, manageable, and easily visualized and maintained. Reversion after repair should also be a simple process. PXTs produced by the heuristic from [ChCh04] can evidently be so complicated that we cannot even fully represent them and their protected routeset in a single diagram.

Notwithstanding, we can use the results to demonstrate, for conceptual appreciation, how PXTs would actually work. To do so, the response of one of the PXTs to an example span failure is illustrated in Figure 4.9 and Figure 4.10. Figure 4.9 shows how the spare capacity of a PXT can be broken up into four protection path segments in response to the failure of a single span. The new coloured lines layered on top of the blue PXT represent these protection paths (the sections of the PXT that remain blue are not required and remain unused). Figure 4.10, in turn, separates these protection paths into 4 subfigures and shows how they are used to protect 4 working paths that are all affected by the span failure. Each working path is protected end-to-end by one of the protection paths formed out of the preconnected channels of the PXT.

These illustrations show that even though the PXT is quite convoluted, each failure causes the preconnected spare capacity in the PXT to be divided up into very simple protection paths. The convoluted nature of the PXTs is a result of so many protection paths being shared between many different working paths to achieve capacity efficiency. So indeed, the PXT concept works, and achieves good capacity efficiency, but it seems to engender extensive operational complexity in the pre-failure state, the restored re-routing state, and the post-repair reversion process. This is a significant revelation, especially since the authors of [ChCh04] motivate their work by a concern about the possible size of *p*-cycles. The fact is that in contrast to the PXT of Figure 4.9, which is only one of 31 PXTs in the complete design and uses 35 hops, the largest possible *p*-cycle in this network has only 11 hops. (The Murakami & Kim graph is not Hamiltonian, but if it was then even a design using a single Hamiltonian *p*-cycle would require a single structure of only 12 hops to protect the entire network.)

More practically, the optimal multi-cycle *p*-cycle design employs eight distinct cycles, none of which is longer than 10 hops. It may be argued that the methods used to produce these designs are different (an online algorithm for PXTs and an offline algorithm for *p*-cycles), so the results are not comparable. But simple algorithms also exist to update a *p*-cycle configuration incrementally with near-optimal efficiency. And because *p*-cycles are formed by cross-connecting spare capacity it is possible to recompute and update the configuration following batches of demand arrivals, keeping always very close to an optimal set of *p*-cycles (see [Grov03], Chapter 10). In addition, the very first paper on *p*-cycles

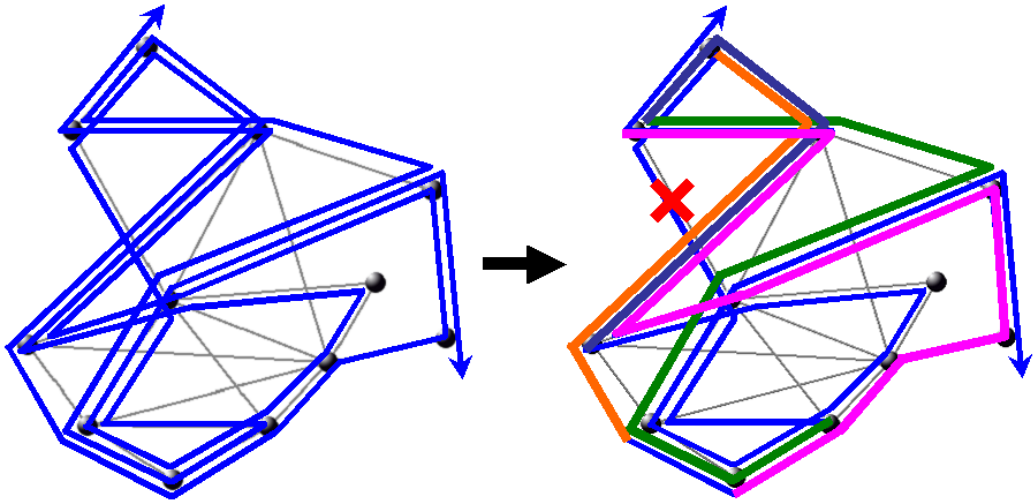


Figure 4.9: Creation of 4 protection paths in a 35-hop PXT in response to the failure of a single span (details shown in Figure 4.10)

[GrSt98] outlined an online distributed self-organizing protocol, which would continually adapt the set of *p*-cycles to a condition close to that of the corresponding problem solved optimally with all demands known at the time. It is unlikely, therefore, that the *p*-cycle solutions here are significantly different from the set of *p*-cycles that would be in place had the demands accumulated incrementally instead. Under known methods, they would be very similar to the optimal solutions here, and thus clearly very different than the set of incrementally evolved PXTs from the PXT algorithm of [ChCh04]. Overall, then, this exercise shows that the PXTs produced by the algorithm from [ChCh04] can be significantly more complex than the cycles used in comparable *p*-cycle designs.

Statistical Characterization of the Design

Visual inspection of the PXTs provides some interesting qualitative insights into the nature of the algorithm, but restorable network design also involves quantitative concerns about capacity efficiency, operational complexity, and other characteristics. Therefore in this Section we calculate various metrics of interest for each PXT in the design to support more quantitative comparison and assessment of the characteristics of PXT designs that may be significant to network operators. Table 4.5 lists these metrics for each PXT in the design. The operational significance of some of these measures is discussed below. In addition, each PXT is assigned a numerical identifier for reference. The metrics abstracted from the

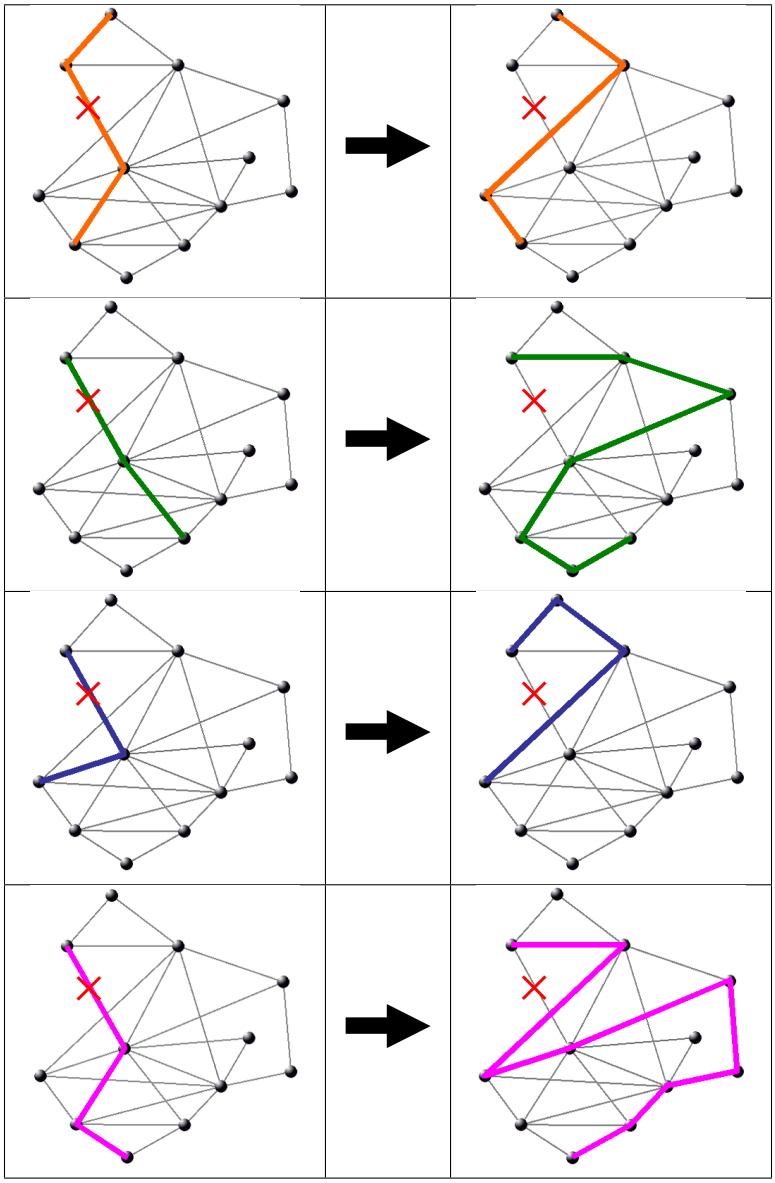


Figure 4.10: Breakdown of the individual protection paths shown together in Figure 4.9

designs and appearing in order as follows in Table 4.5 are:

Hops: Length of the PXT in hops.

Length: Physical length of the PXT.

Copies: Number of copies of this PXT placed in the network. In practice this will always be 1, as the heuristic algorithm internally tracks each PXT individually. Therefore even if two PXTs end up being exactly the same, the algorithm will consider them as separate.

Closed: Whether or not the PXT's tail is preconnected to the head, forming a cyclical structure like a ring or *p*-cycle.

Self node-crossings: The number of times the given PXT loops through a node. This is a significant measure of the complexity of a PXT from a network operator's point of view. Note that such looping at intermediate nodes does not imply cyclical closure of the actual PXT path.

Self span overlaps: The number of times the PXT crosses over a span that it has already covered.

Number of demands protected by the PXT.

Path-structure coincidence count: The number of spans a PXT has in common with the working paths of its own protected demands. Spans that are used by more than one demand are counted multiple times.

Total protected working capacity: The total distance-channel count of working channels protected by the PXT.

Maximum simultaneous protected path failures: The largest number of protection paths that can be simultaneously activated within the PXT in response to any single span failure.

Longest protection path (by hops and length): Length of the longest protection path, by hops or by actual physical length respectively, used by any of the demands protected by the PXT.

Redundancy: This is calculated by dividing the PXT's own total distance-channel count product for spare capacity by the total of its protected working capacity.

Several of these metrics characterize factors affecting the complexity of operation of PXTs as real protection structures in an optical network. The number of self node-crossings and self span overlaps in a PXT both affect the impact that the failure of a single node or span may have on the integrity of a single PXT. To an operator they represent the difficulty of keeping track of information about the PXT throughout its operational life, and especially through the transition between non-failed and failed states. The specific problems that network operators may have with PXTs with a large number of self-intersections will likely vary from operator to operator, and are beyond the scope of this work. However, considering the fact that modern day network operators are used to simple ring and APS systems, the comparative step in complexity to massive structures with over 30 hops that may, for example, cross entire continents multiple times may be perceived as operationally impossible by many.

The worst case maximum number of simultaneous failures that a PXT protects against is another metric that has no limit imposed currently in design, but that may be important to network operators. A PXT that forms a large number of protection paths may need to perform many switching actions due to a single failure. From the standpoint of individual protection segments, one could argue that switching is still simple because each segment will only perform two actions: one at each end-node affected by the failure. However, from an overall network viewpoint, each PXT is a contiguous optical path that must be broken up when it is used to protect against failures. This means that large PXTs may require some type of synchronization to ensure that protection segments on the same PXT do not interfere with each other when they are brought into use as backup paths. Again, this is likely system specific and will not be discussed in detail here, but serves as motivation to reduce this particular metric where possible.

4.2.5.4 Discussion

Complexity Metrics

The metric values in Table 4.5 that represent PXT structural properties reinforce the visual impression that some PXTs can be very complicated. We might assume that this complexity

PXT ID	Hops	Length	Copies	Closed	Self node-crossings	Self span overlaps	Number of Demands Protected	Path-Structure Coincidence Count	Cost of working capacity protected	Maximum simultaneous protected path failures	Longest Protec-tion Path (Hops)	Longest Protec-tion Path (Length)	Cost Re-dundancy
0	6	1132	1	No	0	0	3	2	807	1	4	794	1.40
1	2	327	1	No	0	0	1	0	92	1	2	327	3.55
2	3	608	1	No	0	0	1	0	229	1	3	608	2.65
3	3	460	1	No	0	0	1	0	302	1	3	460	1.52
4	4	713	1	No	0	0	1	0	344	1	4	713	2.07
5	3	607	1	No	0	0	1	0	226	1	3	607	2.68
6	3	608	1	No	0	0	1	0	229	1	3	608	2.65
7	3	608	1	No	0	0	1	0	229	1	3	608	2.65
8	2	327	1	No	0	0	1	0	92	1	2	327	3.55
9	2	443	1	No	0	0	1	0	394	1	2	443	1.12
10	2	415	1	No	0	0	1	0	299	1	2	415	1.39
11	3	602	1	No	0	0	1	0	366	1	3	602	1.65
12	19	2966	1	No	8	4	9	16	2441	3	10	1430	1.22
13	5	940	1	Yes	0	0	7	5	1510	2	4	817	0.62
14	35	5782	1	No	24	17	27	72	7882	4	8	1379	0.73
15	33	5558	1	No	24	17	21	74	6692	4	9	1428	0.83
16	17	2753	1	Yes	6	2	13	19	3662	3	6	1002	0.75
17	3	602	1	No	0	0	1	0	366	1	3	602	1.65
18	5	926	1	No	0	0	2	3	868	1	3	531	1.07
19	14	2111	1	No	5	1	10	13	3117	3	5	823	0.68
20	2	442	1	No	0	0	1	0	391	1	2	442	1.13
21	7	1037	1	No	2	1	4	2	860	2	4	700	1.21
22	17	2707	1	No	7	4	9	18	2495	3	8	1198	1.09

23	23	3747	1	No	12	8	15	38	4764	3	9	1564	0.79
24	40	6470	1	No	29	22	29	91	9505	6	8	1287	0.68
25	17	2747	1	No	7	3	9	12	2514	2	7	1239	1.09
26	2	289	1	No	0	0	1	0	111	1	2	289	2.61
27	3	548	1	No	0	0	1	0	509	1	3	548	1.08
28	36	5798	1	No	26	19	21	67	7202	7	9	1361	0.81
29	2	327	1	No	0	0	1	0	92	1	2	327	3.55
30	6	992	1	No	0	0	3	0	1104	1	5	896	0.90

Table 4.5: Breakdown of the properties of the PXTs generated by the greedy heuristic algorithm for the Murakami & Kim network

is what allows the number of efficient sharing relationships to be high, but the per-structure redundancy values show that high complexity does not always correspond to high capacity efficiency. For example, PXT 15 has 24 self node-crossings but protects 21 demands with a redundancy of only 83%. This lead us to question the real value of allowing such complicated structures to be considered at all. We thought it might be possible to use an alternate design method that would greatly restrict the complexity of the PXTs in the design while still retaining the capacity efficiency of a shared mesh scheme.

Table 4.5 also gives the metric values for the length of the longest protection path segment obtained from each PXT by any of the demands it protects. From these values we can see that even in the longer PXTs the complexity of protection paths themselves does not run away significantly. This is the benefit of a mechanism of the modified Dijkstra's algorithm, called *span rivalry*, that the heuristic uses to enforce the simplicity of protection paths. This reinforces the earlier statement that even though these PXTs are topologically complex, they remain operationally simple in some regards.

Undesirability of Span-Looping PXTs in Transparent Optical Networks

Aside from general objections to high complexity in protection structures that ought to be understandable by human operators, there exists an even stronger objection against looping PXTs that is based on real technical concerns. This problem arises when such looping involves a span overlap, as illustrated (using an actual PXT from our heuristic algorithm solution) in Figure 4.11.

If we assume that all-optical wavelength conversion hardware is unavailable, PXTs in a transparent optical network must have wavelength continuity over all of their preconnected spans. Therefore, in a situation in which the PXT crosses any span twice, a double requirement for one of the wavelengths on that span is created, implying that two separate fibres must be used to support the PXT on the overlapping edge. In other words, these "span self-looping" PXTs create an absolute requirement for either expensive regeneration hardware or the multiplication of active fibres used to support the PXT: one extra fibre for each time a PXT crosses over the same span. Figure 4.12 is a histogram of the minimum number of fibres required by the heuristic design on each span in the network, as set by the maximum number of PXT-loopings on each span in our design (assuming sufficient

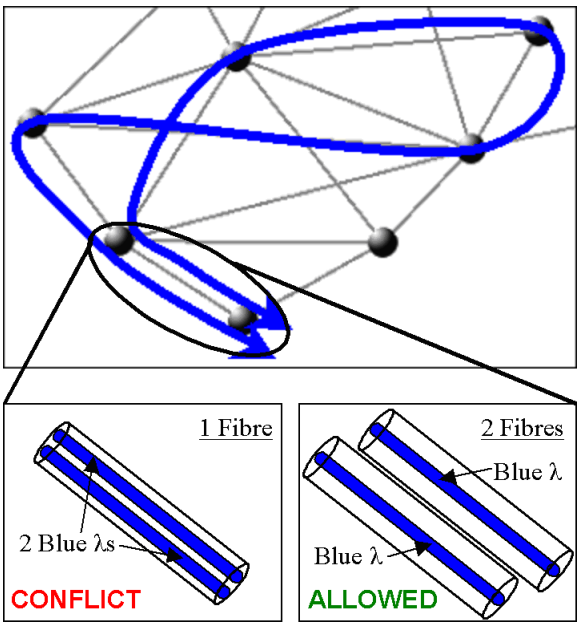


Figure 4.11: A PXT that crosses over the same span twice, requiring two separate fibres on that span to support the operation of two channels with identical wavelengths

wavelengths available on each fibre). The average over all spans is 2.6 fibre pairs per span. Therefore, in transparent DWDM networks it is clear that a non-looping PXT design would provide a major capacity savings simply from elimination of the fibre multiplication effect of span-looping PXTs.

This is an obvious and overwhelming reason to try to prohibit PXTs from ever involving such self-span overlaps as a result of their looping characteristics, which the algorithm from [ChCh04] permits, both for efficiency and because it would be more complex to prohibit this behaviour. Given this observation, it is natural to ask whether it would be practical to alter the heuristic algorithm itself to restrict the complexity of the PXTs it creates, perhaps by limiting them to only be simple trails. This turns out not to be so easily effected in the algorithm. To understand why, we must discuss the modified version of Dijkstra's shortest path algorithm that is used in the greedy PXT design heuristic to find the protection paths for new demands. This algorithm consists of Dijkstra's original algorithm modified to include the concept of "rival" spans for the purpose of restricting the protection paths formed by the algorithm to being simple paths. Intuitively, one might assume that this mechanism could simply be modified slightly in order to force the overall PXTs to be simple as well. The most straightforward approach to take would be to add a new criteria

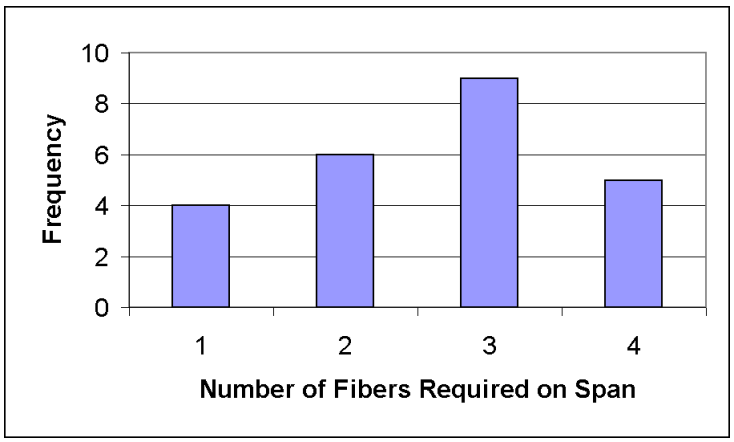


Figure 4.12: Histogram of the lower bound on the number of fibre pairs on each span arising from self-span looping characteristics of the PXTs in the heuristic design

for span rivalry, under which spans are rivals if using them both in the protection path for a demand would result in combining or extending PXTs in such a way that the resulting PXT would be non-simple.

Unfortunately, although this works for span self-disjointness, this scheme cannot be made to properly enforce the node-disjointness criterion for simple PXTs. Basically, we cannot flatly call two PXTs rivals if they contain any nodes in common because PXTs that share a single end-node are joinable on that end-node without that node being used twice. Therefore we need to allow rivalry exceptions for PXTs that share a single end-node. However, if we allow this uniformly then the PXTs may later become joined in a way that is not through this common node (say through an intermediate PXT or other real spans in the network). In this case they will overlap on that one node, making the PXT and any extensions of that PXT non-simple. Therefore, when used to enforce the self node-disjointness property of simple PXTs, the concept of rivalry becomes conditional on the order in which the PXTs are joined together. This makes it impossible to implement this condition within the framework of the greedy heuristic algorithm.

Note, however, that the fibre multiplication problem does not apply to PXTs which loop only across nodes. That type of PXT is still realizable in a network using only a single fibre-pair per span, although it may still be operationally complex. We have seen that enforcing the "span self-disjointness" property by itself, while allowing looping over nodes, is possible within the given heuristic framework, so we can indeed achieve designs

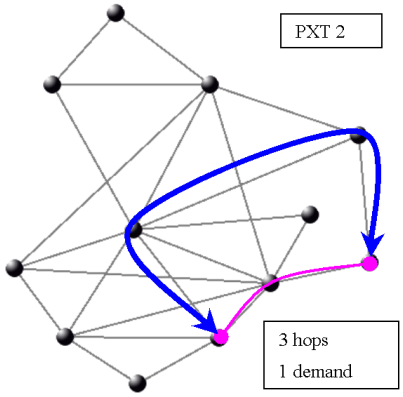


Figure 4.13: 1:1 APS-equivalent PXTs from the heuristic design

without fibre multiplication using a modified version of the PXT greedy heuristic, even though the PXTs will not in general be simple PXTs. We touch briefly on designs that allow this type of PXT in an experiment in Section 4.4.

PXT Lengths and 1:1 APS-Equivalents in the PXT Design

One marked characteristic of this design is a wide range of PXT lengths (from 289 to 6470). This is to be expected from the algorithm because its fundamental operation is to extend the length of PXTs. The algorithm attempts to extend existing PXTs in the network to protect additional demands, and long PXTs will have a higher chance of containing a protection path that can protect another demand. Therefore it is likely that already long PXTs will be made even longer, while short PXTs remain short because extending them to be as efficient as a longer PXT will generally require a larger relative capacity investment.

An interesting discovery arising from the exercise of visualizing each PXT from the design solution (Appendix E) is that PXTs at the lowest end of the length spectrum are actually just instances of dedicated 1:1 APS arrangements. This type of PXT makes up a significant fraction of the PXTs in the design. The design contains 16 individual (unit capacity) PXTs (out of a total of 31) of this type. One such PXT is illustrated in Figure 4.13.

In the context of the details of the algorithm, these 1:1 APS-equivalent cases can be thought of as instances of the "starter" PXTs for a single demand that never lead to any extension opportunities. In other words, no subsequently protected demands can extend these PXTs in an efficient manner for their own protection. This is a significant new insight

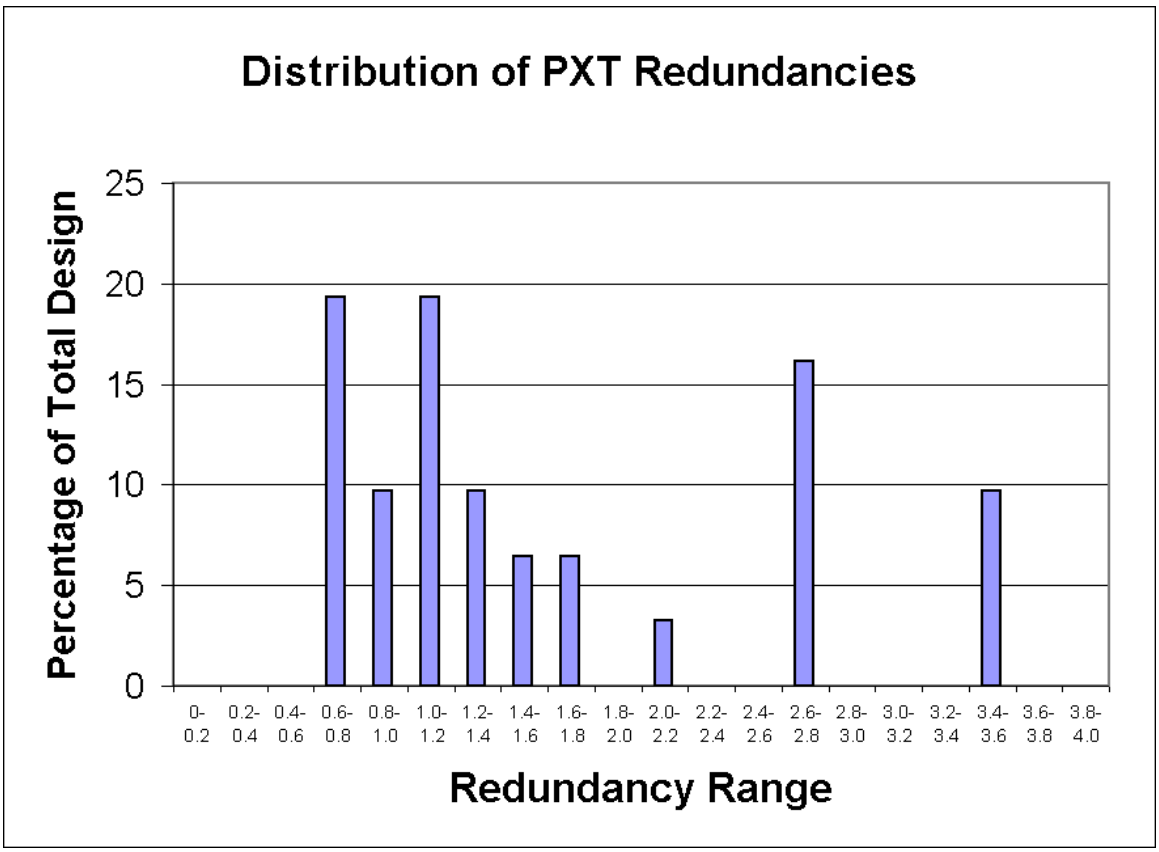


Figure 4.14: The distribution of individual PXTs according to their distance-weighted capacity redundancies in the heuristic design

about PXTs and how they relate to prior 1:1 APS concepts including the suggestion that APS-equivalent PXTs can be required components of an efficient overall preconnected network design. This is slightly counter-intuitive because taken individually no 1:1 APS setup can ever be less than 100% redundant.

Individual PXT Redundancy

The individual PXT redundancy (defined in Section 4.2.5.3 above) is a simple measure of the efficiency of any individual PXT in an overall network design. Of course, by itself, it does not take into account the interactions between many different PXTs in the design, meaning that the optimal design may contain PXTs that individually seem very inefficient, but are a required part of an efficient complete design. Nevertheless, investigating individual PXTs in terms of this metric is a useful way of characterizing the design. Figure 4.14 shows how the PXTs are distributed according to their individual redundancies.

The design contains a group of PXTs over a range of redundancies from 0.6 to 2.2, and then two distinct groupings in the 2.4-2.8 and 3.2-3.6 ranges. These groups of less efficient PXTs correspond to the 1:1 APS-equivalents. Apart from these outlying groups, however, we see that the algorithm generally produces more PXTs with better (lower) individual redundancies, despite the fact that this is not an explicit goal of the algorithm. Therefore this metric may be used a priori to roughly approximate the desirability of (longer) PXTs in the network design.

Need for an Optimization-Based ILP Design Method

At this point our research established a real need for a different type of PXT design method. The reasons are twofold. First of all, it is preferable that protection structures be compact, manageable, and easily visualized and maintained. We have also outlined the undesirability of looping PXTs in a network where wavelength continuity is necessary. The PXT heuristic does not provide a simple way to control these measures. Secondly, the existing algorithm is designed as an online algorithm. It can also be used in an offline manner to solve the "green fields" design problem, but if used in this way its output will depend on some arbitrary ordering of demand arrivals. This shows that it is not tailored to the offline design problem. Therefore we saw a need for an algorithm designed specifically for the offline case.

Traditionally, the method used to solve the offline design problem has been to write ILP models expressing the constraints and to use readily available commercial programs to solve these problems. It is also well known that an ILP can provide precise control over the structures used in a restorable network design (e.g., the *p*-cycle design model, in which the candidate *p*-cycle set is explicitly enumerated and thus can be filtered subject to any number of criteria that the designer wishes). Therefore we proceeded to apply ILP methods to the PXT design problem.

4.3 Optimization-Based Approach to PXT Design

4.3.1 Motivation

As is often the approach when answering basic questions of network science, we initially only wanted only to use ILP methods to establish a lower bound on the cost of the PXT architecture. We intended to develop an ILP-based reference model to discover the fundamental limits to the PXT concept's efficiency and find out how close the heuristic from [ChCh04] could approach truly optimal designs. It occurred to us during the investigation, however, that an ILP-based approach also provides a practical alternative design method in its own right, under which the complexity of individual PXT structures can be easily controlled.

The optimal solution of a complete PXT design problem is very difficult to obtain in general. The main contributors to the complexity of a formal ILP design model for PXT network design are the number of distinct trails that exist in a network (especially if looping is allowed), and the fact that for each such candidate PXT there is a combinatorial explosion in the number of combinations of demands that the PXT may be able to protect. Of all these combinations only a small fraction will be valid because any combination of demands with non-disjoint working routes is not allowed to share protection capacity. Encoding this restriction in the model requires a constraint for each demand pair/PXT combination, resulting in a huge number of constraints for even small problems. However, we would be misguided to allow the complexity of the entire problem, solved to complete optimality, to distract us from the fact that an ILP model of the same problem, solved with only a partially-populated set of candidate structures and protectable demand combinations, can constitute an effective and easily tailored form of heuristic in its own right.

By way of example, the same problem was encountered when formulating the optimization problem for Failure Independent Path-Protecting (FIPP) *p*-cycles [KoGr05], another preconnected network protection architecture. An ILP model with reduced complexity was developed for FIPP *p*-cycles in [KoGr05a] by taking the so-called Disjoint Route Set (DRS) approach, which is applicable to PXT design as well. The DRS algorithm is a heuristic approach that uses ILP methods. The heuristic aspect is introduced when, instead of considering every possible combination of working routes that could be protected

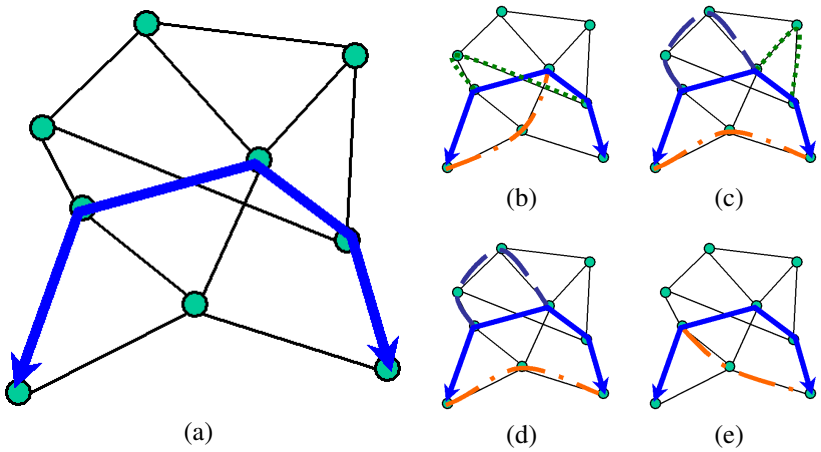


Figure 4.15: A PXT (a) and 4 possible DRSs that it could protect ((b) through (e))

by each PXT, we consider for protection only a limited collection of randomly assembled sets of working routes that are all disjoint from each other. Each of these sets is called a DRS. Because no single span failure can affect more than one working path in a DRS, it is impossible for spare capacity contention to occur on the PXT that protects a DRS. Figure 4.15 illustrates the DRS concept by showing a PXT alongside 4 possible combinations of DRSs that could be protected by it. If the randomly generated set of DRSs for this PXT consisted of only those shown in the Figure, the solver would have the option of using the PXT to protect these 4 different combinations of demands only.

Once the DRSs and a set of candidate PXT structures are generated, the ILP model needs only to assign PXTs to the protection of the DRSs that are present in the generated set. The DRS generation approach, like the approach of generating a limited set of candidate structures, is used to reduce the burden on the solver by reducing the size of an exponentially large set to something manageable at the expense of accepting sub-optimality.

Because this method worked so well for FIPP p-cycles, we decided to adopt it for PXT design as well. We call this an "optimization-based" heuristic PXT design method because, even though the resulting designs will not in general be truly optimal, the ILP solver is used to ground the results against truly optimal solutions in the sense that the results can be improved in the direction of true optimality by providing the model with a more complete set of parameters to work with. The following Section describes the DRS-based PXT ILP model. This model is conceptually identical to the model used for FIPP p-cycles, though

its use for PXT design is novel.

4.3.2 ILP Model

The DRS-based PXT ILP model is as follows:

ILP Model

Sets:

S The set of spans in the network, indexed by j.

P The set of all candidate PXTs, indexed by k.

C The set of all DRSs, indexed by c.

D The set of demand relations, indexed by r.

Input Parameters:

C_j The cost of a unit of capacity (i.e., a single channel) placed on span j.

d_r The number of demand units required by relation r.

δ_j^k Encodes the spans of a PXT. It is equal to the number of times that PXT k crosses span j. Can only be 0 or 1 for simple PXTs.

x_c^k Encodes the protection relationship between PXT k and DRS c. $x_c^k = 1$ if PXT k can offer protection for DRS c. $x_c^k = 0$ otherwise.

ζ_c^r Encodes the relationship between DRS c and demand pair r. $\zeta_c^r = 1$ if DRS c contains demand r. $\zeta_c^r = 0$ otherwise.

β_r^k The number of protection paths PXT k can provide to demand pair r. This can be any integer greater than or equal to zero, as some PXTs may be able to provide multiple protection paths simultaneously.

Decision Variables:

n^k The number of copies of PXT k used in the design.

n_c^k The number of copies of PXT k used to protect DRS c specifically.

s_j The number of spare channels allocated on span j in the design.

Objective Function:

Minimize

$$\sum_{j \in S} C_j \cdot s_j \tag{4.1}$$

Constraints:

$$d_r \leq \sum_{k \in P} \sum_{c \in C} \zeta_c^r \cdot x_c^k \cdot \beta_r^k \cdot n_c^k \quad \forall r \in D \tag{4.2}$$

$$n^k \geq \sum_{c \in C} x_c^k \cdot n_c^k \quad \forall k \in P, \forall c \in C \tag{4.3}$$

$$s_j \geq \sum_{k \in P} \delta_j^k \cdot n^k \quad \forall j \in S \tag{4.4}$$

Constraint 4.2 ensures that all the demand between node pair r can be fully protected by PXTs. Constraint 4.3 ensures that the total number of instances of PXT k provisioned is equal to the sum of all the requirements for this PXT from each individual DRS. Constraint 4.4 ensures that sufficient spare capacity exists to form all of the PXTs selected by the design. This model is conceptually identical to the FIPP DRS model recently presented in [KoGr05a], the only difference being that the set of candidate FIPP p-cycles is simply replaced with a set of candidate trails for our purposes.

Note that this model comes close to being a complete mathematical representation of the PXT design problem, but falls short only because of the DRS assumption. In the generalized design problem, a PXT is allowed to protect sets of non-disjoint working routes as well, as long as their response to joint failure does not result in capacity contention. The DRS simplification forces PXTs to protect only sets of disjoint routes. This is desirable from both the standpoint of computational complexity and operational complexity [KoGr05a], but is nonetheless an approximation of the "full" definition of PXTs from [ChCh04]. Apart from this one assumption, heuristic compromises are introduced only according to how the parameter sets are formed for a particular numerical instantiation of the model. While we could theoretically achieve the full efficiency potential of this model

by fully populating the candidate sets, in practice this is not possible. Generally the model will not be usable to obtain completely optimal DRS-protecting PXT designs because (1) the set of candidate PXTs P provided to the solver will be incomplete (because the set of all possible trails is unmanageably large; in fact, if trails are allowed to repeat spans and nodes, it is infinite), and (2) the set of DRSs C provided to the solver will also in general be incomplete (because the set of all possible DRSs will also be prohibitively large).

4.3.3 Pre-processing to Produce the Model Inputs

A pre-processing data preparation stage is required in order to generate the set of candidate PXTs, the set of DRSs, and other input parameters. This stage was implemented as a stand-alone program that accepts user-defined values for the following parameters:

The minimum number of disjoint route sets containing each demand: The DRS generation algorithm begins by placing a single "seed" route into an initially empty DRS, and then continues by randomly adding working routes to the DRS that are disjoint from all the routes currently contained in the DRS. This is done until no more routes can be added, either because the DRS size limit has been reached or because no more disjoint routes exist. This process is performed using each demand in the network in turn as the seed, and is repeated for each seed a number of times that is defined by this parameter. This ensures that each demand is represented in the given number of different DRSs at the very least, although it may of course also be added to other DRSs as part of the DRS generation process for different seed demands. Additionally, a special set of DRSs is generated, independent of the value of this parameter, which consists of DRSs containing only a single demand. One such DRS is generated for each demand in the network. This is done to allow the possibility of simple 1:1 dedicated APS options in the solution, if this is optimal. More on 1:1-equivalent PXTs is explained in the results discussion.

The minimum number of candidate trails to generate for the protection of each DRS: After the set of candidate DRSs is assembled, candidate trails are generated. To assemble this set, each DRS contributes a set of trails that pass through the source and destination node of each demand in that DRS. A trail in this set may not necessarily

protect all of the demands of the DRS, because it may not be able to provide disjoint protection paths to every working path. So, to ensure an adequate number of protection choices for each demand in the problem, enumeration of these trails continues until each demand in the DRS is protected by a number of trails equal to or greater than the value of this parameter. Therefore the number of trails in the set will always either equal or exceed this value. These trails are generated from shortest to longest, i.e., first the shortest suitable trail is found, then the second shortest, and so on until this criterion is met. The reason for choosing short trails is straightforward; a short trail will be more efficient than a long trail for protecting a given DRS, as it uses less spare capacity to do so.

The maximum number of working paths in a DRS: It was found in [KoGr05a] that restricting the size of DRSs to a reasonable number of demands reduces complexity while not having a significant impact on solution efficiency. The maximum size of any DRS can be limited via this parameter. This control is also of value when overall service availability is considered in the case of multiple simultaneous failures, as this is a direct way to limit the total number of protection relationships any individual PXT will bear.[3]

The data preparation stage also performs working path routing and working capacity assignment. To facilitate comparisons to results for the heuristic from [ChCh04], the ILP-based heuristic uses the same working path routing for demands.

4.3.4 Experimental Method and Test Cases

The ILP model was used to generate a 100% single span failure restorable network design for a single test network. The network used was again the 12 node and 24 span (degree-4) "Murakami & Kim" network topology from Figure 4.8. Span costs were assigned based on the Euclidean distance between nodes in the network as drawn. The demand pattern consisted of three unit capacity connection requirements for every node pair in the network. These are all the same test parameters that were used for the tests of the heuristic in Section

[3]The point is that although any solution from the above model will be 100% single-failure restorable, the susceptibility to dual failures, which is what determines availability in a survivable network, depends directly on how much sharing of protection paths is allowed to occur. (See for example [ScGr04])

4.2.5.2. As before, this is network design in an offline sense, in which we presume to know the demand volumes fully ahead of time, and span capacities are assigned without imposing any pre-determined maximum. However, the same method can also be adapted as an online algorithm; we touch on this later on. Note that, unlike our tests of the heuristic that used a single arbitrarily randomized demand order, the ILP model does not have any inherent ordering or order-dependency on the demands to be protected; protection of the entire set of demands is considered concurrently as a single overall optimization problem.

The pre-processing program used to populate the data set of the ILP model generated 20 DRSs for each "seed" working route and 20 candidate trails per DRS. Maximum DRS size was limited to 10 routes. For this step all candidate trails were also constrained to be simple trails. This both helps to limit the set of candidates to a reasonable size as well as meets the aim of eliminating non-simple PXTs from the resulting network design. This particular test network contains a total of 10,922 such trails (i.e., this is the total number of distinct simple trails that exist between all node pairs). The set of such candidate trails for the problem was found directly by exhaustive depth-first search. Although we could have explicitly limited the maximum length of candidate PXTs at this stage as well, we wanted to produce initial comparative test results using the entire set of "tamed" candidate PXTs for this graph. In this circumstance, the graph itself limits the maximum hop length of any PXT for us however. There being 12 nodes, and with no looping allowed, this means that no candidate PXT in this design will be over 11 hops in length. The ILP model was implemented in AMPL 9.0 and solved using the CPLEX 9.0 MIP solver. An optimal[4] solution was obtained by CPLEX after 2.1 hours.

4.3.5 Results

4.3.5.1 Summary

As in our prior PXT characterization effort, we graphically inspected the tamed PXTs in the design produced by the ILP model. Diagrammatic portrayals of each PXT in this design are given in Appendix F, in which the network design is broken down into figures showing

[4]Use of the term "optimal" in this context refers only to a complete CPLEX termination for this particular problem tableau (which, as mentioned, is only partially-populated in terms of the DRS options available). Because the full problem was not solved, the results do not represent a truly optimal PXT design, but rather an optimal solution of a problem which is an approximation of the full problem.

each unique PXT along with the working paths that it protects. PXTs are drawn with thick blue lines with arrows at the ends. The working paths that the PXT protects are illustrated by the multi-coloured lines that end in circles. The ILP model generates PXT designs by capacitating each candidate PXT a number of times, so the diagrams also indicate how many times each PXT is capacitated. Note that the design employs candidate PXT 36 twice in two separate diagrams. This is not simply a two-channel deep instance of the same PXT used for a common group of demands, but rather the PXT is used twice to protect two different DRSs. In contrast to the portrayals of the heuristic-generated PXTs (Appendix E) it can be immediately appreciated why we refer to the PXTs found in the present results as being "tamed".

The total cost of working routing was 59,696 (identical to the routing for the heuristic on this network, as they use an identical algorithm). The units of cost are distance-channel counts (i.e., if the lengths of spans are taken in km then costs are in units of channel-kilometers). The cost of spare capacity for the ILP design was 57,476. This gives it a redundancy of 96.3%, which is 6.5% higher than the redundancy (89.8%) achieved by the greedy heuristic. This is entirely attributable to the taming restrictions we have now added (that the solution employ only simple trails as candidate PXTs) and the limitations on DRS candidates. The result suggests that the restrictions that improve PXT characteristics and reduce the ILP problem to manageable complexity have a measurable, but not large, effect on the efficiency achievable with the concept.

To check whether the restricted population of the data sets of the ILP problem was significantly limiting the efficiency of the design, several more trials were attempted with parameter values ranging up to 30 DRSs per demand, 40 PXTs per DRS, and a maximum DRS size of 20 demands. In none of these cases did the redundancy of the resulting design improve on that of the original design by more than 1%. Therefore we can consider the ILP-based PXT design to have a network cost redundancy that is close to the best that can be expected from this approach. In other words, it is indeed the taming restrictions of non-looping PXTs and DRS-only protection that account for the reduced design efficiency, and not side-effects introduced by the heuristic compromises that make the design problem solvable. Although we can only partially populate the problem model, the DRS/PXT decision alternatives are sufficiently well represented as to be achieving very nearly global

optimality given the fundamental constraints of this particular model. In addition, these results confirm that this design is typical in terms of efficiency, and not a statistical outlier caused by a set of unusually good or bad randomized choices (i.e., in the randomized selection of demands for DRS generation).

We also compared these results with those for *p*-cycle and span-restorable mesh designs for the same topology and demand pattern (calculated in Section 4.2.5.3). Again, these solutions are completely optimal for their respective architectures. Their spare capacity costs were 51,748 (*p*-cycle) and 46,681 (span-restorable mesh), corresponding to redundancies of 86.7% and 78.2% respectively. Therefore the ILP heuristic algorithm was able to approach the efficiency of an optimal *p*-cycle design within 10%.

4.3.5.2 Statistical Characterization of the Design

For comparative purposes, the same metrics as in Section 4.2.5.3 were calculated for the tamed PXTs from the ILP-based design heuristic. Table 4.6 lists these metrics for each PXT from the design. Each PXT is assigned a numerical identifier for reference.

4.3.6 Discussion

4.3.6.1 Complexity Metrics: Comparing Tamed and Untamed PXT Designs

Of course, because the ILP method restricts all PXTs to only be simple trails, the self node-crossing and self span-crossing values in Table 4.6 are all zero for the tamed design. Similarly, because the DRS restriction allows PXTs to only protect demands that are disjoint from each other (making the network state much simpler to comprehend when in a restored service state), the maximum simultaneous protected failures are all 1 (because no more than one demand in a DRS can fail at once). This emphasizes that the ILP method can be used to easily restrict PXT complexity with a high degree of control. For example, if a network operator deemed that PXTs with a single self-node-intersection were acceptably simple, the ILP-based heuristic could easily admit such PXTs simply by modifying the preprocessing program to generate that type of PXTs as well. Adding such a consideration to the greedy heuristic would be much more difficult, if not totally impractical.

Regarding the longest protection path provided by each PXT, it is interesting to see that this value for the tamed ILP PXTs is not significantly shorter than for the heuristic PXTs.

PXT ID	Hops	Length	Copies	Closed	Self node-crossings	Self span overlaps	Number of Demands Protected	Path-Structure Coincidence Count	Cost of working capacity protected	Maximum simultaneous protected path failures	Longest Protec-tion Path (Hops)	Longest Protec-tion Path (Length)	Cost Re-dundancy
0	2	285	3	No	0	0	1	0	115	1	2	285	2.48
1	2	289	1	No	0	0	1	0	111	1	2	289	2.61
2	2	322	3	No	0	0	1	0	123	1	2	322	2.62
3	2	327	1	No	0	0	1	0	92	1	2	327	3.55
4	2	343	1	No	0	0	1	0	210	1	2	343	1.63
5	2	347	3	No	0	0	1	0	341	1	2	347	1.02
6	2	349	1	No	0	0	1	0	324	1	2	349	1.08
7	2	356	3	No	0	0	1	0	187	1	2	356	1.90
8	2	375	1	No	0	0	1	0	352	1	2	375	1.07
9	3	425	3	No	0	0	1	0	226	1	3	425	1.88
10	2	442	3	No	0	0	1	0	391	1	2	442	1.13
11	2	443	2	No	0	0	1	0	394	1	2	443	1.12
12	3	460	3	No	0	0	1	0	302	1	3	460	1.52
13	3	524	2	No	0	0	1	0	502	1	3	524	1.04
14	3	548	1	No	0	0	1	0	509	1	3	548	1.08
15	3	549	3	No	0	0	1	0	165	1	3	549	3.33
16	3	607	2	No	0	0	1	0	226	1	3	607	2.68
17	3	608	2	No	0	0	1	0	229	1	3	608	2.65
18	7	1274	1	No	0	0	5	2	1627	1	5	979	0.78
19	9	1331	1	No	0	0	8	3	2316	1	9	1331	0.57
20	9	1349	1	No	0	0	6	3	1901	1	6	1012	0.71
21	9	1354	1	No	0	0	8	2	2088	1	5	871	0.65
22	10	1407	1	No	0	0	5	2	1815	1	7	1001	0.78

23	10	1427	1	No	0	0	7	4	2094	1	10	1427	0.68
24	10	1427	1	No	0	0	6	6	2251	1	8	1201	0.63
25	8	1441	1	No	0	0	8	2	2094	1	6	1094	0.69
26	9	1491	1	No	0	0	9	5	2443	1	8	1281	0.61
27	9	1526	1	No	0	0	7	3	1946	1	8	1411	0.78
28	10	1552	1	No	0	0	7	3	1906	1	5	787	0.81
29	9	1565	1	No	0	0	8	3	2147	1	7	1199	0.73
30	9	1580	1	No	0	0	6	3	2090	1	8	1446	0.76
31	10	1678	1	No	0	0	9	1	2205	1	8	1441	0.76
32	11	1685	1	No	0	0	7	2	1779	1	9	1389	0.95
33	10	1696	2	No	0	0	6	4	1687	1	9	1604	1.01
34	9	1702	1	No	0	0	6	2	1949	1	7	1406	0.87
35	10	1722	1	No	0	0	7	4	2447	1	7	1249	0.70
36	11	1732	1	No	0	0	7	4	1989	1	8	1271	0.87
36	11	1732	1	No	0	0	5	4	1796	1	8	1271	0.96
37	11	1737	1	No	0	0	7	4	2249	1	9	1396	0.77
38	11	1774	2	No	0	0	7	5	2310	1	8	1167	0.77
39	10	1823	1	No	0	0	6	3	2035	1	8	1597	0.90
40	11	1850	1	No	0	0	8	3	2263	1	9	1643	0.82

Table 4.6: Breakdown of the properties of the PXTs generated by the ILP-based PXT design heuristic method for the Murakami & Kim network

The average values for the ILP and greedy heuristic designs are almost identical: 786 and 772 (distance units) respectively. The metric also reaches a similar maximum value of approximately 1600 in both networks. This can be explained by the fact that any PXT design will have a built-in upper limit on the degree of sharing that can occur in its PXTs, due to the inherent limitations of both the network topology and the PXT architecture itself. This creates corresponding practical limits on the lengths of protection paths, because longer protection paths require more sharing of the capacity in order to remain efficient. Therefore, we should expect that any two designs of roughly equivalent efficiency, no matter how they are obtained, would display convergent protection path length limits, even if their other characteristics were markedly different. But what the result makes clear is that while both methods are able to formulate a preconnected protection plan with almost identical properties of cost and protection path length, the ILP method is able to "untangle" the highly inter-connected protection arrangements produced by the greedy heuristic into a larger, but more manageable, set of tamed PXT structures.

4.3.6.2 PXT Lengths

The PXT lengths generated by the greedy heuristic ranged from 289 to 6470. In contrast, the lengths in the ILP design are only between 285 and 1850. PXT length is reduced by more than two thirds here as a consequence of only using tamed PXTs in the ILP problem. Furthermore, we could have equally as easily limited their maximum length directly by restricting the candidate PXT lengths even further, if necessary. Other factors being equal, shorter PXTs seem desirable wherever optical power and impairment budgets are an issue, for example, and the use of smaller protection structures also localizes management and reconfiguration effects. Therefore the ability to manage PXT lengths in this way is highly advantageous. Checking the details of the candidate set, we found that the longest PXT candidate given to the ILP model is 2000 distance units long and the shortest is 92 units long. Therefore the PXTs used in the solution span almost the entire range of candidate PXT sizes.

In addition, the lengths of the PXTs that were used in the ILP solution appear to be divided into two distinct groupings, one group of short PXTs of only a few hops, and another group of PXTs of almost the maximum length of a simple trail (11 hops for this

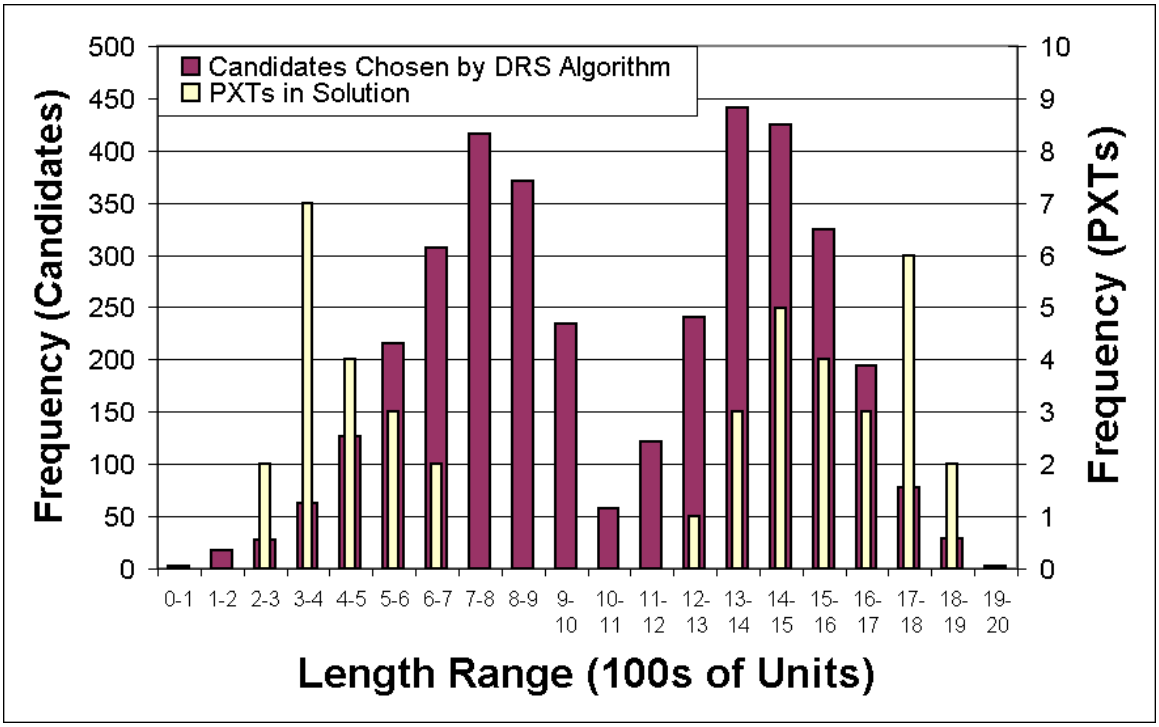


Figure 4.16: Histogram of PXT Lengths in the ILP design (both for the candidate set and the PXTs in the resulting design)

12 node graph). Figure 4.16 uses a histogram to illustrate this grouping tendency. The graph shows the frequency of occurrence of lengths in both the set of candidate trails and in the set of PXTs used by the solution (for the purposes of this histogram, only distinct PXTs in the solution are counted, i.e., multiple copies of the same PXT are not taken into consideration). The set of candidates is already biased into a bimodal distribution in the pre-processing step due to the two distinct sets of DRSs that are generated: regular randomized DRSs and the special single-demand DRSs, described earlier in Section 4.3.3. When the pre-processing program generates the set of trails for the potential protection of each of the DRSs, short PXTs will generally be generated for the single-demand DRSs and long PXTs will generally be chosen for the large DRSs. However, the set of PXTs used in the solution is even more biased towards the extremes of the length spectrum.

4.3.6.3 1:1 APS Equivalences

In Section 4.2.5.4 it was discovered by drawing out the PXTs from the heuristic network design that many of the PXTs produced by the greedy heuristic are simply instances of

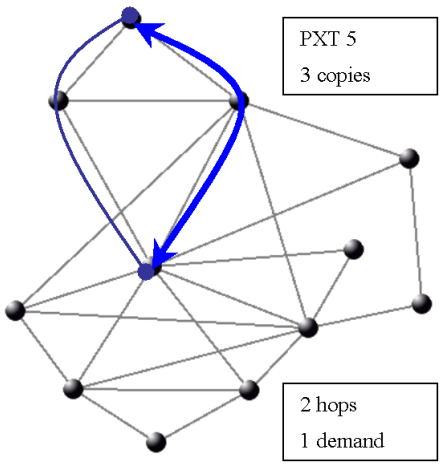


Figure 4.17: 1:1 APS-equivalent PXT from the ILP design

dedicated 1:1 APS arrangements. When the same exercise was performed for the ILP-based design, a similar set of 1:1 APS PXTs was observed. The ILP design has 18 distinct PXTs of this type (out of a total of 41 distinct PXT structures). Again, multiple unit capacity copies of a single structure are not considered in this measurement. These PXTs are shown in Appendix F as PXTs 0 through 17. Figure 4.17 gives an example of one of these PXTs. It turns out that it is mainly these 1:1 APS-equivalent PXTs that make up the lower lobe of the PXT length distribution in Figure 4.16. Because such arrangements protect only one demand in a totally redundant end-to-end manner, they must be short in order to be efficient. Conversely, the rest of the PXTs protect DRSs that contain many demands and so tend to be long in order to maximize the number of protection paths they can provide.

The emergence of the 1:1 APS-equivalent cases from the greedy heuristic was explained as resulting from the algorithm's greedy nature. But the presence of many 1:1 equivalents in the ILP design is unexpected and harder to explain. Because the ILP-based method is closer to being a global optimization, yet we again see the appearance of significant numbers of 1:1 APS-equivalents, we conclude that this is probably a fundamental aspect of efficient PXT network design, not an effect arising from any particular methodology alone. This is further evidence that certain 1:1 APS structures, though individually inefficient, can still be an essential part of low-cost restorable network designs.

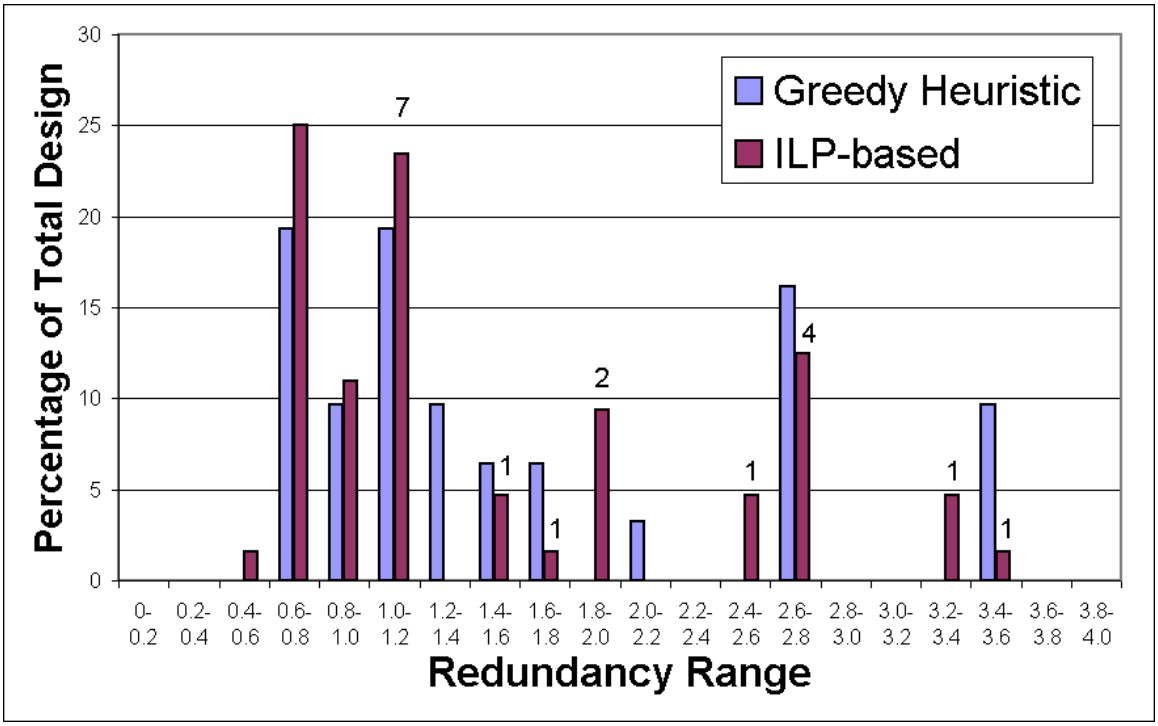


Figure 4.18: Comparison of capacity redundancies of individual PXTs between the greedy heuristic and ILP-generated designs

4.3.6.4 Individual PXT Redundancy

As described earlier, the individual capacity redundancy of a PXT is defined as the ratio of the spare capacity used by that PXT to the total amount of working capacity it can protect. Figure 4.18 compares the capacity redundancy distribution of the PXTs in the greedy heuristic design with those from the ILP-based heuristic design. Overall, both designs show the same general pattern of many PXTs with low redundancies in addition to separate groupings in 2.4-2.8 and 3.2-3.6 ranges. The groupings of PXTs with higher redundancies correspond mostly to the less efficient 1:1 APS-equivalents (although there are 1:1 APS PXTs in lower redundancy ranges as well). The numbers above certain bars in the Figure represent how many of the 18 1:1 APS PXTs in the ILP design are in that range, if there are any.

The fact that the 1:1 APS PXTs are grouped around approximately the same redundancy ranges in both designs suggested to us that both the ILP and the heuristic may have used approximately the same 1:1 APS PXTs. Indeed, inspection of the solutions reveals that 8 of the 11 1:1 APS-equivalents used in the heuristic design are also found in the ILP design.

This indicates that the greedy heuristic is relatively good at finding the individually good 1:1 APS-equivalent PXTs. The fact that these PXTs are not desirable to be extended for the protection of additional demands by the greedy heuristic must translate well into the realm of the ILP optimizer, which also decides that these demands cannot efficiently share their protection with a group of other demands as part of a long PXT.

4.3.7 Further Experiments

Some of the results suggested further questions that were able to be answered by small additional side studies. The following Sections give a brief overview of the methodologies and results obtained.

4.3.7.1 Comparison to FIPP *p*-Cycles and FIPP *p*-Cycle Hybrids

Having investigated the properties of PXTs and PXT design algorithms independently, it is of interest to know how PXTs compare to other protection architectures. We have already mentioned comparable results on a single test network for both *p*-cycles and span-restorable mesh (in Section 4.3.5), which show that PXTs (in this network) can compare favorably to both. However, while we can use these architectures as efficiency references, they are not very comparable structurally to PXTs, because both are span-protecting techniques. A more suitable choice for comparison, as an established high-efficiency path-protecting architecture, are FIPP *p*-cycles. Additionally, as we use the same ILP model to compute designs for both, we can take advantage of this in the same way we did for span *p*-trees and *p*-cycles and also investigate the properties of optimal (or near-optimal) hybrid designs.

Method

Both sets of comparative results (a FIPP *p*-cycle design and a hybrid design) were generated on the same (Murakami & Kim) test network using the same demand pattern, working routing, and DRS parameters as in Section 4.3.4 (20 DRSs per demand, 20 candidate structures per DRS, and maximum DRS size of 10). The FIPP *p*-cycle design was given all simple cycles as the set of candidate cycles. The hybrid design was given both the set of all simple cycles and the set of all simple trails as candidate structures.

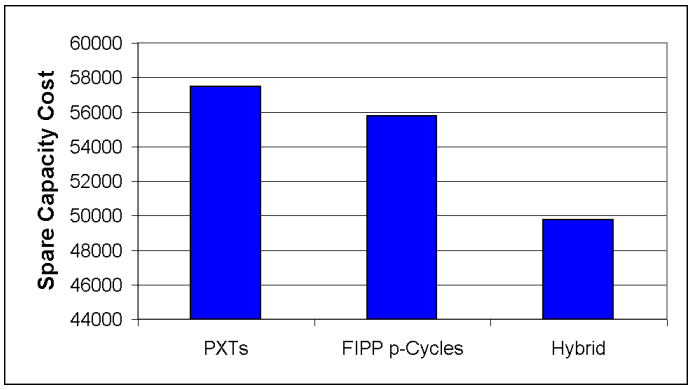


Figure 4.19: Comparison of PXT, FIPP p-cycle, and hybrid designs

Results

The results are illustrated in Figure 4.19. We see that the PXT and FIPP p-cycle designs both have similar capacity costs. Also, we are able to achieve more than a 10% decrease in cost with the hybrid designs. This is in stark contrast to the span-protecting case, where the addition of segments was not able to improve p-cycle designs by greater than 1.6% (see Section 3.3 and following Sections on limiting the tree selection to segments only). This suggests that non-cyclical protection structures are much more suited for path protection than span protection. From these results, we can conclude that PXTs deserve at least equivalent consideration with FIPP p-cycles in terms of efficiency in the field of pre-cross connected, path-protecting restoration strategies. For a more detailed examination of PXTs vs. FIPP p-cycles and hybrid designs, see Section 5.3, which encapsulates a discussion of the contribution of PXT protection into a discussion of path p-trees/FIPP p-cycle hybrids.

4.3.7.2 Further Limitation of PXT Lengths

Given that our results showed that it is possible to limit the complexity (and, as a side-effect, length) of PXTs without giving up much in the way of efficiency, it was interesting to us to investigate the consequences of limiting these characteristics even further. To this end, we performed another experiment in which tamed PXT designs were produced using the same method, network topology, and parameters used in the original design experiment, except that the set of candidate PXTs was further restricted to contain only simple trails less than a certain number of hops in length. Designs were generated with hop limits from 11 down to 4. Reducing the hop limit past 4 results in an infeasible design problem, because in these

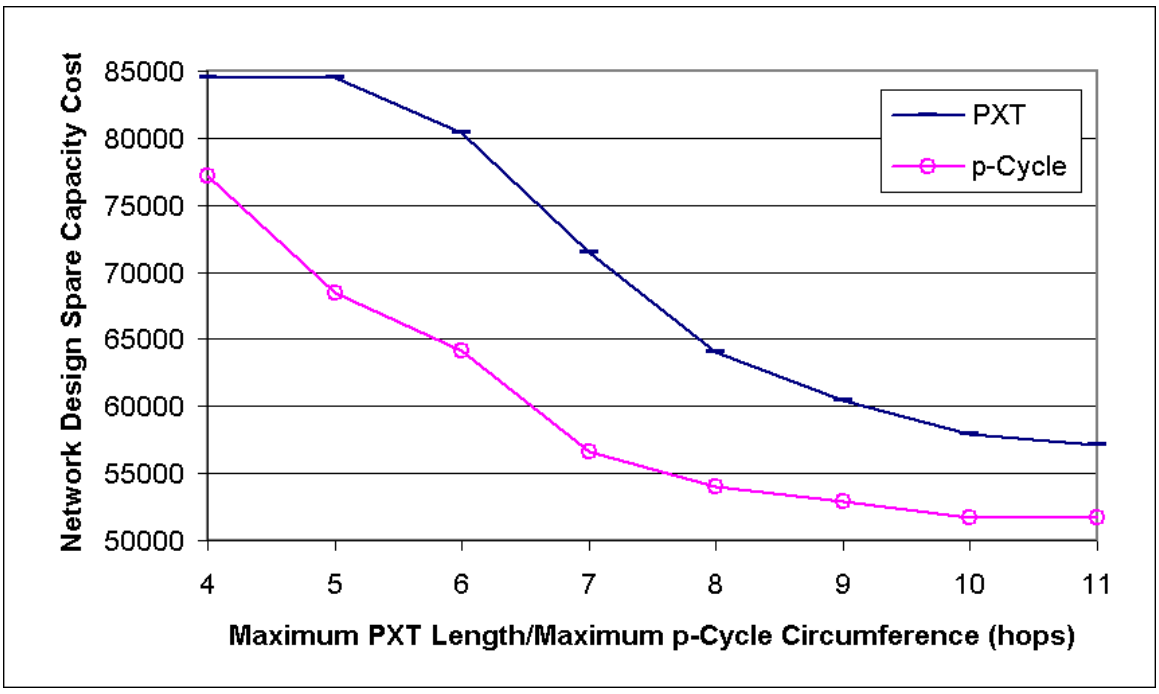


Figure 4.20: The costs of PXT and *p*-cycle network designs in which the set of candidate structures is limited by structure length

cases there do not exist any PXTs long enough to protect certain demands. Because the longest possible trail in the network is 11 hops long, the 11-hop-limited case represents another design created using the exact same data set and parameters as used in Section 4.3.4 (although it will not be identical because the data set is created using a randomized selection of DRSs, as explained in Section 4.3.3).

Figure 4.20 shows the costs of the network designs produced by these 8 experiments, and for comparison, the costs of corresponding *p*-cycle designs produced with the same limits on the maximum circumference of the set of candidate cycles. The *p*-cycle reference designs used the same working path routing as the PXT designs. The PXT curve reproduces a trend that is already well known in the *p*-cycle case, in which the addition of longer structures provides diminishing returns as the allowed length becomes larger. However, this effect is not as pronounced in the PXT case, as the reduction in capacity cost is still significant even as the allowed PXT length approaches the maximum of 11.

To examine this trend in more detail, Figure 4.21 shows the same values from Figure 4.20 normalized to the spare capacity costs of reference designs with no limit on structure size (for PXTs and *p*-cycles respectively). Hence the position of each data point on the

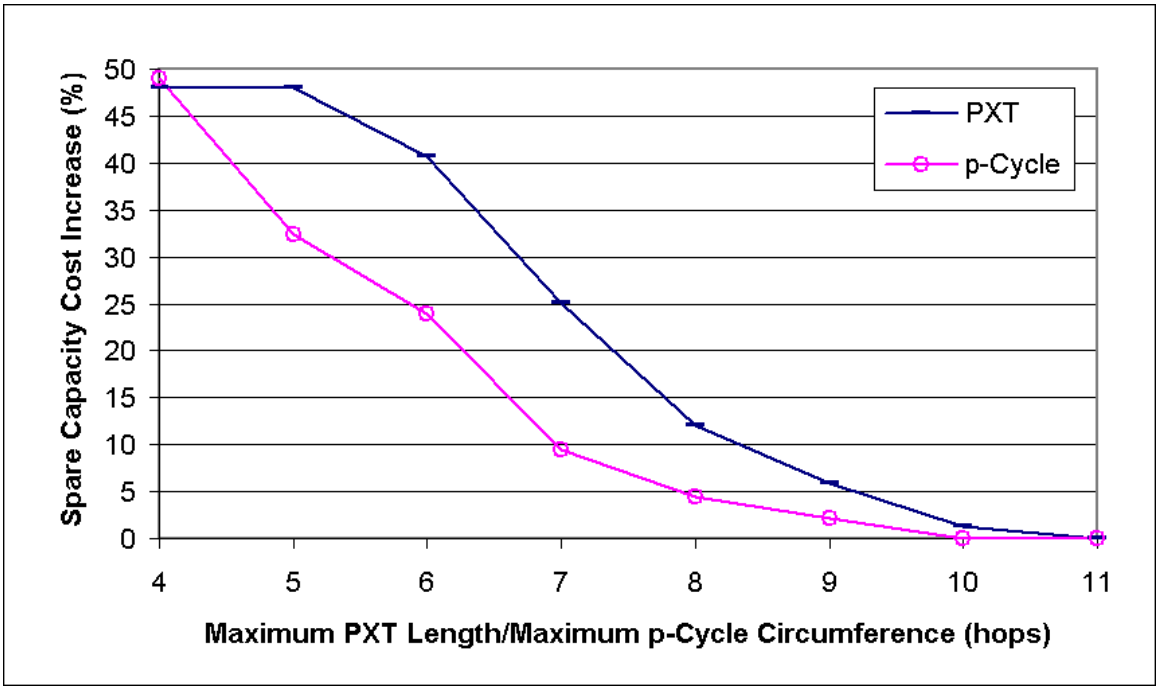


Figure 4.21: The network design costs shown in Figure 4.20 normalized to the cost of of the corresponding reference design with no limit on structure size

y-axis represents the capacity penalty for restricting the structure size to the value on the x-axis. Here the previous observation becomes more pronounced. The *p*-cycle designs only increase in cost by 5% if the size of the candidate cycles is limited to 8, whereas the corresponding limited PXT design is 12% more costly than that of the unlimited case. This phenomenon suggests that PXTs (at least when they are restricted to being simple trails) are more reliant on length for efficient protection than *p*-cycles are reliant on cycle circumference. This makes sense, given that PXTs must always be at least as long as the working paths they protect (assuming shortest-path working routing), whereas *p*-cycles protect on an individual span-by-span basis and thus do not have an explicit lower bound on their size.

We also attempted the converse experiment, in which the limitations on the PXTs were relaxed instead of tightened. We took the approach of allowing the PXTs to loop through nodes (but not over spans), instead of forcing them to be strictly simple trails. As mentioned in Section 4.2.5.4, such a design would also be realizable in a single-fibre-pair network.

However, this dramatically increases the size of the potential candidate set, requiring us to rather arbitrarily limit the PXTs that are included in the problem, which in turn limits our

ability to make comparisons with previous results. We were able to generate a solution to within 1% of optimality for a problem that was given approximately 20,000 of the shortest node-looping PXTs (in addition to all 10,922 simple trails). However, its cost was actually 3% greater than the corresponding design using simple trails only. This small difference, combined with the 1% margin of error and the possibility of cost fluctuations introduced by the pseudo-random DRS algorithm, makes this result inconclusive at best.

4.3.7.3 Comparison of PXT Design Methods as Online Algorithms

We have shown so far that the ILP-based heuristic is a practical alternative to the original heuristic proposed in [ChCh04] when used as an offline algorithm. However, the greedy heuristic was originally proposed as an online algorithm for the protection of demands as they arrive dynamically during network operation. Therefore in this Section we outline a method for using our ILP-based heuristic as an online algorithm for survivable service provisioning, and describe an experiment that compares the performance of the two methods when used in an online sense. The results also provide insight into the workings of the heuristic and highlight the ability of the ILP-based approach to be more selective about its choice of PXTs while at the same time attaining good capacity efficiency.

Approach

The ILP model itself does not have to be changed in order to be used in the online context. Instead, we need only change the way in which the model is used. For online protection, instead of solving the model only once for the protection of an entire set of pre-determined demands, the model is solved each time there is a new demand arrival (or departure, if they are considered as well) to re-optimize the PXT configuration of the network. The routing of existing demands remains undisturbed when new demands arrive, as routing is always done via the shortest paths.

Experimental Method and Test Cases

We use the same approach to characterize online performance as used in [ChCh04] originally and also earlier in Section 4.2.3.1. That is, we consider only random arrivals (not departures) and examine total capacity use as demands are served, without any assumed

finite limits on edge capacities. To test the use of the ILP-based heuristic as an online algorithm and to compare it to the greedy heuristic, we simulated an online service provision scenario under random demand arrivals using the same Murakami & Kim network used above. The random arrival sequence consisted of all of the single-unit demands from the demand pattern used in tests from Section 4.3.4, organized in a randomized arrival order. Both the greedy heuristic and ILP-based method were used to protect these demands in this same randomized sequence.

Two variations of the ILP-based method were used. The first used the ILP model given in Section 4.3.2 with no modifications. The second used a slightly modified objective function:

Minimize

$$\sum_{j \in S} C_j \cdot s_j + \alpha \cdot \sum_{k \in P} n^k \tag{4.5}$$

The first term is the standard spare capacity cost objective function. The second term effects the bicriteria minimization of the total number of PXTs used, in addition to the usual capacity minimization. For our experiments, α was set to 1, which is a small value in this context. The presence of the second term, with small but nonzero weight, causes the solver to effectively choose amongst alternative solutions that are all equivalently optimal in the primary objective function term to find one that contains fewer PXTs. As the technique is used here, the second term is kept so small that its presence will not cause the solver to make concessions in the first term, being in this case the spare capacity cost of the network. (The value of α is thus arbitrary over a wide range and $\alpha = 1$ is just one such suitable value that we could use here.) This modified model is partly motivated by the thinking that the least complex network from an operator's point of view may be one with the smallest number of tamed PXTs, given that tamed PXTs in a network still remain easily understandable by a human designer even if they become long, in contrast to those that may loop. Therefore, if cost-wise equivalent designs exist, we should choose the one that uses the fewest PXTs.

The greedy heuristic is designed to deal with an ordered set of demands in any case, so so its method of use remained the same. In the two ILP-based tests (regular and bicriteria), first each new demand was routed using the same working routing algorithm as before, then the sets of DRSs and candidate trails were re-generated from scratch, and finally the model was used to solve the current cumulative protection problem, i.e., after the arrival of

demand N the problem was solved for the protection of all demands 1 through N together. For all three methods, the network state was recorded in each step.

Results

A graph of the spare capacity totals after each demand arrival for the three design methods is shown in Figure 4.22. Only two different curves appear because the bicriteria ILP model, as mentioned above, is designed to produce networks with almost identically the same costs as those from the regular ILP model. The only difference is that the number of PXTs in the bicriteria designs may be lower. The spare capacity totals are then broken down visually in terms of the average length of the PXTs in the solutions (Figure 4.23) and the number of PXT structures in the solutions (Figure 4.24). These two Figures are a breakdown of Figure 4.22 in the sense that the total amount of spare capacity in a design is, by definition, equal to the product of the average length of the PXTs and the number of PXTs.

The ILP curve in Figure 4.22 is quite close to the greedy heuristic curve, showing that the ILP is able to perform nearly as well as the greedy heuristic in the online context as well, over a wide range of demand volumes, while avoiding the formation of looping PXTs completely. However, what the graph does not show is the difference in computation times. Recall that each step of the heuristic is truly an incremental protection problem for a single new demand, while the ILP method reconsiders the protection of every demand protected to date. The result of this is that each iteration of the heuristic takes a fraction of a second, compared to a few hours for the ILP problems with the highest levels of demand volume. But in networks of the foreseeable future this is not necessarily a problem, as long as a notice period of equal or greater length is given before the demand is put into service.

Regarding the two variants of the ILP-based method, the designs produced by the bicriteria ILP were identical to those of the unmodified ILP in the vast majority of cases. In only 2% of the cases was a design with fewer PXTs found, and then only by at most 3 fewer PXTs. Furthermore, in none of these cases were the network design costs found to be exactly the same, with the only difference between the two designs being the number of PXTs used. In other words, the spare capacity cost of the design always had to be increased slightly in order to accommodate a reduction in the number of PXTs (with a net reduction in the value of the bicriteria objective function, despite the choice of a small value for α).

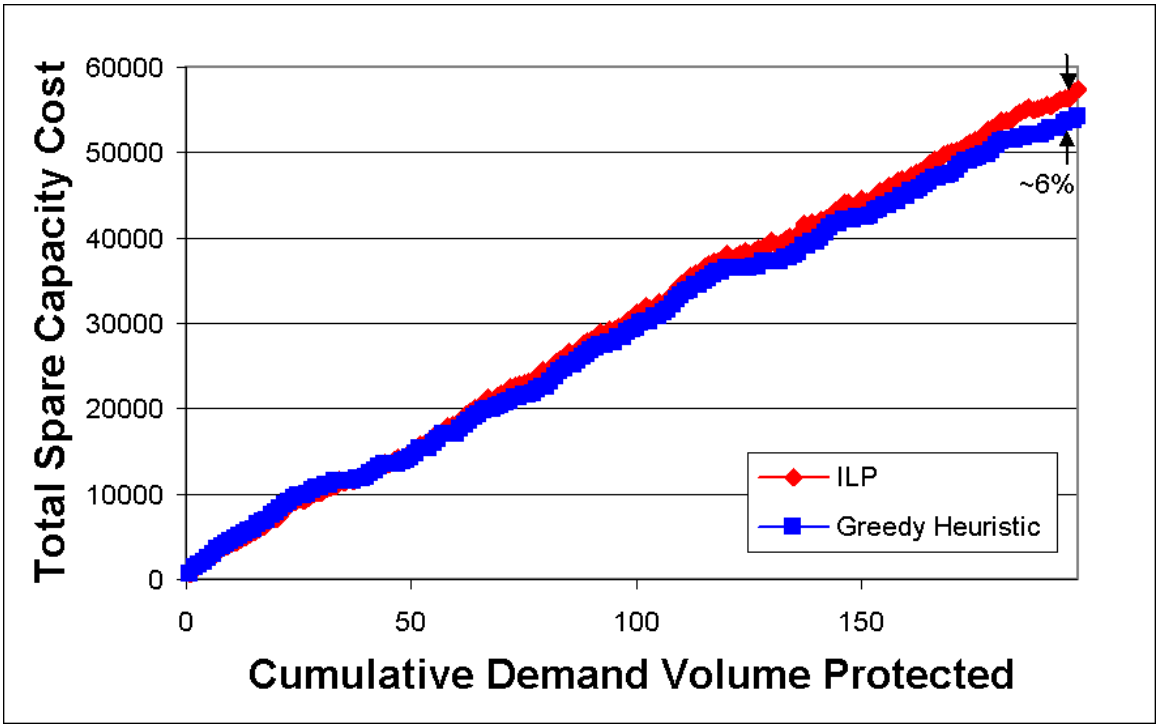


Figure 4.22: Spare capacity costs of the designs produced by each intermediate step of the ILP-based heuristic and the greedy heuristic when used as online algorithms

Therefore we can conclude that it is not a common occurrence that a PXT design produced using this method can be simplified into a design of the same cost that uses fewer PXTs. Designs that are found by this model to be optimal are generally unique.

Figure 4.23 and Figure 4.24 together give a more detailed picture of the differences between the internal workings of the greedy heuristic and the ILP-based heuristic. Because the greedy heuristic grows existing PXTs, it uses fewer, more complicated structures to protect the network as more and more demands arrive. The ILP methods, however, are able to tame the complexity of structures under approximately the same amounts of capacity by growing the number of structures uniformly while keeping the average length stable. Under these methods, the linear growth of capacity in the network comes almost entirely from the number of PXTs used rather than PXT size. Thus we can think of the ILP approach as providing an extra measure of discrimination that is able to select, out of a set of designs with roughly similar capacity efficiency, designs with additional desirable properties. What the heuristic is able to accomplish with a process that is relatively uncontrolled can be accomplished by the ILP model even when the set of structures provided to it is significantly

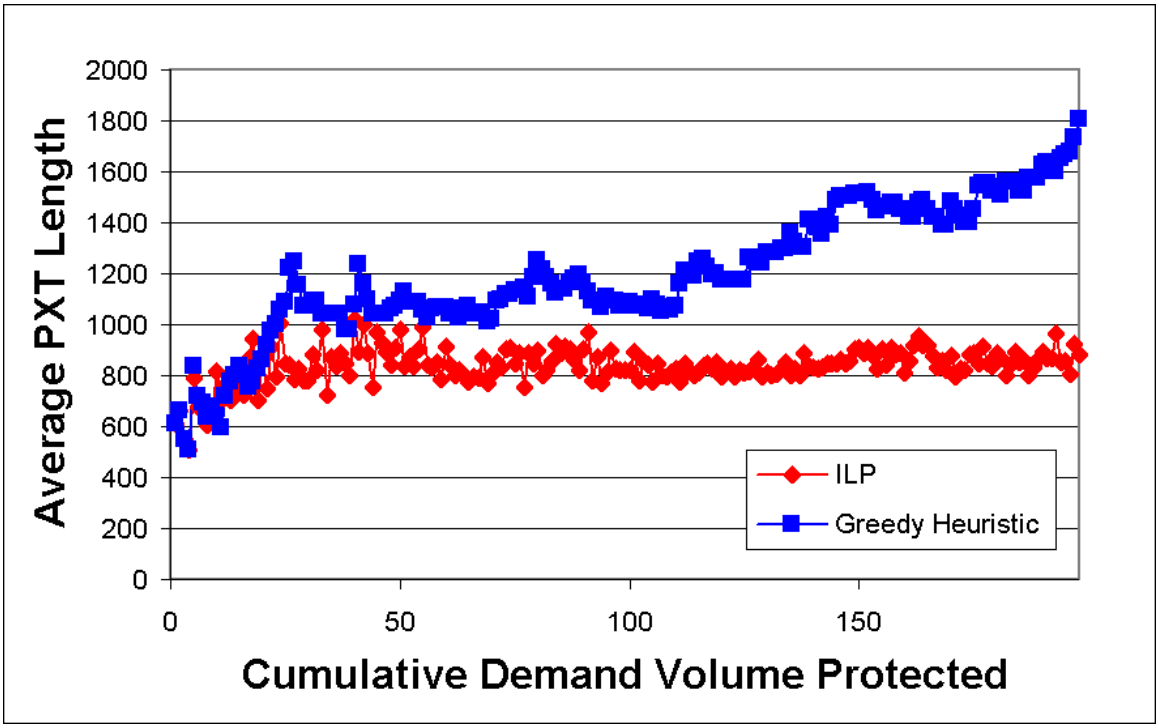


Figure 4.23: Average PXT length in the designs produced by each intermediate step of the ILP-based heuristic and the greedy heuristic when used as online algorithms

more limited.

4.3.8 Conclusions

The results show that the ILP approach to PXT design leads to highly efficient PXT-based solutions and does so using simple, non-looping PXT structures, which we refer to as "tamed PXTs". Additionally, it permits precise control of the maximum length of any PXT. In contrast, the prior heuristic produces PXTs with potentially unbounded length and complexity. We have also demonstrated a method of adapting the ILP-based algorithm to the online protection of incrementally arriving demands. In all cases, the cost efficiencies of our tamed network designs closely approach those found using the original heuristic.

Inspection of the PXTs produced by the ILP method also gave some insights into PXT network design in general. We found that, even though our method is quite different from the original heuristic approach, both methods tend to produce designs containing a significant proportion of PXTs that are equivalent to 1:1 APS arrangements for many of the same demands. The spontaneous emergence of the 1:1 APS architecture for a significant

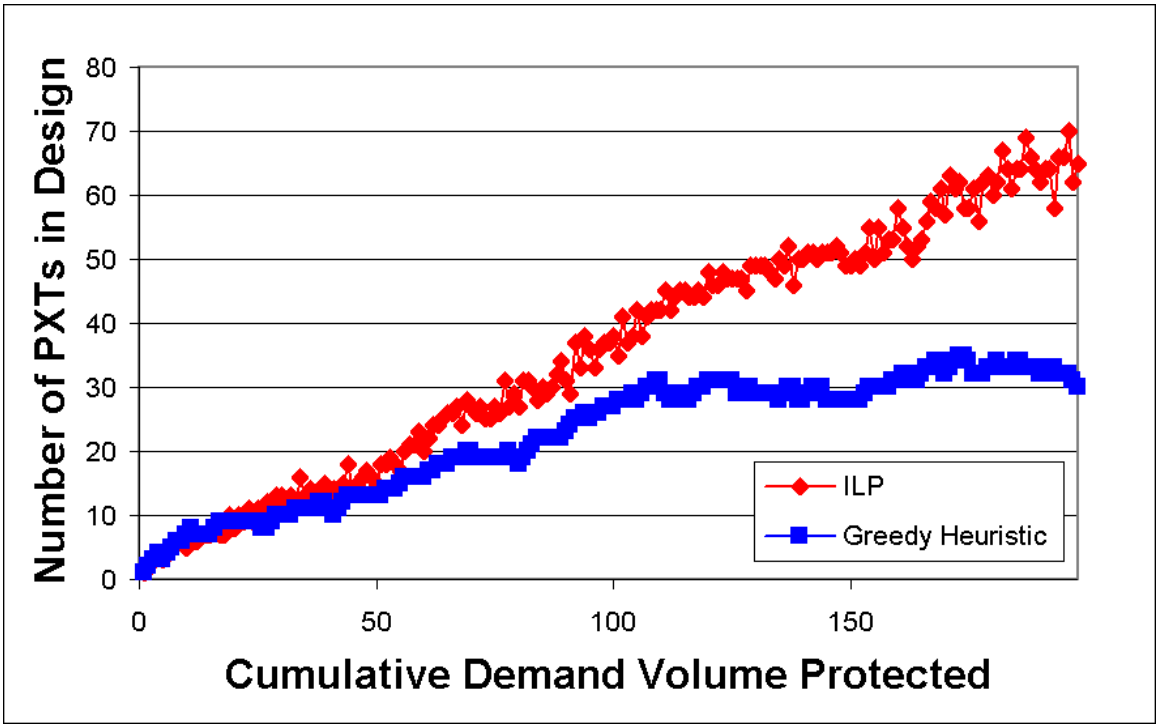


Figure 4.24: Total number of PXTs in the designs produced by each intermediate step of the ILP-based heuristic and the greedy heuristic when used as online algorithms

number of node pairs in both the greedy heuristic and ILP-based designs, when it was not explicitly assumed or intended, suggests that it has some inherent merit in certain cases where a fully pre-cross-connected protection is required. In this regard, one way to view the PXT network designs that were obtained is that they are a hybrid of dedicated (1:1 APS) treatments for selected demand pairs, and shared protection structures for the remainder of the protected demands. Notably, this is similar to the view given previously in the context of ring-mesh hybrid design (see Ch. 11 of [Grov03]) of 'forcer clipping' to explain how cost-effective hybrid survivable architecture designs work in a shared mesh network environment.

4.4 Modified PXT Constraints: Span Self-Disjoint PXTs

4.4.1 Motivation

The two methods for PXT design that we have considered so far, the greedy heuristic and the optimization-based heuristic, have been disassociated not only by their differing design

approaches, but also by the type of PXTs that they allow into designs. The greedy heuristic allows any PXT supported by the definition of the PXT concept, including PXTs that loop over themselves on nodes or spans, but the tests we have done using the optimization-based heuristic has been limited to using only PXTs that are simple trails. In the following Sections we detail our efforts into equalizing the class of PXTs considered by both methods, so they may be more directly compared. In other words, we introduce the class of PXTs that are span self-disjoint, but that may not be node self-disjoint. These PXTs are not allowed to cross the same span more than once, but may cross over themselves at nodes only. The fact that they cannot cross the same span more than once means that there is an upper bound on PXT size in the network, meaning that the set of these PXTs is finite in size. Unlike simple PXTs, however, their ability to loop through nodes more than once gives rise to the possibility of more interesting types of protection relationships, i.e., the creation of multiple protection paths for the same working path in the same PXT (so-called "*p*-cycle-like protection"). These properties make span self-disjoint PXTs an interesting candidate for additional study. We have already dabbled with using this type of PXT in the ILP-based algorithm in Section 4.3.7.2, but the conditions of the study were such that the results were inconclusive. In this Section we aim to adjust the conditions of the study such that a meaningful comparison can take place.

As we noted in Section 4.2.5.4 when investigating the greedy PXT heuristic, there is a rather simple way that the algorithm can be modified to use only span self-disjoint PXTs, even though we have seen that restricting it to completely simple PXTs is impractical. As for the ILP formulation, we simply need to modify the candidate PXT generation algorithm to allow the PXTs it generates to cross over nodes multiple times. The size of this PXT set will certainly be larger than the set of simple trails, but the span self-disjointness criteria places an upper bound on the size of a candidate PXT in the network, meaning that this set is at least finite in size. Note the contrast between the implications for the two algorithmic approaches: span self-disjoint PXTs represent a *restriction* of the greedy PXT heuristic, but a *relaxation* of the ILP approach.

4.4.2 Greedy Heuristic

4.4.2.1 Implementation

Because it is so difficult to enforce strict simplicity on the PXTs produced by the greedy heuristic, it comes as a surprise that the problem becomes quite simple when this restriction is relaxed to only span self-disjointness. To implement this, we only need to make a slight modification to the step that encodes span rivalries in the modified Dijkstra's algorithm. In the original greedy heuristic, this step ensures that all protection paths (not PXTs) are simple paths. However, we can modify this mechanism to suit our needs by also producing rivalries between spans that, if combined into a single protection path, would result in a merger or extension of PXTs such that the resulting PXT would cross the same span more than once. The reason this method works for span self-disjoint PXTs but not fully simple PXTs is due to problems with enforcing rivalries between PXTs that share nodes, as was described in more detail in Section 4.2.5.4.

This modification adds a small amount of complexity to the rivalry generation step, and using the modified Dijkstra's algorithm with more rivalries will increase its complexity even further from that of the standard Dijkstra's algorithm, but overall the time required to run the algorithm still remains reasonable.

4.4.2.2 Experimental Method

The modified greedy heuristic with this additional constraint on PXT complexity was first used to generate designs for the Murakami & Kim network, considering span costs as being equal to span lengths, with a uniform demand pattern of 3 units (as in Section 4.2.5.2 and 4.3.4). The networks used for the greedy heuristic capacity characterization tests in Section 4.2.3.1 (including again the Murakami & Kim network) were also used, again with span costs equal to 1 and a uniform demand pattern of 5 units.

4.4.2.3 Results

The costs of the resultant designs are given in Table 4.7. The average cost penalty of including the simple PXT constraint is approximately 5% if we ignore the K6,6 network. The cost increase is listed as negative here because the simple PXT designs are actually cheaper than the non-simple designs. Recall that this is also the network for which our initial re-

	Trial							
Network	**1**	**2**	**3**	**4**	**5**	**Average Cost**	**Original Cost**	**Cost increase (%)**
3-unit demand pattern, distance-based costs								
Murakami & Kim	55,855	56,350	55,474	58,522	57,848	56,810	53,591	6.0
5-unit demand pattern, hop-based costs								
12-cycle + 3 edges	926	903	928	895	926	915.6	884.1	3.6
3 x 4 grid	641	634	642	620	635	634.4	590.7	7.4
Tietze's graph	401	390	411	390	395	397.4	366.9	8.3
Murakami & Kim	517	518	511	517	510	514.6	510.2	0.9
Icosahedron	200	195	193	195	186	193.8	183.7	5.5
K6,6	186	193	190	190	186	189.0	236.2	-20.0

Table 4.7: Cost increases resulting from enforcing span self-disjointness in the greedy PXT heuristic

sults differed greatly from those found in [ChCh04] (Section 4.2.3.2). This supports the hypothesis that, rather than the difference being due to an error in the report of the original results, the extremely regular nature of this particular topology may render the results more susceptible to subtle differences in implementation of the heuristic. Instead of increasing the cost of the design by limiting the PXT growth options, the simple PXT constraint may counter-intuitively break the heuristic out of a course of PXT growth that ends up having a negative overall impact on the cost of the design in the totally unconstrained case. For example, perhaps the internal orderings of demands and the choice mechanism between equal-length paths cause the heuristic to build an initial "mega-PXT" that subsequently cannot be extended or shred with any more demands, resulting in many inefficient PXTs that follow. Imposing the simple PXT constraint can be thought of as a way to artificially stop the growth process at some point, resulting in a larger set of shorter PXTs with more balanced lengths.

Overall, the cost increase for imposing this constraint is not great, especially considering that it can make the difference between a WDM-based PXT design that can or cannot be

implemented in a single-fibre network without added required wavelength conversion. This further supports the conclusion that limiting the complexity of PXTs does not necessarily require a significant impact on PXT efficiency.

We should also mention the qualitative observation that imposing this constraint caused a significant increase in the runtimes for the algorithm. The increase depends upon the test network, but could be up to a factor of 4 in some cases. Therefore it seems that this constraint causes a significant increase in average case computational complexity for the algorithm. However, because this algorithm is so fast for these test cases to begin with, the runtimes were still quite manageable from a practical standpoint: no more than 5 minutes in the worst case.

4.4.3 ILP-Based Heuristic

4.4.3.1 Implementation

Because the PXTs used in the ILP-based PXT experiments to this point have all been simple PXTs they have also, by definition, been span self-disjoint PXTs in addition to being node self-disjoint. Therefore, while the span self-disjointness criteria for PXTs was an additional constraint for the heuristic, for the ILP formulation it is actually a relaxation on the criteria for the PXTs in the candidate set. Because the ILP model already supports the inclusion of arbitrary PXTs as candidate structures, the only change that must be made is to the candidate PXT generation algorithm itself. The simple depth-first search procedure that is used to generate trails need only be modified to allow traversal through the same node more than once.

The difficulties with this approach are therefore not a result of any complexity in the implementation, but rather of practical limitations on memory and processor speed. Even though the set of span self-disjoint PXTs is finite, it is still orders of magnitude larger than the set of simple trails, meaning that generating the full set is impractical for networks of the sizes we use here. Therefore we need a way of limiting the number of PXTs that are generated that is both consistent and justifiable. For our experiments, we have chosen the method of generating, for each node pair in the network, at least the N shortest span self-disjoint trails between those two nodes. We say "at least" because there may exist many trails with the exact same length such that to include all of them would result in a set with

more than N trails. In this case, to limit the set to exactly N trails would mean arbitrarily choosing between many equal-length trials. Therefore in cases such as these, all of the span self-disjoint trails with equivalent length to that of the N^{th} shortest trail will also be included.

Unfortunately, this means that we will generally be using only a small fraction of all possible span self-disjoint trails in the ILP. Specifically, even though simple PXTs are a special case of span self-disjoint PXTs, the limitations on the set size means that we will not in general be generating all simple PXTs; in the end, only simple PXTs below a certain threshold length will be included, depending on the length of the longest PXT found within the first N span self-disjoint PXTs. Therefore, even though allowing self node-crossing represents a relaxation on the constraints on PXT structure, we might see an increase in design costs as compared to the original ILP because of the incompleteness of the candidate set. Therefore, in order to both facilitate comparison with results for both the heuristic and original ILP, we took the approach of supplementing the set of span self-disjoint PXTs with the entire set of simple PXTs generated earlier (with duplicates removed). This allows us to more directly determine the degree to which span self-disjoint PXTs are able to improve upon simple PXT designs.

4.4.3.2 Experimental Method

Unfortunately, our approach for including span self-disjoint PXTs in the ILP model introduces yet another parameter (N) into our already heavily parametrized DRS-based ILP model, giving us another source of variability in our results. The following results use the largest value of N for which we were both able to complete the PXT generation procedure and also find optimal (or near-optimal) solutions for our design problems. The understanding is that, while these results do not represent the theoretically optimal improvement of span self-disjoint PXTs over simple PXTs, they do indicate the degree of improvement that can be expected for a given degree of effort (indicated by the value of N) put into expanding the candidate structure set. Because this value will be different for different networks (depending on size and connectivity of the topology), we will give the value for each test case separately.

The ILP was used to generate designs for the same networks and demand patterns as

the previous Section (4.4.2.2). For each network we performed 5 trials, each using a different run of the non-deterministic DRS generation algorithm. We also obtained results for designs using only simple trails (as per the original PXT ILP design method from Section 4.3.4) to use as reference designs. For the Murakami & Kim network with the "uniform 5" demand pattern we reused the results for the design from Section 4.3.5 as the reference; for all other cases, 5 "simple PXT only" designs were generated and the average cost was taken as the reference.

Unfortunately, meaningful results using the set of span self-disjoint PXTs were not obtainable for the K6,6 network topology, in which the high degree of connectivity ($\overline{d} = 6$) causes the number of span self-disjoint PXTs to explode even for very small values of N. For this network, the search for span self-disjoint PXT consumes too much memory and fails. The Icosahedron and Murakami & Kim networks are also highly connected, though not to the same degree ($\overline{d} = 5$ and 4 respectively), and so here we must restrict the size of the set of span self-disjoint trails more severely than for the other networks. All other networks have $\overline{d} < 4$.

The designs were solved to full optimality in all cases, except for the Tietze's graph and Icosahedron cases. For designs including the self node-crossing PXTs, the Tietze's graph tests were solved to within 2.6% of optimality on average, and the Icosahedron tests were solved to within 3.6% of optimality on average. For the reference "simple trails only" designs, the Tietze's graph tests were solved to within 1% of optimality and the Icosahedron tests were solved to within 4% optimality.

4.4.3.3 Results

The results are given in Table 4.8. Values are marked with an asterisk where the design could not be solved to optimality. These results show that we can expect a minor cost decrease from including PXTs that cross over themselves on nodes. This decrease is proportional to the number of additional PXTs that we can add to the problem; in the Murakami & Kim and Icosahedron cases we see a smaller decrease.[5] However, it is important to realize just how much we must expand the problem in order to obtain these modest improvements.

[5]Note that the value for Icosahedron are less certain, because the solution mipgap is large as compared to the observed cost difference (3% to 4% vs. 0.3%).

		Trial							
Network	*N*	**1**	**2**	**3**	**4**	**5**	**Average Cost**	**Simple Trails Only**	**Cost Decrease (%)**
3-unit demand pattern, distance-based costs									
Murakami & Kim	900	56,737	56,368	56,848	56,909	56,103	56,593	57,476	1.5%
5-unit demand pattern, hop-based costs									
12-cycle + 3 edges	9000	1155	1150	1155	1150	1155	1153	1244.2	7.3%
3 x 4 grid	9000	698	702	702	701	705	701.6	732.6	4.2%
Tietze's graph	9000	547*	546*	541*	534*	546*	502.6*	542.8*	7.4%*
Murakami & Kim	900	561	552	547	559	560	555.8	568.6	2.3%
Icosahedron	400	316*	308*	310*	315*	312*	312.2*	313.2*	0.3%*

Table 4.8: Cost decreases resulting from allowing self-node crossing in the candidate PXTs for the ILP-based PXT design method (expressed as an added set of span self-disjoint PXTs)

For example, in the "Tietze's graph" case where we see the largest improvement, we use on average about 2700 simple PXTs in the "simple only" case, but about 7400 total PXTs in the combined case. This shows the diminishing returns associated with increasing the size of the candidate set; most of the efficiency of the designs can be attained with simple structures alone.

4.4.4 Comparative Results

Table 4.9 shows the combined data from Tables 4.7 and 4.8, comparing the results for the greedy heuristic and the ILP-based design method using span self-disjoint PXTs (and also comparing them against the original heuristic and ILP designs.) We see that adjusting the two methods to use similar structures does indeed adjust the results closer to each other (as it must, because heuristic costs must increase and ILP costs should decrease), but there is still a large gap in some cases.

The full ILP problem, with all span self-disjoint PXTs and all DRSs included, solved to optimality, should be able to match or surpass the heuristic in terms of cost in all cases,

	Original			*Span Self-Disjoint PXTs*		
	Heuristic Cost	**ILP Cost**	**Cost Difference**	**Heuristic Cost**	**ILP Cost**	**Cost Difference**
3-unit demand pattern, distance-based costs						
Murakami & Kim	53,591	57,476	**7.2%**	56,810	56,593	**-0.4%**
5-unit demand pattern, hop-based costs						
12-cycle + 3 edges	884.1	1244.2	**40.7%**	915.6	1153	**25.9%**
3 x 4 grid	590.7	732.6	**24.0%**	634.4	701.6	**10.6%**
Tietze's graph	366.9	542.8	**47.9%**	397.4	502.6	**26.5%**
Murakami & Kim	510.2	568.6	**11.4%**	514.6	555.8	**8.0%**
Icosahedron	183.7	313.2	**70.5%**	193.8	312.2	**61.1%**

Table 4.9: Comparative results for span self-disjoint PXTs under the greedy heuristic and ILP design methods

so this shows that the practical limitations to the size of the problem mean that we are still far from achieving optimality in some cases (such as the 12-cycle + 3 edges case and the Tietze's graph case, in which the ILP designs cost more than 25% more than the respective heuristic designs). However, the ILP compares most favorably to the heuristic for the most realistic test cases (for the Murakami & Kim network), and even surpasses the heuristic in one of them.

4.4.5 Conclusions

We have seen that it is feasible and practical to introduce the concept of span self-disjoint PXTs to both the greedy heuristic and ILP-based design methods. In the heuristic case it is as an additional constraint that incurs a small cost increase (between 1 and 10%), with the benefit of simplifying the structures involved such that they could be implemented in a single-fibre DWDM network without wavelength conversion. In the ILP case it is as a relaxation on the type of PXTs allowed that results in a small cost savings (between 1 and 7%) at the cost of greatly increasing the number of candidate PXTs in, and hence the complexity of, the design problem. Comparative results show that practical limitations on the size of the ILP problem puts it at a disadvantage when the potential set of candidate

structures is very large.

4.5 Summary and Future Work

This Chapter represents the first in-depth investigation into the PXT architecture and associated design methods. We first characterized the efficiency of the first PXT design algorithm in the literature on our own test cases and proceeded to investigate methods of improving the results using simulated annealing. Results showed that minor improvements were possible given enough processing time. We then pursued a more detailed characterization of the designs produced by this algorithm and found that they had some significant disadvantages in terms of structural complexity. Among the implications was the fact that it could not guarantee that the resulting PXTs could be implemented in a DWDM network without wavelength conversion, and indeed tended to violate this condition more often than not.

We then proposed our own design method, based on standard ILP techniques and adapted from a design method for FIPP p-cycles, that was shown to give results with similar efficiency and improved structural properties. A comparison of PXTs to FIPP p-cycles then showed that PXTs could be comparable in terms of efficiency, and that a hybrid of the two structures could have significantly lower cost. We then saw that even though the original PXT design heuristic was designed to be an online algorithm, our ILP method could be adapted to perform as an online algorithm that gives comparable results. Finally, we saw that the efficiency gap between the two approaches can be narrowed significantly in more realistic test cases if the structure types used by both algorithms are equalized.

By no means does this represent an exhaustive study of PXTs, however. As a novel architecture, there are still many unknowns in the area of PXT network design. In this Chapter we used the DRS method developed for FIPP p-cycles as a tool with which to study the properties of PXTs. However, we did not investigate methods of fine-tuning this tool to suit the properties of PXTs specifically. Future work may investigate the effect of modifications to the DRS method on the efficiency of PXT designs, e.g., variations in parameters such as the size and composition of the DRS set, as well as the sizes and composition of the DRSs themselves. The results could point the way towards a method of streamlining the ILP model, allowing us to obtain comparable results with smaller data

sets. This would further allow deeper investigations into properties of PXTs as we would be able to solve problems with larger structure sets.

Chapter 5

Path-Protecting *p*-Trees

5.1 Introduction

Having investigated span-protecting pre-cross-connected trees in Chapter 3, it is natural to extend the concept to end-to-end path protection. However, we first required the work in Chapter 4 to establish an ILP framework for path-protecting architectures in general (as opposed to FIPP *p*-cycles alone, as the concept was originally developed), in order to apply this approach to tree-based protection. This Chapter covers our definition and investigation of the path-protecting *p*-tree concept. First, though, we must address some issues with the concept of tree-based path protection.

5.1.1 Background

5.1.1.1 The Path *p*-Tree Concept

The *p*-tree concept requires some further revision when it is extended to path protection, because the working model of degree-N cross-connection as a splitting function (see Section 3.1.1.1) directly implies some problems with the formation of protection paths in the *p*-tree structures. In the context of single span failures alone, this concern only arises in the path protection case because of the possibility for two paths protected by the same structure to fail simultaneously due to the failure of a single span.

Such non-disjoint path failures are handled by other architectures (e.g., PXTs and FIPP *p*-cycles) by providing disjoint protection paths for the failed working paths within the protection structure. Even though the two working paths may fail simultaneously, they can be protected at the same time by the same structure because they will not contend for the

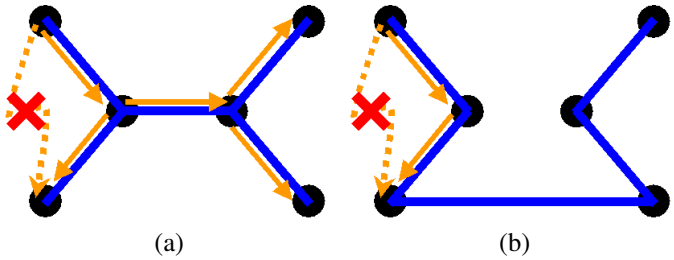


Figure 5.1: Path protection in (a) trees, with associated propagation of restored-state flow due to splitting, and (b) PXTs, where restored-state flow is intercepted only by the destination node

same spare capacity. The same may not be true for path *p*-trees, however, if one assumes a splitting model for the behaviour of protection signals at nodes with degree greater than 2. This problem is illustrated in Figure 5.1, which shows the restoration action of both a path-protecting *p*-tree (a) and a PXT (b) for the failure of a single protected *unidirectional* working path (dashed arrow).

In the *p*-tree case, due to the broadcast behaviour of splitting the restoration flow at degree-3 nodes, the restoration flow propagates throughout the entire tree. This broadcast restoration state flow will interfere with the protection paths for any other working paths that may have been affected by the same failure. In the PXT case, however, the restoration flow can be intercepted (dropped) at the destination node of the failed flow. In this case, the remaining spans of the PXT are unaffected by the failure and thus could be used for the protection of another working path if it were to fail jointly with the original working path.

This fundamental difficulty with the control of the propagation of restored state flow within branching structures greatly complicates the use of *p*-trees for the simultaneous protection of multiple failed working paths. In cases more complicated than the one depicted in Figure 5.1(a), the failure of a working path will cause propagation of restored state flow throughout part, but not all, of the *p*-tree (including but not limited to the spans on the protection path where this is appropriate and expected) because of the fact that the end-node of the failed path possesses enough information to be able to intercept (terminate) this flow. To be most efficient, we would like to be able to form additional protection paths out of the capacity in the remaining unaffected portions of the tree, but taking this into consideration would greatly complicate *p*-tree design. The alternative would be to postulate a type of

signaling between p-tree nodes to allow nodes other than the end-node of a failed working path to terminate restoration flow to avoid this situation, with the corresponding increase in operational and design complexity that this suggests.

Instead it seems to be reasonable to, at least initially, make the simplifying assumption that path-protecting p-trees should not be allowed to protect against the failure of multiple simultaneously failed working paths. Of course, this does not preclude the possibility of sharing protection between multiple working paths across *different* single-span failure scenarios, as long as none of the working paths have any such scenarios in common (i.e., they are all span-disjoint.) This restriction is essentially the same as the DRS restriction for FIPP p-cycles, although the motivation behind it is different. This is the assumption we will continue with in the following studies, with the associated benefit that it allows us to use the DRS models developed in [KoGr05a] and Section 4.3.2.

5.1.1.2 Previous Path p-Tree Literature

The general literature survey in Section 2.4 provided a summary of the existing literature that discusses path-protecting trees. We will now describe how this literature relates specifically to the path p-tree concept as defined in this thesis.

The group of publications [LaSt02, GrCo03a, GrCo03b] investigates the use of tree structures, called backup trees, used for the protection of entire end-to-end paths. Backup trees are developed in these works as a method for reducing spare capacity requirements in restorable MPλS networks in particular. As a result, many restrictions are imposed on backup trees that cause them to behave somewhat like very special cases of the more general path-protecting p-tree concept.

The first major difference is that these trees offer unidirectional protection only, instead of the protection for bidirectional flows offered by p-trees. This restriction is able to solve some problems with tree-based protection by forcing the protection flows to only travel "up" the tree, from the leaf nodes to the root node, such that different protection paths may merge but never split. This means that, while hardware selection of incoming flows at the nodes of the tree may be needed, splitting of flow is never required, and therefore the complications and restrictions that result from flow splitting do not apply. As such, these trees do not provide "true" tree-like protection so much as they do a mechanism to share a

single channel between different flows, allowing capacity savings. This is a direct result of the label-directed nature of the MPλS networks that this scheme was designed to protect.

Second, as suggested by the above restriction, it is required that within a single backup tree, all protection flows provided by the tree must terminate at one node: the root. In other words, a backup tree can only protect a unidirectional flow that terminates at its root. Again, this is because of label-switching. Because all protection flows that transit a node are assigned the same label, their ultimate destination must be the same: the network has no way of distinguishing them. This is significantly different from the operation of *p*-trees because a *p*-tree can provide any portion of its preconnected capacity as a backup path, using any nodes as the origin and destination so long as they are somewhere on the tree. Therefore the type of trees studied in [LaSt02, GrCo03a, GrCo03b] are uniquely suited to the protection of MPλS networks, making use of label switching concepts to combine protection flows into branching structures that, although they resemble trees, are really quite different from the kind we study here. So we can be sure that the work in this thesis on path-protecting *p*-trees does not duplicate the previous studies of [LaSt02, GrCo03a, GrCo03b].

There is also the range of publications on the subject of red/blue trees [MeFi99, ZhXu08, XuCh02, XuCh03], a scheme in which two unidirectional trees are established such that any span failure will leave any pair of nodes connected through at least one of the two trees. Despite having the appearance of tree-based protection, this is actually quite different from our concept of path *p*-trees because the red/blue tree arrangement is really just a representation of a combined protection routing plan for multiple failure cases, and does not define the capacitated protection structures or channels by itself. Therefore, just as with the backup tree concept, red/blue trees do not represent tree-based pre-cross-connected protection as path *p*-trees do. Furthermore, the reduction of the protection options to just the routes found in the red/blue trees prevents options such as splitting of the protection of a working path between multiple structures. Also, because the trees are unidirectional, restored state paths will generally be different for both directions of a bidirectional demand. Therefore the red/blue tree concept is also quite separate from the concept of path *p*-trees presented here.

From this survey of literature on the topic of tree-based path protection it seems that

true preconnected, branching protection structures have not as of yet been given serious consideration in the field. Even [Stam97], which investigates protection using arbitrary preconnected structures, only does so in the span-protecting context. Therefore this Chapter represents the first foray into the study of path *p*-tree protection.

5.1.2 Goals and Objectives

Path *p*-trees are the most complex protection architecture studied in this thesis thus far, owing to the fact that they combine the complexities of tree-based protection and path protection. Because true tree-based path protection had not been studied before, our initial goal for the project was simply to determine if standard ILP-based design methodologies could be applied to the problem at all, or whether some other approach would need to be taken (e.g., custom heuristics, as for PXTs in [ChCh04] and cycles, trees, etc. in [Stam97]). Given that ILP-based methods would be effective to some extent, our intent was to use these methods to the greatest effect possible to determine characteristics of path *p*-trees such as characteristic efficiency and their ability to achieve improvements via hybrids with other architectures. In this case we chose FIPP *p*-cycles, making this a path-protecting analogue to our research into hybridization of span *p*-trees with "regular" *p*-cycles.

5.2 Capacity Efficiency Characterization

5.2.1 Motivation

We wanted to begin the study of path-protecting *p*-trees by using standard ILP methods to create optimal (minimum capacity) restorable network designs based on the architecture, both to serve as a reference for future studies and also in order to investigate their properties. It was also our goal to be able to compare path *p*-trees to other path-protecting architectures, such as PXTs and FIPP *p*-cycles, in the areas of both capacity efficiency and the properties of the specific structures used in the designs.

5.2.2 Initial Method

We again followed the standard practice of using an ILP model to design *p*-tree based networks. However, path *p*-trees combine the computational problems of both tree-based

protection and path protection. We must deal with both the huge size of the candidate tree set (as with span *p*-trees) and the large number of combinations of paths that may be protected by each of these trees (as with PXTs and FIPP *p*-cycles). Because of this, we incorporated the complexity managing approaches of these prior studies, limiting the size and nodal degree of our candidate trees and extending the DRS ILP model to the path *p*-tree design problem.

5.2.2.1 ILP Model

The design model used for path *p*-trees is mathematically identical to the one used for PXTs (see Section 4.3.2). Therefore it will not be reproduced in its entirety here. Just as the *p*-cycle model can be adapted for use with span *p*-trees by appropriately setting parameters to represent trees instead of cycles (as discussed in Section 3.2.1), so can the PXT model be adapted for use by path-protecting *p*-trees. The DRS generation step must be modified slightly in order to compute the protection parameters for demands and DRSs using trees instead of trails, but this is a technical change and not a conceptual one. It still determines whether or not the structure (whether tree or trail) can provide a disjoint protection path for each working path.

5.2.2.2 Test Cases

The path-protecting *p*-tree model was used to generate designs for the networks from the 15 node network family. For each network, the DRS generation program was given all trees with a maximum size of 7 spans and a maximum degree of 3 as the set of candidate *p*-trees; this is the same set of trees that was given to the span *p*-tree model in Section 3.3. The DRS parameters were set to generate 10 DRSs per demand and 10 trees per DRS, with a maximum DRS size of 10. These parameter values are quite limited compared to the values that we used for the PXT designs in Section 4.3.4, but it is necessary to set them this low in order for the problems to solve in a reasonable amount of time (due to the increased complexity overall of the tree problem as compared to the PXT problem).

We also compared these path-protecting *p*-tree designs against designs from other architectures: span-protecting *p*-trees, FIPP *p*-cycles, and *p*-cycles. For span-protecting *p*-trees, the designs found in Section 3.3 were used. For FIPP *p*-cycles, designs were found using

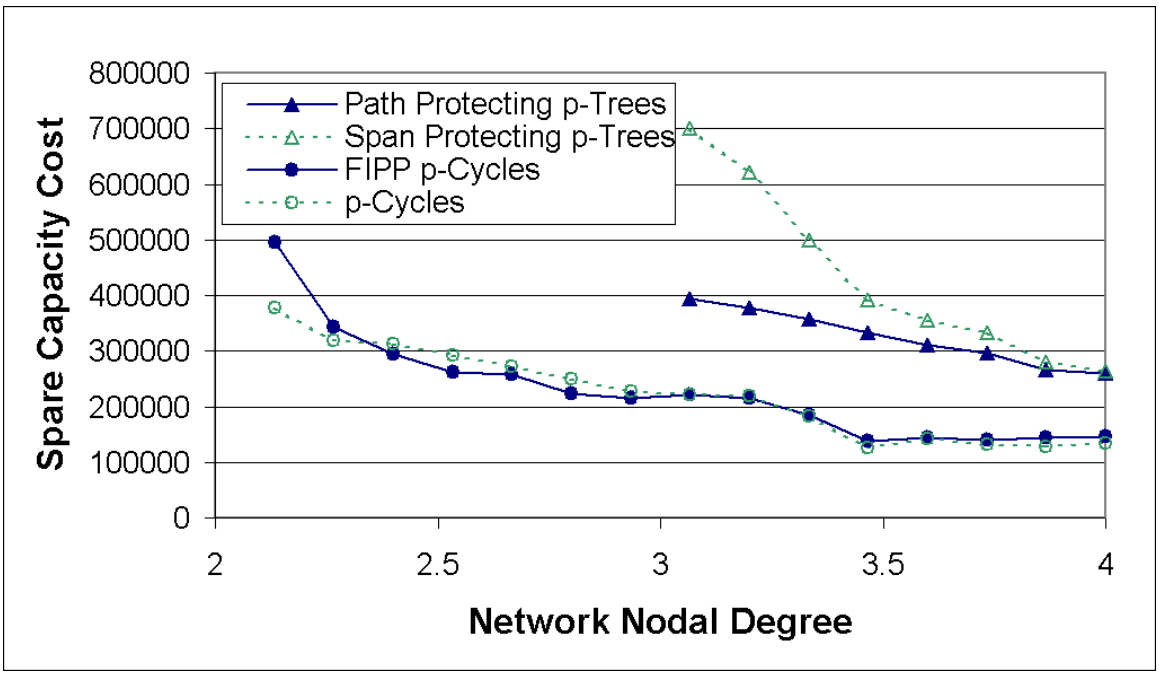


Figure 5.2: Costs of path-protecting *p*-tree designs compared to those of other architectures for the 15 node network family

the FIPP DRS ILP method, given 10 DRSs per demand, 10 cycles per DRS, and a maximum DRS size of 10, i.e., the same parameters as used for the path *p*-tree DRS method. For *p*-cycles, optimal designs were computed.

5.2.3 Initial Results

The costs of the resulting designs are given in Figure 5.2. The span-protecting architectures use dashed lines and hollow markers, while the path-protecting architectures use solid lines and markers. The tree-based architectures use triangular markers while the cycle-based architectures used circular ones. Note that, because the FIPP *p*-cycle designs were found using the same DRS generation parameters that were used for path *p*-trees, these designs are less optimal than FIPP designs that could be found using larger parameter values (due to the much smaller set of candidate cycles as compared to candidate trees). All designs have been solved to within at least 1% of optimality.

We see first of all that path *p*-trees suffer from the same problem we saw with span *p*-trees; the limited size of trees in the candidate set means that 100% single span failure restorable designs cannot be found for the sparsest of the networks. In this case, this is

because there are some demands in these networks for which there does not exist any trees that can protect them, because their shortest eligible protection path is longer than the tree size limit of 7 spans. Therefore both *p*-tree architectures suffer from this serious issue when attempting to obtain results that are close to optimality.

Also note that the difference between span and path *p*-tree efficiencies becomes extremely pronounced as network connectivity is reduced to the minimum limit at which feasible designs still exist. This effect may occur because of the restrictions on the sets of candidate tree structures. We can see how this would affect span-protecting tree designs more severely than path-protecting tree designs in the cases where the network nodal degree (i.e., connectivity) is barely high enough to allow the presence of a feasible design using the given sets of trees. In these cases, using path-protecting *p*-trees, the working paths with shortest protection paths of length 7 will have very few *p*-trees available to protect them (i.e., only 7-hop degree-2 path *p*-trees, basically PXTs), but these structures will still hopefully not be too individually redundant, especially considering that other demands may have the opportunity to share their protection with this *p*-tree. However, using span *p*-trees in the same networks, there will now be some individual spans with minimum protection path lengths of 7 hops, which will *each* have to be protected by a 7-hop degree-2 span *p*-tree (*p*-segment). In this case, opportunities for sharing with other spans still exist, but the individual redundancy considering only the span in question and the 7-hop *p*-segment will be much higher (on the order of a 7:1 ratio or 700% assuming roughly equal span lengths). Therefore we expect that the low-connectivity span *p*-tree design will contain much more inefficiency that is "locked-in" due to the topology and limited number of protection alternatives.

These observations seem theoretically correct, but were called into question when we discovered problems with the soundness of our design methodology via analysis of the structural properties of the path *p*-trees themselves.

5.2.3.1 Structure Analysis

Just as in the span-protecting case, we mapped out the structures used in our designs in order to gain further insight into the characteristics of efficient path *p*-trees. In doing so, we found that by far most of the protection in these designs was provided by APS-equivalent

degree-2 degenerate trees (i.e., PXTs). We have seen that APS-type protection can be common in efficient designs (e.g., in pure PXT designs, as in Section 4.17), but the usage in this case seemed excessive, with more than 95% of protection in all designs provided in this way.

To discover why this might have occurred, we looked into the data sets generated for the DRS problem (DRSs, candidate structures, and their protection relationships), and discovered that, for the most part, protection relationships between trees and DRSs were only being discovered for the single-demand DRSs, and not for the DRSs that were built up from single demand "seeds" (see the DRS method description for PXTs in Section 4.3.3). The result is that each structure, whether true tree or segment, almost always only had the option of APS-type protection. This also explains why almost all structures in the resulting designs were segments: a tree will always be less efficient than a segment for APS protection, as any branches on the tree that are not part of the APS protection path are extraneous.

This is a major flaw in the design process, as it almost entirely rules out shared protection for trees, making it impossible for us to perform meaningful comparisons to other architectures. The reason for this result is due to a negative interaction between three factors: the limits on tree sizes, the DRS filling method, and the properties of p-tree protection. For FIPP p-cycles (the architecture for which the DRS method was initially developed), we almost always want large DRSs as a cycle that protects many demands will generally be more efficient. That is why the DRS method builds the DRS up in size until no more demands can be added (either because of routing conflicts with the existing demands or because the size cap has been reached). In the DRS method, the requirement for a cycle to protect a DRS is that all of the end-nodes of the paths in the DRS are on the cycle, meaning that large DRSs will generally require large cycles. However, for cycles, at worst 1 unit of protection can be offered to any path in a DRS it protects, so even if we require our cycles to be large by generating DRSs containing many paths with many distinct end nodes, we are guaranteed to receive some "return" in terms of increased protection.

For PXTs the situation is different because a path with its end nodes on the PXT may not be able to be protected at all. However, we are still able to generate all of the PXTs in a realistically sized network, so that means we still have large PXTs available to cover our large DRSs, even though it may well provide zero units of protection to certain demands.

For p-trees, however, we must limit the size of our trees, so we do not, in general, have trees that are large enough to protect our large DRSs. This is why we found very few protection relationships for our non-trivial DRSs in the design problem input files. In this situation, the solver has no option but to use the single-demand DRSs, and so we get designs consisting mainly of APS protection.

5.2.4 Modified Design Method

We have seen that the DRS method, as used for PXTs in this thesis, is not appropriate for path p-tree designs under the structure size constraints we must use for our experiments here. In light of the explanation above, we decided to alter the DRS model for path p-trees (see Section 4.3.2) in order to accommodate their special properties. The fundamental problem was that the DRS protection parameter x_c^k was not being set for any DRS c except for the set of trivial DRSs containing just one demand. This is because to set $x_c^k = 1$, tree k must contain all of the end-nodes of all demands in DRS c. But under our current design method, DRSs are large and trees are size-limited, meaning that it is very unlikely that a large enough tree exists to protect any of our DRSs. However, note that this criterion for setting x_c^k is actually somewhat arbitrary. What if we were to relax this criterion and instead set $x_c^k = 1$ in some cases where tree k did not contain all end nodes of the demands in DRS c? In this case, k could not possibly protect these demands, so $\beta_r^k = 0$ for each of them. This means that even if k is used to protect this DRS, the actual protection contribution in the protection constraint will be 0, due to the multiplication with β_r^k.

This shows that we can in fact set $x_c^k = 1$ for any combination of k and c and still have a valid model that produces accurate results. The only difference is that the more tree/DRS combinations in which we set $x_c^k = 1$, the more protection opportunities the solver will have and the better the optimal result will be. This means that setting $x_c^k = 1$ for all tree/DRS pairs will produce the best results, but we cannot do this in practice because the size of the model would become much too large to solve. Therefore we need to choose a smart way in which to set the values of x_c^k. For PXTs and FIPP p-cycles, the end-node containing constraint seemed to work well, but we have seen that the situation is quite different for path p-trees.

Therefore we needed to find a better way to set the values of x_c^k. Two options present

themselves. The first is to maintain the same DRS generation method (generating large DRSs) and find a way to set $x_c^k = 1$ for the non-trivial DRSs so as to give the solver better protection options. The second is to change our method of DRS generation such that the "default" method of setting x_c^k works well, and leave that part of the design method unchanged. We decided on the prior method, as it allows us to generate designs using the same data files containing the same sets of pre-generated DRSs and trees used for our previous experiments. In addition to requiring less effort and time, this makes the results more easily comparable with our our prior results, as the only difference in the data sets will be the values of the x_c^k parameter.

For generating our new designs, redefined x_c^k such that $x_c^k = 1$ for any DRS c and tree k for which the tree could protect *at least one* of the demands in c. This is vastly different from our original approach, which sets x_c^k to 1 only when a tree contains all of the end nodes of all of the demands in c, and is unconcerned with the details of protection (except indirectly through the fact that a path may be protected only if it has both end nodes on the tree). Table 5.1 gives the fraction of the tree/DRS combinations in which $x_c^k = 1$ under both the original approach and our redefinitions of x_c^k for 3 of our test networks (16 node, 23 node, and 30 node). As we would expect, the number of possible protection relationships greatly increases as a result of this strategy, by a factor of ~5 for the 16 span network and over a factor of 100 for the more highly connected networks.

This leads us to expect that the results for these new design problems should be much better, and more importantly more representative of the true efficiency of path p-trees. However, the complexity of the problem also increases, because the solver has many more options to consider. More precisely, the number of variables will increase, because the set of per-DRS protection variables (n_c^k) is defined over all combinations of DRS c and PXT k such that $x_c^k = 1$. The number of constraints will not change, although the number of terms in constraints 4.2 and 4.3 (the DRS protection and PXT allocation constraints) will increase, perhaps indirectly influencing complexity by increasing the "linkage" between the variables involved in these constraints. However, we are still able to solve these problems to optimality quickly. We are aided by the fact that we can use the values of the variables in our initial solutions as a starting point for these new solutions, as any existing solution is still a valid solution for the new problems. This saves time by allowing us to start from

Network	Original DRS method	$x_c^k = 1$ when any demand is protectable
15 node, 16 span	0.0135	0.0630
15 node, 23 span	0.00339	0.363
15 node, 30 span	0.00242	0.288

Table 5.1: Fraction of possible DRS protection options for candidate trees under the improved approach to setting $x_c^k = 1$ for selected test networks

within approximately 10% to 20% of optimality, instead of starting from scratch.

5.2.5 New Results

The plot in Figure 5.3 shows the design costs of the new designs appended to the original results from Figure 5.2. The old results have been replaced with red crosses and the curve for the new results takes on the formatting of the old. Increasing the number of available protection relationships vastly decreases the cost of the path p-tree designs. Our original observations remain valid; if anything, the difference between span and path p-trees has become more pronounced.

Although these results are now less directly comparable to those for FIPP p-cycles (because they now use different variations on the DRS method), they are also more indicative of the characteristic capacity efficiency of path p-trees. At the very least, we can say that their design costs are now comparable to those for p-cycles and FIPP p-cycles, although still significantly higher. Certainly, they are much more comparable than the span-protecting p-tree designs were. This is understandable considering that PXTs are a special case of path-protecting p-trees, and we have already seen that PXTs can have efficiencies comparable to FIPP p-cycles (see Section 4.19). Therefore path p-tree designs must have costs equal to or less than costs for a PXT design with an equivalent restriction on PXT length.

Figure 5.3 also shows that the transition from span protection to path protection has a much more drastic effect for trees than for cycles. While FIPP p-cycle designs usually have roughly the same costs as their span-protecting p-cycle counterparts in this network family, path p-tree designs are almost always cheaper than the span p-tree designs. This can be understood intuitively as a function of a tree's ability to spread its branches further apart in a network than a cycle can spread, given the same amount of capacity. For example, in a 6 span cycle no node is more than 3 hops away from any other node, because of a cycle's

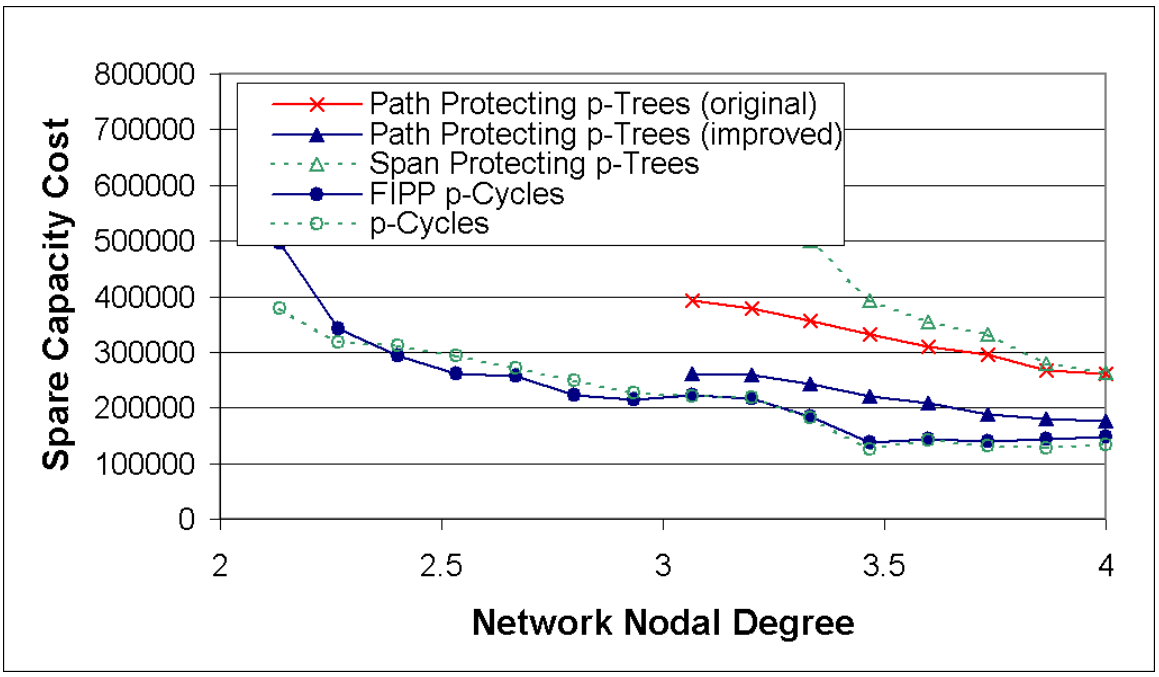


Figure 5.3: Costs of path-protecting *p*-tree designs generated using a more representational DRS method

closed nature. Two nodes at either end of a 6 span tree, however, may be up to 6 hops away. Under a span protection framework, and assuming realistic network topologies in which there tend to be more spans between close nodes than nodes that are far from each other, this means that a cycle of a certain size is more likely to be able to protect many spans than a tree of the same size, because the nodes in the cycle are more likely to be close and thus have spans between them. Large and spread-out trees are wasted under this framework, as spans between distant nodes in the trees are not likely to exist. Therefore we expect the set of span-protecting trees to be much less dense with efficient structures than the set of cycles, a fact that was confirmed by our tests on span-protecting *p*-trees. However, under path protection with a full demand matrix (as we have here), working paths exist between every node pair in the network, near and far alike. Therefore the utility of larger trees is increased and the same set of trees becomes denser with good possibilities for protection when full end-to-end paths are considered.

5.2.5.1 Structure Analysis

Because of the great decrease in cost for the path *p*-tree designs after our improvements to the DRS method, we might expect that the designs are now making much more use of "true" *p*-trees. Analyzing the details of the *p*-tree structures used shows that this is true. Table 5.2 shows the number of true trees and PXTs used in each of the designs, as well as the proportion of total structures that are true trees. First of all, the new designs use many fewer structures; this is a direct result of being able to use larger structures instead of mostly short 1+1 APS-equivalent PXTs. As for the degree of tree protection, while the initial designs used only a small amount, the new designs can use almost as many true trees as PXTs in some networks. Because the total number of trees is so large, we cannot illustrate them all in an Appendix. However, we do give some select examples to illustrate certain points later in this analysis (Figure 5.5).

One characteristic that is common across both the old and new results is that there is a general trend towards less tree protection in the higher degree networks (although the decline is hardly regular). This runs counter to our expectations based on the explanatory framework we have built so far concerning the effective "protection reach" of different types of structures vs. their total size. In a sparsely connected network, the shortest possible protection path between any two nodes is long, and becomes shorter as spans are added. Therefore, we would expect to require more PXTs in a sparse network to provide these long protection paths. However, the opposite is occurring here.

Under the modified DRS method, the issue of protection path length may no longer be paramount, because the set of protection relationships for each demand is now very densely populated, with each demand facing many choices for protection from PXTs and trees alike. Instead, trees are likely being used here as special case "fixes" in certain areas where the use of PXTs would be suboptimal (as seen with both *p*-trees and *p*-segments with respect to *p*-cycle designs in Chapter 3). Figure 5.4 shows an extreme case in which the only possible (non-cycle) structure that can protect both demands at once is a tree. This is the case because the end-nodes of both demands are degree-2 nodes, and therefore any non-cycle structure that protects them must terminate at both of those end-nodes, meaning that either we must use 1+1 APS protection for both, or we need to use a tree with its branches terminating at each of the 4 end nodes. As network connectivity increases, the

	Original Designs			*Increased DRS Protection*		
Network	**Number of True Trees**	**Number of Segments (PXTs)**	**% True Trees**	**Number of True Trees**	**Number of Segments (PXTs)**	**% True Trees**
23 span	12	464	2.5%	90	111	44.8%
24 span	9	460	1.9%	70	138	33.7%
25 span	12	470	2.5%	67	147	31.3%
26 span	3	508	0.6%	80	132	37.7%
27 span	6	493	1.2%	79	118	40.1%
28 span	0	504	0%	85	111	43.4%
29 span	0	518	0%	49	172	22.2%
30 span	0	518	0%	60	162	27.0%

Table 5.2: The amount of "true" tree vs. PXT protection in both initial and improved path *p*-tree designs

(a) Protection paths (b) Tree

Figure 5.4: An artificial case in which shared protection for two demands must be provided by a tree

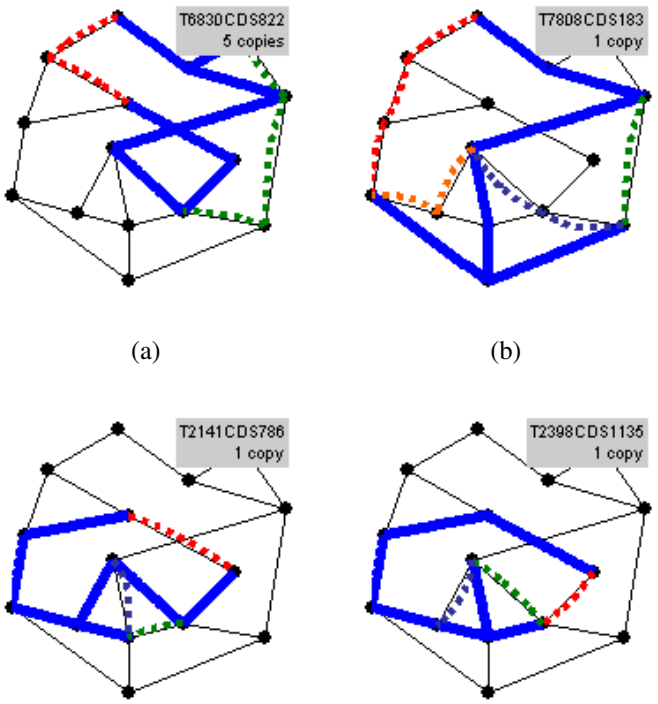
number of demands with degree-2 end nodes will decrease, freeing up options for shared protection on PXTs in addition to trees.

Figure 5.5 illustrates this point using examples from our solutions, all of them actual trees extracted from our design for the 23 span network. Subfigure (a) is the most similar to our artificial example because a simple PXT is not able to protect both demands simultaneously. The reader can verify, by tracing out the possible protection paths that *do not* combine to form a tree, that such a structure would have to be either a cycle or a non-simple PXT. In this case this occurs because each of the demands has one degree-2 end node, combined with a kind of low-degree "choke point" that exists at the other end of the network. This shows that not only are degree-2 end nodes a factor, but low connectivity throughout the network can produce a requirement to use trees for shared protection. Subfigure (b) shows a similar situation involving more demands. Here, the red and orange demands

share one degree-3 end node, as do the purple and green demands, creating again a situation where the protection structure must terminate at these nodes. The branching structure of the tree allows it to reach both of these nodes, as well as all the other end nodes of the demands, to complete the protection paths. These examples show that using *p*-trees allows increased opportunities for sharing in low degree conditions. As connectivity increases in the network, these situations occur less frequently, as there are more routings that will allow a linear structure to connect all end-nodes of the demands in a DRS. Essentially, low connectivity will require more of the routing flexibility of *p*-trees.

Of course, there is nothing in the design problem that mandates that we must use the exact DRSs in the examples given in Figure 5.5, protecting these particular groupings of demands by the same structures. We can imagine that a different DRS arrangement for these demands would allow groupings of end nodes that would be more friendly to easy access by PXTs instead of trees. This may indeed be the case, but the fact is that we see more of these groupings for the more sparsely connected networks, indicating that they are more necessary for efficient protection in these cases. Again, as in the span-protecting case (and even more so here because of the necessarily incomplete nature of the DRS-based design problems), we cannot derive certain rules for path *p*-trees to determine when they must be used and when they must be avoided for efficient protection. We are simply advancing a statistical argument that, we feel, strongly suggests a higher suitability of *p*-trees for sparse networks.

Another interesting feature gleaned from inspecting the trees in our designs is the observation that, as with our PXT observations in Section 4.3.5, there are many trees that could have their protection capabilities greatly enhanced by minor alterations that would transform them into cycles (that is, FIPP *p*-cycles in the path-protecting case). Such examples are shown in Figure 5.5 (c) and (d). In both cases, we can just add one span and move another to make the trees into cycles that still protect all of the pictured demands because of a cycle's ability to protect demands that overlap with it. This suggests that the benefits of path *p*-tree protection in low-degree cases may diminish or disappear entirely in the presence of FIPP *p*-cycles. This further motivates the following comparison of path *p*-trees with FIPP *p*-cycles and an investigation of hybrid designs.



(a) (b)

(c) (d)

Figure 5.5: Examples trees from path *p*-tree designs

5.3 Hybridization with FIPP *p*-Cycles

5.3.1 Motivation

At this point, it was natural to continue on the same investigative course that we took with span *p*-trees: hybridization with a cycle-based architecture. The purpose of this is twofold: first to determine the degree of improvement that trees can provide over cycles alone, and second to gain insights into the properties of situations in which trees and cycles can complement each other, this time under the paradigm of path protection. In the case of path protection, the relevant cycle-based architecture is FIPP *p*-cycles.

5.3.2 Initial Method

Because we can find path *p*-tree and FIPP *p*-cycle designs using the same DRS ILP model, we are able to follow the same approach used to hybridize span *p*-trees with *p*-cycles. In other words, we can generate a design problem that uses both trees and cycles simultaneously as candidate structures. The DRS generation element, however, adds some complexity to this hybrid approach. Because the generation of candidate structures depends on the set of DRSs that is generated prior, there are a number of ways that the problems could be merged together. For example, it would be possible to first generate both DRSs and candidate structures independently for both trees and cycles as normal, and then merge the two DRS sets and candidate sets together. Another approach is to first generate only a single set of DRSs, then generate both trees and cycles for the protection of these DRSs simultaneously. We opted to take this second approach initially, as it most closely satisfies the philosophy of treating trees and cycles impartially, as generalized structures instead of conceptually separate architectures.

First, the set of DRSs is generated as normal, as per the PXT DRS or pure path *p*-tree DRS method. Next, only the set of cycles is searched in an attempt to find at least the designated (by user-assigned parameter) number of structures that can protect each DRS. After this stage, if there are any DRSs left that still require more structures to protect them, only then is the set of trees searched as in the normal path *p*-tree DRS method. The intent was for this search to then continue until each of the DRSs in question have the designated number of protection relationships from trees and cycles. The combined tree and cycle

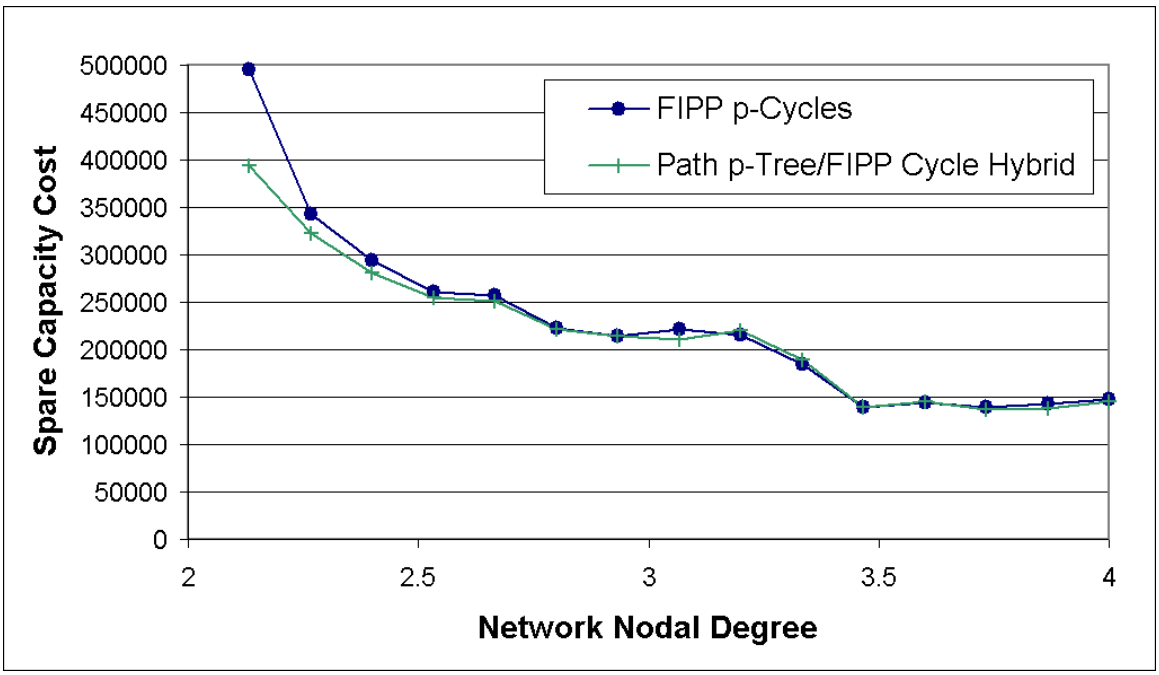


Figure 5.6: Spare capacity costs of initial hybrid path *p*-tree and FIPP *p*-cycle designs

candidate set is then used as the candidate set for the problem.

Note that these experiments were performed before the problems with the DRS method for path *p*-trees were identified, and therefore the DRS protection relationships were underrepresented for the path *p*-tree candidates, as with our initial results for pure path *p*-tree designs. Therefore these initial results do not truly represent the ability of path *p*-trees to supplement FIPP *p*-cycles. However, the results of the exercise were successful in terms of motivating us to improve this approach to the hybrid path *p*-tree/FIPP *p*-cycle problem, so we will discuss them briefly before moving on to discuss our further modifications.

5.3.3 Initial Results

The costs of the hybrid designs are given in Figure 5.6, along with the costs of the FIPP *p*-cycle-only designs. It shows that the hybrids are hardly ever able to improve significantly on the pure FIPP designs. Initially it was assumed that this was a situation analogous to the span-protecting case in Section 3.4, where cycles naturally dominate trees in terms of efficiency, but we later discovered the problems with the original DRS method in conjunction with *p*-trees and realized that we could not draw strong conclusions about the characteristic efficiency of *p*-trees from these results.

At the time, however, this was unknown to us, so this poor showing for path p-trees motivated us to search for other causes. Indeed, we discovered a different problem, this one isolated to the hybrid design process. We inspected the output of the DRS generation algorithm to determine whether or not it was biasing the problem given to the solver, and discovered that, for the networks containing 23 to 30 spans, the candidate structure generation process was actually generating zero candidate trees for use in the problem. Therefore for all of these networks the solved problem can be considered to be identical to the FIPP problem (the differences in results being caused by differences in different runs of the non-deterministic DRS generation algorithm). Even in the other cases where trees are used, this number is often quite small. For example, the 22 span network is only given 3 candidate trees, as compared to 134 cycles. Trees only outnumber cycles in the data sets for networks with 19 spans and below (even though the size of the full tree set will always be much greater than the number of cycles). Therefore the problems will already be heavily biased in favour of cycles in the majority of cases, and these results do not provide a valuable comparison between trees and cycles, regardless of how well populated the set of potential protection relationships is.

This is all a direct result of the candidate structure selection method, which is biased heavily in favour of cycles. Because the entire set of cycles is checked first, before any trees are considered, it is possible that the process will terminate entirely before any candidate trees are discovered at all, due to the large number of protection relationships that cycles are able to provide alone. In a way this is no different from the ordinary DRS method for FIPP p-cycles, which sorts cycles by length and thus prioritizes short ones over long ones. It is still justifiable as a practical design methodology, because in general we do not expect trees to be as efficient as cycles in any case, so we only wish to consider them if we cannot find a desired number of possible protection choices among cycles. However, it does interfere with our efforts to compare path p-trees and FIPP p-cycles theoretically, because trees are unduly eliminated early in the heuristic DRS/structure generation step instead of being compared fairly against cycles by an impartial solver, as was our intention. In order to obtain useful results, we needed to resolve two problems with our design algorithm: both the under-representation of tree candidates and the underpopulation of their potential DRS protection relationships.

We must also mention for completeness that a bug was also discovered in the implementation of the candidate structure generation procedure. As stated above, the intent was for this procedure to generate first cycles, then trees, until the total number of protection relationships for each demand exceeded the minimum defined by the relevant parameter. The bug, however, caused the protection relationship count to become reset when moving from cycles to trees. This means that in any case where there were not enough cycles to cover the required N protection relationships, trees were then generated to provide all N relationships, no matter how many had already been dealt with by using cycles. This bug was fixed for the following tests. Our design methodology already had enough problems at this point that our results had no real theoretical value in any case (though they did have value in terms of informing our attempts to develop a more fair version of the design process). However, note that, unlike our other problems, this bug actually biased the solutions partially *in favor* of trees, by generating more than were intended (at least, in situations where trees were generated at all).

5.3.4 Modified Method and Experiments

The issue with underrepresented DRS protection in trees was fixed in a similar way as it was for the pure p-tree designs in Section 5.2.4. For the hybrid designs, this involved changing the values of x_c^k for *all* structures, trees and cycles included (in order to be fair to both architectures, as per the intent of the exercise). As with the pure p-tree case, we used initial solutions with limited numbers of tree protection options as starting points for solving our new designs. We also took the same approach, at first, of setting $x_c^k = 1$ in all cases where at least one demand was protectable, but unfortunately we found that this resulted in problems that would not progress significantly away from their initial conditions and towards optimality in a reasonable amount of time, because of the complexity of the problem. We were still able to achieve results within approximately 5% of optimality for most networks (solutions were found for the 16 span through 26 span networks), but in the more densely connected networks the original design would report more than 10% from optimality, with negligible change after over a day. Therefore we needed to find an alternative method with reduced complexity for these networks.

The modified approach was to set $x_c^k = 1$ only in cases where *more than half* of the

demands in a DRS were protectable by the structure in question. This method still increases the number of available protection relationships, just not to such a great degree as before. Using this approach, most designs were able to be solved within 5% of optimality. It is these results (for the ">50% protection" approach) that we present in the following Section. Note that this method, unlike the first, can actually remove protection relationships in certain cases, because there may be trees that pass through all the end nodes of demands in a DRS that can actually only protect less than half of the demands because the protection paths are not routed disjointly from the working paths. However, this approach will still increase the number of protection relationships on average. Furthermore, it is still more suited to trees than the original DRS "end-node intersection" approach, because it takes protection capabilities directly into account.

As for the issue with the generation of the structures themselves, we wanted a method that would remove as much of the bias as possible from the selection process in order to be able to use the hybrid design problems to make a fair comparison between FIPP *p*-cycles and path-protecting *p*-trees. Such a method would ignore the distinction between trees and cycles, treating them as generalized path-protecting structures and sorting them in an order for consideration that is as "neutral" as possible. The first such approach that comes to mind is simply total randomization of the combined set of trees and cycles. However, this adds an extra element of non-determinism to the DRS algorithm which already has issues with repeatability due to its random nature.

The method we decided on was to combine the set of cycles and trees together and then sort sort the combined set by total structure size (the sum of the lengths of all on-structure spans). This maintains the original order of the tree and cycle sets individually, as they are normally sorted by length separately, while at the same time giving both types of structures a chance to be chosen to protect DRSs, as they are interleaved into the same list. This is the approach taken in the following experiments.

This method is not without its flaws. For example, sorting the combined structure set by length with the expectation that smaller structures protecting the same DRS do so more efficiently ignores the fact that a cycle will likely be able to provide much more protection to a DRS than a tree of the same size. While a FIPP *p*-cycle can always provide either 1 or 2 protection paths to a working path that has both of its end nodes on the cycle, a

Network	*Cycle-Biased Selection*		*"Fair" (Length-Sorted) Selection*	
	Number of Cycles	**Number of Trees**	**Number of Cycles**	**Number of Trees**
16 spans	3	258	3	258
17 spans	7	598	7	598
18 spans	14	844	14	844
19 spans	24	1143	24	1332
20 spans	43	50	36	2109
21 spans	73	38	58	3321
22 spans	134	3	89	4177
23 spans	186	0	132	4358
24 spans	294	0	213	4585
25 spans	423	0	308	3861
26 spans	575	0	456	2901
27 spans	664	0	491	3276
28 spans	772	0	649	3625
29 spans	992	0	805	3084
30 spans	1286	0	1009	3082

Table 5.3: Number of candidate structures for path p-tree/FIPP p-cycle hybrid designs under the original (cycle-biased) and new (combined list sorted by length) candidate structure selection methods

tree may not even be able to provide 1 such path if the structure is not disjoint from the working path. Therefore it may be argued that this method of sorting the structures gives undue preference to trees, considering their characteristic inefficiency. The weaknesses of the approach must simply be kept in mind when drawing conclusions from the data.

The parameters used to perform the following experiments were identical to those performed in the previous Section. Using the new candidate structure selection approach, trees were indeed more densely represented in the candidate sets. Table 5.3 gives a breakdown of the number of each type of structure under both the old and new methods. The Table shows how, under the original method, trees are initially well represented, but become dominated by cycles suddenly as the number of spans is increased to 20. We also see that under the new method the number of cycle candidates is reduced slightly (because of the increased contribution of trees), but that large numbers of both structures are still included across all of the test cases.

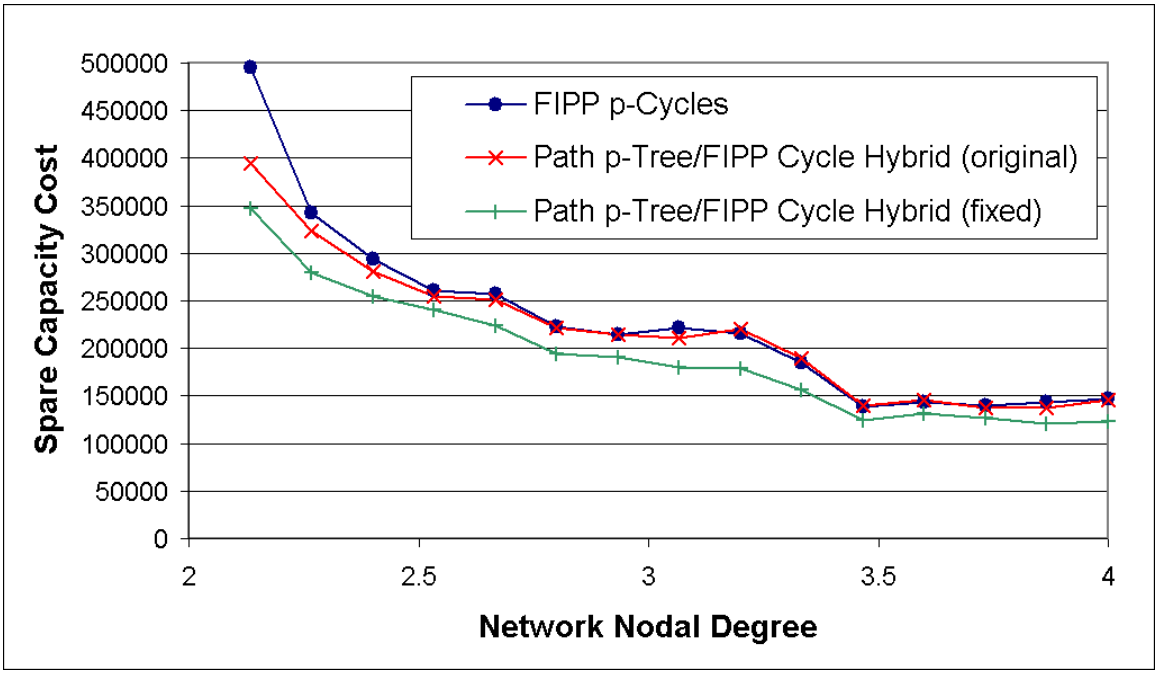


Figure 5.7: Spare capacity costs of hybrid path p-tree and FIPP p-cycle designs with "fair" candidate structure selection

5.3.5 Results of Corrected Experiments

Figure 5.7 shows the costs of the new results as compared to the pure FIPP designs and the original, flawed hybrid designs, discussed above. The curve for the hybrids from Figure 5.6 is replaced with red crosses and the curve for the new hybrids takes on the formatting of the old. The Figure shows a large improvement in all cases, as expected from our twofold effort to improve the protection ability of trees in the design problem. As a result, the hybrids can now improve on the FIPP p-cycle designs by a significant margin. The average improvement is 14.5%, with a maximum of 30% in the 16 span topology. Regardless of whether this method can be said to value tree and cycle candidates "equally", it certainly allows the tree candidates to be be present in amounts that can provide improvements in many cases over cycles, without over-representing them to the extent that their characteristic inefficiency becomes dominant over cycles. Note that in all cases where results for the "$x_c^k = 1$ where any protection is possible" method were available, they did not improve upon the results in Figure 5.7 by more than ~3%, so we can be sure that the method used here was sufficient to provide enough protection options for trees to capture their true efficiency.

The extreme improvement in the 16 span case is understandable, given that there are

only three candidate cycles in this network, meaning that there are not many different choices available for protection using FIPP p-cycles alone. Therefore the design is able to benefit greatly even from the opportunity for 1+1 APS protection that is introduced by the inclusion of degree-2 path p-trees (i.e., PXTs). Indeed, only one true tree structure was used in this design, with the rest being 1+1 APS-equivalent PXTs. We will discuss the details of the structure composition of our designs more in the following Section.

By now we have seen that non-cyclical protection has much more relevance in a path-protecting context than a span-protecting context, so the ability of trees to improve upon cycle designs in this same context is not entirely surprising. As to whether this improvement is due to actual tree protection, or if it is really mostly PXT protection, that will be determined from the following study of the structural characteristics of these designs.

5.3.5.1 Structure Analysis

Again, we mapped out the structures used by our designs in order to determine how exactly path p-trees are used to enhance FIPP p-cycles. Table 5.4 gives a breakdown of the percentage of structures in each design that are cycles, PXTs, and true trees. As we can see from the Table, while PXTs can have a significant presence in FIPP p-cycle design, true tree protection is extremely rare. Indeed, 9 of the designs do not use any tree protection at all, including those for the 6 most highly connected networks. The few true trees that are used in these designs are illustrated in Appendix G.1.

This seems to suggest that tree protection, as in the span-protecting case, becomes irrelevant when the possibility of cyclical protection is introduced. However, there may be a feature of the DRS methodology being used here that interferes with the potential of tree-based protection. We say this because our partial results for the original approach (in which $x_c^k = 1$ whenever protection is possible) show a significantly different composition of structures for many designs. In these designs, PXTs can still make up a significant proportion of the structures (an average of 21.5%), but there are significantly more trees: an average of 5.0% of structures are trees over all designs. The 23 span network has the most trees: approximately 50% of structures are cycles, 30% are PXTs, and 20% are true trees. This is also the network with the greatest capacity difference between the results (about 3.3%). On average the number of trees is still small, but is significantly greater than the results

	% of Total Structures		
Network	**FIPP *p*-Cycles**	**PXTs**	**True Trees**
16 spans	70.8%	28.6%	0.5%
17 spans	88.6%	11.4%	0.0%
18 spans	80.2%	19.8%	0.0%
19 spans	84.4%	15.6%	0.0%
20 spans	79.3%	19.8%	0.8%
21 spans	84.6%	13.5%	1.9%
22 spans	75.7%	21.6%	2.7%
23 spans	59.6%	39.5%	0.9%
24 spans	55.7%	43.5%	0.9%
25 spans	57.6%	42.4%	0.0%
26 spans	98.3%	1.7%	0.0%
27 spans	88.6%	11.4%	0.0%
28 spans	78.1%	21.9%	0.0%
29 spans	59.8%	40.2%	0.0%
30 spans	49.0%	51.0%	0.0%

Table 5.4: Contribution of cycles, true trees, and PXTs to protection in hybrid FIPP *p*-cycle/path *p*-tree designs

reported in Table 5.4. This suggests that most of the capacity efficiency of the design is due to cycles, but that a significant difference in the efficiency of the non-cyclical protection can occur when we eliminate potential protection relationships for lightly protected DRSs (i.e., DRSs for which less than 50% of the demands can be protected). The true trees in these solutions are illustrated in Appendix G.2.

To understand this effect, we looked at the properties of the structures themselves, specifically the trees and PXTs in the two designs for the 23 span network, where we saw the most disparity between our two approaches. Examining these structures showed that almost all of them in the ">50% protection" case are 1+1 APS-equivalent PXTs; only one structure is a true tree that protects multiple demands. In contrast, the PXTs and trees in the "any protection" design often protect multiple demands; there are only 6 out of 32 PXT copies that are equivalent to 1+1 APS arrangements. However, the number of demands protected by these types of structures rarely exceeded 3. This is understandable because we are only able to include small trees in our data set, so it is likely that they can only protect small sets of demands. Unfortunately, to determine if larger trees could be more efficient,

we would have to add them to the design problem as well, which we have explained is not possible given the constraints on computing power that we are under. However, as a practical matter we have seen that it is possible for small path p-trees in hybrid tree/cycle designs to handle the niche of protecting small DRSs. The FIPP p-cycles handle the large DRSs, which they are especially suited to, while small trees and PXTs are efficient for protecting the few remaining demands.

5.3.6 Conclusions

Overall, path p-trees become rather unimportant when cycles are introduced to the design problem, as PXTs and FIPP p-cycles are able to efficiently handle by far most of the protection in the network. Even when we are able to give the maximum advantage to path p-trees by allowing them to protect the widest possible variety of DRSs, the number of tree structures is small and the decrease in capacity cost is minimal (although definitely measurable). If path p-trees have a place in efficient network design, it will be in special cases where capacity efficiency is of utmost concern and designers are able to identify areas in an otherwise FIPP p-cycle design where a small group of demands is more efficiently protected by a small tree.

5.4 Summary and Future Work

Characterization of the efficiency of path-protecting p-trees showed that they are more comparable to cycles and trails than their span-protecting counterparts, but that they are still at a significant disadvantage as a standalone architecture. Structural analysis of the designs showed that they contain significant numbers of true trees (as opposed to PXT equivalents). As a byproduct of these investigations, we also discovered that the DRS method, as used for FIPP p-cycles and PXTs, is not directly suited to the path p-tree design problem, and must be modified to obtain meaningful results. Investigations into hybridization with FIPP p-cycles showed that the utility of true trees decreases significantly in the presence of cycles, although again not as severely as in the span-protecting case. Tree/cycle hybrids are able to improve to a noticeable degree upon pure FIPP p-cycle designs, although this improvement is mostly due to PXT equivalent p-trees.

The fact that tree-based, preconnected protection may be useful, even in a small fraction of cases, means that there may be merit in a further investigation of the meaning of the concept of degree-3+ cross-connection and its device-level implementation. On a more fundamental level, this Chapter establishes a hierarchy of protection efficiency: tree protection is dominated by PXT protection, which is dominated in turn by cycles. This hierarchy, which we have seen holds in both span- and path-protecting cases, is one of the fundamental contributions of this thesis.

Unfortunately, computational constraints have impeded a thorough investigation of this topic, even more so than in Chapter 3 on span-protecting p-trees. Not only do we have to perform experiments with an incomplete tree set, we have also seen how the DRS-based approach to path-protecting structure design can easily lead to false conclusions if we do not carefully inspect the different effects that the implementation can have on different types of protection structures. Basically, the DRS approach, necessary for the feasibility of optimization-based path-protecting structure design, upsets the fairness and neutrality of the combined ILP approach to hybrid design that we initially found so desirable. The investigation and discussion here forms only an initial foray into the area of path p-tree, which we hope will become increasingly accessible with increases in computer power and improved design methods.

That said, the next steps for the research of path p-tree design methods would almost certainly need to address these computational issues. For example, a method of thinning the population of candidate trees, while at the same time allowing structures of a variety of sizes, would allow us to make more definitive statements about the usefulness of large vs. small trees. Another approach would be to abandon the ILP-based design method entirely and investigate heuristics: the greedy PXT design heuristic may be easily adapted to p-tree protection. We have already seen that the heuristic can do as well as or better than our ILP-based method in the PXT case (because it considers structures that our ILP approach does not), so it may be even more suited to the p-tree case where the sheer size of the solution space weighs heavily against the ILP approach.

Chapter 6

Cross-Architecture Considerations for the Protection of Transparent Optical Networks

6.1 Introduction

To this point, we have been focusing very narrowly on comparisons between pre-cross-connected architectures, usually in terms of their capacity efficiency, without considering more broadly the implications of implementing these architectures for the protection of real optical networks. This Chapter collects a number of studies, related by the fact that they all investigate practical considerations of protection in optical networks using the architectures studied previously in this thesis (span and path *p*-trees, *p*-cycles and FIPP *p*-cycles, and PXTs).[1]

[1] Some of the work on the HAVANA project in the first part of this Chapter has been published at DRCN 2007 and ICC 2009:

A. Grue, W. D. Grover, M. Clouqueur, D. Schupke, J. Doucette, B. Forst, D. Onguetou, D. Baloukov, "Comparative Study of Fully Pre-Cross-Connected Protection Architectures for Transparent Optical Networks," *Proceedings of the 6th International Workshop on Design of Reliable Communication Networks (DRCN 2007)*, La Rochelle, France, 7-10 October 2007, pp. 1-8.

A. Grue, W. D. Grover, M. Clouqueur, D. Schupke, D. Baloukov, D. Onguetou, B. Forst, "CAPEX Costs of Lightly Loaded Restorable Networks Under a Consistent WDM Layer Cost Model," *to appear in the proceedings of IEEE International Conference on Communications (ICC 2009)*, Dresden, Germany, 14-18 June 2009.

The work presented in the second part of this Chapter on failure localization has been published in IEEE Communications Letters:

W. D. Grover, A. Grue, "Self-Fault Isolation in Transparent *p*-Cycle Networks: *p*-Cycles as their Own *m*-Cycles," *IEEE Communications Letters*, vol. 11, no. 12, December 2007, pp. 1004-1006.

6.2 Comparative Study of Protection Architectures for Transparent Optical Networks

In 2005, the Network Systems research group (of which I am a member) at TRLabs was approached by researchers at Siemens (now Nokia Siemens Networks, or NSN) regarding the possibility of a cooperative research project on the topic of protection architectures for transparent optical networks. The project, named HAVANA (High AVAilability Network Architectures), was to focus specifically on networks implemented using NSN-specific networking technology, and would address several areas of practical interest to NSN, such as dual failure restorability and cost modeling.[2]

The work described in this Chapter was done as part of this collaborative project. Results from comparative studies with other architectures performed by other researchers are credited to the appropriate project members.

6.2.1 Goals

We have already discussed the general motivation behind the investigation of network protection technologies specific to transparent optical networks. However, the treatment given to the problem so far in this thesis has been largely theoretical in nature, focusing on studies of relative theoretical purity over large families of artificially generated networks. The HAVANA project, in contrast, was to focus on the more practical aspects of network design for a single network under the constraint of transparent networking. The goals for the project can be expressed threefold as:

1. Characterize and compare several protection architectures
2. Consider a mix of many "real world" design constraints simultaneously
3. Develop a set of tools for achieving goals 1 and 2

This thesis will focus mainly on the first two goals, as the implementation details of the tools used to produce our results are not, in general, of academic interest.

[2]My own role in the project was that of a Research Engineer for the first year, and as a Senior Project Engineer for years 2 and 3. This role involved coordination of my own research activities with that of a team of up to 5 other Research Engineers involved with the project, as well as producing deliverables (e.g., summary slides and reports) and putting together and presenting conference papers based on project results.

6.2.2 Overview and Methodology

The study was performed in three distinct stages, corresponding to the three years of the project. At each stage, the goals of the project were reevaluated and new investigative tasks were outlined. The project covered the following areas:

Year 1

- Basic restorable network design with spare capacity minimization objective
- Dual failure restorability analysis of single failure restorable designs
- Implications of wavelength continuity constraint
- Implications of length limitations on protection paths

Year 2

- Single node failure restorability analysis of single span failure restorable designs
- Implications of the constraint that working and protection paths must share the same wavelength (same-wavelength protection)

Year 3

- Cost modeling

The study followed an "accumulative" approach, under which the initial basic single span failure restorable designs were carried forward into subsequent investigations and modified according to the findings, i.e., adjusted to satisfy wavelength continuity, protection path length limits, etc. The result at the end of year 2 of the project was a set of "best feasible" designs that were then used for the cost modeling exercise.

My role as a researcher in this project was to perform the above investigations for PXTs and (to a lesser extent) path- and span-protecting *p*-trees. However, as the study was fundamentally comparative, results for other architectures are also provided in the following Sections, and credited to the appropriate authors. The other architectures included in the study were DSP, *p*-cycles, and FIPP *p*-cycles.

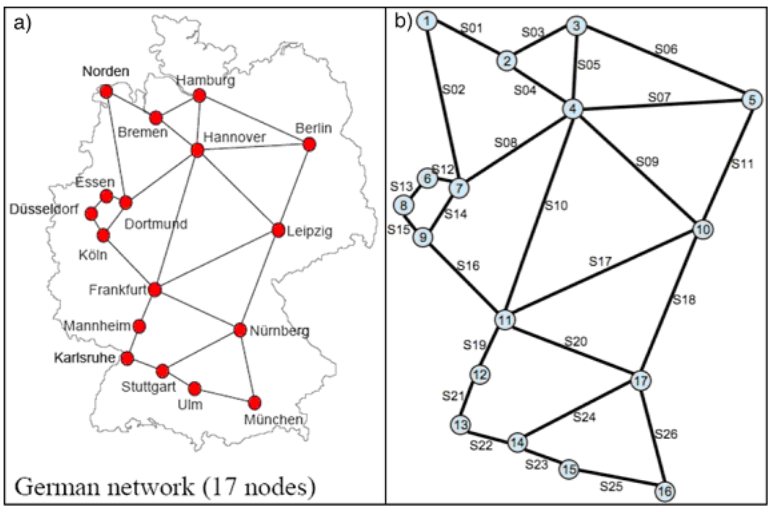


Figure 6.1: Topology for test network TestSet0 that was used for the study

6.2.3 Study Parameters

As mentioned, the study focused largely on the design of restorable networks for a single network topology and demand pattern (named "TestSet0"). The network topology used was the "Germany" network shown in Figure 6.1. Besides being of special interest to Nokia Siemens Networks, it is also a standard topology that is used in many other simulations found in the literature (e.g., for DSP in [GrKo05]). The network has 17 nodes and 26 spans, with an average degree of $\overline{d} = 3.06$. The demand pattern was provided by NSN as an example on a "realistic" demand matrix that could be expected from such a network. This matrix is described in Appendix B.11. Demand is expressed in units of wavelength channels.

6.2.4 Basic Restorable Network Design

6.2.4.1 Method

For PXTs, basic single span failure restorable network designs were generated for PXTs using both the greedy heuristic (Section 4.2) and ILP-based heuristic (Section 4.3) methods. The greedy heuristic for PXTs requires no parameters and simply runs until the network is completely protected. However, the results will vary depending on the order that demands are protected. The program was run 10 times, each time with a new randomized demand order, so as to determine the extent of the effect of demand order on the resulting capacity

cost. Costs were found to vary up to 10% between different runs (i.e., $\pm 5\%$). The design with the lowest cost was chosen as the reference design for further discussion.

For our ILP experiments, we enumerated all of the simple trails in the network as candidate PXTs. There are 13,640 such trails in the network under study. The parameters used for the ILP method (outlined in Section 4.3.3) were 30 DRSs per demand, 30 candidate trails per DRS, and a maximum DRS size of 15 demands. After experimentally testing several permutations of parameters, these values were found to produce good designs in a reasonable amount of time.

For *p*-trees, the set of candidate trees was limited in the manner described in Section 3.2.2. The span *p*-tree model was given as candidates all trees with a maximum size of 9 spans and maximum degree of 4, a total of 136,690 trees. The path *p*-tree model, being a DRS model, is more complex and therefore cannot handle as many candidates given the same computational power and time. So the path *p*-tree model was given the set of all trees with maximum size of 8 spans and maximum degree of 3, a total of 45,997 trees. The DRS parameters used for the path *p*-tree model were 20 DRSs per demand, 20 candidate trees per DRS, and a maximum DRS size of 10 demands.

Each of these four design problems (heuristic PXT, ILP PXT, span *p*-tree and path *p*-tree) was solved twice, using two different models for capacity cost. Under the "distance-based" model, the cost of a unit of capacity on a span was assumed to be equal to its length. Under the "hop-based" model, the cost of a unit of capacity on a span was assumed to be 1 everywhere. These models can be seen as the two extremes of a continuum over which capacity cost has both a constant "setup" component (e.g., transponders and amplifiers at the end-nodes) and a transmission component that varies depending on span length (e.g., amplifiers/regenerators at periodic locations on the span to ensure sufficient SNR at the receiver). The "distance-based" model is one in which the constant setup cost is assumed to be negligible in comparison to the costs of amplification and regeneration; in the "hop-based" model the opposite is true. Which of these models is more accurate will depend on the relative costs of the real-world components, but detailed cost modeling was not a concern at this point in the study, so both models were used in the absence of more specific information.

The working routing method used for the ILP-based and heuristic-based PXT designs,

as well as for path p-trees, was shortest-path routing with modifications used to eliminate the "trap" case that would result in designs not completely restorable against single failures (see Section 4.3.3). The working capacity cost was 166 units for the hop-based cost model and 23934 units for the distance-based cost model. Because span p-trees do not protect end-to-end paths, the "trap" situation is not a concern, so we simply used a basic shortest-path routing algorithm. This algorithm was also allowed to split flow (demand bundles) in a balanced manner between working routes with identical lengths, a situation that will occur frequently when the hop-based cost model for spans is used. Demand splitting like this was not allowed in the working routing algorithm for path p-trees or PXTs, because the complexity of DRS-based models is sensitive to the number of distinct working routes used in the working capacity assignment.

6.2.4.2 Results

For PXTs, The ILP models required several hours to solve to full optimality. In contrast, the heuristic required only a few seconds to generate a solution in each case.

The span p-tree design solved to optimality (a full CPLEX termination with an optimal design) in only a few minutes. The path p-tree design required several hours to solve to full optimality in the hop-based case, but only a few minutes for the distance-based case. Interestingly, the actual CPLEX solution time for path p-trees was only a fraction of a second; by far the bulk of the time was used to generate the columns and rows of the ILP problem itself from the AMPL model and its data.

The results of all 8 experiments are summarized in Table 6.1. The data in this Table reflects properties of PXTs and p-trees established earlier in this thesis. For example, the greedy heuristic PXTs use fewer structures to achieve the same amount of protection. Further calculation establishes that the average lengths of greedy heuristic PXTs in these designs are 12 (hop-based) and 1950 (distance-based), whereas the average lengths for the ILP designs are 5.6 (hop-based) and 790 (distance-based). Also, p-trees are again shown to be very capacity inefficient. This, combined with the uncertainly surrounding the feasibility of their implementation (e.g., what does a fully pre-cross-connected degree-3 node mean?), puts them at an extreme disadvantage with respect to the other architectures in this study where practical implementation issues are at the forefront.

Design		Spare Capacity (channel-km)	Redundancy	Number of Structure Copies Used	Number of Unique Structures Used
Hop-Based Cost Model	ILP-Based PXTs	209	126%	32	25
	Heuristic PXTs	168	101%	14	14
	Span *p*-Trees	303	183%	47	23
	Path *p*-Trees	271	163%	70	51
Distance-Based Cost Model	ILP-Based PXTs	28915	121%	32	29
	Heuristic PXTs	23398	98%	12	12
	Span *p*-Trees	45313	189%	65	23
	Path *p*-Trees	37533	157%	71	48

Table 6.1: Summary of PXT and *p*-tree "low capacity" reference designs (100% single span failure restorable)

The results also show quite a large gap between the efficiency of the designs produced by the ILP-based PXT design approach and the heuristic-based PXT design approach. In our studies on the Murakami & Kim network in Chapter 4, even though the heuristic was able to outperform the ILP-based approach, the gap was not nearly so large. This suggests that, for this particular test network and demand pattern, limiting the PXTs to being simple trails in the ILP model is a significant limiting factor on capacity efficiency. However, the complex nature of the self-intersecting PXTs that the heuristic produces is still a significant reason to consider the ILP-based approach to be advantageous. Also, we must keep in mind that in the end, the true "real world" cost of these designs depends not on span-based transmission costs, but on the node hardware required to support the capacity plan, and we have not yet seen how these two cost measures are correlated. Therefore any conclusions about characteristic cost at this point must be considered to be tentative.

Figure 6.2 gives a graphical comparison of the redundancies of the PXT and *p*-tree designs to the other architectures included in the study. The redundancy of 1+1 APS (173%) is given as a horizontal red line on the Figure. This Figure highlights the low efficiency of

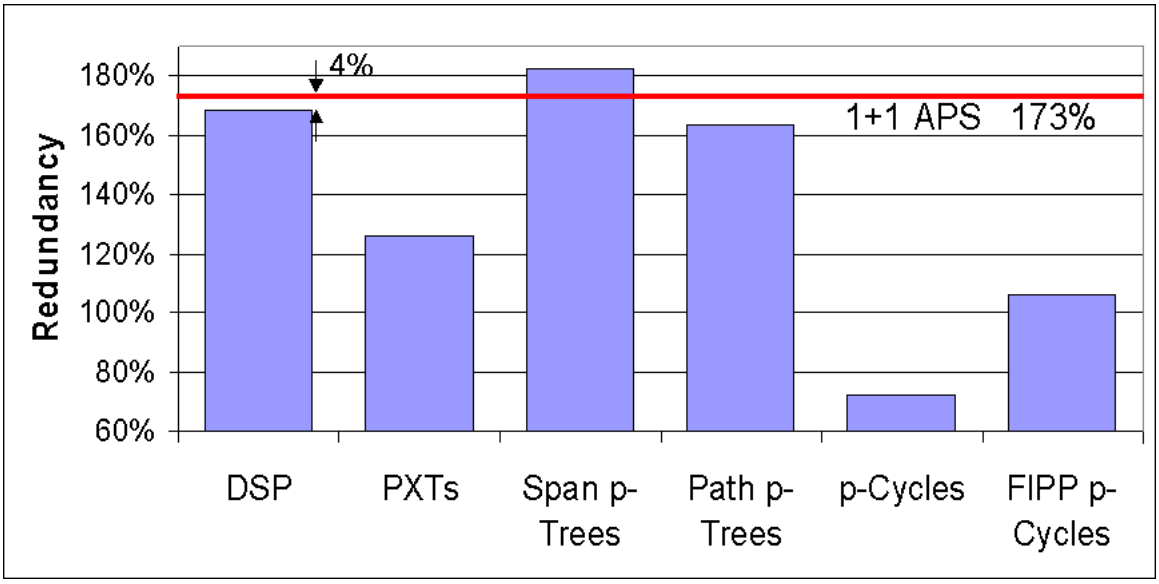


Figure 6.2: Comparison of spare capacity redundancy values of initial low capacity designs for all HAVANA architectures

p-trees. Path *p*-trees use nearly as much capacity as APS and DSP, showing that path *p*-trees do not make significant use of the ability to share protection capacity between disjoint working paths. Span *p*-trees are even more inefficient than APS; this is possible because span *p*-trees are span-protecting and therefore we cannot say that APS is strictly a degenerate case of span *p*-trees (otherwise we would expect span *p*-trees to be as costly or less costly than APS in all cases). We can see that PXTs and FIPP *p*-cycles compare favorably to each other, but *p*-cycles are able to attain much higher efficiency than any other architecture, path-protecting or otherwise.

The results for APS and DSP were produced by Brian Forst (see also [FoGr07, Fors09]), the results for *p*-cycles were produced by Diane Onguetou, and the results for FIPP *p*-cycles were produced by Dimitri Baloukov.

Effect of DRS Parameters on Results

The ILP problems used for these experiments were incomplete due to the large size of the candidate sets for the PXT, span *p*-tree, and path *p*-tree problems (as explained in Sections 4.3, 3.2.1, and 5.2, respectively). Therefore some attention should be given towards the efforts taken to determine that these results were close enough to optimal to be meaningful and comparable with each other and the other architectures in the study.

When we were initially experimenting with *p*-tree generation to determine how large we could feasibly make the candidate set, we generated *p*-tree designs using size limits of both 8 and 9, and degree limits of both 3 and 4, and all four combinations of those two variations. The (9,4) combination was finally chosen as the reference design that is described in Table 3, but the costs of the solution in all four instances was actually exactly the same. This tells us that, for this network at these specific tree degree and size limits, there is no benefit in considering larger trees for protection.

Because the path *p*-tree model takes so long to generate solutions, we did not spend as much time looking into variations of candidate tree parameters for path *p*-trees. However, we did attempt one more experiment to test whether or not our initial design fairly represented the efficiency of path *p*-trees. In this test, the DRS parameters were increased to 30 DRSs per demand, 30 candidate trees per DRS, and a maximum DRS size of 15. Despite these increases, the cost of the resulting design was 277, higher than the cost of 271 for the initial design. This is possible because of the random variations introduced by the nondeterministic nature of the DRS generation algorithm. Despite this unpredictability, it is reasonable to make the general conclusion that there are probably not major gains to be made by further increasing the DRS parameters from those used in our original test. We decided to keep the original design as the "official" path *p*-tree reference design because of its lower cost, even though the size of its populated data set was smaller.

Additional runs of the PXT ILP model were also performed. In initial tests, the hop-based model was run with only 20 DRSs per demand, 20 candidate trails per DRS, and a maximum DRS size of 10 demands. However, there was no difference between the cost of this design and the reference design, described above, which used the parameter values (30, 30, 15). This does not necessarily mean that the designs were exactly the same (indeed, this is unlikely, as the DRS selection procedure is randomized). It just serves to show that the DRS parameter values used are more than adequate, as the increase in the size of the populated data set between the two tests did not produce any improvement.

The distance-based model was also run again with the same parameters to determine whether wide variance should be expected between different runs. If the variance was wide, it would be wise to perform averaging over a wide set of test runs to obtain more reliable characteristic capacity cost data. The difference in the results was only 1%, however, so

the existing design was considered to be adequate.

6.2.5 Dual Span Failure Restorability

As basic low-capacity design of p-tree and PXT networks has already been investigated in detail, the above exercise was undertaken only to obtain reference designs to carry forward into further investigations. The first such investigation was an analysis of dual span failure restorability ("span R_2") in the reference designs. In other words, given the existing designs with guaranteed 100% span R_1, how much span R_2 can we obtain "for free", given the existing spare capacity allocation, protection structures, and predefined protection actions?

Span R_2 was calculated by simulating each design's response to every possible dual failure scenario and recording the total numbers of damaged working routes, as well as the total number of these working routes that could be restored. For the purposes of this analysis, it is assumed that nodes are only capable of performing a single, pre-determined switching action in the event of failure. In other words, we assume that nodes have no ability to adaptively respond to multiple failures; whenever a failure is detected on a working path, the same restoration response is attempted regardless of possible prior failures and recovery actions. If this attempt fails due to a failure on the protection path, then the demand simply remains unrestored. The dual failure restorability of each architecture was expressed using the ratio of demand flows restored to the total number of demand flows affected (damaged) by either of the two failures, over all dual failure scenarios (i.e., $R_2 = \frac{\text{flows restored}}{\text{flows affected}}$).

6.2.5.1 Method

To perform the R_2 analysis, scripts were written to perform a simulation of the network designs' responses to each possible dual-failure scenario. Once the script analyzes each of these dual failure responses to determine the number of affected demands that are restorable, the R_2 value is computed. This basic method was used for all architectures in the study. A high-level pseudocode of this algorithm is as follows:

```
1. For each protection structure in the design


    a. Make an explicit assignment of the protection paths it
```

```
    provides to the spans/demands it protects (depending on
    whether the architecture is span/path-protecting,
    respectively).

2. For each dual span failure scenario [s1,s2]:

    a. Trigger the failure of span s1:

        i. Mark failed working paths as affected and switch them
        to their designated protection paths.

    b. Trigger the failure of span s2:

        i. All protection paths activated in step 2.a.i. that
        contain s2 are considered unrestorable but all of their
        spare capacity is still held and considered unusable.
        ii. All working paths unaffected in step 2.a. that contain
        s2 are marked as affected and switched to their protection
        paths (and marked as restored) if possible, i.e., if the
        required spare capacity is still available.

    c. Count the total number of working paths affected by the
    failure scenario as well as the number of restored paths.

3. Compute  from the total number of affected and restored
working paths over all failure scenarios.
```

Step 1 (explicit assignment of protection structures) is necessary for many of our designs because they may contain a certain degree of *overprotection*.[3] That is, it is possible that some of the spans/paths in the network are protected by more structures than they need to be, meaning that they have a choice of which protection structure they can use. This has no effect on span R_1 (always 100% for our designs), but when it comes to R_2 calculations, the precise assignment of structures may affect the R_2 value. We took the approach of

[3]This does not apply to the PXT heuristic designs because the greedy algorithm explicitly assigns PXTs to working paths as it proceeds.

simply performing an arbitrary assignment of structures to protected spans/paths, because if the overprotection is minimal the effect of variations in the assignment on R_2 should be equivalently minimal. A more thorough approach would be to use another ILP model to find an assignment that maximizes R_2, but this would introduce a great deal of complexity into the problem for minimal benefit.

6.2.5.2 Results

Table 6.2 summarizes the results of the span R_2 analysis. For PXTs, it is not clear why the heuristic design has so much higher R_2 restorability than the ILP design in the hop-based case, whereas in the distance-based case they are practically identical. Simple random chance is one possible explanation. However, it is noteworthy that the heuristic does not in any case exhibit a lower R_2 value than the ILP-based solution, even though the heuristic designs are significantly more capacity-efficient. This may merit further investigation, as it suggests that the advantage afforded by the complex PXTs used by the heuristic is not reduced cost, but rather higher availability.

We also see that the p-tree designs have significantly higher span R_2 than the PXT designs in most cases. This is a natural result of their lower capacity efficiency. A high degree of sharing of the protection capacity in protection structures will result in an efficient design, but when a dual failure occurs, any two working paths affected by the failures that share protection capacity will contend for this capacity, and only one at most will be restorable. Also, a high degree of sharing will usually result in longer protection paths, as protection paths are detoured to make sharing relationships possible. But a longer protection path will also make it more likely that a second failure in the network will affect the protection path, making the working path unrestorable. Therefore we see that protection capacity sharing increases the likelihood of the two types of failure events (two failed working paths contending for capacity and failures on both a working path and its protection path) that cause unrestorability of dual span failures. Our p-tree designs, being less efficient, have less sharing and therefore are able to fully restore a larger number of dual span failure events.

In terms of a comparison to the other architectures considered by the study, the results for PXTs and p-Trees are near the upper end of the spectrum. The architecture with the

Design		Demands Affected	Demands Restored	R_2 Restorability (Restored / Affected)
Hop-Based Cost Model	ILP-Based PXTs	8116	5457	67%
	Heuristic PXTs	8116	6288	77%
	Span p-Trees	8290	6746	81%
	Path p-Trees	8116	6713	83%
Distance-Based Cost Model	ILP-Based PXTs	8468	5945	70%
	Heuristic PXTs	8468	5967	70%
	Span p-Trees	8684	7162	82%
	Path p-Trees	8468	7030	83%

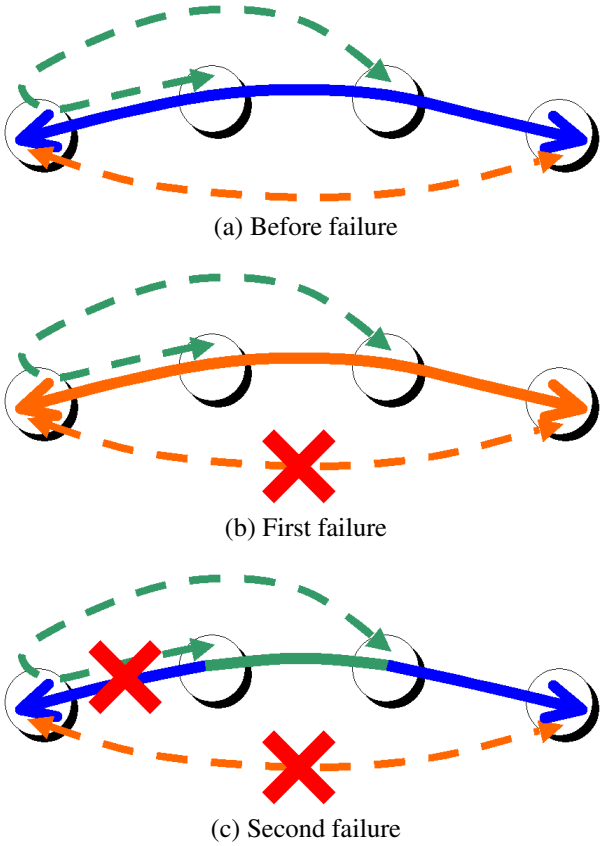
Table 6.2: Summary of dual span failure restorability in PXT and p-tree restorable network designs

highest span R_2 value was DSP with R_2=85%, unsurprising considering the design has a redundancy very close to that of APS. We can see that span and path p-trees both come close to this value with redundancies in the range of APS and DSP as well. PXTs, however, have R_2 values more in the range of p-cycles (R_2=66%). This is a surprising result, considering that the redundancy of p-cycles is quite a bit lower than that of PXTs (~70% vs. ~120%). This speaks highly of the ability of p-cycles to restore so many dual failures using so little spare capacity.

The results for APS and DSP were produced by Brian Forst (see also [Fors09]), the results for p-cycles were produced by Diane Onguetou, and the results for FIPP p-cycles were produced by Dimitri Baloukov.

6.2.5.3 Effect of Allowing Protection Path Stub Release

The pseudocode representation of the R_2 analysis method explicitly states in step 2.b.i that spare capacity that is used for the protection against the first failure is never considered available for use by other protection paths that may potentially be formed in response to the second failure. This is the case even if the second failure causes the failure of the initial protection path. However, path-protecting architectures may have an additional recourse available to them if this occurs. Even though the protection path loses integrity due to a



(a) Before failure

(b) First failure

(c) Second failure

Figure 6.3: Spare capacity stub release in a dual span failure scenario for PXTs

failure, all units of spare capacity on that path that are not also on the failed span (called the *stubs* of the failed path), are still intact, and thus could be *released* and used instead to protect against further failures. This may allow further demands to be restored in the event of a second failure, increasing R_2 restorability.

The spare capacity stub release concept for dual failures is illustrated in detail in Figure 6.3. Subfigure (a) shows a PXT structure (solid line) protecting two paths (dashed lines). In subfigure (b) one path has failed and the entire structure is used to protect it (symbolized by the structure changing to the colour of the failed path). In subfigure (c) a failure occurs on the PXT, meaning it can no longer be used to protect the first path. The undamaged sections of the PXT are released, and part of it is then used to protect the other demand, which has now also failed. The structure shown in this Figure is a PXT, but the same concept applies to path p-trees.

As explained above, it was assumed for the purposes of the initial study that the network was not intelligent enough to support this functionality. However, we calculated protection

path stub release R_2 values for the PXT and path p-tree designs anyway, in order to determine the R_2 gains that could be made were this degree of network intelligence available. The effect was however found to be small. For the p-tree designs, the number of restorable demands increased by only 3 (out of over 8000 affected demands in each case) in both the hop and distance-based path p-tree designs. For PXTs the effect was slightly greater; R_2 increased by approximately 1% for both the ILP and heuristic designs. These gains are likely not large enough to justify the price of requiring centralized tracking of network failure states (although if such a feature is available, it can be used to attain a small increase in R_2 and, therefore, availability).

6.2.5.4 Further Outlook for p-Trees

At this point in the study we had identified both quantifiable advantages and disadvantages of p-trees: the designs tend to be approximately 60% more capacity redundant than PXTs, while seeing a corresponding increase in span R_2 of about 10%. This alone is not enough to totally rule p-trees out of practical usage, as network operators might be willing to pay more for increased availability (perhaps for demands of higher availability gold or platinum QoP classes). They also remained of high theoretical interest in general. For the purpose of the HAVANA project, however, with its focus on practical problems and solutions, they were dropped from further studies due to the questions surrounding the feasibility of their implementation in real networks. The decision was made to move forward with the traditionally preconnected (i.e., degree-2) structures only, in order focus more closely on the defined goals of the project. Therefore we will discuss only PXTs for the remainder of this Chapter.

6.2.6 Assigning Wavelengths to Lightpaths

To this point, a restorable PXT design has consisted of simply a working capacity routing plan and a set of PXTs. This defines for us the capacity requirements on each span, as well as the cross connection configuration for the OXC/OADM equipment. However, there is a third set of variables that is required when we consider this design in a WDM context: the assignment of physical wavelengths to each lightpath in the network. For the purpose of this study, we assume a network with no wavelength conversion at all. Therefore each working

path and each PXT is one lightpath end-to-end. The initial design process assumed the plentiful availability of wavelengths on each span, but a real network has a limited number of wavelengths per fibre. Also, even if the number of available wavelengths on a span is sufficient to support the wavelength channel requirement (aggregate of spare and working capacity) on that span, it may still be impossible to find a wavelength assignment for the network without wavelength conflicts, due to the wavelength continuity constraint for paths and PXTs. These were the issues to address when solving the wavelength assignment problem for PXTs.

6.2.6.1 Method

Before delving into the details of the wavelength assignment method, we must explain the fact that this study was only done on the ILP-based PXT designs, and not the designs generated by the greedy heuristic. This is explained by the discussion in Section 4.2.5.4 on the topic of wavelength continuity in heuristic PXT designs. For heuristic-based PXTs, either we will have multiple fibre requirements on many of the network spans, or we must introduce wavelength conversion into the network. The first option would make the wavelength assignment problem trivial (as the multiple fibres would result in a large surplus of available wavelengths in the network), and the second option is not in the spirit of the current investigation (although the more general design problem in which wavelength conversion is available at a cost is certainly still of interest).

The approach taken for wavelength assignment was to use a secondary ILP model to assign wavelengths to each of the continuous wavelength paths in the designs, such that the total number of wavelengths used in the assignment was minimized. We use the term "secondary model" because it uses the outputs of the first model as parameters for the new wavelength assignment problem; it is not a combined model that both solves the network design and wavelength assignment plan simultaneously. The model used was as follows:

ILP Model

Sets:

S The set of spans in the network, indexed by s.

P The set of *all wavelength-continuous lightpaths* in the network (i.e., both working paths and PXTs), indexed by p.

W The set of wavelengths available in each fibre, indexed by w.

Input Parameters:

δ_s^p Encodes the spans on lightpath p. $\delta_s^p = 1$ if lightpath p crosses span s, 0 otherwise.

Decision Variables:

u_w^p A binary variable that determines whether or not lightpath p uses wavelength w.

a_w A binary variable that is 1 if wavelength w is used anywhere in the network, and 0 otherwise.

Objective Function:

Minimize

$$\sum_{w \in W} a_w \tag{6.1}$$

Constraints:

$$\sum_{p \in P \, s.t. \, \delta_s^p = 1} u_w^p \leq 1 \quad \forall s \in S, \forall w \in W \tag{6.2}$$

$$\sum_{w \in W} u_w^p = 1 \quad \forall p \in P \tag{6.3}$$

$$a_w \geq u_w^p \quad \forall p \in P, \forall w \in W \tag{6.4}$$

Constraint 6.2 ensures that there are no wavelength conflicts by stating that each wavelength on a span can only be used by at most one path that crosses that span. Equation 6.3 ensures that each path is assigned exactly one wavelength. Equation 6.4 calculates the overall usage of each individual wavelength in the network based on whether or not it is used by any lightpath. The objective function then minimizes the total number of wavelengths

used. The wavelength continuity constraint is implicit in this model because the wavelength variable u_w^p sets the wavelength usage on a per-path basis, not for each individual path hop.

Note that, for the purposes of determining if a feasible wavelength assignment exists for a PXT design, this model does not even have to be run to optimal termination. To obtain such an assignment, the model only needs to be run until a feasible integer solution is found. Such a solution represents a valid wavelength assignment, although it may not use the minimum number of wavelengths possible. Because our goal with this model was mainly to analyze the feasibility of wavelength assignment, the objective function is somewhat of an arbitrary placeholder, present only because the solver requires it to guide the solution process.

6.2.6.2 Test Cases

As mentioned above, this model was only run on the ILP-based designs. Also, it was determined at this point that carrying through hop-based and distance-based designs was duplicating effort unnecessarily. We decided, with the help of our colleagues at NSN, that the hop-based model of span cost was likely to be more realistic than the distance-based model for their purposes, so distance-based designs were dropped from the study. Therefore, from now on results apply only to the hop-based cost model design case.

The first test case was run on the initial low capacity 100% span R_1 ILP-based PXT reference design (using the hop-based cost model). The problem was given two wavelength bands to work with, each containing 20 wavelengths. A feasible wavelength assignment was expected for this case, as the largest total capacity requirement of any single span in the design was only 24 wavelength channels out of the maximum of 40, leaving plenty of room for the solver to find a workable solution. In contrast, the goal of the second test was to create a PXT design using only a single band of 20 wavelengths. Obviously we could not use the original reference design for this purpose, as we know it must contain at least 24 wavelengths. Therefore we needed to first re-solve the PXT ILP model to create a new design with at most 20 wavelength channels of capacity on any span. We did this by using the exact same populated data set as for the initial low capacity design test in Section 6.2.4, and adding a single additional constraint to the model that prevented any span from carrying more than 20 units of total capacity. The wavelength assignment model

was then run on this modified design, this time given only a single wavelength band with 20 wavelengths.

6.2.6.3 Results

40 Wavelength Design

As expected, the first test using 2 bands of 20 wavelengths each was able to find a valid wavelength assignment. The model solved for an optimal design containing the absolute minimum of 24 assigned wavelengths in only a few seconds.

20 Wavelength Design

For the first part of this test, in which the PXT design was re-solved with the additional 20-capacity-unit cap on each span, the solver was able to find the new design in approximately the same amount of time as it required to find the original design. The capacity cost of this design was 213 as compared to 209 for the original reference design, a cost increase of only 2%. Note that it was not guaranteed that such a design would be found. It is conceivable that capacity forcing effects might create a minimum capacity floor for the network that would exceed 20 units for some spans, meaning that the 20-unit cap constraint would render the problem infeasible. The fact that such a design was found, and with such a low increase in spare capacity cost, indicates that PXTs are flexible enough to support rearrangement of the design to accommodate capacity cap limits on spans.

After this new design was found and the wavelength assignment model was run, an assignment using only 20 wavelengths was then found in only a few seconds. Note that although the 20 unit capacity limit per span is a necessary condition for a 20 wavelength design, it is not a sufficient condition; the wavelength continuity constraint for paths means that even though a span may require only 20 wavelengths, some of the 20 paths that cross the span may need to be assigned the same wavelength in order to accommodate other areas of the design. If we had found this problem to be infeasible, a reasonable approach would have been to try a capacity cap of 19, 18, etc. until a 20 wavelength design was found.

6.2.7 Transparent Optical Path Lengths

Another characteristic of real networks that has an impact on restorable network design is the practical limit on the lengths of transparent optical paths. If a signal is sent transparently on an optical fibre, even with regular amplification, there is a limit to how far the signal can be sent before the SNR degrades unacceptably. Therefore, a network either needs to be designed carefully such that no lightpaths exceed this limit, or the design process must include the placement of 3R regenerators. Regeneration, however, involves OEO conversion of the signal and therefore violates transparency of all regenerated paths. Therefore, in order to maintain the advantages of transparency for all paths and to avoid the cost of regeneration equipment, it is advantageous to simply control path lengths a priori.

This may not be a practical approach to take with existing systems, but it is a valid approach for a theoretical study in which we can return to the original "green fields" design problem whenever necessary to consider a new requirement or design factor. In the following Section, we first analyze the optical path lengths in the PXT reference designs, and then go on to develop a method of restricting the path lengths in these designs.

6.2.7.1 Analysis of Reference Design

The equation defining the limitations on preconnected path length due to optical losses is as follows:

$$length + (hops - 1) \cdot 80km \leq 2000km \tag{6.5}$$

The path length limit, as given in Equation 6.5, is 2000 km. The *length* of the path is given in km. However, note that the left hand side expression is not solely a length measurement but also incorporates the number of *hops* in the path. This is because the path undergoes a retransmission loss at each node which is the equivalent of the loss over 80 km of fibre. This expression can therefore be thought of as the "equivalent length" of the path for transparent transmission purposes. Therefore the following analysis is of this "equivalent optically transparent length" (or "transparent length" or "optical reach", etc.) of the lightpaths in the PXT design, and not the actually physical length (although in practice these values will be quite close for this network).

Our first exercise was to inspect our wavelength-assigned PXT designs and record the transparent length of each lightpath (both protection and working paths). If this value is greater than 2000 km for any path, then the network design as it stands requires the use of regeneration hardware at some nodes in order to support the establishment of the paths in question. Note that we do not consider the transparent length of entire PXTs to be important in this exercise; even though a PXT is pre-cross-connected end to end, each individual working path may only use a small subset of this preconnected path as a protection path. The end-to-end pre-cross-connection is performed only to ensure that any sub-path formed as a protection path will be pre-cross-connected as well. In the failure state, we will not in general be transmitting across the entire PXT end-to-end (although if there is an entire-PXT protection path, the length of the PXT will be reflected indirectly as the length of this path in the following analysis).

It is important to note that, as with R_2 analysis, an extra step must be taken before performing this path length analysis for the ILP-based design to explicitly assign working paths to the PXTs (and protection paths within those PXTs) that protect them. Again, this is because of the possibility of overprotection. For working paths that are protected by more PXTs than they need to be, the choice of PXT assignment matters, as the lengths of the protection paths will generally be different depending on which PXT is chosen. For our initial analysis, we did not place any special criteria on PXT assignment. In other words, the assignment was arbitrary. We proceeded in this way with the understanding that, if transparent reach limit violations were discovered, we would have to investigate the possibility of rearranging the PXT assignment to satisfy the reach condition.

In-depth optical reach analysis of the heuristic-based designs was not performed because we already knew that there were significant problems with the implementation of that type of PXT in a single-fibre network without wavelength conversion.

6.2.7.2 Analysis Results

Figure 6.4 contains a histogram giving the distribution of the equivalent lengths of the protection paths in the ILP PXT reference design. The red bar represents the reach limit of the network of 2000 km. Most (91%) of the protection paths lie below this limit, but 9 paths exceed it. The longest path has an equivalent length of 2907 km, almost 1.5 times

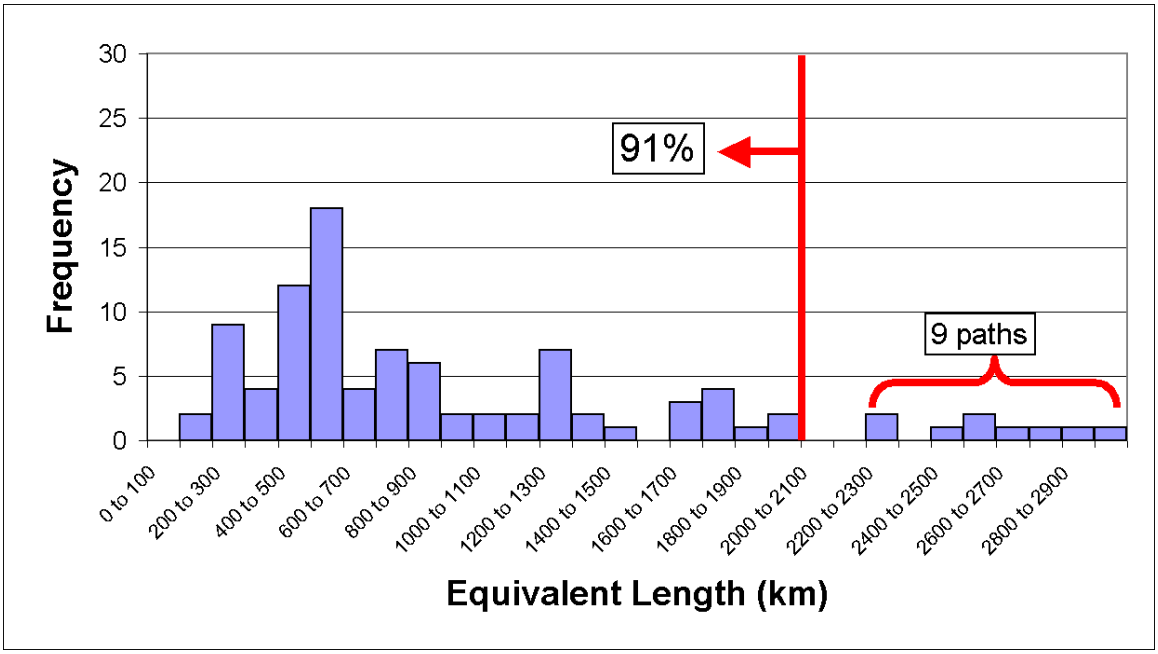


Figure 6.4: Distribution of equivalent transparent path lengths in the original reference ILP PXT design (protection paths)

the maximum limit. Figure 6.5 is the corresponding histogram for working paths, using the same axes. Unlike the protection paths, all of the working paths are significantly under the 2000 km limit, a fact that is assured by shortest-path routing in this particular network. Therefore the working routing is not a concern as far as transparent reach is concerned, but we still must make some changes to the protection layer of the design at the very least in order to satisfy this restriction.

Before trying to modify the protection layer to shorten these 9 overlong paths, we first investigated the possibility discussed above, that we might be able to rearrange the PXT assignment for overprotected demands such that the protection paths of these demands would become shorter. Because we can rearrange protection paths only for overprotected demands, the set of overprotected demands in the network must be protected by at least one of our 9 problematic paths, otherwise we cannot make any gains using this approach. Unfortunately, even though this design contains 3 overprotected demands, none of their protection paths are among those above the 2000 km limit. Therefore we needed to make changes to the PXT configuration itself in order to solve this problem.

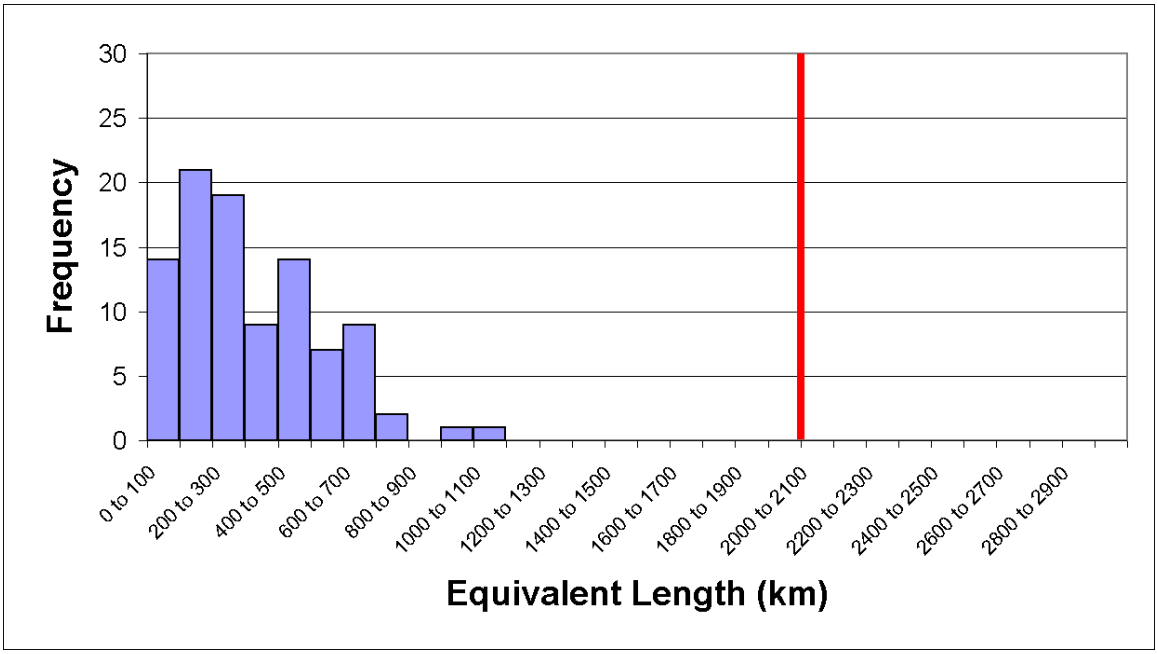


Figure 6.5: Distribution of equivalent transparent path lengths in the original reference ILP PXT design (working paths)

6.2.7.3 Modified ILP Method with Path Length Limits

Altering the ILP-based PXT design approach to take transparent reach restrictions into account is straightforward. Recall from the definition of the PXT ILP model in Section 4.3.2 the parameter β_r^k that determines whether PXT k is able to offer protection to demand r. This parameter is 1 (for simple PXTs) if k can protect r and 0 otherwise. We can therefore enforce transparent reach restrictions on any PXT design problem by manipulating this parameter to ensure that the parameter is never 1 for a combination of k and r such that the protection path used by r in k is longer than 2000 km. The constraints of the model do not have to be modified at all.

To test this method, we executed a post-processing step on the same data set used to populate the ILP PXT problem for the original reference design (recall that the same data set was also used for the 20-wavelength restricted design). This step involved iterating through every PXT and examining the demands for which β_r^k had been initially calculated as 1. If the protection path for this demand was too long, we toggled the value of β_r^k to 0. Note that this type of procedure could also be done directly during the DRS generation procedure itself instead of as a post-processing step, and that this might produce slightly

better results, as the DRS method would then be generating its DRSs with accurate information as to which PXT could protect which demand. The advantage of post-processing the existing data set, however, is the fact that experiments run using this modified data set will use the same DRS and PXT sets used by the initial experiment, making the results more easily comparable. For example, we know that the cost of the resulting design must increase, as it is a more highly constrained version of the original problem. In contrast, if we were to run the DRS method again, this time excluding directly the protection paths that are too long, the randomized set of DRSs would be completely different, and the cost of the design would be affected by this in an unpredictable way, reducing our ability to judge the effect of this modification on the capacity cost of the design. In essence, this method allows us to modify one variable at a time and measure the effect.

After obtaining our modified data set, we solved the PXT design problem with this data, along with the 20-wavelength constraint discussed in Section 6.2.6. We then verified that this design could be given a valid 20-wavelength assignment. The result was then a design that satisfied both the optical reach and 20-wavelength limitations of our problem definition.

6.2.7.4 Results with Explicit Path Length Constraint

The resulting design had a spare capacity cost of 220 units. Recall that the original design had a cost of 209, and the 20-wavelength design a cost of 213. Therefore including reach considerations increases cost by 7, another 3%. Together the two constraints raise the cost by 5% over that of the initial design. The histogram of the protection path optical length requirements is shown in Figure 6.6. All protection paths are below the 2000 km reach limit because paths longer than this are explicitly forbidden by the post-processing step for the model's data set. As for the working paths, there is no explicit check in this method that ensures that they are short enough. As we have mentioned, this is not a concern for shortest-path routing in this network, but for the design of general networks the working routing must be inspected to ensure that it too meets the requirements for transparent reach. Of course, if shortest-path routing does not satisfy these requirements, regeneration hardware will be strictly required at some point in the network, the placement of which becomes an entirely different problem.

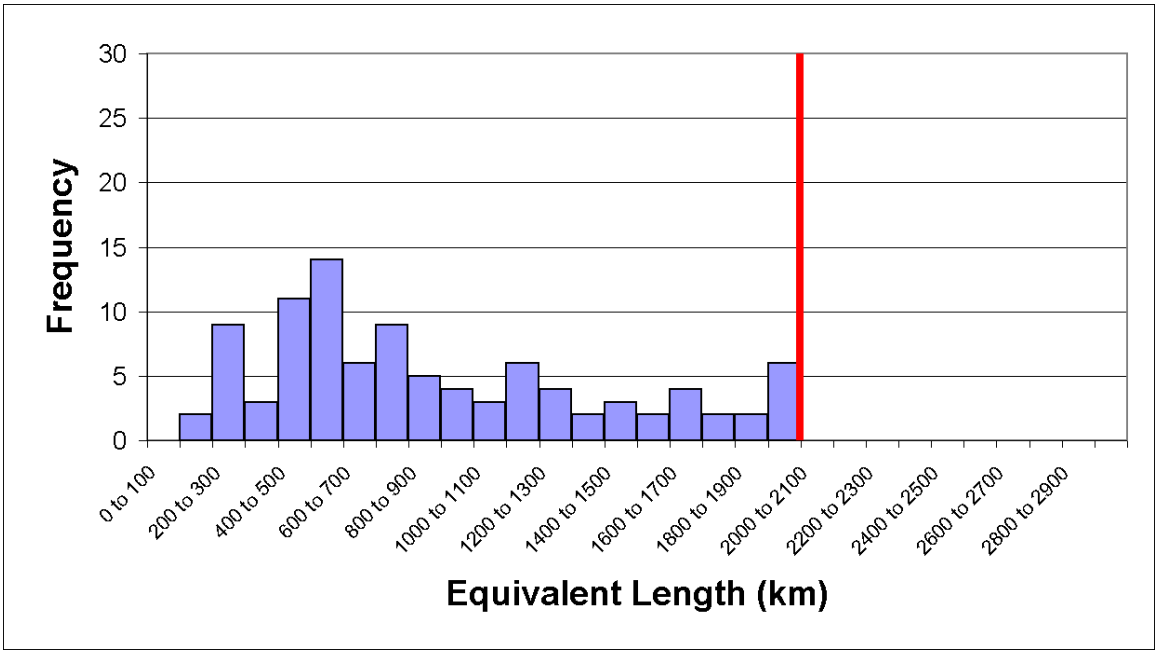


Figure 6.6: Distribution of equivalent path lengths for protection paths in a PXT design with explicit path length limit and 20 wavelength constraint

The result of this work was a modified reference design that satisfied all of the restrictions considered by the study at the time. Therefore we used this "best feasible" design as the standard design for all subsequent investigations. For the comparable work on path length restrictions in *p*-cycle networks that also came out of this project, see [OnGr08].

6.2.8 Same-Wavelength Protection

As we are going over each substudy of the project chronologically, some words now should be spent on the topic of same-wavelength protection. However, the content of this investigation is essentially repeated in Section 6.2.11 (cost modeling), where it comes into play when considering optically switch vs. electronically switched protection. Therefore most of the discussion was incorporated into Section 6.2.11, with this Section only including a brief explanation of the concept and discussion of the problem.

6.2.8.1 Motivation and Approach

In the first year of the project, it was assumed that the hardware used for protection switching was able to switch transmission from a working path on one wavelength to a protection path on a different wavelength if necessary. At the outset of the second project year, how-

ever, it was pointed out by our colleagues at Nokia Siemens Networks that enforcing the constraint that protection must occur on the same wavelength as the working path could possibly carry non-negligible cost benefits. Also, the capability to have a working and protection path on different wavelengths may not even be present in all network technologies. This motivated us to study the implications of imposing this additional protection constraint on the architectures under study.

For this initial consideration, we restricted ourselves to the level of a thought experiment only, in answering the question "What are the theoretical effects of same-wavelength protection on our pre-cross-connected protection architectures?" The idea was that important observations could be made before running actual experiments, observations that would be important when continuing the investigation forward to discover the actual qualitative cost benefits of same-wavelength protection. Several observations were made for each architecture and compiled into a report for NSN for this part of the study. However, all of these observations come out in the cost modeling study in Section 6.2.11, so we will not discuss them in detail here.

6.2.9 Node Failure Restorability Analysis

Usually the study of restorable networks is confined to investigating restoration against span failures. Indeed, all of the designs created for the TestSet0 network so far have been created with the aim of providing full restorability against all single span failures. This follows from the fact that spans represent a fundamentally much more vulnerable medium (fibre) than nodes (DCS, OXC, whole CO buildings, etc.). However, node failures do happen (catastrophic power failure, building fires, etc.), even if they occur at a much lower rate than span failures, so it is of some interest to determine how our protection architectures behave in their presence.

6.2.9.1 Motivation and Approach

It is a commonly touted feature of path-protecting architectures that they are able to protect against node failures as well as span failures. See, for example, [ZhZh06], which states that "The significant advantage of *p*-cycle based path protection over link protection is the node failure recovery capability". It is true that, under end-to-end path protection, it is as easy

to perform a path switch as a result of a failed node as it is for a failed span. When the end nodes detect a loss of light, they will switch to the protection path in the same manner no matter what type of failure scenario exists on the working path (i.e., any number of failed nodes and spans in any combination will produce the same failure state from the point of view of the end nodes). Span protection cannot achieve this, because a failed node cannot be used to switch directly; the closest we can achieve to span protection is to route around the failure at the two nodes one hop removed from the failed node. However, this node failure protection *capability* does not translate into an automatic node failure restorability *guarantee* if the network is not explicitly designed to take this into account. Unless the network is designed with node failures in mind, a sufficient number and arrangement of intact protection paths may not exist after a node fails. This may not be acceptable if the system needs to be designed for extremely high availability.

Having established this fact, the question remains whether or not this is even an issue for networks designed to protect against span failures. It may be the case that many node failures are protected by default, with only a small portion being unprotected, even when the network is not designed specifically for this objective. Conversely, it may also turn out that a high proportion of failures are unprotected, requiring a network redesign to attain acceptable availability levels. The first objective of the node failure restorability investigation was to quantify the degree of node failure restorability attainable via by the PXT reference design. We then investigated both the methods required for adjusting the designs to provide 100% node failure restorability, as well as the effect this adjustment has capacity efficiency.

6.2.9.2 General Properties of the TestSet0 Network

In this Section, we discuss some properties of the German TestSet0 network that are independent of the particular protection architecture used and also important to the node failure restorability study. Recall that a shortest-path (by hops) demand routing is used in most of our designs (with modifications where necessary to support restorability). For each node, the resulting traffic handled by that node can be broken down into demand that that bypasses (transits it), and demand that it sources or sinks. When a node failure occurs, any demand that originates or terminates at the failed node is irreparably lost. Only working paths that transit the node can be restored by being switched to protection paths. We call these paths

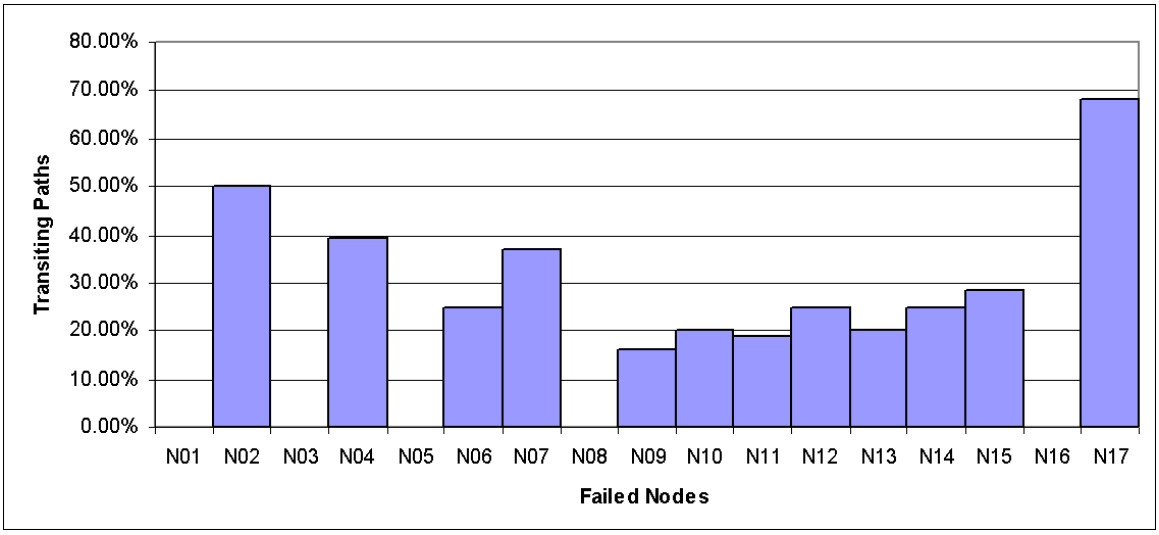


Figure 6.7: Percentage of demand at each node in the HAVANA Germany network that transits that node

affected transiting paths, or simply *transiting paths*. Figure 6.7 shows the percentage of paths at each node that are transiting paths (Figure provided by Diane Onguetou). 26% (69 out of 263) of all affected paths over all single node failures are transiting paths, meaning that at most 26% of the demand affected by all single node failures could be restored in a best-case scenario. Therefore the most useful metric for node failure restorability is to consider only the percentage of transiting affected paths that can be restored, as this value can theoretically always be increased to 100% by adding a suitable amount of protection.

6.2.9.3 Study Methodology

Node R_1 Calculation Script

Single node failure restorability was first calculated using an automated script, in a manner analogous to the method used for span R_2 calculation in Section 6.2.5. This script first makes a direct assignment of PXTs to protected working paths (as in the R_2 case, multiple assignments may be possible in the case of overprotection; the chosen assignment is arbitrary). It then iterates through each possible node failure sequentially, noting for each failure both the amount of demand affected by the failure and the amount of this demand that is restorable. At the end of the inspection process, the restorability fraction is calculated as the total number of restored paths divided by the total number of affected (transiting) paths.

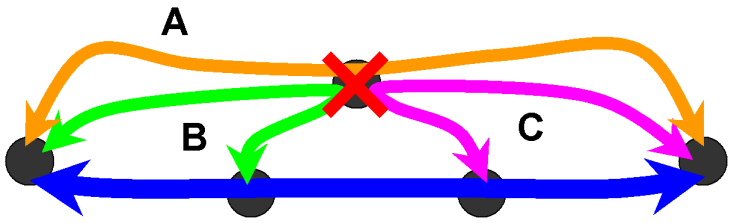


Figure 6.8: A type of PXT protection where the order of protection reactions will affect the amount of restored flow

In a network only designed to survive span failures, not all paths may be restorable in the face of a node failure. For PXTs, this may be caused by three separate factors. First of all, a node failure may cause contention between multiple failed paths that are all assigned to be protected by the same PXT. Secondly, a node failure may cause both the failure of a working path and the failure of the protection path on the PXT that would normally protect that demand. Both of these cases result in a network failure state in which some of the failed demand is unrestorable because of the insufficient allocation or structuring of PXT resources. The final case is the case in which the demand is fundamentally unrestorable because of the failure of one of its end nodes, as already discussed.

Note that, if more than one working path is in contention for the use of a PXT due to a node failure, assignment of protection by this script is done on a "first come, first served" basis, with no regard as to whether that assignment is optimal. This can result in sub-optimal results if 3 or more paths are in contention. Figure 6.8 illustrates a situation where this can occur. Assume working paths A, B, and C share a node (but not a span) in common. If this node fails and working path A is chosen for protection first, it will block out B and C, resulting in only 1 protected working path instead of 2. Unlike the span R_2 case, this problem cannot be solved by considering the failure event as two separate failures, ordered in time, because the failure of the node is considered to be a single event that affects all failed working paths simultaneously.

The node R_1 analysis script was run first on the initial low capacity PXT design generated in year 1 of the project (i.e., the minimum-cost design only, not considering optical path length restrictions or wavelength continuity constraints). It was then run also on the finalized 20-wavelength, 2000 km optical path length limited PXT design.

ILP Approach

Despite the possibility of situations shown in Figure 6.8, assuming this occurs with relative rarity, we expect the analysis script approach to closely approximate the maximum node failure restorability that can be attained. Furthermore, assuming the network is sufficiently "dumb", the "first come, first served" process is likely the best type of reaction we can expect from it in practice. For example, it would be much more operationally complex for the end nodes of multiple failed working paths to be able to communicate and organize themselves such that the greatest number of paths is restored in all cases. Therefore, the results obtained from this approach can be thought of as a good indication of *practically* attainable node failure restorability. However, to address any doubts concerning the optimality of the results obtained using the script, a more advanced method was also used to evaluate *optimally* obtainable node failure restorability. Under this method, a script again investigates the total affected and restored demand volume in each node failure scenario, except this time instead of assigning backup paths in the simple "first come, first served" iterative way, a small ILP problem is computed in order to find the optimal assignment in each case.

The following model is solved once for each combination of node failure and protected DRS to determine the optimal protection assignment for that DRS in the case of that node failure (avoiding the trivial cases where no working path in the DRS is affected by the node failure). Even though it is solved many times, the model is simple and has a minimal data set (the problem being local to a single DRS), and therefore solves extremely fast in each case; total solution time is negligible.

ILP Model

Sets:

S The set of spans in the network, indexed by i.

A The set of demands that are affected by the failure, indexed by r.

Parameters:

θ_r^i Encodes the protection path of demand r (in the DRS considered in this problem) as protected by the PXT under consideration. This value is 1 for span i if the protection path crosses that span and 0 otherwise.

n The number of copies allocated to the current cycle used to protect the current DRS. This is obtained from the value of the variable n_c^k from the PXT design model solution (see Section 4.3.2).

a_r The amount of protection offered by the current PXT to demand r in the current DRS. This value is calculated a priori to settle instances of over-protection, as described above.

Variables:

p_r Indicates the number of demand units of demand r that receive protection under this failure scenario.

Objective Function:

Maximize

$$\sum_{r \varepsilon A} p_r \tag{6.6}$$

Constraints:

$$p_r \leq a_r \quad \forall r \in A \tag{6.7}$$

$$\sum_{r \in A \, and \, \theta_r^i = 1} p_r \leq n \quad \forall i \in S \tag{6.8}$$

Note that the failed node is not represented explicitly in this model, but is represented implicitly by the combination of failed demands in the DRS (i.e., set A). Constraint 6.7 ensures that each demand will be assigned an amount of protection no greater than the amount given it by the PXT. The actual protection value in the solution may be less than this in the node failure case because of contention with other demands. Constraint 6.8 limits the total protection offered to demands that have overlapping protection paths to an amount no greater than the number of PXT copies allocated to the current DRS. It is this

constraint that controls contention due to multiple path failures. The objective function simply maximizes the total amount of protection. Summing up this objective function value over all of the individual problems for a given design will give us the total maximum number of restorable demand units.[4] The node R_1 analysis method based on this ILP model was used to check the results obtained using the naïve "first come, first served" script.

6.2.9.4 Results

Even though the analysis method needed to be greatly modified for us to be able to determine the absolute upper bound values for node R_1, the values obtained using the two methods were identical in all cases. This suggests that there is not much to be gained by pursuing absolutely optimal restoration behaviour in the case of node failures, and that the naïve "first come, first served" approach is adequate.

Over all single node failures, a total of 263 demand units were affected. Of these demands, 194 are fundamentally unrestorable, having the failed node as an end-node. This is twice the number total demand in the network (97), because each demand pair has 2 end nodes that will occur once each over all single node failures. Therefore only 69 out of the 263 affected demand units are potentially restorable. The two node R_1 analysis methods found that 51 of these 69 were restorable in both designs (both the original low-capacity design and the and the "constraints-feasible" design). This corresponds to a single node failure restorability value of 74%.

6.2.10 Design for Single Node Failure Restorability

We have seen that PXTs are indeed able to achieve a significantly large proportion of the theoretical maximum node failure restorability in networks designed only for restorability of single span failures. Given this fact, the question arises of how easy it is to move from this to 100% node failure restorability, either by augmenting existing designs or redesigning the network with modified design methods.

[4] That is, optimal given the protection assignment performed at the beginning of the script to obtain the values of a_r, but we have seen in the past that the choice of this assignment makes a negligible difference, e.g., in dual failure restoration.

6.2.10.1 Method

When designing for full node failure restorability, we would like to produce a design that is comparable to the existing reference design, one that uses the same set of candidate PXTs and DRSs, rather than creating a design that is totally distinct. This suggests an approach in which we augment the existing design rather than beginning again from scratch. However, the added node failure restorability constraint has the side effect that some DRSs that were considered in the initial design will become invalid (because some working paths in a DRS, while span disjoint, may not be node disjoint), and some PXTs that were considered for protection of a certain DRS will likewise become invalid (because it may share a node with one or more of the working paths in that DRS). Therefore it really makes more sense to begin the design process over from scratch, with the added node restorability constraint.

Initial Method: Modification of DRS/PXT data

Our initial approach was similar to the method used in Section 6.2.7.3 to create designs that satisfied the optical path length limitation requirement. Under this approach, the DRS and PXT sets used for the initial design problem are modified to screen out the invalid cases produced by the node restorability constraint. The result was a data set in which all paths in each DRS were node-disjoint and all PXTs were node-disjoint from their protected working paths.

This process is more complex than the optical path length case, because here there are many possible choices that can be made to modify the data to satisfy the new constraints. For example, if paths P_1 and P_2 in some DRS are not node-disjoint, the constraint may be satisfied by either removing P_1 or removing P_2 from the DRS (or both), but it is not clear a priori which choice is preferable. The script was written to remove working paths on a first-considered basis, i.e., the DRSs are scanned first for any conflicts with demand 1, and demand 1 is removed from the DRS in these cases. This continues on for demand 2, demand 3, etc. Then all PXTs are examined and the protection relationship parameter is changed to zero in any instances where a protection path and the protected working path would not be node disjoint.

This script was run on the same data set used to produce the original reference design. Unfortunately, the solver reported that the resulting problem was infeasible, because no

protection possibilities existed for some demands. Evidently this process removed too many protection relationships. Therefore another approach had to be found.

Secondary Method: Re-calculation of PXT and DRS Data Set

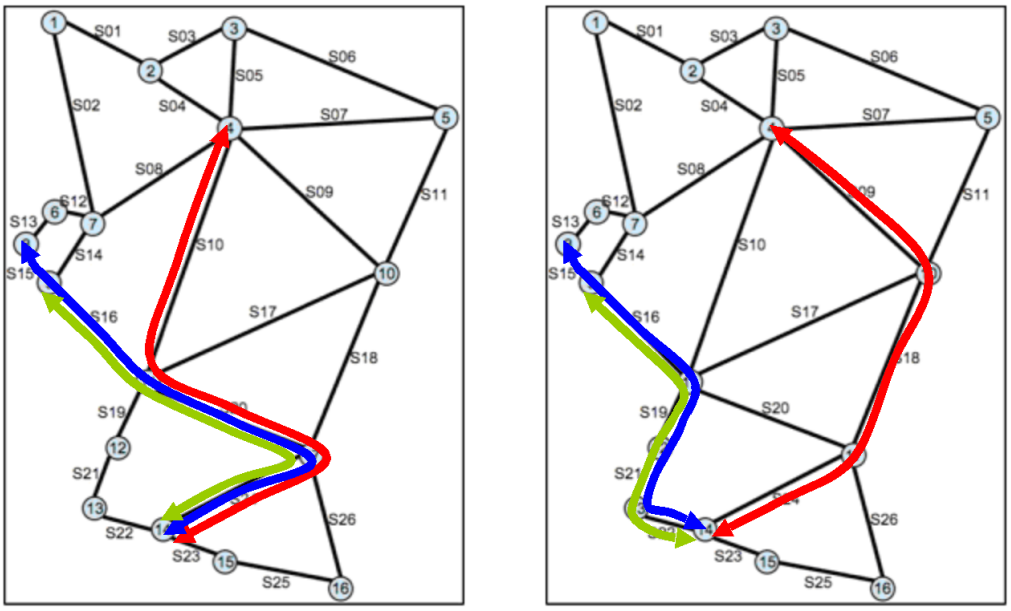
Because the existing data could not be modified to support node failure restoration, the only recourse was to re-compute the PXT and DRS sets from scratch, taking node disjointness into account from the very beginning. Adjusting the program to do just that is simply a matter of modifying the span-disjointness tests for DRSs and PXTs to node-disjointness tests instead. Unfortunately, recalculating the parameter sets removes any significant correlation between this test and the previous reference designs. Therefore 5 data sets were calculated this way in order to obtain any idea of the characteristic difference between the node- and span-restorability cases.

Unfortunately, all 5 of these cases reported infeasibility of the problem in AMPL (i.e., fundamental infeasibility of the constraint set). Furthermore, the infeasibility was because of the unrestorability of the same three demands in each case. Upon further inspection the reason was found to be obvious; these demands are routed along working paths such that, although other span-disjoint paths between the end nodes exist in the network, no *node*-disjoint paths exist, which is a requirement for the possibility of node failure restoration at all. Therefore, full single node failure restorability is not even possible without adjusting the working routing of the network. The working routes in question are shown in Figure 6.9 (a).

Finalized Method: Working Routing Adjustment and Data Set Re-calculation

Accounting for the above discovery requires that we depart even further from the initial reference designs. To enable full node R_1, the demands in Figure 6.9 were re-routed along the shortest routes for which there also existed at least one alternate node-disjoint path between the end-nodes (recall that the original routing only guaranteed the existence of a span-disjoint path). This resulted in an increase in working capacity; total working capacity increased from 166 to 168. The modified working routes for the three problematic demands are illustrated in Figure 6.9 (b).

After this modification was made, the data set was again re-computed 5 times, as in our



(a) Working paths in original design with no possible node-disjoint protection paths

(b) Rerouting of original working paths to allow node failure restorability

Figure 6.9: Illustration of the need for working path rerouting to enable full node failure restorability

previous method. Note that we cannot simply use the same DRSs and PXTs as in the reference design (as in our initial method), as that data is all dependent on the specific demand routing that was used initially and found to be insufficient for node failure restorability. Our 5 data sets were then used to create 10 single node failure restorable designs: 5 regular designs and 5 designs under the 2000 km reach limit. We also tried to create 5 designs under both the reach limit and 20 wavelength limit, but these designs were found to be infeasible. The node failure restorability constraint simply requires too much capacity for the design to fit under the 20 wavelength channels per-span limit.

6.2.10.2 Results

The 5 "regular" single node failure restorable designs had spare capacity costs of 287, 290, 290, 286, and 287 respectively, an average of 288. Combined with a working capacity cost of 168, this means they had an average total cost of 456. The original design had a total cost of 375 (166 working and 209 spare). This is an average cost increase of 22% to go from 74% to 100% node failure restorability.

The costs of the 5 optical path length-constrained designs were 287, 291, 292, 288, and

287, an average of 289. The average total cost is 457. The cost of the original (length and wavelength constrained) design was 386 (166 working and 220 spare). This is a cost increase of 18%, again to go from 74% to 100% single node failure restorability.

All designs were checked by hand to ensure that they were able to attain 100% single node failure restorability. The observed cost increases are understandable considering that we have to protect an additional 18 out of the 69 transiting path failures; 35% more paths are protected for an approximate cost increase of 20%.

6.2.11 Cost Modeling

Up until this point, we have been using spare capacity (or total capacity) as a surrogate for real network cost. As explained in Section 6.2.6, the "hop-based" capacity cost model we have been using assumes that network costs will be dominated by per-span setup costs for lightpaths (costs that must be paid regardless of span length). However, in reality network costs will contain some mixture of hop-based and distance-based considerations, in addition to other components that are not related directly to wavelength channel count (e.g., path-terminating equipment, whole fibre port costs, etc.).

The purpose of the work in this Section was to replace our basic wavelength-channel based cost model with a more realistic one and to determine the effects this would have on our observations regarding cost. Because we are modeling WDM transport networks, the new model needed to apply to the WDM layer. We decided to use the "NOBEL cost model" from [GuLe06], at the recommendation of our NSN colleagues. The advantage of using a standardized model is that it gives us some confidence that the choice of relative cost values is realistic, and therefore that the conclusions we can draw from the cost results in the following Section can tell us something about how to design real networks using these architectures.

After identifying the model, our next step was to identify and execute architecture-specific approaches for reducing cost under this model, and to study these results comparatively between all architectures. Under guidance from NSN, we then made some modifications to the model to better represent the type of protection switching they support, and studied the effect of these changes on the results.

6.2.11.1 The NOBEL Cost Model

The NOBEL cost model is a cost model for the WDM layer recently developed by the European NOBEL project. It describes a set of normalized CAPEX costs for the span equipment, node architectures, and transmission equipment required by a WDM network.[5] All costs in the model are normalized to that of a single 10G transponder. The collaborators on the NOBEL project worked together to develop a set of normalized costs that would be accurate enough to be realistic, but general enough to apply across a variety of different vendor equipment. This model was used for the study because it was both standardized and freely available. A model based on NSN-specific equipment would, of course, produce results more relevant to networks based on NSN equipment, but at the expense of preventing us from forming more general conclusions about the cost of protection in other types of WDM networks.

The tables of model components and corresponding costs from [GuLe06] are reproduced here as Table 6.3 (trimmed to contain only the components used for this study). MTD refers to the *Maximum Transmission Distance* that the equipment is capable of. Equipment of different MTD cannot be mixed and matched within the same path (e.g., a 750 km transponder card may use different encoding from a 1500 km card). Equipment that is used by paths of a variety of lengths (e.g., many different paths may use the same in-line amplifier or segment of dispersion-compensating fibre) must have an MTD large enough to accommodate the longest such path.

The paper also contains an example of the implementation of an optical path using the components of the model, shown in Figure 6.10. An optical path consists of a sequence of optical links connected by OADM/OXC devices. These links are made up of a sequence of optical spans (distinct from the network topological sense of the term "span"), each with an in-line amplifier and length of dispersion-compensating fibre (DCF). The model also requires a dynamic gain equalizer (DGE) every four spans. Each transparent OADM/OXC requires a transparent node amplifier at the ingress/egress of every fibre port. Paths are added/dropped at each OXC via tunable 10G transponder cards. Further switching of the

[5] A second version of the NOBEL model has since been published in [HuGu08]. This is a multi-layer model that also takes into account the cost of elements in the IP/MPLS layer, the Ethernet layer, and the SDH/OTN layer. Because our focus was on the WDM layer only, we did not consider this new version of the model.

Cost per...	10G Transponder Card	In-line amplifier	Dispersion compensating fibre (per 80 km span)
MTD = 750 km	**1**	3	0.9
MTD = 1500 km	1.4	3.8	1
MTD = 3000 km	1.9	4.7	1.2

(a) Reach dependent equipment

Cost per...	OADM (2 fibre ports)	OXC ($N = 3$ to 5 fibre ports)	OXC ($N = 6$ to 10 fibre ports)
40 channels	11.8	$5.35 \cdot N + 2$	$5.85 \cdot N + 2$

(b) Capacity dependent equipment

Cost per...	Transparent Node Amplifier	Dynamic Gain Equalizer (every fourth amplifier site)	10G equivalent EXC switch port
Single unit	1.25	3	0.28

(c) Other equipment

Table 6.3: Costs of relevant components in the NOBEL model from [GuLe06]

electronic signals may occur between the OXC and the client, but this is not shown here.

With regards to switching equipment, the baseline node equipment model is given in Figure 6.11. An arbitrary node consists of a single OADM/OXC interfaced to an *Electronic Cross-Connect* (EXC). Some of the transponders interfaced with the add/drop ports of the OXC are exposed directly to the client, and are only switched optically. Others are interfaced intermediately to the EXC, which allows both electronic and optical switching of signals (more on this in the discussion of protection switching). A given node may not have an EXC at all if no electronic switching is required. The difference between the presence and absence of the EXC under this model is irrelevant in any case, as an EXC is priced incrementally according to the number of ports required, with a zero base cost. Therefore in the following discussion we are only concerned with the number of ports used at each node.

As for protection structures, protection switching, and the transmission equipment required to support working and protection paths in tandem, the NOBEL paper does not cover resiliency and therefore does not give standard or recommended models of protection-

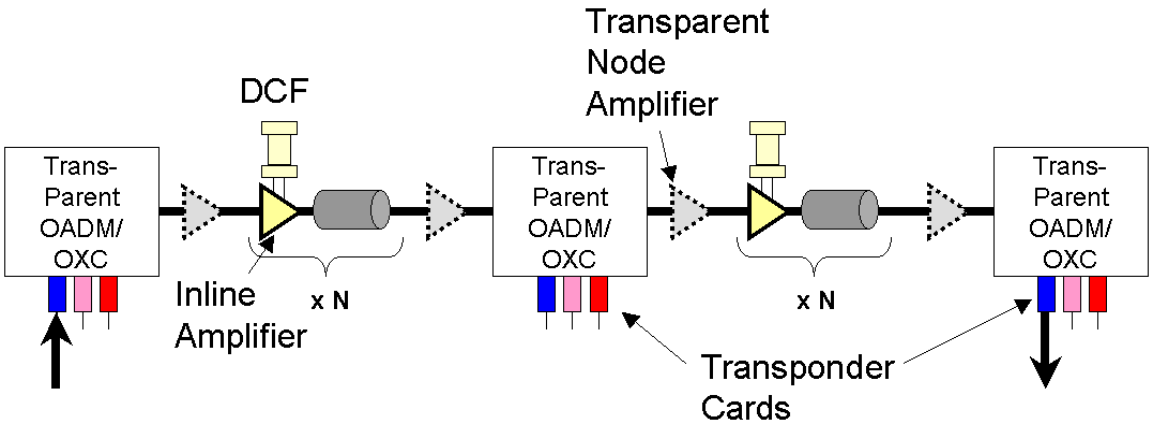


Figure 6.10: Elements of an optical path under the NOBEL model

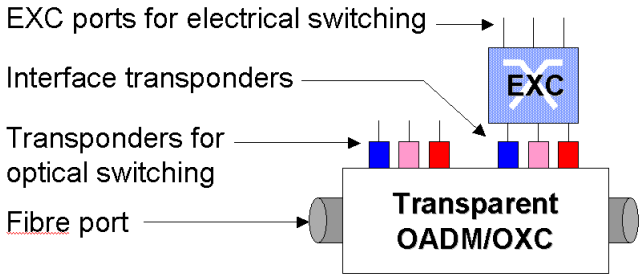
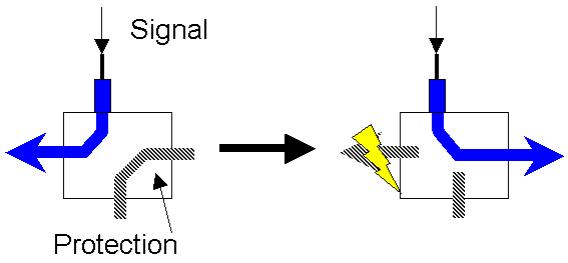


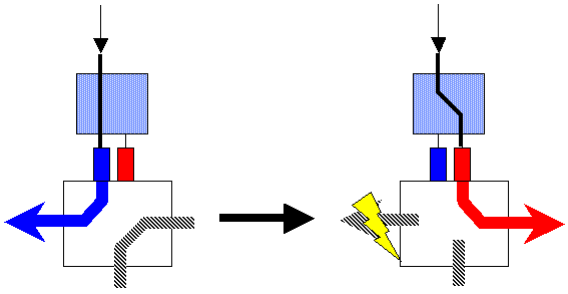
Figure 6.11: Baseline node architecture for the NOBEL cost model study

related items. Therefore part of the cost evaluation exercise was to first establish our own model of protection switching using NOBEL components. We have already mentioned that each node contains the capability for either switching signals electronically or optically (or both, if required). Therefore we considered two different types of protection switching: optical and electronic. Optical protection switching utilizes the OXC to switch a signal output from a single transponder between two different end-to-end concatenated wavelength paths. Electronic switching switches the signal in the electronic domain from one transponder to another. Illustrations of optical and electronic switching using the baseline node architecture from Figure 6.11 are given in Figure 6.12 (a) and (b) respectively.

Because optical switching does not change the transponder being used for the path, it implies that working and protection are transmitted on the same wavelength (of the tunable transponder card). Electronic switching, on the other hand, has the freedom of being able to use different wavelengths for the working and protection paths if necessary, because the signal is routed to an entirely new transponder via the EXC. However, we pay for the flexibility of electronic protection switching with higher cost. From Figure 6.12, we can see that



(a) Optical switching



(b) Electronic switching

Figure 6.12: Implementations of optical and electronic protection switching using NOBEL model components

optical switching requires only a single transponder (per end-node of the lightpath) in addition to the standard node switching equipment. Electronic switching, however, requires use of 3 ports on the EXC (one client-side and 2 for the working and protection transponder), and two transponders. Assuming 750 km MTD equipment, a transponder has a cost of 1 and a port on the EXC costs 0.28. Therefore the cost is 1 for optical switching, compared to 2.84 for electronic switching, a nearly threefold cost increase. Considering that a network may contain an aggregate of hundreds of these protected wavelength paths, this can add up to a large difference in cost. Therefore we would prefer to use optical switching wherever possible to reduce costs.

6.2.11.2 Method

Producing models for all of the pieces of a restorable network design gave us a method of representing our restorable network designs in the context of the NOBEL cost model. The next step was to tabulate equipment requirements for our network designs to generate some initial cost results. This Section covers this process for PXTs only; the same process was performed by others for the other architectures, but we will only present the final results for comparative purposes. We would have liked to use the reference "best feasible" reference

designs (described in Section 6.2.7.4), as the intention was to carry them forward to further stages of the project, but we discovered some minor modifications that needed to be made first so that cost evaluation on them would make sense.

Path Lengths

The path length limitation investigation in Section 6.2.7 used 2000 km as the limit on the equivalent length of the optical paths. However, we have seen that the NOBEL model describes equipment with MTD capabilities up to 3000 km, with divisions at 750 km and 1500 km. Therefore it does not make sense to perform cost evaluation on designs artificially limited to 2000 km; we should either allow the design to use up to 3000 km paths, or limit the design to 1500 km to achieve cost savings through MTD reduction. Therefore we did not use the path length limited PXT designs, using instead the previous designs with no limits on path lengths.

Wavelength Assignment

Many features of the cost model are wavelength-dependent (e.g., the cost difference between same- and different-wavelength protection), so we needed to use a design with wavelengths assigned to all paths for this investigation. We could have used one of the designs generated in the wavelength assignment portion of the HAVANA study (see Section 6.2.6), but we immediately observed that these assignments (both for 20 and 40 wavelengths) were designed under conditions that did not have cost reduction in mind, and that could be improved by simple adjustments to the ILP model. More specifically, the wavelength assignment ILP model used to generate those results used an objective function that minimized the total number of different wavelengths used in the network, a metric which should have no direct impact on cost under this model. On the other hand, we have observed that there is a direct cost benefit to using optical protection switching wherever possible. Because this requires the working and protection paths to use the same wavelength, we decided we would do better to find a wavelength assignment that could maximize the amount of same-wavelength protection in the network.

To do so, we made the following modifications to the ILP model described in Section 6.2.6.1:

Modified ILP Model:

New Parameters:

o_q^p This binary parameter is equal to 1 iff q is a PXT, p is a working path, and q protects p. Essentially, it encodes our preference as to whether paths p and q should use the same wavelength and be counted towards the amount of same-wavelength protection used in the network (to be maximized in the objective function).

New Variables:

d_q^p This binary variable is 1 iff paths p and q use different wavelengths and 0 otherwise. It is defined over only all p,q such that $o_q^p = 1$, because these are the only paths whose wavelength relationship we are interested in.

Modified Objective Function:

Minimize

$$\sum_{p,q \varepsilon P \, s.t. \, o_q^p = 1} d_q^p \tag{6.9}$$

New Constraints:

$$d_q^p \geq u_w^p - u_w^q \quad \forall p,q \in P, \forall w \in W \tag{6.10}$$

$$d_q^p \geq u_w^q - u_w^p \quad \forall p,q \in P, \forall w \in W \tag{6.11}$$

The objective function simply minimizes the number of working path/PXT pairs using different-wavelength protection (maximizing the number of cases where optical protection switching can be used). Constraints 6.10 and 6.11 differ only in the order of subtraction on the right hand side. These constraints set the different wavelength protection variable d_q^p according to which wavelengths are used by the paths p and q. If, neither path uses a given w, the constraint reduces to $d_q^p \geq 0$. If, however, path p uses wavelength w and path q does not, Equation 6.10 reduces to $d_q^p \geq 1$, and 6.11 reduces to $d_q^p \geq -1$. If path q uses a wavelength that p does not, we see the two same resultant inequalities, albeit reversed. In both cases, d_q^p, being binary, is set to 1.

This model was used to generate another 40 wavelength solution to the wavelength assignment problem, this time with same-wavelength protection maximized. The resulting design had 77 working paths (out of 97, approximately 80%) using same-wavelength protection.

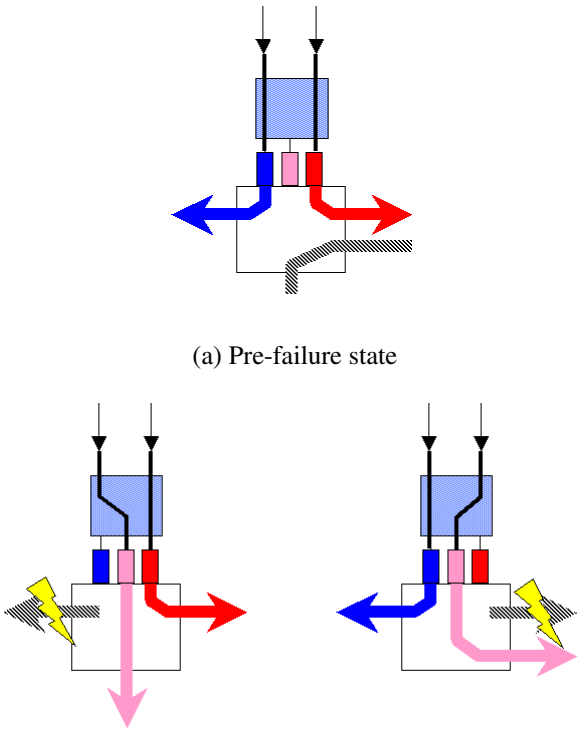
Equipment Enumeration Method

After finalizing the design that would be used for cost evaluation, we were then able to enumerate the required equipment based on the NOBEL model and our own models for protection switching. At this point we must mention some final additional assumptions that were used specifically in the case of PXTs. In certain special cases in which there are two (or more) working paths that terminate at the same node, and both paths use electronic protection switching, we assumed that the paths can both share the transponder that is used to access the PXT, i.e., only one PXT access transponder is required in total. This situation is illustrated in Figure 6.13. The EXC controls access to the PXT by switching client signals to the access transponder. If the access transponder needs to transmit in different directions depending on which way the protection path is formed in the PXT, we can use the switching functionality of the OXC to switch transmission to the appropriate fibre port. This is a rare situation, and has only a small effect on cost, but should be mentioned nonetheless.

NOBEL costs were calculated via two methods. First, equipment was tabulated manually using a spreadsheet to perform many of the calculations (e.g., longest path crossing a span that sets equipment MTD, transponder requirements for working paths and protection switching, etc.). Then the expertise developed by this process was used to develop an automated cost calculation script run entirely from within AMPL on the solution values produced by the CPLEX solver. The script was tested to ensure that it produced the same results as those obtained by inspection via the spreadsheet.

6.2.11.3 Initial Results

The initial results of this exercise are presented in Figure 6.14, compared with costs for the other architectures. Total costs are broken down into node costs (cost of OXCs, including fibre ports, and transparent node amplifiers), span costs (amplifiers, DCF, etc. required to support the optical links), and transmission costs (cost of transponders and EXC ports). For



(a) Pre-failure state

(b) Failure 1

(c) Failure 2

Figure 6.13: The sharing of protection access transponders in path-protecting shared architectures

all of the following cost modeling exercises, results for APS and DSP were produced by Brian Forst (see also [Fors09]), the results for *p*-cycles were produced by Diane Onguetou, and the results for FIPP *p*-cycles were produced by Dimitri Baloukov.

Note that there were many different cost results produced for the other architectures under many different conditions, in an attempt to find their own strategies for cost reduction (this process for PXTs will be outlined in the following Sections). The conditions under which the specific costs presented in Figure 6.14 were obtained will not be described here in detail, as it would require too much discussion that would be tangential to the subject at hand (PXTs). Rest assured that these results are comparable to our initial PXT results in that they are subject to similar network conditions (e.g., 3000 km maximum path length), although the design methods have been varied from traditional approaches in some cases to produce results that are more characteristic of minimum cost designs under the parameters of this particular test network.

The most immediately interesting feature of Figure 6.14 is that the cost of PXTs (and

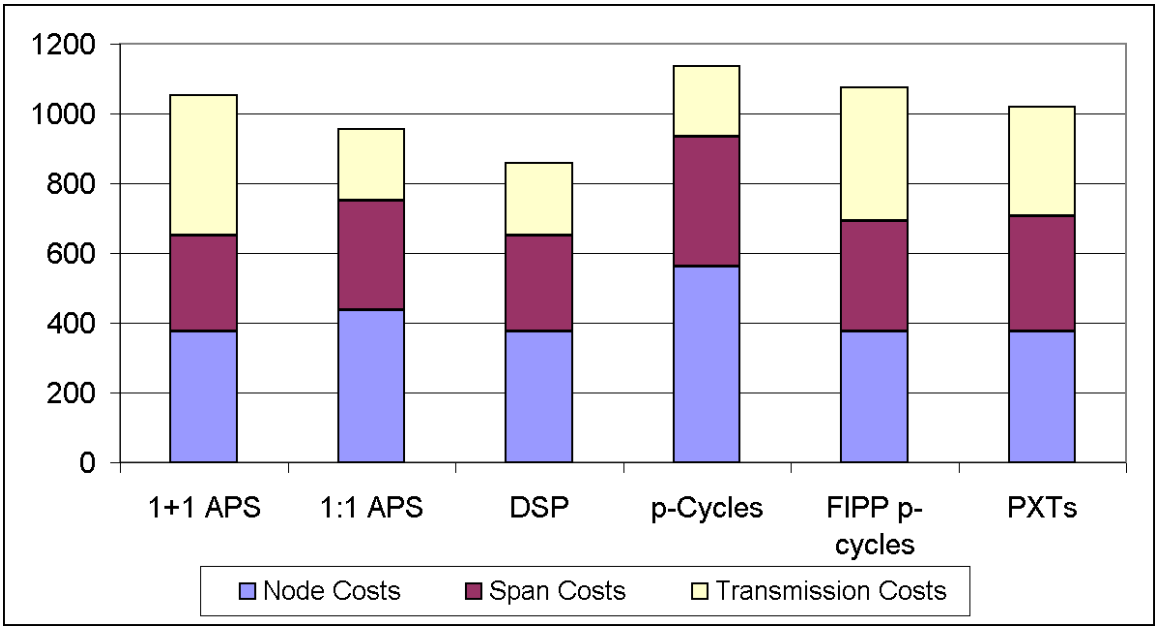


Figure 6.14: Initial NOBEL cost comparison between PXTs and other HAVANA architectures

indeed the other shared architectures as well) is actually greater than that for some of the dedicated protection architectures (1:1 APS and DSP). This development was initially a shocking one, as the results seemed to contradict the widely held assumption in the field that shared protection architectures will enable the deployment of more cost-effective networks. However, even though DSP uses a significantly greater number of spare wavelengths for its design, the shared architectures are more expensive by a noticeable margin.

Focusing on PXTs in particular, we can begin to understand why this is so. Figure 6.14 shows that there is no difference in the node costs; both designs use the same number of OXCs and OADMs at the same nodes, because the number of fibres and fibre ports in both designs are identical at every node (1 fibre per span). This observation highlights an important fact: under the NOBEL model, the fibre is the fundamental cost-setting component for both node and span costs. Node costs increase incrementally with the number of fibre ports, and span equipment is allocated on a per-fibre bases, serving all of the wavelengths in that fibre. As long as the level of load is low enough, the fibre-level granularity of the cost model will essentially cover up the differences in wavelength usage and "level the playing field" between shared and dedicated protection, from a cost perspective.

Looking at the differences in span and transmission costs reveals the true cost differ-

entiators for a single-fibre network in the NOBEL model. Given that the same amount of span equipment is required by both DSP and PXT designs, the only cost difference is in the MTD of the equipment used, and here PXTs are at a disadvantage. Shared protection achieves savings by detouring protection paths in intelligent ways such that the possibilities for sharing are maximized; if sharing were not possible, we might as well use shortest-routed protection paths, as with APS. PXTs, then, naturally uses longer protection paths than DSP. Because the longer paths cross more spans, and the MTD of span equipment is determined by the longest path to cross that span, most of the network spans end up having MTD requirements near that of the longest path in the network, which for this PXT design is very near the limit of 3000 km. DSP, on the other hand, with its working and protection routing that is near-shortest-path, requires only MTD 1500 km equipment, achieving savings on this front over PXTs.

A similar reasoning applies to transmission costs. The number of transponders in a design is bounded from below by the number of paths. This bound is constant across all architectures, being independent even from the number of fibres. Designs will exceed this bound depending on the amount of different-wavelength protection (because of the need for an access transponder of a different wavelength). DSP has a double advantage here; because sharing only occurs between multiple units of the same demand, DSP can achieve 100% same-wavelength protection. However, as described in Section 6.2.11.2, PXTs can only achieve about 80% same-wavelength protection, meaning more transponders are used. Then, in addition, because protection paths in PXT designs are long, the MTD requirements for the transponders are higher, increasing costs even further.

This concept was illustrated concretely by an exercise that was performed on the PXT design during the cost evaluation process. It was observed, while developing the cost tabulation spreadsheet, that the length of one of the protection paths in the network actually exceeded 3000 km by a small amount. This was a mistake, as it was simply assumed that no path in the original (unrestricted path length) design exceeded 3000 km. To fix this, the working path that used this overlong protection path was "split off" from its protecting PXT, had its wavelength changed, and was given a new PXT to protect it end-to-end in a dedicated manner (essentially a 1:1 APS arrangement). Both the original PXT and the new one used same-wavelength protection for this demand, so no access transponder

was required in either case. This splitting exercise saw the design cost *decrease* by 1.8 because the length of the protection path was now below 750 km and therefore the costs of the transponders at both endpoints of the path were reduced by 0.9 (from 1.9 to 1.0). The increase in wavelength utilization (because of the creation of the new PXT) has no effect on cost.

In retrospect, then, it should be obvious that our traditional capacity minimization shared protection approaches are not suited to lightly loaded networks under the NOBEL model, because the high modularity of fibre-level capacity elements hides the impact of wavelength loading on costs, causing other effects (MTD and optical protection switching) to become the cost-determining factors. These results also agree with the results of similar work on the CAPEX costs of path protection [StCo08], published during the course of our study, that show a similar minor difference between the costs of shared and dedicated protection in the transparent case. In our case, we can see that this is because most of the cost is simply invested in the fixed "start-up" equipment for the fibre infrastructure, and therefore has very little variability. An additional exercise calculating the cost of an unprotected network (i.e., working paths only) showed that this fixed initial investment was upwards of 95% of the cost of the DSP design; the 5% increment is only due to increased MTD requirements for upgrading some transponders to support the longer protection paths. The variability in this cost increment will tend to favour the architecture with shorter paths (to reduce MTD), and fewer wavelength dependencies between paths (so as to enable maximum same-wavelength protection). This provides a clue towards developing methods to reduce costs for PXTs under the NOBEL model, which we will discuss in the following Section.

Note that even though the shared architectures are more expensive, they still have an advantage in terms of using up less of an essential network resource: capacity. This is beneficial when considering demand growth, for example; the shared architectures will be able to sustain more growth before provisioning new fibres or upgrading transmission rates (e.g., from 10G to 40G). The trade-off in this case is therefore one of increased cost outlay now vs. expansion cost later. The higher cost of shared protection can therefore be considered not as a liability, but as an investment.

6.2.11.4 Cost Improvement

After determining that short paths and same-wavelength protection were the key to low cost, our next step was to apply these properties to PXT designs. As explained in Section 6.2.11.2, we already optimize the same-wavelength protection in our designs before evaluating costs. Therefore the obvious approach to reduce costs further is to limit path lengths. Here we run into a conceptual difficulty that has the potential to reduce the problem to absurdity: APS is theoretically a special case of PXTs (if all PXTs protect a single working route), and APS achieves 100% same-wavelength protection as well as shortest-cycle working and protection routes, so therefore the lowest cost PXT design under this framework should be simply the APS design.

Obviously this reduction does not produce any insights of value into PXTs. Instead, we took a halfway approach by limiting the length of PXTs in the design while at the same time maintaining "true" PXT characteristics like spare capacity sharing. This was done in exactly the same way as in Section 6.2.7.3, by eliminating protection relationships in the ILP problem data file where protection paths would be too long. This time, the limit was set to 1500 km, so that we could reduce the MTD of all equipment in the network to at most the 1500 km category. The new total costs are illustrated in Figure 6.15, along with those for APS and DSP as before, as well as new costs for FIPP *p*-cycles, which are also able to take advantage of the path length limitation strategy.

The Figure shows that path length limitation is an effective way of reducing cost for both FIPP *p*-cycles and PXTs. Now the span costs are also equal across DSP, FIPP *p*-cycle, and PXT designs, in addition to node costs (as before). This is because all spans now use 1500 km MTD equipment in all three designs. However, DSP still has an advantage in terms of transmission costs, because it still uses fewer transponders with lower MTD. Even though all *span* equipment is 1500 MTD, DSP still has more paths that are short enough to fall into the 750 km MTD category. In addition, FIPP *p*-cycles and PXTs are still not able to achieve 100% optical protection switching (the sharing relationships that impose constraints on the wavelength assignment are not affected directly by the reduction of path lengths).

The exercise was repeated again to try to reduce all paths below 750 km, but this results in an infeasible problem because there are some working paths in the network for which the

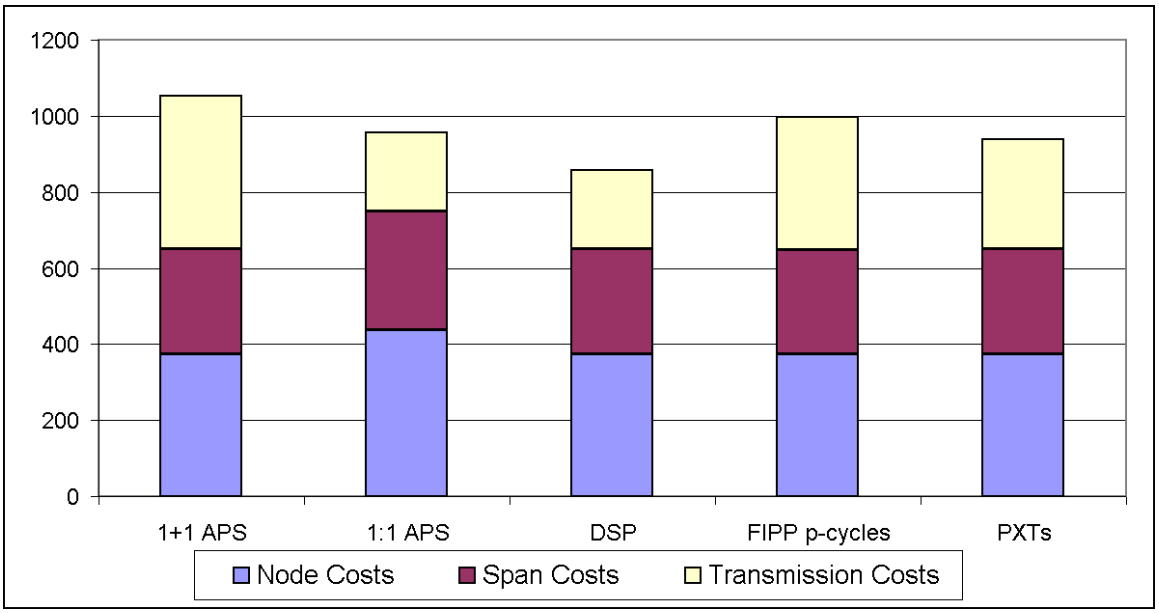


Figure 6.15: NOBEL cost comparison between dedicated architectures, FIPP *p*-cycles and PXTs under 1500 km path length constraint

shortest possible protection path is longer than 750 km. Therefore we stopped our efforts to reduce cost at this point. In any case, the results of this exercise prompted our colleagues at Nokia Siemens Networks to propose some alterations to the cost model such that it would more closely correspond to their own networking solutions.

6.2.11.5 Effect of Demand Scaling

Our observations about the cost determining factors for this combination of cost model and test case suggest that we might see a dominance of shared architectures over dedicated ones under more highly loaded network conditions. More demand would require more working and spare capacity, pushing the capacity requirements on many spans into the multiple fibre region, which would multiply the cost of span and node equipment (due to more fibre ports on the OXCs). Shared architectures would be at an advantage, because they would use fewer fibres. To test this prediction, we compared DSP and PXT designs under a demand scaling factor of 10 (i.e., all demands in the network multiplied by a factor of 10). We solved for an optimal DSP design and a 1500 km path length limited PXT design under this demand pattern and then evaluated their costs under the NOBEL model. We assumed 40 wavelengths per fibre, as before.

Unfortunately, the cost calculation process in the multi-fibre case is not so simple, for several reasons. First of all, because we can no longer assume just one fibre on all spans, we must multiply the number of available wavelengths on all span to accommodate 10 times the number of paths. This makes the wavelength assignment ILP extraordinarily more complex; in our case, it was found to be essentially intractable. Secondly, the large number of fibres means that many nodes now have a requirement for more than the maximum of 10 fibre ports specified in the NOBEL model, so we need multiple OXCs and/or OADMs at these nodes. This introduces additional requirements on the routing of paths, as we must make sure that every path transiting a node does so on the same OXC. If we avoid this by taking into account interconnection of OXCs, we must incorporate the additional costs for doing so into the model. Finally, calculating the "true" cost of such a design is difficult because the MTD of the equipment for a fibre is defined by the longest path that traverses it, and for any given span we will have a combinatorial number of choices as to how to route the many different paths that cross that span within several different fibres. Calculating how to best do this for all spans in the network is an optimization problem in itself.

Due to these issues, we were not able to calculate precise design costs. Instead we calculated approximate costs under the following assumptions:

1. Optical protection switching is used everywhere.

2. The MTD for the span equipment of *all fibres* on a span is defined by the longest path that crosses that span.

3. A node with multiple OXCs/OADMs has the capability to switch any path from any fibre port to any other fibre port (i.e., we assume OXC interconnection "for free").

The first assumption is required because we cannot solve the wavelength assignment problem (or at least, not in any optimal sense that gives meaning to the amount of optical protection switching that results), so we simply assume the best case. Based on past experience, this is being generous to PXTs, because sharing means that we have less flexibility for wavelength assignment. However, if the pool of overall available wavelengths increases (as it must when fibres are multiplied), finding same-wavelength protection will become easier, so this is not totally unrealistic. The second assumption is to avoid the fibre-specific

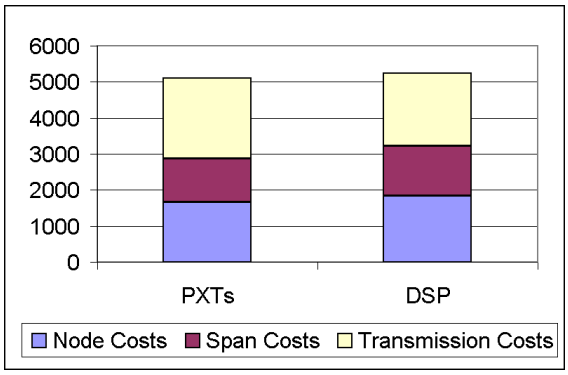


Figure 6.16: NOBEL cost comparison between DSP and PXTs with tenfold demand scaling

routing problem, and is conservative for both architectures, although likely more so for PXTs, because PXT designs tend to have a larger spread of path lengths, and thus it is more likely that we would be able to intelligently "pack" certain fibres with short paths to reduce MTD on those fibres. The third assumption is also somewhat realistic, as local OXC interconnections could use cheap, short-range optics. Therefore we have reason to believe that the results of this investigation, while not perfectly accurate, should still be representative for the high-demand, multi-fibre case.

A comparative cost breakdown of the two designs is given in Figure 6.16. As the Figure shows, PXTs are able to achieve lower cost than DSP in this case and under the assumptions listed above. Node and span costs are lower for PXTs, as predicted, simply because of the lower number of fibres and fibre ports; the DSP design uses a total of 125 fibres, whereas the PXT design uses only 113. Transmission costs are higher for PXTs because the protection paths are longer, and therefore the transponders require a higher MTD in more cases, as expected. However, this is not enough to offset the savings due to fibre reduction.

These results support the predicted correlation between demand volume and lower relative costs for shared architectures. We would expect this margin to only increase with higher demand as the modularity of a single fibre becomes less of a contributing factor. Also, this analysis does not take into account any modularity-aware design methods that could more efficiently pack wavelengths into fibres and could therefore potentially reduce the fibre requirements of PXTs as compared to DSP even further. Therefore these results may be conservative for PXTs.

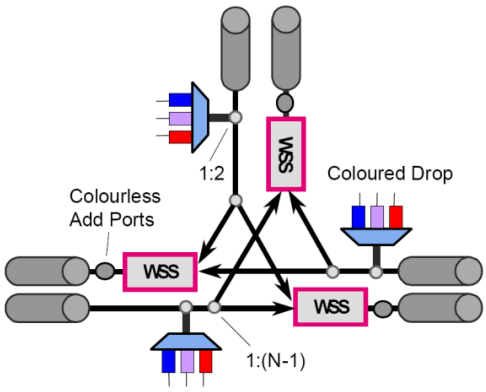


Figure 6.17: An OXC based on the WSS architecture

6.2.11.6 Effect of Client Protection Switching

Communications with Nokia Siemens Networks regarding the NOBEL cost model results revealed that one of our assumptions about our optical switching capability was in fact incorrect, or at least not in accordance with the capabilities of NSN networking equipment. Specifically, our colleagues pointed out that freely available optical switching of added/dropped paths (i.e., optical protection switching of client signals at path end-nodes) is not possible under the WSS (wavelength selective switch) architecture for OXCs. An illustration of a 3-port OXC using the WSS architecture, taken from [GuLe06], is given in Figure 6.17. The important point is that under this architecture, the colourless add ports are *after* the WSS, "pointing" directly into the fibre. This means that we cannot use the capability of the WSS to perform protection switching, because added signals enter the fibre immediately.

This greatly affects our assumptions about low cost design under this model, because it means that cost reduction by using optical protection switching is not possible. Instead, a form of electronic switching must be used for all paths, regardless of whether or not the wavelength of the backup path is the same. The NSN team suggested modeling this switching capability as a set of client protection switching cards (PSC) at each node that interface between the client optical signal and the OXC. A diagram of this proposed arrangement is shown in Figure 6.18. The idea is that a node would have a bank of these PSC cards, one for each client signal to be added/dropped at that node. Note that this is slightly different from the EXC implementation, because a bank of PSC cards would only have the flexibility to switch each client signal between 2 paths, as opposed to the flexibility of a full EXC switch

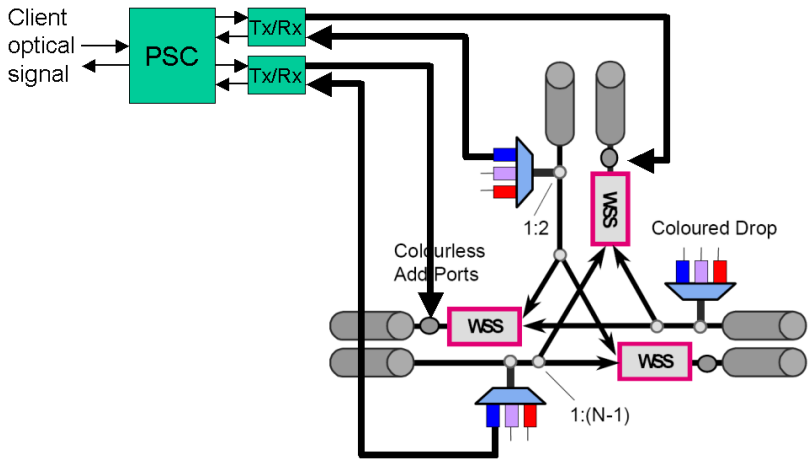


Figure 6.18: A PSC interfaced to a WSS-based OXC

matrix (which would be unnecessary in this situation in any case).

The cost model for the set of PSC cards at a node consists of a base cost for the card "rack" and one initial card and an incremental cost for any more cards in addition to the first. We assumed a 10% cost increment per card, giving the following expression for PSC costs at a node:

$$C \cdot (1 + (N-1) \cdot 0.1)$$

Here, N is the number of cards at a node. For this investigation we assumed $C = 1$, which NSN considered to be roughly accurate. We also assumed that all cards at a node (across any number of different clients) are grouped together into one rack to amortize the base cost C (as opposed to different clients using their own separate racks).

This new model adds a small wrinkle to our protection switching model in the special case where protection transponders are shared (e.g., the case shown in Figure 6.13 for the original NOBEL model). For the unmodified NOBEL model we could rely on the EXC switch fabric to control access to the shared protection transponder, but using PSC cards we require additional equipment to do this. The example in Figure 6.18 shows the basic case with one working transponder and one protection transponder, but if two client signals need the option of using the same protection transponder, we need an arrangement like that shown in Figure 6.19. Here, another PSC is "cascaded" with the two client-exposed PSCs and interfaced with the protection transponder so that either client PSC can access the protection transponder depending on which working path fails. The alarm lines of

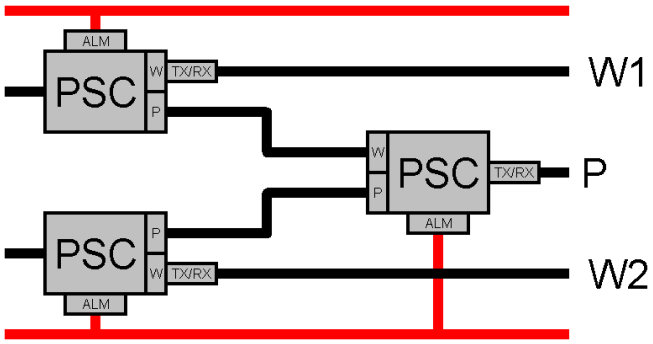


Figure 6.19: Cascade arrangement of PSC for protection transponder sharing

W1 and W2 are connected such that if W1 fails but W2 does not, the protection PSC will be switched to its W port and the W1 PSC will access the transponder. If W2 fails and W1 does not, the protection PSC will be switched to its P port and the W2 PSC will access the transponder. If both fail, then W2 will still access the transponder and W1 will be unrecoverable. The orientation of the protection PSC can be flipped vertically in this diagram to change which path has priority in a dual failure scenario.

Larger cascade arrangements are also possible when the number of working paths sharing the protection transponder is 3, 4, etc. In general, the number of cascaded cards will be equal to $2N-1$ where N is the number of client signals that need to share (so $N-1$ more cards then if sharing were not required). This consideration adds a small cost increase for instances of protection transponder sharing, but does not have an overall large contribution in cost, because situations like this are rare in PXT designs and the cost increment is only 0.1 for every card. So, for example, even if 5 cards needed to share (which never happens in our designs), the added cost would only be $4 \cdot 0.1 = 0.4$, in designs where cost values are on the order of 1000. However, this is still an important implementation detail.

The costs for our network designs under this modified design model are illustrated in Figure 6.20. As stated, this model forces electronic protection switching to be used everywhere, reducing cost variability even further. The only remaining cost variability factor is essentially transponder MTD, meaning that the architecture with the shortest paths is the cheapest. Even then, the possible variability is low, given that none of these designs use equipment with MTD greater than 1500 km, yet topological constraints mean that it is not possible for many paths to be shorter than 750 km. Figure 6.20 shows that node and span costs are the same in all cases, with a minor variation in transmission costs due

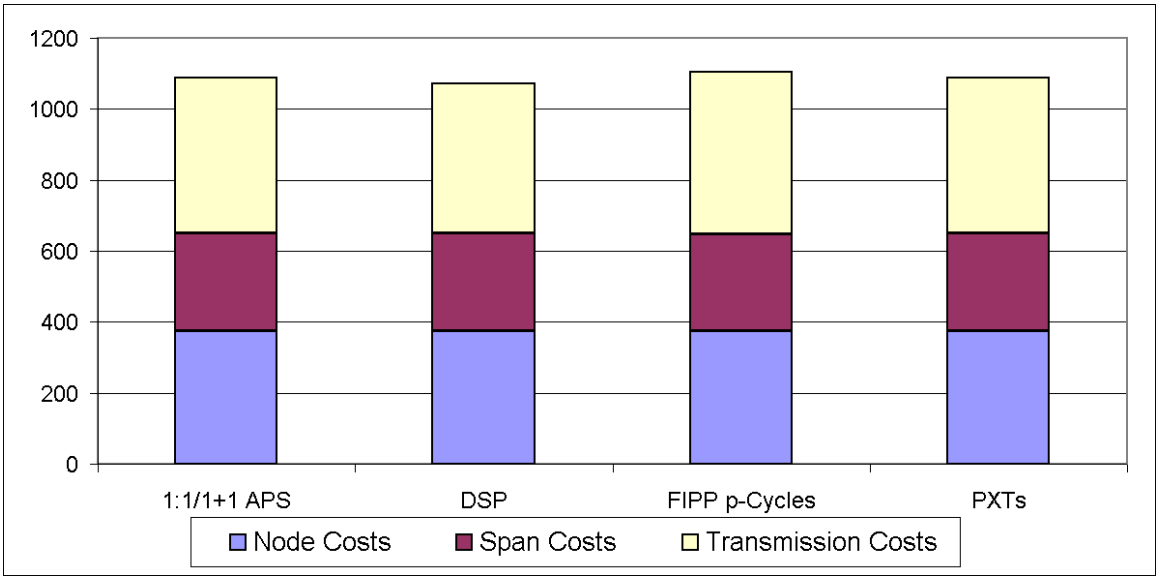


Figure 6.20: NOBEL cost comparison between dedicated architectures, FIPP *p*-cycles and PXTs with PSC modification

to transponder MTD differences. Again, FIPP *p*-cycles and PXTs are at a disadvantage because protection paths are longer because efficient sharing requires that protection paths are deviated from the shortest path.

Overall, this model has the effect of compressing the variability across any set of intelligently-designed networks, regardless of the architecture. However, we must keep in mind that although the costs of the designs are very similar, their network utilization levels can be vastly different. Table 6.4 contains the numerical results presented in Figure 6.20, with additional data regarding the utilization levels of the network. The "Remaining Wavelength Capacity" column shows how many free wavelength channels are unused over all spans in the network. This column represents how much more room there is in the design for additional paths in general. The "Load on Maximally Loaded Span" column shows how many wavelength channels are utilized on the most heavily loaded span. Because we assume 40 wavelengths per fibre and a single-fibre network, the maximum this value can be is 40. This column shows how close the network is to requiring a major capacity upgrade in the worst case that additional paths use the maximally loaded span. The data from both of these columns shows that PXTs are able to achieve significantly less network utilization than APS and DSP under the same approximate cost. A similar pattern was also seen in the "regular" NOBEL case (i.e., before the PSC modification). FIPP *p*-cycles are not able

Architecture	Node Costs	Span Costs	Transmission Costs	Total Cost	Remaining Wave-length Capacity	Load on Maximally Loaded Span
1:1/1+1 APS	374	275	438	1087	**587**	40
DSP	374	275	422	1071	**595**	36
FIPP p-Cycles	374	273	458	1105	**495**	33
PXTs	374	275	440	1089	**639**	28

Table 6.4: NOBEL cost and network wavelength usage comparison with PSC modification

to achieve low utilization in this instance because the design method used for this test case sacrifices capacity efficiency to achieve cost-reducing savings in other areas.

6.2.11.7 Conclusions

These tests have shown that for restorable transparent optical networks, under a realistic cost model for the WDM layer, the spare capacity metric is not an accurate predictor of relative system costs. In fact, under the conditions used here, spare capacity usage is negatively correlated with cost. Under the NOBEL model, it is more accurate to say that network capacity is a resource that is purchased by the relatively fixed start-up costs of the single-fibre network, and a protection architecture that uses less of this capacity is therefore more cost-efficient. The predicted cost savings of low capacity does not appear as an actual reduction in the cost of the initial design, but rather manifests in the long term as a greater time delay before capacity upgrades (either through outlay or requisition of additional fibre or via transmission system upgrades) are required due to demand growth. This is a valuable point of view to keep in mind when presenting shared mesh protection concepts to network operators going forward.

6.2.12 Project Summary

Overall the HAVANA project succeeded in its goals of comparing a wide variety of architectures across several simultaneous real world design constraints. Perhaps the most valuable insight of the study came at the end, where the implementation of the NOBEL cost model cast a new light on our perception of the relationship between spare capacity

efficiency and network cost.

As for PXTs specifically, the results of the project continued to support the conclusion that they are a viable strategy for path protection in transparent networks, at least on par with FIPP *p*-cycles in terms of network utilization and design cost, and perhaps superior when considering wavelength continuity and optical path length constraints. The DRS ILP-based design approach for PXTs holds up well under a WDM layer cost model, even more so than for FIPP *p*-cycles, because of their increased flexibility and suitability towards purely optical protection switching. Both path and span *p*-trees, on the other hand, did not perform favourably in the study, as expected from the results of the work in Chapters 3 and 5.

6.3 Failure Localization in *p*-Cycle Networks

Although it was not part of the HAVANA study, and was motivated by work from an entirely different source, the following study follows in the same vein of investigating problems surrounding the implementation of transparent optical networks. We take a method proposed for localization of single span failures and apply it to restorable *p*-cycle networks, with an eye towards the use of this combined localization and protection scheme in transparent networks.

6.3.1 Background

The failure localization problem is relevant to restorable networking in general, but is particularly significant in the transparent case. In an opaque network, every optical line signal is electronically processed at each node, so a span failure is easily localized to the span on which the signal is lost. But in an optically transparent network, loss-of-light will propagate along the length of the affected path to the end-nodes. This is acceptable for end-to-end path switched protection, but if we are using span protection (e.g., *p*-cycles), we need to know the precise span that has failed so the end nodes can activate the loopback or break-in switching action. To achieve low cost, optically transparent switching nodes typically do not have fast-acting abilities to sense and isolate such failures. Methods are available following the failure to sectionalize the fault, but they are generally slower than required

for activation of a protection switching response. However, a promising idea for fault localization, only recently proposed and studied in the work of Zeng, Huang and Vukovich [ZeHu06, Zeng07], takes a distributed approach with the elegant idea of an *m*-cycle cover of the graph.

The idea behind *m*-cycles is to find a special cycle cover of the graph such that the combination of cycle failures in a failure state will encode the location of the failed span. If a network graph has a cycle cover in which the set of cycles covering each span differs by at least one cycle, then, when a span fails, the span can be uniquely identified by the combination of covering cycles which display an alarm state. This assumes one signal monitor per cycle, which can be placed at any node on the cycle. This central idea was studied in the Ph.D. thesis by Zeng [Zeng07]. As so far considered, the idea is to use the result of the *m*-cycle localization process as input to a separate protection or restoration scheme. The *m*-cycle scheme itself is agnostic about the survivability mechanism employed, and could be used as the activating input for any span-protecting scheme. Furthermore, to our knowledge, all existing work on *m*-cycles has been performed in the absence of any considerations regarding the protection mechanism itself. In [2] the main focus is to find a cycle cover that maximizes the degree of fault localization at minimum cost (some combination of total cycle length and number of cycle monitors). A branch and bound algorithm was developed to produce near-optimal solutions for the minimum cost *m*-cycle cover problem.

The idea of *m*-cycles strongly resonated with us as being very similar to *p*-cycles, if not in concept than at least in structure. Both concepts revolve use the idea of a cycle formed out of preconnected spare channels, though they are used to different ends. Also, to implement *p*-cycles in a transparent network would require enhanced failure localization capabilities. Therefore it seemed natural to us to attempt to combine the two concepts. This Section outlines our attempts to answer the question "can *p*-cycles also serve as *m*-cycles?" More precisely, can a group of span-protecting *p*-cycles as a set also serve as an *m*-cycle cover of the graph for the purposes of fault localization?[6]

It was not initially obvious that *p*-cycle network design could simultaneously include *m*-cycle properties without significant added cost because the design goals of both schemes

[6]Dr. Grover originally proposed these questions in Fall 2006 as a response to the work in [Zeng07], after which followed our joint investigation of the topic.

seemed to be at cross-purposes. *m*-Cycles must constitute a cycle-cover, in which only the on-cycle spans of the *m*-cycles are important, whereas *p*-cycles are most efficient when they "stand off" from the working span capacities they protect (i.e., straddling span relationships are the most preferred from an efficiency standpoint). Therefore our goal was to develop a method to design a combined *p*-cycle/*m*-cycle network with full span failure protection and failure localization, to determine if this combination was feasible and practical.

6.3.2 Method

As usual, we took an ILP approach to the problem, representing both the *p*-cycle restoration constraints and *m*-cycle localization constraints in a single model to achieve both properties with the same set of cycles. For *p*-cycles, we began with the basic *p*-cycle model, given in Section 3.2.1. We then added the constraints for the *m*-cycle cover problem. One phrasing of the problem is that, for every pair of spans in the network, there must be at least one *m*-cycle that crosses one span but not the other. In addition, every span must be covered by at least one *m*-cycle. Represented mathematically, these constraints are, respectively:

$$\sum_{k \in P \, s.t. \, \delta_i^k \oplus \delta_j^k = 1} n_k \geq 1 \quad \forall [i,j] \in S^2, i \neq j \tag{6.12}$$

$$\sum_{k \in P \, s.t. \, \delta_j^k = 1} n_k \geq 1 \quad \forall j \in S \tag{6.13}$$

Note that constraint 6.12 alone is enough to guarantee the *m*-cycle localization property, except that without constraint 6.13 it will allow the case in which there is some span in the network that is covered by no cycles (this is possible even if the *p*-cycle constraints are enforced as well if the span is protected only as a straddler). Although it would be guaranteed that no other span failure would result in none of the cycles failing, in reality this would be indistinguishable from the case in which no failure had occurred at all. Constraint 6.13 explicitly adds this additional case that will be present in all problems and must be distinguished from actual span failure events.

A model using the variables, parameters, and sets from the original *p*-cycle model and these two constraints alone would provide optimal solutions to the *m*-cycle model, a more generalized version of the branch-and-bound method in [Zeng07]. Combined with the *p*-

cycle constraints (see 3.2.1), we have a model that guarantees both span failure restorability and localization in the same design.

6.3.3 Caveat: Chain Subnetworks

Before continuing to the test cases, we must point out one of the pitfalls inherent in *m*-cycle localization, as it affects which networks we are able to use and places limitations on the results. Namely, the fault localization by *m*-cycles cannot be 100% when degree-2 nodes exist in the network. For two spans to produce different *m*-cycle alarm combinations, there must be at least one cycle that passes over one span, but not the other. But if a cycle passes through one span in a degree-2 chain, it necessarily passes through the others. The work in [Zeng07] recognizes this and proposes supplementing *m*-cycle designs in such cases with individual span monitors for the failure of spans in these chains. Here we accordingly either consider design for full localization only in chain-free networks or we seek designs that achieve maximum localization rather than 100% localization in networks with chains.

6.3.4 Test Cases

This combined design model was solved on three test networks also used in [Zeng07], illustrated in Figure 6.21 (a) (b) and (c). The Bellcore and NSFNET networks contain chain subnetworks, so full localization is not possible. Two approaches were taken to obtain comparative results in these cases. In one, chains were eliminated to obtain the *meta-mesh* topologies (see [GrDo02] for a description of the meta-mesh concept) with all nodes having degree-3 or greater. The modified topologies are shown in Figure 6.21 (d) and (e). Under the second approach, the design model was adapted to solve for maximum fault localization instead of total localization. The fault localization levels achieved are then the same as the best results for these networks in [Zeng07]. This approach allows assessment of any extra capacity needed to endow a *p*-cycle network with the same (either complete or partial) levels of localization achievable for optimal *m*-cycle design alone. In these tests, span costs are all set to 1 and the working capacity distribution is that arising from least-hop routing of one lightpath between every node pair.

An additional five tests for the effects of demand volume and pattern were performed on network 15n30s1 in Figure 6.21 (f). This is a topology of degree-4 with no chains. Span

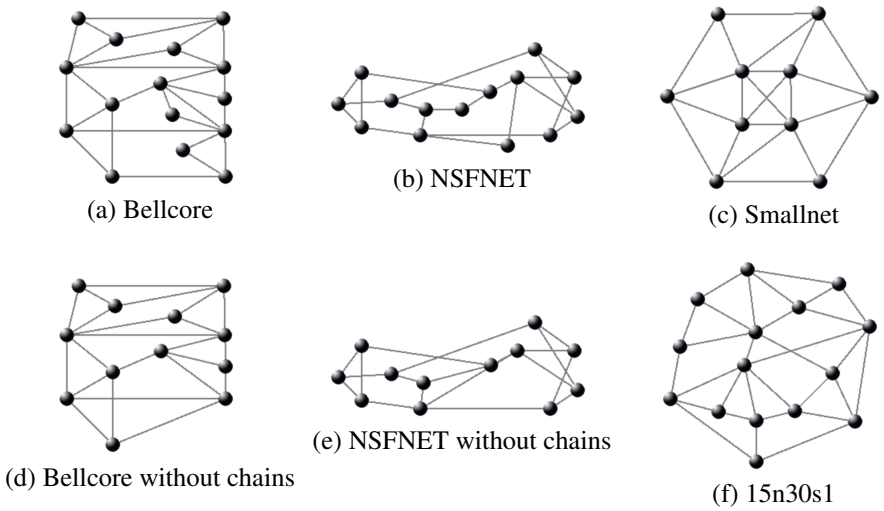


Figure 6.21: Test topologies for combined p-cycle/m-cycle designs

costs here are the Euclidean distances between the end-nodes of each span. The baseline test on this network uses uniform random demand in the interval $[1 \ldots 10]$ between each node pair, routed via shortest paths. In the next three tests, these demand volumes are divided by 2, 4, and 6, and rounded down to the nearest integer value. The final test uses a unit demand between every node pair. The purpose of these tests was to determine the effect of the magnitude of the demand pattern on the results.

6.3.5 Results

Results are summarized in Table 6.5. In one case (the "Divided by 6" demand pattern on the 15n30s1 network), the solver had to be terminated before an optimal solution could be found, because it was simply taking too long to solve. The value is marked with an asterisk in the Table, along with the distance from optimality of the current best integer solution at the time of termination (3%). In fact, we saw that the complexity of the problem in the 15n30s1 network (in terms of computation time) increased as the demand volume decreased.[7] We can explain this because of the fact that, as demand volume goes to zero, the problem tends towards the pure m-cycle problem (no demand needs to be protected, but

[7]The "Divided by 6" demand pattern is the most sparse amongst our 15n30s1 tests. Because the demands in the original demand pattern have values from 1 to 10, and we round down to integer values when dividing by 6, this demand pattern mostly has demands with 0 magnitude, with the highest demand value being 1 ($\lfloor \frac{10}{6} \rfloor = 1$). Therefore it is actually a less heavily loaded demand pattern than even "Unit demand", which has a full matrix of single unit demands.

Network Topology	Demand Pattern	Pure *p*-Cycle Design Cost	Design Cost With *m*-Cycle Property	Cost Increase (%)
15n30s1	Random 1-10	133792	134131	0.25%
	Divided by 2	58117	58623	0.87%
	Divided by 4	23045	23578	2.31%
	Divided by 6	11547	12442* (3%)	$\leq$7.75%
	Unit demand	26749	27106	1.33%
Bellcore (with chains)	Unit demand	151	156	3.31%
Bellcore (chains removed)	Unit demand	79	83	5.06%
NSFNET (with chains)	Unit demand	139	139	0%
NSFNET (chains removed)	Unit demand	92	92	0%
Smallnet	Unit demand	38	45	18.42%

Table 6.5: Combined *p*-cycle/*m*-cycle designs compared to pure *p*-cycle designs

every span still requires failure localization under this formulation). This problem involves strictly binary decisions ("Should this cycle be placed or not?"), and ILP problems using binary variables are classically very difficult for ILP solvers to handle. This is one of the reasons that [Zeng07] used a custom branch-and-bound algorithm to solve the problem.

An example of an *m*-cycle set, taken from the Smallnet network design, is given in Figure 6.22. We can determine by inspection that each span failure will produce a unique failure "symptom" in the cycles. In other words, if we consider the failure state of each cycle as a bit, each single span failure will produce a unique 6-bit word. This sets a maximum limit of the number of total localizable failure scenarios in this design of $2^6 - 1 = 63$ (subtracting 1 for the non-failed network state). This logarithmic relationship between the number of *m*-cycles and the number of localizable failures is noted in [Zeng07].

Results for the set of tests on the 15n30s1 network illustrate that increasing the demand volume decreases the incremental fractional cost of adding the *m*-cycle requirement to a *p*-cycle design. This makes sense because the total spare capacity for survivability is at least monotonically increasing with network demand (for *p*-cycles and for any scheme restorability scheme), whereas the *m*-cycle cover is a purely topology dependent fixed cost type of investment for the network as a whole. Or, to look at it another way, as the working capacity of a network increases it becomes increasingly easy to find a subset of cycles to

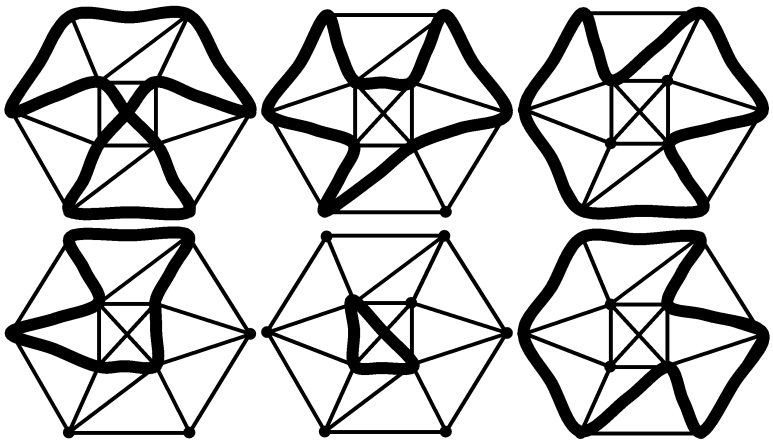


Figure 6.22: The *m*-cycle cover set for the Smallnet network topology

realize the fault localization cover requirement within a restorable *p*-cycle design. This explains the relatively high excess capacity result (18%) for Smallnet, and gives a general insight about how sheer volume of demand in a network will inevitably dilute the relative cost of incorporating *m*-cycle monitoring in the network design. The Smallnet network's small size combined with the unit demand pattern means that it has very little demand volume with which to dilute this cost.

The main finding in Table 6.5 is that it seems to be feasible and economic to integrate *m*-cycle functionality into a *p*-cycle network design. The contrasting tendencies of *m*-cycles and *p*-cycles to prefer on-cycle versus straddling span relationships are not so difficult to reconcile after all. However, the economics of this will depend on the demand volume in the network. At the very least, we can say that it is economic in the unit demand pattern case in realistically sized networks. This result for the unit demand pattern has special significance, because it can be thought of as the full-communication (communication between every node pair) demand pattern with the minimum possible volume of working capacity against which to dilute or amortize the added cost of the *m*-cycle cover set. We would expect this condition to be met in most modern transport networks.

6.3.6 Conclusions

We have shown that it is not difficult or necessarily costly to obtain *m*-cycle fault localization in *p*-cycle networks. The added cost of perturbing the *p*-cycle design to include an *m*-cycle cover can even be zero in certain cases. At the very least, it is a fixed cost

for a given topology and hence becomes less significant to the total cost of a network as its demand increases. The designs that we have produced here are really still *p*-cycle designs, the same as those produced by the original model, except with the *m*-cycle cover property incorporated into them. This can be thought of as a method to increase the utility and efficiency of *m*-cycles as presented originally in [Zeng07], where they are only used for monitoring and are not themselves part of the capacity usable for protection. In our designs, the same *p*-cycle in a transparent network can be centrally monitored for loss-of-light as part of the *m*-cycle cover, and then be activated as part of the survivability solution immediately afterward. This potentially opens up span protection via *p*-cycles as a viable option for transparent networks.

Chapter 7

UPSR-Like *p*-Cycles

7.1 Introduction

The one concept that ties together all of the studies presented so far in the thesis is that of spare capacity sharing. Indeed, sharing is the primary attribute of mesh-based restorable networking: taking advantage of the mesh topology to share spare capacity between topologically dispersed working paths. To use a simple dedicated protection scheme (such as APS or UPSR rings) in a mesh topology seems to be a waste of its highly interconnected nature.

However, dedicated protection does carry with it certain advantages. For example, availability will generally be higher (because availability in a single failure restorable network is a function of dual failure restorability, which will tend to increase with higher spare capacity redundancy). Also, the switching of dedicated protection is inherently simpler. This raises the question of whether it is possible to take advantage of some of the properties of mesh networks while still retaining the advantages of dedicated protection. This Chapter introduces a new architecture called "UPSR-like *p*-cycles" that incorporates these properties. In this Chapter we develop and analyze methods of designing restorable networks using UPSR-like *p*-cycles, and in the process investigate several of their characteristics.[1]

[1]Some of the work in this Chapter has been accepted as a paper at the Workshop on Reliable Networks Design and Modeling:

A. Grue, W. Grover, "UPSR-like *p*-Cycles: A New Approach to Dual Failure Protection," to appear in the *Proceedings of the Workshop on Reliable Networks Design and Modeling (RNDM 2009)*, St. Petersburg, Russia, 12-14 October 2009.

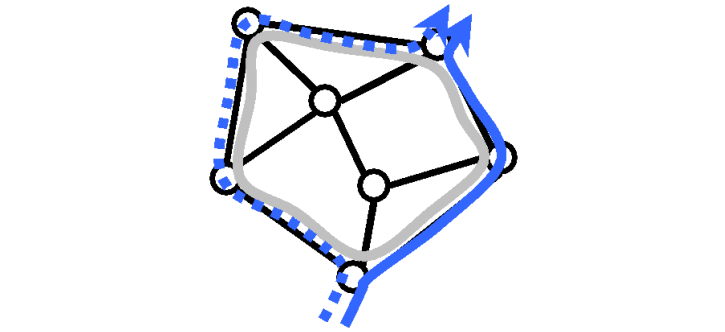
7.1.1 Background

As mentioned above, the availability of dedicated protection will be higher in general because of higher resilience to dual failures [ClGr02]. However, this statement only applies as an average over all demands and failure scenarios, and does not mean that a network using dedicated protection and designed for single failure restorability only will be able to provide guaranteed dual failure restorability to a class of high priority demands. In this regard, dedicated and shared protection are on equal footing; extra considerations must be made for both the capacity plan and failure response in order to handle dual failures. A number of techniques have been proposed to enable guaranteed dual failure restorability for high priority QoP classes with *p*-cycles, including straddler-routing [KoGr05b], a multi-QoP approach using span-restorable mesh restoration [GrCl05] or *p*-cycles [ClGr05], and reconfigurable *p*-cycles [ScGr04].

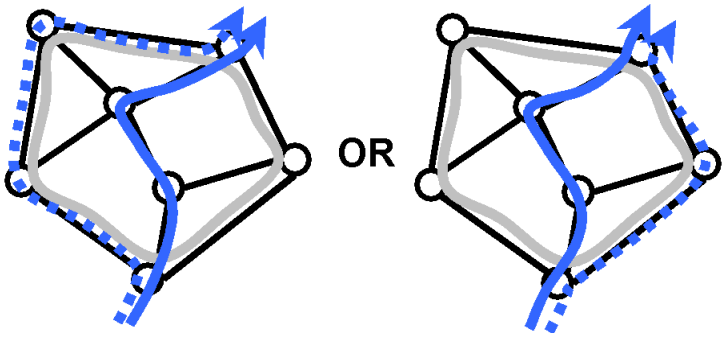
However, it occurred to us that resilience to dual failures could also be enabled in a *p*-cycle architecture by combining the principle of path-protecting *p*-cycles (see [KoGr05]) with the dedicated protection approach of UPSR rings.[2] In a UPSR ring, the working signal is simultaneously circulated in both directions around the cycle from the source to the destination, such that a single failure on either side of the ring will leave the other path intact. A UPSR-like *p*-cycle would incorporate this simultaneous circulation principle with the mesh-like routing of working paths in traditional *p*-cycles (i.e., removing the requirement for working paths to be routed strictly on-cycle).

This bears some similarity to the approach proposed recently in [Kama06], which also employs simultaneous circulation of working and protection signals in *p*-cycles for fast protection switching. The approach in [Kama06] shares the *p*-cycle by making use of network coding principles. This approach can handle at most one path failure at a time, and is especially susceptible to dual failures because of the network coding requirements. In contrast, UPSR-like *p*-cycles can provide enhanced R2. If a working path is routed as a straddler, there are two possible protection paths around both sides of the cycle. Should the default protection path fail in addition to the working path, a high-priority demand might preempt traffic on the other side of the cycle and use some of that capacity to create a

[2] Dr. Grover first proposed this new architecture in Winter 2007, originating and providing the impetus for the line of work followed by this Chapter.



(a) UPSR



(b) UPSR-like *p*-cycle

Figure 7.1: Routing in UPSR and UPSR-like *p*-cycles

second protection path. See Fig 7.1 for an illustration of this difference between UPSR and UPSR-like *p*-cycles.

This additional routing flexibility not only introduces the possibility of cost savings over UPSR by reducing the length of working routes, but leaves open the possibility of further optimization of structure capacity by making choices for protection routing within the structure itself. As shown in Fig. 7.1(b), there are two routing options available for a straddling working path on a UPSR-like *p*-cycle. By taking advantage of routing options in a mesh network, we can combine the benefits of UPSR protection with some of the well-known efficiency of *p*-cycles (see [GrSt98]).

7.1.2 Goals

The purpose of this study is to answer some of the research questions regarding UPSR-like *p*-cycles. What is the best way to route the protection paths? How would the routing problem be solved in an ILP, and do there exist better (faster, simpler) methods to solve this problem? How can we design efficient networks using UPSR-like *p*-cycles? What savings

can they achieve over regular UPSR, and how do they compare to more efficient schemes? To what extent can they provide enhanced R2 for priority services? These questions will be answered over the course of the following sub-studies.

7.2 Single-Ring View with Straddlers Only

7.2.1 Problem Description

As we were starting from the ground-up with a new architecture, we decided to begin by treating only the issues with designing a single UPSR-like *p*-cycle. This problem is non-trivial, unlike the regular UPSR design problem, because of the fact that working paths may now be disjoint from the cycle. This means that we now have to make a decision as to which direction the protection path will take around the cycle. This problem has some similarities to the "BLSR sizing problem", which is the problem of routing demands on a BLSR ring such that the capacity of the ring is minimized (discussed further in [MoGr98]). The problems are different, however, as the protection paths here do not share spare capacity (in order to maintain the 1+1 APS property of simultaneous transmission along both the working and spare paths). We also further simplify the initial investigation by only considering straddling working paths (i.e., completely disjoint from the cycle) for protection, and solving the routing decisions for these paths.

The objective of this problem is taken to be the minimization of the total link capacity used by the cycle. For the purpose of this investigation, the UPSR-like *p*-cycle is assumed to use an identical amount of capacity on every span of the cycle. If this were not true, and the cycle were allowed to use a variable amount of capacity on each span, the design problem would become trivial as it would only be necessary to route each protection path along the shortest route around the cycle in order to achieve minimum capacity usage. But if the UPSR *p*-cycle is seen as a single structure of uniform capacity (as with a regular UPSR, e.g., a separate, reserved "pipe" that protects a certain set of demands or path segments as a single encapsulated structure), the problem then becomes one of finding protection routings that overlap in a way such that the cycle capacity "ceiling" forced by the most heavily capacitated span is minimized.

A way of solving this problem that intuitively seems to be near-optimal would be to

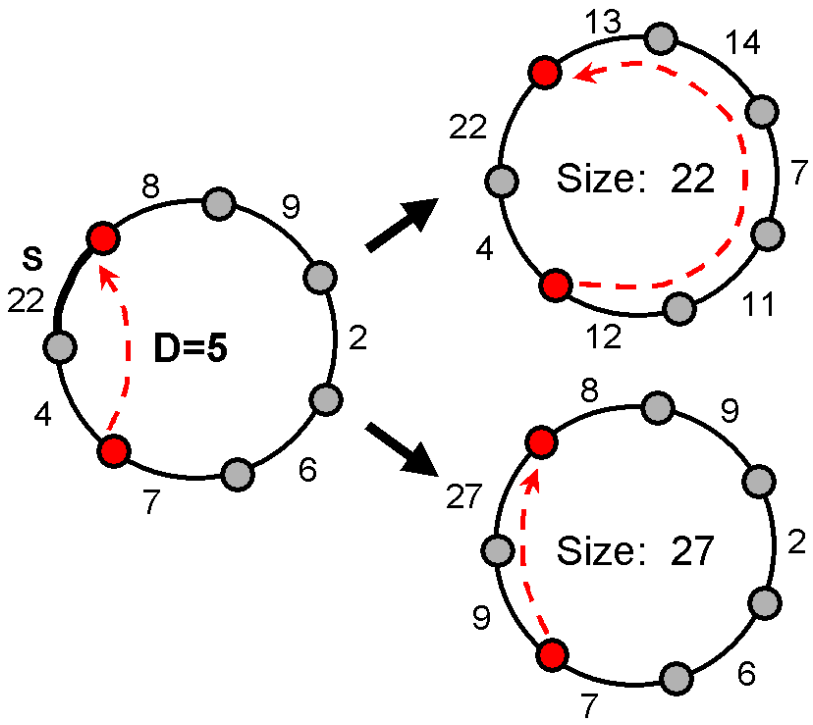


Figure 7.2: Potential suboptimality of shortest-path protection routing on a UPSR-like *p*-cycle

simply route all protection paths along the shortest available side of the cycle. We would expect this approach to reduce the ring cost in an indirect way by minimizing the total amount of utilized spare capacity in the ring. However, we might also expect that in some situations this will result in unintelligent routing decisions that unbalance the capacity distribution on the cycle, producing a suboptimal result. Such a situation is illustrated in Figure 7.2, in which span *S* is already heavily loaded by other demands and therefore should be avoided. Another approach may be to balance the capacity on the cycle by attempting to split the capacity of each working route in two ways around the cycle (assuming the splitting of working routes is allowed). Of course, the way to *guarantee* that the cycle capacity is minimized is to use an ILP model to optimize the objective function over the entire space of design possibilities. All three of these approaches (shortest-path, split routing, and full optimization) were tried and are compared in the following Section.

7.2.2 Experimental Setup

In total, five approaches to the ring loading problem were investigated. An ILP-based approach was used to implement each of them. The first three approaches are summarized as follows:

(a) Shortest-path: Each working route is allocated protection on the shortest protection route around the cycle (by hops). In case of ties, the solver makes an intelligent choice (including possibly splitting protection paths between both halves).

(b) Equal split: Each working route is allocated half of its protection around each half of the cycle. In case of odd amounts of demand, the solver again makes an intelligent routing choice.

(c) Optimization: The optimizer freely chooses the directions of all protection paths.

During the course of the study, it was recognized that, if this scheme is to replicate the simplicity and fast switching of 1+1 APS, it may be desirable for the protection paths (and working paths) to remain unsplit, i.e., each end-to-end demand should be served by only a single working and protection route (so that, for example, the protection signal does not need to be split into two lower-rate signals at the transmitter and then recombined at the receiver). This introduces the following "no-splitting" variants to schemes a and c (not b for obvious reasons):

(d) Shortest-path without splitting: Each working route is allocated protection on the shortest protection route around the cycle (by hops). In case of ties, the solver still makes an intelligent choice, but this choice is binary; splitting is not allowed.

(e) Optimization without splitting: The optimizer freely chooses the directions of all protection paths, but all protection paths for the same demand must follow the same route (i.e., no splitting).

7.2.2.1 ILP Model

The above 5 schemes were implemented using the following ILP model as a base:

ILP Model

Sets:

D The set of demands to be protected by the cycle being loaded, indexed by r.

S The set of spans on the cycle, indexed by i.

P_r A set of protection paths on the cycle for each demand r in D. Can contain either 1 or 2 paths depending on whether the demand is routed on-cycle or straddling, respectively. Indexed by q.

T_r^q The set of cycle spans contained in protection path q for demand r.

Parameters:

d_r The number of demand units to be protected by the cycle for demand r.

α Scaling value for the bicriteria objective function.

Variables:

s_i The amount of spare capacity used for protection on span i of the cycle.

h The highest amount of spare capacity used on any one span of the cycle (sets the total UPSR-like p-cycle capacity).

u_r^q The number of copies of protection path q allocated to protect demand r.

Objective Function:

Minimize

$$h + \alpha \sum_{i \in S} s_i \tag{7.1}$$

Constraints:

$$s_i \leq h \quad \forall i \in S \tag{7.2}$$

$$\sum_{q \in P_r} u_r^q = d_r \quad \forall r \in D \tag{7.3}$$

$$s_i = \sum_{r \in D} \sum_{q \in P_r \, s.t. \, i \in T_r^q} u_r^q \quad \forall i \in S \tag{7.4}$$

More accurately, the above model implements optimization approach c) as described above. The objective function will find the design with the minimum ring capacity (h), that also uses the minimum total capacity (the second term in the objective function). This term is weighted with α such that it will never be greater than 1, so that it will not have an impact on the value of h (an integer). This was done to prevent the solver from finding a design that, though "optimal" in the sense of using a minimally sized ring, uses obtuse routing decisions resulting in sub-optimal capacity usage that nonetheless remains below the optimal spare capacity "ceiling". We would rather obtain designs using the best routing possible, so that we can draw more general conclusions about intelligent routing in UPSR-like p-cycles.

Constraint 7.2 is what establishes the value of h based on the largest spare capacity usage among all of the spans on the cycle. Constraint 7.3 ensures that enough protection paths are allocated to protect each demand. Constraint 7.4 calculates spare capacity based on protection path usage. Protection paths are not shared, so this calculation is a simple sum. Note that the model implicitly allows splitting of restored flow around the cycle, as there is nothing preventing a straddling path from claiming a nonzero amount of protection on both of its possible protection paths in P_r.

This model solves very quickly, because it is a local problem for a single cycle only and therefore only needs to consider a handful of spans and demands at once. It is not difficult to modify it to handle design approaches a) and b) as well by simply adding constraints. To implement design approach a), the following constraint is added to the model:

$$u_r^q = 0 \quad \forall r \in D, \forall q \in P_r \, s.t. \, q \neq \text{shortest path in } P_r \tag{7.5}$$

This constraint set only really applies to straddling demands where 2 potential protection paths exist (for non-straddlers, P_r will only contain one path and therefore no constraint will be generated). This is simply a mathematical way of saying that if one path is longer than the other, it cannot be used (i.e., the shorter path must be used instead). Note that this leaves open the possibility of splitting between two equal-length shortest routes, as

described above in a). Note also that we could achieve this same effect by pre-processing the data to remove strictly shortest protection paths.

To implement design approach b), the following constraint is added to the model:

$$u_r^{q_1} - u_r^{q_2} \leq 1 \quad \forall r \in D, \forall q_1, q_2 \in P_r \, s.t. \, q_1 \neq q_2 \tag{7.6}$$

This constraint again only applies in the straddling case, and states that the difference in usage between the two protection paths must never exceed 1. Therefore, if the amount total amount of demand is odd, the difference in usage between the two paths will be exactly one, and if the amount of demand is even, the difference will be zero. This allows a certain amount of leeway in cases when the amount of demand is odd, but keeps to the spirit of investigating the effects of the general design policy of equally splitting protection wherever possible.

As mentioned, this model, along with the above 2 optional constraints, implements strategies a) through c) as described above. To represent the no-splitting concept for strategies d) and e), however, we must modify the original model in this way:

Variable Modifications

u_r^q This time it is a binary variable that is 1 iff demand r uses protection path q for protection and 0 otherwise.

Constraint Modifications

Constraint 7.3 needs to be modified to take into account the fact that protection path usage is now a binary variable:

$$\sum_{q \in P_r} u_r^q = 1 \quad \forall r \in D \tag{7.7}$$

Constraint 7.4 must be modified for a similar reason:

$$s_i = \sum_{r \in D} \sum_{q \in P_r \, s.t. \, i \in T_r^q} u_r^q \cdot d_r \quad \forall i \in S \tag{7.8}$$

This new model ensures that no splitting will occur by making the routing decision a binary one. The above changes to the basic model (i.e., without supplementary con-

straints 7.5 and 7.6) implement strategy e) (fully optimized routing without splitting). To implement strategy d) (shortest-path routing without splitting), we simply use this modified model along with supplementary constraint 7.5. Note that, although some of these routing strategies could be implemented using more basic approaches than ILP, it is convenient for us to represent them here as ILP problems both because of the guarantee of the optimality of the solutions and because our ability to solve the problems follows directly from our ability to represent them all mathematically. The models are simple enough that solution times are trivial in any case.

7.2.2.2 Test Cases

This problem only concerns the routing of a single ring. Therefore, the only variability in the test cases consists of the ring size (in terms of spans or nodes) and the demand pattern (all of which are demands that originate and terminate on the ring). This is much simpler than a whole-network design problem, in which we might consider any arbitrary network graph topology. This means that we need concern ourselves less with the characteristics of the ring (being in this case only its size), and focus more on the characteristics of the demand pattern.

In keeping with this, the test cases consist of many variations on demand patterns over a small number of different sizes of rings. To start off with, we consider the simplified case of straddling protection only, i.e., the case where every demand has a choice between two different protection routes. The parameters of the straddling-only test cases are summarized in Table 7.1. The label "All straddlers" means that every node pair in the ring (except adjacent nodes) carries demand that we assume is routed as a straddler with respect to the cycle. We exclude adjacent nodes from this set, because we assume in these cases that any working flow would be routed directly along the span linking the two nodes, preventing this path from being a straddler. Therefore this case represents one of maximal routing freedom, while still taking into account the implications of shortest-path routing.

After completing trials 8 through 28, the method was automated to calculate the results for a much larger number of test cases so as to derive some statistical conclusions over a large set of randomized tests. These random tests again used a ring of 6 nodes/spans and a full set of straddling demands. In each test case the magnitude of each demand was

Test Case	Ring Size (Number of Nodes/Spans)	Demand Pattern
1	6	All straddlers (9 total): 5 7 6 9 4 5 8 6 3
2	6	5 straddlers (demand pattern from case 1 with 4 randomly deleted): 5 9 4 8 3
3	6	All straddlers, unbalanced demand values: 1 5 7 7 12 11 15 18 16
4	8	All straddlers (20 total): 5 7 6 9 4 5 8 6 3 5 7 6 9 9 4 5 4 6 7 6
5	9	All straddlers (27), balanced demand values
6	9	All straddlers, unbalanced demand values (linearly increasing from 1 to 27)
7	6	All straddlers, unbalanced demand values (linearly increasing from 1 to 9)
8 through 28	6	All straddlers, uniformly random distribution between 0 and 20 (21 randomly generated cases)

Table 7.1: Summary of UPSR-like *p*-cycle single ring design test cases (straddling protection only)

Test Case	Loading Strategy				
	a) Shortest-Path With Splitting	**b) Equal Split**	**c) Optimized With Splitting**	**d) Shortest-Path Without Splitting**	**e) Optimized Without Splitting**
1	23	26	23	26	26
2	15	15	15	15	15
3	38	46	38	41	41
4	46	60	46	49	49
5	59	81	57	59	59
6	178	187	136	178	142
7	18	23	18	19	19
8 through 28 (averages)	41.8	———	41.7	47.3	46.0

Table 7.2: Summary of results (ring costs) for UPSR-like *p*-cycle single ring design (straddling protection only)

assigned a random value between 1 and 20. The optimal ring costs were calculated under approaches a) through e) for 20,000 randomized trials.

7.2.3 Results

Table 7.2 gives the results of the experimental trials outlined in Table 7.1. The first thing that the Table demonstrates is that equal split routing is uniformly worse than shortest-path routing, while not being significantly simpler (both make their routing decisions independently for each demand on the ring without considering the global ring situation), so we should be able to disregard this as a viable strategy. The real competition is between optimized routing and shortest-path routing, with shortest-path routing being simpler at the expense of being occasionally less efficient. What Table 7.2 shows, however, is that shortest-path routing produces the same ring size as a full optimization in a surprising number of cases. This suggests that in many cases the optimal local choice is in fact the optimal choice overall. If it is true that shortest-path routing is near-optimal in most cases, then we may be able to perform whole-network optimal design for this architecture under the simplifying assumption of shortest-path routing.

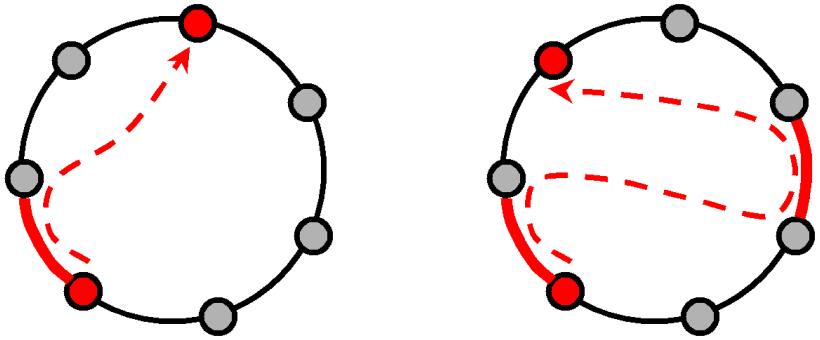
However, the results show that there are certain cases where shortest-path routing is not

so good. The most striking case is case 6, which was specifically constructed as a case to foil the shortest-path strategy. To expand on the description given in Table 7.1, it is a test case with a 9-span ring and a full demand pattern of 27 straddling demands. The magnitudes of the demand varied linearly from 1 to 27. The test was constructed this way so as to provide an unbalanced set of demands that would create a situation in which it was not ideal for many of the demands to be routed shortest-path, similar to the case illustrated in Figure 7.2.

Tests 8 through 28 were conducted in order to determine how often optimization would result in improvement over shortest-path routing. The tests were continued until a difference in the ring size between the two methods was observed, i.e., in only the last out of the 21 test cases were the results any different. The automated trials expand on this result. Over all 20,000 trials, strategies a) and c) (that allow splitting) differed in only 9.5% of the cases, and the results for shortest-path routing used only 0.6% more ring capacity on average. When the same test was performed for the non-splitting case (i.e., strategy d) vs. e)), the results differed in 36.5% of the cases, but overall the rings using shortest-path loading required only 0.9% more capacity. Therefore we can be quite confident that, in the case where rings are used to protect sets of straddling demands of roughly comparable magnitude, shortest-path loading is nearly optimal.

7.3 Extension to On-Cycle Protection

To this point we have considered the simpler case of UPSR-like *p*-cycles that only protect straddling demands. However, there is no reason why we cannot extend the concept to the protection of demands whose working paths are not routed disjoint from the cycle. In fact, this case is even simpler in terms of planning, as there can be at most one viable protection path for such a demand. Figure 7.3 (a) illustrates this fact. Figure 7.3 (b) shows a case where no protection is possible because the working path intersects both possible protection paths. This case is analogous to the Z-case for FIPP *p*-cycles (see Figure 2.8). For FIPP *p*-cycles, one protection unit can still be provided to the path, because the restoration action will implicitly "choose" the direction in which restoration occurs depending on the failure (even though the restoration action can still be designed to be failure-independent).



(a) On-cycle path that can be protected

(b) On-cycle path that cannot be protected

Figure 7.3: On-cycle path protection cases for UPSR-like *p*-cycles

For UPSR-like *p*-cycles, however, this is not possible, as transmission must occur on the working and backup paths simultaneously. Theoretically, such protection would be possible if backup transmission occurred simultaneously on both halves of the cycle; then, no matter what span failure occurred, at least one path would remain intact. However, such an arrangement would effectively reserve the entire cycle for use of that demand, making it very capacity inefficient. Therefore we do not consider this case in the following study.

Because on-cycle paths have only one possibility for protection, there is no real routing problem for these paths considered in isolation. Routing an on-cycle path is a one-step process of assigning dedicated spare capacity along the (only possible) backup path. What is interesting about the extension for on-cycle protection is the effect that this will have on the routing decisions for straddling paths. The capacity mandated by on-cycle paths represents a kind of "spare capacity floor" that serves as an input to the straddling protection problem. If this capacity is zero, or equal on every span, there is no impact on the routing of straddlers, and the same optimal solution as in the all-straddler case will still apply. When this capacity floor is relatively even, we would similarly expect that the straddler solution would not change greatly and that our observations about straddler routing so far would not be affected. If this capacity floor is relatively uneven, however, it may have a significant effect on straddling path routing.

7.3.1 Experimental Setup

Table 7.3 describes the individual test cases used. Again, in addition to the test cases in this Table, randomized trials were also used to further test the characteristics of the designs. The random tests again used a ring of 6 nodes/spans. The demand pattern contained a full set of both straddling and on-cycle demands. The set of straddling demands consisted of all demand pairs between non-adjacent nodes in the ring (each with 2 possible protection paths), as in the straddler-only test cases. The set of on-cycle demands consisted of all demand pairs between adjacent nodes (each with only 1 possible protection path, as we assume that shortest-path working routing will cause the working paths to only ever reside on the single span between these two nodes), combined with all demand pairs between non-adjacent nodes, each represented twice, once for each side of the cycle that might contain the overlap with the working path. As mentioned above, on-cycle demands do not represent routing decisions and only provide a minimum capacity floor for the cycle, so accounting for all these various possibilities may simply be equivalent to generating a randomized capacity floor for the cycle. The full exercise was performed for the sake of representing all possibilities in the problem, and because the problem is simple enough to solve that including these additional details does not impede the solution process.

In each test case the magnitudes of the demands assigned random values between 1 and 20. The ring costs were calculated under approaches a) through e) for 20,000 randomized trials.

7.3.2 Results

The results for the test cases from Table 7.3 are given in Table 7.4. As for the automated trials, strategies a) and c) (that allow splitting) differed in 34% of the cases. However, on average the results for shortest-path routing used only 1% more ring capacity. When the same test was performed for the non-splitting case (i.e., strategy d) vs. e)), the results differed in 53% of the cases, but overall shortest-path loading required only 1.2% more capacity. Again, shortest-path routing appears to be near-optimal. However, note that this is only the case where the capacity floor from on-cycle demands is (statistically) flat.

To incorporate a non-statistically flat distribution of on-cycle demands, the simulation

Test Case	Ring Size (Nodes/Spans)	Demand Pattern
1	6	Full matrix (15 total) with both balanced straddlers and on-cycle demands
2	6	Full matrix (15 total) with balanced straddlers but unbalanced on-cycle demand
3	6	Full matrix (15 total) with balanced straddlers (in the range of 0 to 10) and one on-cycle demand that is routed around exactly half of the cycle (magnitude 20)
4	9	Full matrix (36 total) with both balanced straddlers and on-cycle demands
5	9	Full matrix (36 total) with balanced straddlers but unbalanced on-cycle demands
6	9	Full matrix (36 total) with balanced straddlers (in the range of 0 to 10) and one on-cycle demand that is routed around half of the cycle (5 spans, magnitude 20)

Table 7.3: Summary of UPSR-like *p*-cycle single ring design test cases (straddling and on-cycle protection)

Test Case	Loading Strategy				
	a) Shortest-Path With Splitting	**b) Equal Split**	**c) Optimized With Splitting**	**d) Shortest-Path Without Splitting**	**e) Optimized Without Splitting**
1	39	44	39	40	40
2	97	109	97	99	99
3	39	46	39	41	39
4	88	111	88	88	88
5	99	123	97	99	99
6	77	101	71	77	71

Table 7.4: Summary of results (ring costs) for UPSR-like *p*-cycle single ring design (straddling and on-cycle protection)

was re-run, this time removing all on-cycle demands including one (arbitrary, randomly chosen) span in their protection path (to cause a reduction in the spare capacity floor on this span and the surrounding spans). For these tests, the results differed in 77% of the cases, but shortest-path loading still requires only 3.5% more capacity on average. So the difference was greater, but still minor, even in this most extreme case in which the methodology was designed to maximally depress the spare capacity floor on a single span.

7.3.3 Conclusions

The exercises performed to this point can be taken as convincing evidence that shortest-path routing of protected demands is near-optimal in the "average worst case" scenario. In other words, if a network designer were to follow this policy, he would not, on average, tend to produce designs that were excessively inefficient in terms of spare capacity usage (as compared to the truly optimal designs). However, this does not rule out special cases where exceptional routing decisions may be made to achieve savings. As usual, the discretion of a human designer must be applied.

7.4 Full Network Design Problem

The results so far have given us a good indication of how to properly load a single UPSR-like *p*-cycle. We have seen that, on a cycle with a balanced distribution of protected paths, shortest-path routing performs almost identically to optimized routing. We can use this information to inform our study of the design of entire networks protected by UPSR-like *p*-cycles.

7.4.1 ILP Models

As with the ring loading problem, we can imagine two types of policies for UPSR-like *p*-cycles: with splitting allowed, and with splitting prohibited. We present models to design UPSR-like *p*-cycle-protected networks under both of these policies below. Again we assume that each ring is sized according to the maximum amount of capacity used on any span of the ring.

ILP Model

Sets (both models):

D The set of demands to be protected in the network, indexed by r.

S The set of spans in the network, indexed by i.

C The set of cycles that can be used as UPSR-like p-cycles, indexed by c.

S_c The set of spans crossed by cycle c.

P_r^c The set of protection paths provided to demand r by cycle c (will be empty if demand r cannot be protected by cycle c). Indexed by q.

$T_r^{c,q}$ The set of spans contained in protection path q of cycle c for demand r.

Parameters (both models):

d_r The number of demand units to be protected for demand r.

Variables (both models):

s_i^c The amount of spare capacity used on span i on cycle c.

s_i The total amount of spare capacity used on span i.

h_c The size of ring c.

$u_r^{c,q}$ Usage of protection path q on cycle c for demand r. The exact sense of this variable will depend on the splitting policy, as in the single ring loading problem. In the splitting case, it is an integer, and represents the number of times the path is used. In the non-splitting case it is binary and represents the all or nothing decision of using this path for demand r.

Objective Function (both models):

Minimize

$$\sum_{i \in S} s_i \tag{7.9}$$

Constraints (both models):

$$s_i^c \leq h_c \quad \forall c \in C, \forall i \in S_c \tag{7.10}$$

$$s_i \geq \sum_{c \in C \, s.t. \, i \in S_c} h_c \quad \forall i \in S \tag{7.11}$$

Constraints (splitting allowed):

$$\sum_{c \in C} \sum_{q \in P_r^c} u_r^{c,q} \geq d_r \quad \forall r \in D \tag{7.12}$$

$$s_i^c \geq \sum_{r \in D} \sum_{q \in P_r^c \, s.t. \, i \in T_r^{c,q}} u_r^{c,q} \quad \forall c \in C, \forall i \in S_c \tag{7.13}$$

Constraints (no splitting):

$$\sum_{c \in C} \sum_{q \in P_r^c} u_r^{c,q} \geq 1 \quad \forall r \in D \tag{7.14}$$

$$s_i^c \geq \sum_{r \in D} \sum_{q \in P_r^c \, s.t. \, i \in T_r^{c,q}} u_r^{c,q} \cdot d_r \quad \forall c \in C, \forall i \in S_c \tag{7.15}$$

There are obvious parallels between this model and the model for the single-ring case. Constraint 7.10 is constraint 7.2, generalized to the case where many rings are present in the network. Constraint 7.11 is new, but only aggregates the total capacity from all rings onto each span. Protection constraints 7.12 and 7.14 are the whole-network versions of equations 7.3 and 7.7, with additional summation terms because protection is now spread over a set of many cycles. Cycle capacity constraints 7.13 and 7.15 are the equivalents of constraints 7.4 and 7.8. The only addition is the new superscript c to apply the constraint to the entire set of cycles.

Note that the constraints have been changed from strict equalities (as in the single-ring design) to more relaxed inequalities. This was done to reduce the complexity of the model by relaxing the constraints, as the model has quite long solution times for realistically sized networks (as we will see in the next Section). The resulting designs will still be optimal, but note that this allows for a certain amount of "sloppiness" in the solutions, e.g., the protection variable $u_r^{c,q}$ may be set such that certain demands are overprotected in the solution, as long as this overprotection does not increase the total ring capacity for cycle c past the optimal amount.

7.4.2 Initial Tests

The version of the above model with protection splitting allowed was first run on a network with 15 nodes, 30 spans, and a full demand pattern ranging from 1 to 10 units between each node. Demands were all routed shortest-path. Unfortunately, the model was found to be quite complex. Several days were required to even obtain a feasible integer solution (however, this initial solution was optimal to within a 0.6% mipgap). In the non-splitting case, a feasible solution was not even found after several days. The complexity of the non-splitting model is higher because the real-valued variables become binary variables, which are notoriously problematic for ILP solvers. Therefore the non-splitting model was not considered further in these tests.

The complexity of this model is not surprising considering that this problem has similarities to the FIPP *p*-cycle design problem. Although UPSR-like *p*-cycles do not need to deal with the concept of DRSs (disjointness is not a concern because every path receives dedicated protection), the model must still keep track of the assignment of each protection path on each cycle to each demand. Explicit assignment of protection is a concern unique to path-protecting design models, and greatly multiplies the number of variables in the model.

Another problem with solving designs for this architecture is that, because the Z-case cannot be protected, every working path must have at least one unique span-disjoint path between its endpoints to serve as the protection path. Now, even though a bi-connected network is guaranteed to contain at least 2 such span-disjoint paths between ever node pair, if working path routing is performed via a naïve shortest-path approach, the working path may be chosen such that no disjoint path can then be found (the so-called "trap topology"). Therefore we took care to perform a modified type of shortest-path routing that avoids this case (as in Section 4.2.2.1). In the rest of the following test cases, this type of demand routing was used.

7.4.3 Shortest-Path Routing Approximation

To get results from this model in more reasonable lengths of time, we could simply restrict ourselves to smaller networks. However, another alternative is to use the insights gained in the single ring design case to build approximations into the model that reduce its complexity

Test Case	Nodes	Spans	Demand Pattern
NSFNET	14	21	1 unit between every node pair
Bellcore	15	28	1 unit between every node pair
ARPA2	21	25	1 unit between every node pair
Smallnet (small demand pattern)	10	22	1 unit between every node pair
Smallnet (larger demand pattern)	10	22	Uniform random (1-10)
15n30s1 family	15	16 to 30	Uniform random (1-10)

Table 7.5: Summary of UPSR-like *p*-cycle full network design test cases

while not sacrificing much in terms of the quality of the solutions.

In the single ring design case, we saw that the shortest-path routing method would often give optimal solutions. Therefore we modified the original model to only allow protection for straddlers along the shorter arcs of the cycles that could protect them. This can be done in a variety of ways, such as adding a constraint to enforce shortest-path routing (as done above in the single ring case), modifying the input data set to only contain the shortest protection paths, or modifying the model itself to only consider a single protection path for each demand. We chose the pre-processing method, removing the longest protection path for each straddling demand from the input data completely. In cases where two protection paths of the same length existed, one was removed arbitrarily. This is the preferred method for reducing problem complexity, as it cuts down the data set used by the model. The approach of adding constraints to the model would only increase complexity.

7.4.4 Experimental Setup

Two questions interest us here. First of all, how good are the results produced by a model that uses the shortest-path routing approximation as compared to the original? And second, does this reduction of variables and constraints significantly reduce solution times? With this in mind, both the original model and the model using shortest-path routing only (versions with protection splitting allowed) were run on the test cases given in Table 7.5 to compare solution times and capacity efficiency. The NSFNET, Bellcore, and Smallnet topologies have been shown earlier in Figure 6.21. ARPA2 is shown in Figure 7.4.

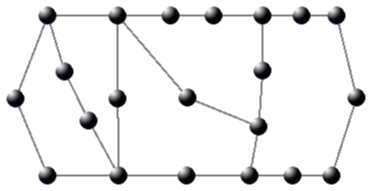


Figure 7.4: ARPA2 network topology

7.4.5 Results

A summary of the results is given in Table 7.6. The results show that, in the full network design case, the difference between full optimization and the shortest-path approximation is even more negligible than in the single ring case; the cost premium never exceeds 1%. Also, many designs show what seems to be a cost decrease; this would be impossible were the designs fully optimized, but in this case is simply because the shortest-path designs are able to be solved to a tighter mipgap in a shorter amount of time. Unfortunately, the CPLEX instance used to solve these problems is run on a shared processor, so accurate timing information is not available. However, it was consistently observed that the shortest-path approximation model reached designs closer to optimality in less time than the fully optimal model when solved under conditions of similar load on the CPLEX server.

7.4.6 Design Properties

We have seen in Sections 7.2 and 7.3 that shortest-path routing often approaches the efficiency of optimal routing, even in cases where it would seem to be at a disadvantage. The numerical results from the full network experiments seem to also bear out this assertion, with there being negligible difference between the shortest-path and fully optimized versions of the same design problem. How do we explain this feature? Qualitatively, it is intuitive to suggest that the wide number of possible cycle covers available to protect each demand makes it very likely that a configuration of rings exists in which every ring serves well-balanced load (the case in which shortest-path routing performs the best). To further investigate the details of this phenomenon, we looked more closely at the properties of our full network designs.

Test Case		Full Model		Shortest-Path Approximation		Cost Difference
		Cost	Mipgap	Cost	Mipgap	
NSFNET		340	1.17%	339	0.68%	-0.3%
Bellcore		366	1.27%	366	0.43%	0.0%
ARPA2		1637	0	1637	0	0.0%
Smallnet (small demand pattern)		104	3.85%	104	3.73%	0.0%
Smallnet (large demand pattern)		588	1.02%	588	0.37%	0.0%
15n30s1 family	16 spans	4450	0	4450	0	0.0%
	17 spans	3642	0	3642	0	0.0%
	18 spans	3352	0	3352	0	0.0%
	19 spans	3202	0	3202	0	0.0%
	20 spans	2977	0	2978	0	0.0%
	21 spans	2783	0	2788	0	0.2%
	22 spans	2538	0.07%	2542	0	0.2%
	23 spans	2348	0.07%	2356	0	0.3%
	24 spans	2236	0.24%	2250	0.06%	0.6%
	25 spans	2022	0.22%	2033	0	0.5%
	26 spans	1857	0.09%	1874	0	0.9%
	27 spans	1777	0.51%	1785	0.16%	0.5%
	28 spans	1686	0.59%	1692	0.10%	0.4%
	29 spans	1603	1.13%	1598	0.25%	-0.3%
	30 spans	1528	0.62%	1526	0.33%	-0.1%

Table 7.6: Comparison of optimal UPSR-like *p*-cycle designs to those solved using the shortest-path protection routing approximation method

7.4.6.1 Comparison to APS

One way to evaluate the effectiveness of the UPSR-like *p*-cycle designs is to compare them to 1+1 APS designs on the same network. Both architectures use dedicated protection with simultaneous transmission of working and protection signals; the only difference is that UPSR-like *p*-cycles are confined to arranging the backup paths into cyclical structures. Therefore UPSR-like *p*-cycles can only be strictly less efficient on the same network (given the same working routing), and an analysis of the cost gap between the two will reveal how closely UPSR-like *p*-cycle designs can approach their maximum efficiency limit, both when designed optimally and when designed under the shortest-routing only constraint.

Table 7.7 gives a comparison between 1+1 APS and UPSR-like *p*-cycles in the designs for the 15 node network family. The cost values for APS and UPSR-like *p*-cycles under shortest-path-only protection are given, as well as figures for the percentage increase in spare capacity over APS. The results for the fully optimal model are excluded, as they are so similar to the shortest-path results. The Table shows that the increase can be significant, but more so in networks with low nodal degree. In the more highly connected networks the cost premium is less than 5%.

The Table also gives some other metrics of interest. The "% Unused Spare Capacity" column measures how much of the capacity in the UPSR-like *p*-cycle designs is not used for protection paths. In any given UPSR-like *p*-cycle, some spans of the cycle may not be loaded to the entire capacity of the ring. This metric records how much of this spare capacity "slack" there exists in the entire network design. This is not a factor for a 1+1 APS design, because each APS structure is allocated to exactly fit the size of the protected working path. Taking this unused capacity into account, we can compute the "% Extra Used Capacity" metric that measures how much more capacity is used for protection in the UPSR-like *p*-cycle designs compared to the APS designs. This metric captures extra protection capacity in UPSR-like *p*-cycles due to the cycle-routing constraint. The Table shows that this increase is marginal over the entire range of nodal degrees, averaging only 3%.

The relatively low values for both the "unused spare capacity" metric and the "extra used capacity" metric together show that the optimizer is able to find a combination of cycles out of the vast number of possibilities that both evenly loads the cycles and also does

Network Spans	APS Spare Capacity	UPSR-Like *p*-Cycles (Shortest-Path Protection)			
		Cost	% Extra Cost Over APS	% Unused Spare Capacity *	% Extra Used Capacity Over APS
16	4028	4450	10.5%	8.0%	1.6%
17	3134	3642	16.2%	9.0%	5.2%
18	2950	3352	13.6%	8.9%	2.9%
19	2838	3202	12.8%	8.5%	2.4%
20	2701	2978	10.3%	5.5%	4.1%
21	2559	2788	8.9%	5.4%	2.6%
22	2352	2542	8.1%	4.4%	3.3%
23	2144	2356	9.9%	4.0%	5.5%
24	2080	2250	8.2%	3.7%	4.2%
25	1900	2033	7.0%	4.1%	2.6%
26	1773	1874	5.7%	2.3%	3.3%
27	1709	1785	4.4%	2.2%	1.6%
28	1607	1692	5.3%	2.9%	1.4%
29	1534	1598	4.2%	1.4%	2.6%
30	1472	1526	3.7%	0.7%	2.8%

Table 7.7: Comparison of UPSR-like *p*-Cycle design costs to the 1+1 APS lower bound

not detour the lengths of protection paths much beyond the shortest-path APS lower bound. This suggests that shortest-path routing may be used the vast majority of the time, even in the fully optimal design results, the same conclusion that we drew from our observations regarding the efficiency of shortest-path routing closely approaching the efficiency of optimal design. We will further investigate the frequency of shortest-path routing in the designs in the next Section.

7.4.6.2 Shortest-Path Routing Properties

All results up to this point have suggested that shortest-path protection routing for UPSR-like *p*-cycles is near-optimal. To confirm this, we directly analyzed the routing details of the designs generated by our full optimization model. The results in Table 7.8 provide an analysis of the frequency of shortest-path routing in these designs for the 15 node network family. The first column lists the networks from least to most connected. The next three columns provide the analysis for all protected demands. A column for the total amount of protection is provided because, as mentioned earlier, overprotection is possible, and therefore total protection may vary slightly between different designs. There are a total of 518 demand units to be protected in the demand pattern used, so we see that overprotection is rare.

The next two columns give the amount of shortest-path protection as both an absolute value and a percentage of all protection. We can see that shortest-path protection is extremely common in well connected networks but tapers to a small value in sparse networks. This is because sparse networks will not have many straddling protection opportunities, and thus most working paths will need to be protected as on-cycle paths, with their protection paths being routed along the longer side of a cycle. For example, in the most extreme case when the network has an average nodal degree of 2, there will only be a single cycle in the entire network, and all demands will need to have their protection routed along the longer side of the cycle (because working routing is done via shortest paths).

The next three columns take this effect into account by only reporting the results for demands that are protected as straddling paths (i.e., demands for which there actually exists a choice for protection routing). Here we can see that over all network designs, the shortest-path routing decision is very common. At least 85% of straddling-protected demand is

Network Spans	All Protection			Demand Routed as Straddlers		
	Total Protection	Amount Protected Shortest-Path	% Protected Shortest-Path	Total Protection	Amount Protected Shortest-Path	% Protected Shortest-Path
16	518	25	4.8%	0	0	———
17	518	60	11.6%	9	9	100.0%
18	518	68	13.1%	24	24	100.0%
19	519	76	14.6%	35	34	97.1%
20	518	95	18.3%	51	45	88.2%
21	518	175	33.8%	80	68	85.0%
22	518	244	47.1%	127	118	92.9%
23	518	255	49.2%	136	124	91.2%
24	518	290	56.0%	182	162	89.0%
25	519	326	62.8%	194	175	90.2%
26	518	425	82.0%	250	229	91.6%
27	518	460	88.8%	288	275	95.5%
28	518	471	90.9%	312	296	94.9%
29	520	486	93.5%	319	304	95.3%
30	518	482	93.1%	324	312	96.3%

Table 7.8: Analysis of shortest-path routing in fully optimized UPSR-like *p*-cycle designs

routed shortest-path in all of these designs. Also, it is possible that there exist equivalent-cost solutions to these in which shortest-path routing is even more common (because the current model does not prefer these solutions over other equal-cost designs explicitly).

Therefore we see that not only does the shortest-path protection routing simplification on our model not affect the resulting cost, but that fully optimized designs naturally tend to shortest-path protection routing. This shows conclusively that shortest-path routing is preferable in the vast majority of cases. This information should help simplify the design process for network engineers who wish to implement UPSR-like p-cycle protection.

7.4.6.3 Comparison to UPSR

Though we have gained insight into how to best design networks protected by UPSR-like p-cycles, the question remains how much of an advantage they really provide over regular UPSR. Evidently, there is the advantage of flexibility; UPSR-like p-cycles provide dedicated protection but do not require working paths to be routed along the ring like regular UPSR. This means that the design of the working and protection layers in the network can be decoupled, allowing both layers to be more agile and enabling more flexible dynamic service provisioning. But we would also like to have a quantitative comparison in terms of the gain in capacity efficiency that is also a result of this increased routing flexibility.

Obtaining such a comparison is difficult for several reasons. First of all, UPSR-like p-cycles have been treated as a SCP design problem only, while UPSR design is inherently JCP; working paths and rings are laid out simultaneously in order to support efficient ring loading. Also, because UPSR design is a relatively complex problem (due in part to its JCP nature), it is generally not solved using ILP methods but instead using metaheuristics such as tabu search. Therefore optimal UPSR designs are not available in general for realistically sized networks, making comparisons difficult. We could attempt to formulate a JCP version of the UPSR-like p-cycle model, but we have already seen that the SCP model can be difficult to solve, and a JCP version would be exponentially more so (similar to attempting regular UPSR design using ILP).

One comparison we can make with relative ease is to compare the results for UPSR-like p-cycle to those obtained via an SCP approach to UPSR design. As mentioned, this will not generally result in optimal UPSR designs, so such a comparison will not give a fair

representation of the two architectures in terms of optimal efficiency. However, this kind of comparison is useful in the sense that it will reveal the extent to which UPSR-like *p*-cycles benefit from their ability to protect non-co-routed working paths. Or, put another way, it will show how inefficient basic UPSR can be in comparison to UPSR-like *p*-cycles when used to protect an identically-routed working layer, in the case where the routing of this working layer may not be ideal for UPSR.

ILP Model

Creating a SCP UPSR model is quite simple. In fact, the UPSR *p*-cycle full network design model detailed in Section 7.4.1 can already function as a UPSR model, as long as the parameter set is pre-processed to eliminate all protection relationships that are not allowed under a UPSR framework (i.e., only keep protection relationships for working paths that are completely co-routed on the cycles that protect them). This pre-processing step will reduce the size of the parameter set, making the model easier to solve for UPSR than UPSR-like *p*-cycles.

Test Cases

This model was run on the 15 node network family with the same demand pattern as in the previous tests (Section 7.4.4).

Results

Table 7.9 shows that the cost premium of SCP-designed UPSR over SCP-designed UPSR-like *p*-cycles increases approximately linearly from 0 for the most sparse network to nearly 40% for the most highly connected. For the degree ~3 networks (22 and 23 spans) the premium is roughly 10%. We can understand this trend by the fact that as the network becomes less highly connected, the number of non-UPSR-like protection relationships decreases. In the extreme case where the entire network is simply a degree-2 cycle, the entire network simply becomes one large ring, and UPSR-like *p*-cycles reduces to the degenerate case of basic UPSR (all working paths are on the ring).

Note that, as stated previously, this is UPSR designed via SCP methods, and thus not representative of true UPSR efficiency (the costs obtained here give a loose upper bound on

Spans in Network	Cost (UPSR-like p-cycle)	Cost (UPSR SCP)	Cost Increase
16	4450	4450	0.0%
17	3642	3691	1.3%
18	3352	3466	3.4%
19	3202	3334	4.1%
20	2977	3152	5.9%
21	2783	3014	8.3%
22	2538	2780	9.5%
23	2348	2613	11.3%
24	2236	2529	13.1%
25	2022	2411	19.2%
26	1857	2262	21.8%
27	1777	2232	25.6%
28	1686	2206	30.8%
29	1603	2131	32.9%
30	1528	2129	39.3%

Table 7.9: Comparison of UPSR-like p-cycle designs to UPSR SCP designs

those for JCP UPSR designs). However, we would expect this effect to be present in the JCP design case as well, due to this same observation about graph connectedness. Therefore we can expect more significant gains from UPSR-like p-cycles in more highly connected networks. Indeed this is true for all mesh-like protection schemes. It is notable that this effect is relevant even in the case where the protection scheme in question is not shared. Therefore UPSR-like p-cycles are able to take advantage of mesh topologies without using sharing, which is usually considered to be an important feature of mesh-based protection.

7.4.6.4 Dual Failure Restorability

We began this Chapter by motivating UPSR-like p-cycles with an explanation of their potential for improving dual failure restorability, and specifically by noting their potential for providing guaranteed dual failure restorability for priority services. Therefore we now inspect the dual failure properties of the full-network UPSR-like p-cycle designs we have generated.

Method

Our span R_2 analysis method is similar to the method used in Section 6.2.5 for Project HAVANA. We use this method to compare two different policies. The first policy uses only the default single failure response to recover from failures. If a dual failure affects both a working path and its backup path, the path is not restored (and neither is any capacity on its backup path used). This applies to both on-cycle and straddling working paths. Under the second policy, we allow preemption of other cycle capacity to restore straddling paths in the case of a simultaneous working path and protection path failure. In other words, in each dual failure scenario, we first protect any straddling paths that are made unrestorable by the dual failure by preempting capacity around the opposite side of the cycle, and only then do we proceed to protect any other paths (straddling or on-cycle) affected by only one of the two span failures. On-cycle paths are of course unrestorable when hit by both failures, as they only have one possible protection path on the cycle. We compare the results of these two dual failure response policies in the following Section.

Results

The results are given in Figure 7.5 and Figure 7.6. The plotted values represent the percentage of demand units that are restorable over all dual failures. Figure 7.5 compares the results for both straddling and on-cycle paths separately under the two policies, whereas Figure 7.6 gives the total average R_2 for all paths in each case. There are no data points for straddlers in the 16 span (least connected) network because no demands are routed as straddlers in this design. In Figure 7.5 data points for straddlers use filled circles, while data points for on-cycle demands use unfilled circles. In both Figures, the data curves for the preemption policy uses the solid blue lines, while curves for the non-preemption policy use the dashed red lines.

We can see from Figure 7.5 that using the preemption policy greatly increases the average dual failure restorability of straddlers (especially for the more sparse networks) while not greatly affecting that of on-cycle demands. Further, Figure 7.6 shows that the preemption policy does not have a negative effect on total dual failure restorability; rather, overall average R_2 is increased by a small amount. Therefore not only does this policy allow us to increase R_2 for a specific set of priority demands, it also has a positive effect on R_2 in

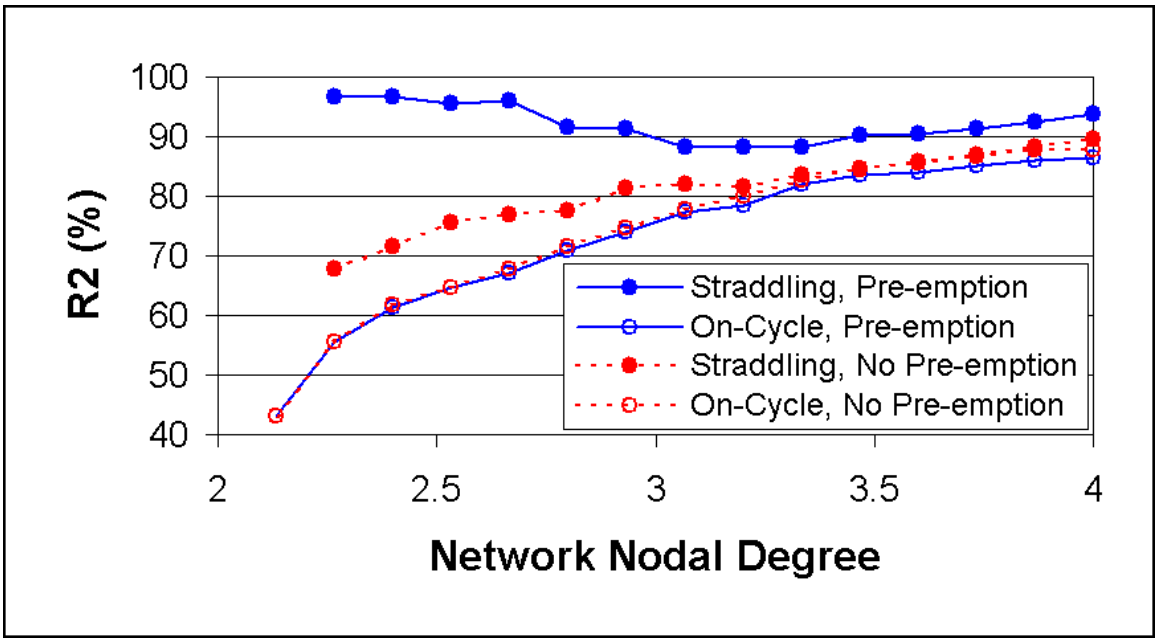


Figure 7.5: Average dual failure restorability for straddlers and on-cycle routed demands in UPSR-like *p*-cycle designs

general, which may make it worthwhile to adopt even when there is no need to implement a QoP class scheme. We can easily understand this effect based on the simple fact that the preemption policy essentially creates new, shared protection paths for straddling demands in certain scenarios, and thereby makes more use of the existing spare capacity to protect against failures.

However, even though we can use this policy to achieve a high degree of R_2 for straddling demands, we cannot use it to guarantee 100% dual failure restorability. This is because even though we allow straddlers to preempt the protection of on-cycle demands, there still exist certain dual failure scenarios in which two or more straddlers contend for the same capacity. In order to be able to guarantee full R_2 for a certain set of demands, we must go further in modifying the network design to accommodate them. We will do so in the next Section.

7.4.7 Conclusions

We have learned many things about the design of UPSR-like *p*-cycle-protected networks through the above exercises. First of all, we have seen that shortest-path routing is a useful simplifying assumption for UPSR-like *p*-cycle ILP models. It removes routing decisions,

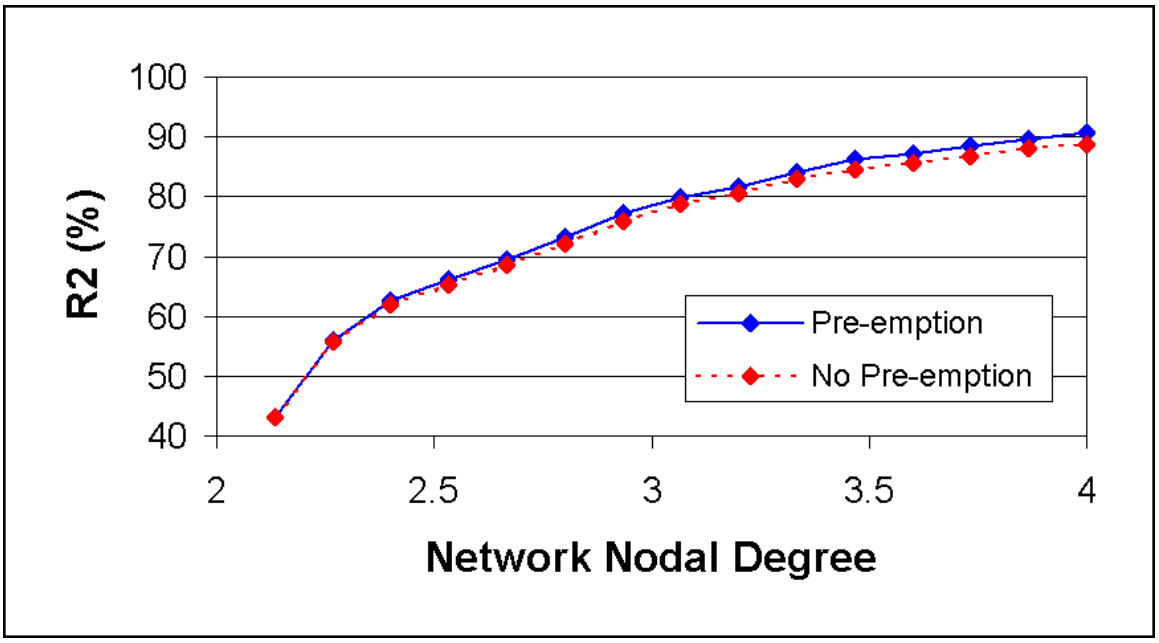


Figure 7.6: Average overall dual failure restorability for all paths in UPSR-like *p*-cycle designs

reducing the design problem to something akin to the *p*-cycle problem, where each cycle can provides a fixed amount of protection for a fixed configuration of working paths, given a fixed investment. It does this while carrying with it only a mild spare capacity penalty. Going on to compare UPSR-like *p*-cycles to other architectures, we first find that they are nearly as efficient as dedicated protection can be, averaging 5 to 10% more expensive than APS designs on the same network. The comparison to UPSR further shows that its *p*-cycle like adaptability to varying working layer routings gives it a significant flexibility advantage. Finally, we have seen that we can provide superior dual failure restorability for straddling demands by preempting other traffic on the *p*-cycle to allow for a second protection path. Going further to provide 100% R_2 for a select class of high priority demands without using preemption will be considered in following Section.

7.5 Quality of Protection Levels

So far we have only considered designing UPSR-like *p*-cycles for one type of protection: restorability of single span failures with dedicated protection paths for fast, UPSR-like protection switching. Our analysis of dual failure restorability has been confined only to

the R_2 obtainable from 100% span R_1 designs. The following Section first covers a multi-QoP approach to UPSR-like p-cycle design in which we define distinct single-failure and dual-failure recoverable classes and investigate how we can produce designs to satisfy these requirements. We then move on to consider QoP classes distinguished instead by speed of protection, and investigate how to design networks using such a scheme.

7.5.1 Dual Span Failures

The most straightforward way to guarantee protection from dual span failures using UPSR-like p-cycles is to allow a straddling path to use both halves of a cycle as dedicated protection paths (all three paths transmitting simultaneously). This allows up to 2 of the 3 disjoint paths to fail without affecting the integrity of the signal. As a bonus, it would also allow for improved error correction in the unfailed state (e.g., bit or word-based voting between the 3 received signals). Note that this approach would not work for demands that cannot be routed as straddlers, e.g., demands for which there does not exist a disjoint simple cycle between its end-points. However, such a demand could not be protected (in a dedicated manner) from dual failures in any case, because this means that two disjoint protection paths cannot be found at all.

A more flexible approach would be to allow any working path to be protected by any two disjoint protection paths, whether on the same cycle or on different cycles. However, note that in the same-cycle dual failure protection case, the working path reserves the entire cycle for protection (much the same as the hypothetical Z-case single failure protection scenario described in Section 7.3). In essence, this represents a perfectly loaded ring, the ideal scenario for this architecture. Therefore, even though the multi-ring option provides more flexibility, it is questionable whether or not it can improve over the simple single-cycle case.

Because the single-cycle dual failure approach reserves an entire cycle for the use of the protected path, dual-span failure restorable demands do not interact with other demands in the design problem. For any dual-span failure restorable demand, the optimal configuration is to use the shortest disjoint cycle between its endpoints as a UPSR-like p-cycle for protection. Therefore the design problem can proceed in two stages: first, find the shortest disjoint cycle between the endpoints of each dual-failure restorable class demand and allo-

cate a sufficient number of copies. Then, perform the normal UPSR-like p-cycle design on the remaining demands.

In contrast, allowing dual failure protection across multiple cycles is more complex to design for, as the model must take into account disjointness of each pair of protection paths in the overall cycle set (to ensure that a dual-failure protected demand is protected by two paths that cannot fail due to the same span failure). This is another reason for preferring the single-cycle approach. Therefore we performed tests to determine if the multi-cycle approach could demonstrate an advantage in terms of capacity efficiency that would justify this fundamental increase in complexity.

7.5.1.1 ILP Model

To support dual failure protection in our UPSR-like p-cycle ILP model, we must add to it another class of dual failure protected demands. The following modifications to the optimal UPSR-like p-cycle model (with splitting allowed) from Section 7.4.1 will allow it to solve for fully optimal dual failure protection for this class of demands.

ILP Model

Sets (new):

D_1 and D_2 These sets are subsets of the set D from the original model, consisting of all single failure protected and dual failure protected demands, respectively. $D_1 \bigcup D_2 = D$ and $D_1 \bigcap D_2 = \emptyset$.

Constraints:

Constraint 7.12 must be modified to apply to only demands in D_1, as it defines protection requirements for single failure protection only, i.e.:

$$\sum_{c \in C} \sum_{q \in P_r^c} u_r^{c,q} \geq d_r \quad \forall r \in D_1 \tag{7.16}$$

The following constraints must be added to handle protection for demands in D_2:

$$\sum_{c \in C} \sum_{q \in P_r^c} u_r^{c,q} \geq d_r \cdot 2 \quad \forall r \in D_2 \tag{7.17}$$

$$\sum_{c \in C} \sum_{q \in P_r^c \, s.t. \, i \in T_r^{c,q}} u_r^{c,q} \leq d_r \quad \forall r \in D_2, \forall i \in S \tag{7.18}$$

All other aspects of the model remain the same. These modifications separate the protection constraints into two categories; one for the single-failure restorable demand class and one for the dual-failure class. We are already familiar with constraint 7.16, its scope has just been limited to span R_1 class demands. Constraint 7.17 ensures that there is enough protection to protect each demand twice over. However, this alone is not sufficient, as we also must make sure that each half of this double protection is disjointly routed. This is what constraint 7.18 accomplishes. This is done by ensuring that, for every span in the network, the total amount of protection that crosses that span does not exceed the magnitude of the protected demand. This way, if a dual failure affecting both demand r and the span in question occurs, we know that there will be some other combination of routes in the network that provide at least d_r units of protection.

7.5.1.2 Experimental Setup

Multi-QoP protection introduces another degree of freedom into our test cases, as the network may now contain single failure and dual failure protected demands in any relative proportion. With this in mind, we compared the fully optimal model given above with the simple two-step method outlined at the beginning of Section 7.5.1 (i.e., R_2 demands separated out and allocated as straddlers to their shortest cycles, then R_1 demands protected via ILP as usual) over a range of R_1/R_2 demand ratios on the same network. The network chosen was the 15 node, 22 span network from the 15 node network family. This network was chosen because it was reasonably connected (average degree of ~3) and because the full UPSR-like p-cycle network design could be solved quickly.

The test cases were formed by starting with the all-R_1 case (as in previous experiments), and then "converting" 5 demand pairs at a time to R_2-protected demands. The result is not a series of tests in which the amount of actual R_2-protected demand volume increases linearly (because the magnitude of the 5 randomly chosen demands may not be consistent between each step), but it does provide us with a set of tests in which the amount of R_2-protected demand varies slowly from none to the all of the demand in this network. Overall there were

Number of R_2-Protected Demands	Optimal Design Cost	Mipgap	R_1 Alone	Mipgap	R_2 Alone (Single-Cycle Protection)	Single-Cycle R_2 Protection Total Design Cost	Cost Increase for Simplified Method
0	2538	0.07%	2538	0.07%	0	2538	0.00%
5	2664	0.04%	2455	0.04%	222	2677	0.49%
10	2851	0%	2348	0%	517	2865	0.49%
15	3030	0%	2217	0%	839	3056	0.86%
20	3152	0.04%	2160	0.08%	1031	3191	1.24%
25	3251	0.03%	2106	0.05%	1196	3302	1.57%
30	3366	0.07%	2042	0%	1382	3424	1.72%
35	3476	0.06%	1972	0%	1586	3558	2.36%
40	3627	0.04%	1882	0%	1850	3732	2.89%
45	3708	0%	1855	0%	1977	3832	3.34%
50	3849	0.03%	1767	0%	2217	3984	3.51%
55	3990	0.03%	1714	0%	2463	4177	4.69%

Table 7.10: Comparison of optimal R_1/R_2 QoP designs to designs using same-cycle protection for R_2-class demands

50 demand pairs out of 105 for which dual failure protection was impossible. Therefore the number of R_2-protected demands varies from 0 to 55, for a total of 12 test cases.

7.5.1.3 Results

Table 7.10 shows the results for the above test cases. Mipgap values are given for completeness, but they are all very low, corresponding at most to a possible variation in 1 or 2 units of capacity. The Table shows that, although the optimizer can find some better alternatives to single-cycle protection for R_2-class demands, the total cost difference between the two does not exceed 5%, even when fully half of the demands in the network require R_2 protection. This shows that the separable optimal R_1/single-cycle R_2 design method can be used to simplify UPSR-like p-cycle R_1/R_2 QoP designs without sacrificing much efficiency.

7.5.2 Protection Speed Classes

It is reasonable to assume that, even if the fast protection provided by UPSR-like p-cycles is desirable for some demands in the network, there will be some paths that carry traffic

of a less critical nature for which it is acceptable to have a brief delay inherent in switching actions typical of mesh-based strategies (such as those for a BLSR ring or *p*-cycle). Therefore it is of interest to consider how this type of protection might be implemented in UPSR-like *p*-cycle networks, including supporting both "fast" UPSR-like and "slow" mesh-like protection as different protection classes in the same network.

Because the capacity used for "fast protection" class demands is dedicated, it cannot be shared for use by this type of demand. Instead, a UPSR-like *p*-cycle that provides this class of protection must reserve some number of its channels for shared protection for any "slow-class" flows it protects. We assume that the ring can reserve a different number of shared channels for this purpose on every span, and that this reserved capacity, combined with the dedicated reserved capacity for UPSR-like operation, produces the per-span capacity requirements that together define the ring's size. Otherwise, slow-class flows would reserve entire unit *p*-cycles by themselves, and the design problem would be trivially separable into two design problems for *p*-cycles and UPSR-like *p*-cycles independently.

In the following investigation, we assume that the rings use regular *p*-cycle protection (i.e., span protection) for slow-class flows. FIPP-type *p*-cycle protection would also be possible, but introduces more complexity into the design problem (i.e., path disjointness constraints, requiring some type of DRS approach). A hybrid of UPSR-like *p*-cycle protection and span *p*-cycle protection offers opportunities for capacity savings, as the solver uses shared *p*-cycle capacity to "fill up" the capacity slack in the UPSR-like *p*-cycle design caused by unbalanced ring loading. However, since we have seen that the rings in UPSR-like *p*-cycle designs are in general loaded very evenly, we anticipated ahead of time that these gains would not be very great.

7.5.2.1 ILP Model

Again, we can modify the basic UPSR-like *p*-cycle model introduced in Section 7.4.1 to allow for *p*-cycle type protection. Again, we use the model with splitting allowed because of its lower complexity. The following changes must be made:

ILP Model

Sets (new):

D_f and D_s Set D_f is a subset of set D from the original model, consisting of all demands that receive fast-class protection. D_s is a new set of demands that represent the working capacity allocated to the working paths of the to slow-class demands (routed shortest-path), broken down into individual single-span demand pairs. This is explained in more detail below.

Variables:

σ_i^c The amount of shared spare capacity reserved on span i on cycle c.

Constraints:

Constraint 7.10 must be changed to accommodate the new type of capacity:

$$s_i^c + \sigma_i^c \leq h_c \quad \forall c \in C, \forall i \in S_c \tag{7.19}$$

This constraint sizes each ring according to a combination of both its shared (p-cycle) capacity and dedicated (UPSR-like p-cycle) capacity.

Constraint 7.11 must be modified to apply only to fast-class demands that use dedicated capacity:

$$s_i^c \geq \sum_{r \in D_f} \sum_{q \in P_r^c \, s.t. \, i \in T_r^{c,q}} u_r^{c,q} \quad \forall c \in C, \forall i \in S_c \tag{7.20}$$

The only change here is that D has been changed to D_f.

The following constraint must then be added:

$$\sigma_i^c \geq u_r^{c,q} \quad \forall c \in C, \forall r \in D_s, \forall q \in P_r^c, \forall i \in T_r^{c,q}$$

This is an equivalent to constraint 7.20 that calculates shared capacity requirements for slow-class demands. The sum disappears from the inequality because the capacity is shared between different demands on the same span. Therefore it is not defined by the sum, but rather the greatest of all capacity requirements for that cycle from all protection paths of all slow-class demands over that span.

To use this model to allocate p-cycle protection for slow-class demands, we only need recognize that, mathematically speaking, span protection is a special case of path protection when the paths are only one hop long. With this in mind, we form the set D_s,

containing $|S|$ demands, corresponding to a one-hop working path for every span in the network (these demands do not overwrite or replace any existing one-hop working paths from the original D). Then, we perform shortest-path working routing on the demands that belong to the slow-class of protection. The resulting span capacities are then assigned to their corresponding "single-span demands" in D_s as their demand volumes. The original end-to-end slow-class demands are then removed from the original D, as they are effectively replaced by the span-by-span equivalent demands in D_s. Then, as long as protection paths are properly computed for these single spans, the model will provide protection for these spans under regular p-cycle type protection, using the shared capacity allocated to each UPSR-like p-cycle. In other words, this is a method of "tricking" a path-based model into behaving like a span restoration model by creating a set of single-span working paths equivalent to shortest-path routed working capacity totals on each individual span.

7.5.2.2 Experimental Setup

The above model was used to create network designs across a range of ratios of slow/fast-class protected traffic. The 15 node, 22 span network topology from the 15 node family was again used. The test cases were formed by starting with the base case (in which all 105 demand pairs were protected with fast-class protection, i.e., the 15 node, 22 span case from Table 7.6), and "converting" 5 demand pairs at a time to slow-class protection, proceeding in this way until all 105 demand pairs were protected using regular p-cycles. The demand pairs converted in each step were randomized, and are therefore not the same as those chosen in the test cases for Section 7.5.1.2. The entire range (from 0 to 105 demand pairs) could be covered this time, as there are no topological limits on either class of protection. We call this method the "joint" method in the following discussion.

As a comparative benchmark, the same test cases were solved using a non-joint design method, i.e., the fast-class and slow-class demands were considered as completely separate working layers and designed independently as two totally separate ILP problems, and not allowed to share the use of the same rings. Such designs must be at least as expensive as the joint designs, as they are unable to take advantage of capacity synergies between the two protection types (e.g., capacity in one protection "layer" that is extraneous to that layer but can be used in a meaningful way by the other layer).

The non-joint results were computed in the following way. The UPSR-like p-cycle layer was solved using the regular UPSR-like p-cycle model described in Section 7.4.1 (or by using the hybrid model from this Section with all demands in D_s removed or scaled to zero). The p-cycle designs were solved first by using the joint model with all demands in D_f scaled to zero, and then checked against results obtained from the traditional p-cycle SCP model given the same working capacity distribution. These results were found to agree in all cases, validating the p-cycle protection portion of the hybrid model. However, the p-cycle SCP model was found to solve faster and was able to be quickly solved to strict optimality in all cases, while the hybrid model could not be. This is understandable, as the hybrid model solves for additional information that is extraneous to the pure p-cycle case (e.g., the specific protection paths used for every span on every protecting cycle).

7.5.2.3 Results

The results shown in Table 7.11 indicate that the optimal hybrids are all but identical in cost to the designs solved as disjoint problems. In fact, some cases show a cost decrease in the disjoint case. This is impossible assuming true optimality of the solutions (any valid non-jointly designed hybrid is a valid solution of the joint model as well), but the discrepancy in this case occurs because the non-joint designs are able to be solved to a tighter mipgap.

This result strongly suggests that it may be not only a trend but also a provable statement that the joint model cannot improve at all on a design solved non-jointly. However, no obvious proof is forthcoming at this point. Indeed, it is intuitive that an optimal p-cycle design that effectively wastes minimal capacity would not be able to offer any spare capacity synergies to a UPSR-like p-cycle design, but this does not strictly rule out the possibility of ring-sharing between the two architectures to achieve capacity savings.

We performed an inspection exercise on the above designs to determine how much of this ring-sharing was actually occurring. First of all, the designs were re-solved using a slightly modified model in which constraint 7.12 was replaced by a slightly modified constraint with the inequality replaced by a strict equality (so that the designs would not contain any overprotection), using the existing designs as a starting point for the solver. Then, each ring was analyzed to determine if the spare capacity used by the two classes of demands was separable into disjoint UPSR-like p-cycles and regular p-cycles which

Number of Demands Using *p*-Cycle Protection	**Joint Design Cost**	**Mipgap**	**UPSR-like *p*-Cycle Cost Alone**	**Mipgap**	***p*-Cycle Cost Alone**	**Total Non-Joint Design Cost**	**Increase (Absolute Cost)**
0	2538	0.06%	2538	0.06%	0	2538	0
5	2541	0.10%	2491	0.09%	50	2541	0
10	2462	0.13%	2363	0.05%	97	2460	-2
15	2406	0.13%	2228	0	176	2404	-2
20	2399	0.12%	2101	0	296	2397	-2
25	2244	0.05%	1916	0	328	2244	0
30	2198	0.09%	1860	0.05%	339	2199	1
35	2156	0.13%	1774	0	382	2156	0
40	2099	0.05%	1663	0	436	2099	0
45	2043	0.10%	1560	0.06%	483	2043	0
50	1969	0.08%	1440	0	529	1969	0
55	1895	0.10%	1272	0	623	1895	0
60	1834	0.10%	1194	0	640	1834	0
65	1814	0.08%	1109	0	705	1814	0
70	1790	0.10%	1068	0	722	1790	0
75	1735	0.06%	885	0	850	1735	0
80	1668	0%	689	0	979	1668	0
85	1594	0.08%	531	0	1063	1594	0
90	1512	0%	436	0	1076	1512	0
95	1449	0.17%	332	0	1117	1449	0
100	1273	0%	153	0	1120	1273	0
105	1152	0.12%	0	0	1152	1152	0

Table 7.11: Comparison of optimal hybrid UPSR-like *p*-cycle/regular *p*-cycle designs to non-jointly solved hybrid designs

had the same combined ring size as the original hybrid ring. It turns out that most of the designs are almost entirely separable in this way. In each of the designs, there is at most one (topological) ring that is shared in a non-separable way. Separating out the *p*-cycle and UPSR-like components of each ring (allocating new rings in these non-separable cases because of the removed ring-sharing component) increases the cost of the designs by at most 4% in the worst case, and only by 1% on average over all test cases. This is on top of the fact that the results show that separable designs can be computed via separate ILP models for no increase in capacity whatsoever.

Perhaps efficiency gains through joint design are not impossible at all, but simply an exceedingly rare event, given how tightly balanced each layer can be designed individually. We cannot determine this from the tests above. What we can conclude, however, is that the joint model offers little, if any, benefit over a disjoint approach. Furthermore, we can say that hybrid designs themselves also hold little appeal in terms of capacity benefits. This means that network operators have the option of treating their network as two independent protected networks (for *p*-cycle and UPSR-like protected services respectively), even using equipment from different vendors for the unique needs of both architectures if desired, without incurring extra capacity costs.

7.5.3 Conclusions

In this Section we have developed methods for extending UPSR-like *p*-cycles to two different QoP scenarios: combined single/dual span failure protection, and combined fast/slow protection switching classes. These exercises have shown that designing for these scenarios can be accomplished easily using a separable ILP model approach. For dual failures, a simple approach in which dual-failure restorable demands are protected by a double-redundant ring (essentially 2+1 APS) produces near-optimal results, and a relatively simple ILP model can be used for true optimization if necessary. For demands that do not require fast protection, a *p*-cycle hybrid with ring sharing was considered, and it was found that the optimal design in this case can be obtained by designing the fast and slow layers as independent networks.

7.6 Summary and Future Work

In this Chapter we have considered many variations on the design and operation of UPSR-like *p*-cycles. We began with an ILP model for finding optimal solutions to the simple single-cycle design case, but found that it was possible to obtain near-optimal results in most cases by following a simple shortest-path routing policy. This insight helped us in the next Section when we found that the complexity of the whole-network design problem was high when we allowed a choice for protection routing along either side of the cycle. Instead, we developed a simplified model that enforced shortest-path routing while still allowing selection of the protection cycle for each working path, and found that the results compared favorably to other dedicated protection architectures (APS and UPSR). We also demonstrated that these UPSR-like *p*-cycle designs could provide significantly enhanced R_2 to paths protected as straddlers.

We then investigated the possibility of providing different QoP classes to demands in a UPSR-like *p*-cycle network. Our study of dual span failure protection showed that it was possible to attain near-optimal designs that provide R_2 protection for a subset of demands by simply protecting these demands with dedicated cycles on which protection is transmitted both ways around the cycle (essentially 2+1 APS). Then, our study of protection speed classes showed that it was possible to attain near-optimal designs that provide "fast" (1+1 APS-style) and "slow" (shared mesh speeds) protection switching classes by designing hybrid UPSR-like *p*-cycle/regular *p*-cycle networks via a separated, two-step design approach.

As with any new networking concept, there are many opportunities for expanded work on several topics. We present a few such ideas here.

7.6.1 Single Failure Protection for the Z-Case

We have mentioned that the so-called "Z-case" protection is technically possible with UPSR-like *p*-cycles if both sides of the ring are used to transmit 2 simultaneous protection signals, effectively using the dual-failure protection mechanism to guarantee single span failure protection. Dual span failure protection would be guaranteed only in cases where the working path fails at a point that is not also on the cycle. We have excluded this

case from the study because its inherent capacity inefficiency can be appreciated a priori, but the question remains as to what effect, if any, its inclusion might have on design cost. Also, it may have benefits in other arenas, e.g., using the three transmitted streams in the non-failed case to perform advanced error correction.

7.6.2 Hybrid with FIPP *p*-Cycles

We have investigated the scenario where demands that do not require the near-instant-switching capabilities of UPSR-like *p*-cycles are protected using the span-protecting *p*-cycle mechanism with a bank of shared spare capacity that lies on the same topological rings. In another scenario, although these demands may not require fast protection switching, it may still be desirable to switch them as entire end-to-end paths. For this, we can easily envision a similar hybrid in which the shared protection mechanism is FIPP *p*-cycles instead of span *p*-cycles. The design problem for this case is more complex as the disjointness of working paths that share the same spare capacity must be ensured, requiring a shift of the model to a DRS-like approach.

7.6.3 JCP Design for UPSR-like *p*-Cycles

So far we have only considered SCP ILP models for designing UPSR-like *p*-cycle-protected networks. However, as mentioned above, UPSR networks have traditionally been designed using JCP methods. This is mainly due to the necessity of routing working paths on the rings themselves, as UPSR are fundamentally structures that require coordinated planning of both the working and protection layers of the network. Although UPSR-like *p*-cycles do not have this limitation, it would still be of interest to explore JCP methods, both to determine the possible gains in capacity efficiency and also to allow a more "apples to apples" comparison to UPSR. Investigations along this avenue may involve either JCP ILP models, or custom JCP metaheuristics, in the same way that the complexity of the UPSR design problem has lead to design approaches such as tabu search.

Chapter 8

Closing Discussion

8.1 Summary of Thesis

The objective of this thesis was to deepen the body of knowledge on strategies for efficient protection in transparent optical networks and to propose and characterize several new architectures designed for this purpose. We began with a survey of existing protection strategies with an eye towards identifying those that satisfy the preconnection constraint. We then outlined a number of new schemes and described the existing literature related to them. These schemes included PXTs, span p-trees, and path p-trees.

We first performed a thorough analysis of span p-trees, developing a method to produce optimal span p-tree designs and then using this tool to compare span p-trees with other span-protecting architecture (p-cycles and p-segments). In the process we gained an understanding of how trees, segments, and cycles are fundamentally able to protect against failed spans in an efficient way. The key finding here was that tree protection in a span-protecting context is fundamentally inferior to segment-based protection, which is in turn inferior to cycle-based protection.

We then turned to PXTs, beginning with an analysis of the state-of-the-art design algorithm. A detailed analysis of the results of the algorithm showed a weakness in the high complexity of the resulting structures, the implication being that the algorithm could not be used to design PXT networks for transparent optical networks without wavelength conversion capability. We then proposed our own design approach based on ILP methods and showed that it was able to produce designs of comparable efficiency with greatly improved PXT characteristics, both when used as an offline or online algorithm.

We then proceeded to examine tree protection in a path-protecting context with path p-trees. We saw that the existing DRS method was not suitable to path p-tree design and introduced changes that allowed us to obtain more meaningful results. We then used this method to analyze the properties of path p-trees, both as a standalone architecture and in conjunction with FIPP p-cycles. The results showed that trees were not efficient structures in general, but could be used to greater effect in path-protecting designs than p-trees in a span-protecting context. Combined with our span p-tree results, these findings cemented the efficiency hierarchy of cycles over trails over trees.

Following this was an outline of HAVANA, a co-operative research project with researchers at Nokia Siemens Networks that investigated the implications of many constraints on transparent optical networks on a variety of protection architectures. We presented the results for PXTs, along with comparative results for other architectures from other researchers. The results showed that PXTs tended to strike a middle ground between cost-effectiveness and flexibility in the face of constraints on structure length, wavelength continuity constraints, and requirements for enhanced availability such as node failure restorability or dual failure restorability. We then presented a related study demonstrating a method for introducing failure localization capability into p-cycle networks through hybridization with m-cycles.

Finally, we introduced the concept of UPSR-like p-cycles, an architecture that uses dedicated protection to provide fast and simple protection switching while still taking advantage of the mesh topology to achieve savings over regular UPSR. We developed ILP-based design methods for the architecture with complexity such that near-optimal designs could be obtained in a reasonable amount of time. Results showed that we can achieve designs that are comparable to APS in terms of efficiency and noticeably better than SCP-based UPSR designs. We also showed how UPSR-like p-cycles could be used to provide enhanced R_2 to a class of high priority services, and then developed methods for implementing QoP classes with differentiated availability and protection speeds in UPSR-like p-cycle networks.

8.2 Main Contributions

Overall this thesis has contributed significantly to the field of network design in the area of preconnected protection for transparent optical transport networks. Our contributions can be summarized as follows:

- Improved understanding of the optimality of the existing pre-cross-connected trail design algorithm and properties of the designs produced with it.

- Development of a new design method for efficient pre-cross-connected trail networks with improved structural properties.

- Development of design methods for capacity-efficient span p-tree and path p-tree under computational limitations.

- Development of design methods for PXTs under constraints on wavelength continuity, path length, and node failure restorability.

- Development of a design method for the low-cost integration of failure localization capabilities into p-cycle networks.

- Improved knowledge of the relative capacity efficiencies of cycles, trails, and trees (both span-protecting and path-protecting).

- Improved knowledge of hybrid designs and the individual roles of cycles, trails, and trees in hybrid protection.

- Introduction of the UPSR-like p-cycle concept and the development of fast and efficient design methods for UPSR-like p-cycles.

- Characterization of UPSR-like p-cycle designs in terms of efficiency and enhanced R_2 properties.

- Development of design methods for differentiated QoP levels for services in UPSR-like p-cycle networks.

8.3 Other Contributions

The Ph.D. work associated with this thesis also resulted in the following publications, patents, and technical reports, which are given in chronological order.

8.3.1 Journal Papers

1. A. Grue, W. D. Grover, "Characterization of pre-cross-connected trails for optical mesh network protection," Journal of Optical Networking, vol. 5, no. 6, June 2006, pp. 493-508.

2. A. Grue, W. D. Grover, "Improved method for survivable network design based on pre-cross-connected trails," Journal of Optical Networking, vol. 6, no. 2, February 2007, pp. 200-216.

3. A. Grue, W. D. Grover, "Comparison of *p*-Cycles and *p*-Trees in a Unified Mathematical Framework," *Photonic Network Communications*, vol. 14, no. 2, October 2007, pp. 123-134.

4. W. D. Grover, A. Grue, "Self-Fault Isolation in Transparent *p*-Cycle Networks: *p*-Cycles as Their Own *m*-Cycles," *IEEE Communications Letters*, vol. 11, no. 12, December 2007, pp. 1004-1006.

8.3.2 Peer-Reviewed Conference Papers

1. A. Grue, W. D. Grover, M. Clouqueur, D. Schupke, J. Doucette, B. Forst, D. Onguetou, D. Baloukov, "Comparative Study of Fully Pre-Cross-Connected Protection Architectures for Transparent Optical Networks," *Proceedings of the 6th International Workshop on Design of Reliable Communication Networks (DRCN 2007)*, La Rochelle, France, 7-10 October 2007, pp. 1-8.

2. A. Grue, W. D. Grover, M. Clouqueur, D. Schupke, D. Baloukov, D. Onguetou, B. Forst, "CAPEX Costs of Lightly Loaded Restorable Networks Under a Consistent WDM Layer Cost Model," *Proceedings of the IEEE International Conference on Communications (ICC 2009)*, Dresden, Germany, 14-18 June 2009.

3. A. Grue, W. Grover, "UPSR-like *p*-Cycles: A New Approach to Dual Failure Protection," to appear in the *Proceedings of the Workshop on Reliable Networks Design and Modeling (RNDM 2009)*, St. Petersburg, Russia, 12-14 October 2009.

8.3.3 Patents Pending

1. W. Grover, A. Grue, "Self-Fault Isolation in Transparent p-Cycle Networks", US Patent Application No. 12/204,564, TRLabs, submitted September 4, 2008.

8.3.4 Technical Reports and Presentations

1. A. Grue, B. Forst, D. Onguetou, D. Baloukov, J. Doucette, A. Kodian, W. D. Grover, "Project HAVANA: Comparative Study of Fully Pre-cross-connected Protection Architectures for High Availability Transparent Optical Networking," TRLabs Technical Report ST-06-01, Edmonton, Canada, 9 November 2006.

2. A. Grue, B. Forst, D. Onguetou, D. Baloukov, J. Doucette, A. Kodian, W. D. Grover, First-year Slide Decks Produced for Nokia Siemens Networks, Project HAVANA, 2006

3. A. Grue, B. Forst, D. Onguetou, D. Baloukov, W. D. Grover, Second-year Slide Decks Produced for Nokia Siemens Networks, Project HAVANA, 2007.

4. A. Grue, W. D. Grover, J. Doucette, B. Forst, D. Onguetou, D. Baloukov, "High Availability Network Architectures (HAVANA): Comparative Study of Fully Pre-Cross-Connected Protection Architectures for Transparent Optical Networks", Paper presentation at DRCN 2007, La Rochelle, France, 10 October 2007.

5. A. Grue, W. D. Grover, B. Forst, D. Baloukov, D. Onguetou, "High Availability Network Architectures (HAVANA): Application of the NOBEL Cost Model", Invited talk given to Nokia Siemens Networks, Munich, Germany, 19 September 2008.

6. A. Grue, W. D. Grover, B. Forst, D. Baloukov, D. Onguetou, "High Availability Network Architectures (HAVANA): Overview and Wrap-up", Presentation given to Nokia Siemens Networks, Edmonton, Canada, 10 December 2008

7. A. Grue, B. Forst, D. Onguetou, D. Baloukov, W. D. Grover, Third-year Slide Decks Produced for Nokia Siemens Networks, Project HAVANA, 2008.

8.4 Future Research Directions

8.4.1 Segment-Protecting Architectures

The work in this thesis has focused on architectures that either protect individual failed spans or entire end-to-end paths. However, there does exist an intermediate possibility, called *segment protection*, in which working paths can be divided up into sub-segments of arbitrary length, each with its own protection path. This approach is very complex, both computationally and operationally, because it is a general approach that incorporates span and path protection as special cases as well as the entire range in-between. If a way was found to obtain near-optimal designs for segment-protecting preconnected architectures, however, it would reveal much about how the efficiencies of different types of preconnected structures are affected by the length of the protected segments.

8.4.2 Design Using Joint Capacity Placement

For the most part, this thesis has limited itself to the study of spare capacity placement formulations only, i.e., we have assumed fixed shortest-path routing of demands. This disregards situations where it might be possible to coordinate the routing of the working and protection layers to achieve some benefit, such as higher availability, lower capacity utilization, etc. One possible avenue of future research would be to investigate the design problems from this thesis in the context of joint capacity placement and determine what effect this would have on the efficiency of preconnected protection structures (especially the less efficient ones such as p-segments and p-trees).

8.4.3 Improved Design Methods

This category serves as a catch-all for any research that would enable the solution of more complete design problems. We have seen numerous times throughout this thesis that computational limitations prevent us from obtaining truly optimal designs, which can limit the certainty of our conclusions. We have taken pains to work around these limitations

whenever possible, but some problems remain beyond our grasp (e.g., completely optimal path *p*-tree solutions). One promising method is column generation [JaRo07], which avoids the need to generate large sets of candidate structures. Other possibilities include custom heuristics or metaheuristics, such as the *genetic algorithm* ILP approach (GA-ILP) [OnGr08].

Bibliography

[BaGr07] D. Baloukov, W. D. Grover, A. Kodian, "Towards Jointly Optimized Design of Failure Independent Path Protecting p-Cycle Networks," *OSA Journal of Optical Networking*, vol. 6, no. 12, December 2007, pp. 62-79.

[Bhan99] R. Bhandari, *Survivable Networks: Algorithms for Diverse Routing*, Kluwer Academic Publishers, 1999.

[BlWa02] U. Black, S. Waters, *SONET and T1: Architectures for Digital Transport Networks*, 2nd Edition. Upper Saddler River, New Jersey, Prentice Hall, 2002.

[Brua92] R. A. Brualdi, *Introductory Combinatorics*, 2nd Edition, Englewood Cliffs, New Jersey: Prentice-Hall, 1992.

[ChCh04] T. Y. Chow, F. Chudak, A. M. Ffrench, "Fast Optical Layer Mesh Protection Using Pre-Cross-Connected Trails," *IEEE/ACM Transactions on Networking*, vol. 12, no. 3, June 2004, pp. 539-547.

[Clou04] M. Clouqueur, *Availability of service in mesh-restorable transport networks*, Ph.D. Dissertation, University of Alberta, Edmonton, Canada, Spring 2004.

[ClGr02] M. Clouqueur, W. D. Grover, "Availability analysis of span-restorable mesh networks," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 4, May 2002, pp. 810-821.

[ClGr05] M. Clouqueur, W. D. Grover, "Availability analysis and enhanced availability design in *p*-cycle-based networks," *Photonic Network Communications.*, vol. 10, no. 1, July 2005, pp. 55-71.

[Craw93] D. Crawford, "Fiber Optic Cable Dig-ups: Causes and Cures," *Network Reliability: A Report to the Nation - Compendium of Technical Papers*, National Engineering Consortium, Chicago, USA, June 1993.

[Dant63] G. Dantzig, *Linear Programming and Extensions*, Princeton, New Jersey, Princeton University Press, 1963.

[Douc04] J. Doucette, *Advances on Design and Analysis of Mesh-Restorable Networks*, Ph.D. Dissertation, University of Alberta, Edmonton, Canada, Winter 2004.

[Flan90] T. Flanagan, "Fiber network survivability," *IEEE Communications Magazine*, vol. 28, no. 6, June 1990, pp. 46-53.

[Fors09] B. Forst, *Analysis and Understanding of Demand-wise Shared Protection*, M. Sc. Thesis, University of Alberta, Edmonton, Canada, Spring 2009.

[FoGr07] B. Forst, W. D. Grover, "Factors Affecting the Efficiency of Demand-wise Shared Protection," *Proceedings of the 6th International Workshop on Design of Reliable Communication Networks (DRCN 2007)*, La Rochelle, France, 7-10 October 2007.

[FoGa03] R. Fourer, D. M. Gay, B. W. Kernighan, *AMPL: A Modeling Language For Mathematical Programming*, 2nd Edition. Pacific Grove, California, Brooks/Cole – Thompson Learning, 2003.

[Free02] R. L. Freeman, *Fiber-Optic Systems for Telecommunications*. New York: Wiley, 2002.

[GaJo79] M.R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, 1979.

[GrCo03a] A. Groebbens, D. Colle, S. D. Maesschalck, I. Lievens, M. Pickavet, P. Demeester, "Efficient protection in MPλS networks using backup trees: part one–concepts and heuristics," *Photonic Network Communications*, vol. 6, no. 3, November 2003, pp. 191-206.

[GrCo03b] A. Groebbens, D. Colle, S. D. Maesschalck, I. Lievens, M. Pickavet, P. Demeester, "Efficient protection in MPλS networks using backup trees: part two–simulations," *Photonic Network Communications*, vol. 6, no. 3, November 2003, pp. 207-222.

[Grov87] W. D. Grover, "The selfhealing network: A fast distributed restoration technique for networks using digital cross-connect machines," *Proceedings of the Global Telecommunications Conference (GLOBECOM '87)*, Tokyo, Japan, 15-18 November 1987, pp. 1090-1095.

[Grov89] W. D. Grover, *Selfhealing Networks - A Distributed Algorithm for k-shortest link-disjoint paths in a multi-graph with applications in realtime network restoration*, Ph.D. Dissertation, University of Alberta, Edmonton, Canada, Fall, 1989.

[Grov94] W. D. Grover, "Distributed Restoration of the Transport Network," *Network Management into the 21st Century*, editors T. Plevyak, S. Aidarous, IEEE / IEE Press co-publication, ISBN 0-7803-1013-6, Chapter 11, pp. 337-417, February 1994.

[Grov03] W. D. Grover, *Mesh-Based Survivable Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking*. Upper Saddle River, NJ: Prentice-Hall PTR, 2003.

[GrCl05] W. D. Grover, M. Clouqueur, "Span-restorable mesh networks with multiple quality of protection (QoP) service classes," *Photonic Network Communications (Kluwer)*, vol. 9, no. 1, January 2005, pp. 19-34.

[GrDo02] W. D. Grover, 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, January 2002, pp. 47-61.

[GrMa98] W. D. Grover, M. H. MacGregor, "Method for Preconfiguring a Network to Withstand Anticipated Failures," U. S. Patent 5850505, December 15, 1998.

[GrMa94] W.D. Grover, M. H. MacGregor, "On the potential for spare capacity preconnection to reduce crossconnection workloads in mesh-restorable networks," *Electronics Letters*, vol. 30, no. 3, 3 February 1994, pp. 194-195.

[GrSt98] W.D. Grover, D. Stamatelakis, "Cycle-oriented distributed pre-configuration: ring-like speed with mesh-like capacity for self-planning network restoration," *Proceedings of the IEEE International Conference on Communications (ICC '98)*, Atlanta, GA, 8-11 June 1998, pp. 537-543.

[GrKo05] C. G. Gruber, A. M. C. A. Koster, S. Orlowski, R.Wessäly, A. Zymolka, "A Computational Study for Demand-wise Shared Protection," *Proceedings of the 5th International Workshop on Design of Reliable Communication Networks (DRCN 2005)*, Island of Ischia, Italy, 16-19 October 2005, pp. 421-428.

[GuLe06] M. Gunkel, R. Leppla, M. Wade, A. Lord, D. Schupke, G. Lehmann, C. Fürst, S. Bodamer, B. Bollenz, H. Haunstein, H. Nakajima, J. Martensson, "A Cost Model for the WDM Layer," *Proceedings of the International Conference on Photonics in Switching (PS 2006)*, Herakleion, Crete, Greece, 16-18 October 2006, pp. 1-6.

[HeSo07] W. He, A.K. Somani, "Comparison of Protection Mechanisms: Capacity Efficiency and Recovery Time," *Proceedings of the IEEE International Conference on Communications (ICC 2007)*, Glasgow, Scotland, 24-28 June 2007, pp.2218-2223.

[Herz94] M. Herzberg, S. Bye, "An optimal spare-capacity assignment model for survivable networks with hop limits," *Proceedings of the Global Telecommunications Conference (GLOBECOM '94)*, San Francisco, CA, USA, November 1994, pp. 1601-1607.

[HuGu08] R. Huelsermann, M. Gunkel, C. Meusburger, D. Schupke, "Cost modeling and evaluation of capital expenditures in optical multilayer networks," *OSA Journal of Optical Networking*, vol. 7, no. 9, September 2008, pp. 814-833.

[ILOG09] ILOG, "ILOG CPLEX," accessed 29 July 2009, available online: http://www.ilog.com/products/cplex/, 2009.

[JaRo07] B. Jaumard, C. Rocha, D. Baloukov, W. D. Grover, "A column generation approach for design of networks using path-protecting p-cycles," *Proceedings of the 6th International Workshop on Design of Reliable Communication Networks (DRCN 2007)*, La Rochelle, France, 7-10 October 2007.

[Kama06] A. Kamal, "1+N Protection in Mesh Networks Using Network Coding over *p*-Cycles," *Proceedings of the Global Telecommunications Conference 2006 (GLOBECOM 2006)*, San Francisco, CA, USA, November 2006, pp. 1-6.

[Kart00] S. V. Kartalopoulos, *Introduction to DWDM Technology: Data in a Rainbow*. IEEE Press, Piscataway, NJ, 2000.

[KiLu04] S. Kim, S. Lumetta, "Capacity-Efficient Protection with Fast Recovery in Optically Transparent Mesh Networks," *Proceedings of the First International Conference on Broadband Networks (BROADNETS 2004)*, San Jose, CA, USA, October 2004, pp. 290-299.

[KoGr05] A. Kodian, W. D. Grover, "Failure-Independent Path-Protecting p-Cycles: Efficient and Simple Fully Preconnected Optical-Path Protection," *Journal of Lightwave Technologies*, vol. 23, pp. 3241-3259, October 2005.

[KoGr05a] A. Kodian, W. D. Grover, J. Doucette, "A Disjoint Route Sets Approach to Design of Failure-Independent Path-Protecting p-Cycle Networks," *Proceedings of the 5th International Workshop on Design of Reliable Communication Networks (DRCN 2005)*, Island of Ischia, Italy, 16-19 October 2005, pp. 231-238.

[KoGr05b] A. Kodian, W. D. Grover, "Multiple-quality of protection classes including dual-failure survivable services in p-cycle networks," *Proceedings of the 2nd International Conference on Broadband Networks (BROADNETS 2005)*, Boston, MA, USA, 3-7 October 2005, vol. 1, pp. 231-240.

[KoZy03] A. M. C. A. Koster, A. Zymolka, M. Jäger, R. Hülsermann, C. Gerlach, "Demand-wise Shared Protection for Meshed Optical Networks," *Proceedings of the 4th International Workshop on Design of Reliable Communication Networks (DRCN 2003)*, Banff, Canada, 19-22 October 2003, pp. 85-92.

[KoZy05] A. M. C. A. Koster, A. Zymolka, M. Jäger, R. Hülsermann, "Demand-wise Shared Protection for Meshed Optical Networks," *Journal of Network and Systems Management*, vol. 13, no. 1, March 2005, pp. 35-55.

[LaSt02] T. Lan, K. Steenhaut, A. Nowe, "Shared backup tree protection in MPλS networks," *Proceedings of the 8th International Conference on Communication Systems (ICCS 2002)*, Amsterdam, Netherlands, 25-28 November 2002, vol. 2, pp. 1237-1241.

[LiWa06] T. Li, B. Wang, "On Optimal *p*-Cycle-Based Protection in WDM Optical Networks With Sparse-Partial Wavelength Conversion," IEEE Transactions on Reliability, vol. 55, no. 3, September 2006, pp. 496-506.

[LiHa06] Y. Lin, H. S. Hamza, J. S. Deogun, "Path-based Protection in WDM Networks with Differentiated Quality-of-Protection," *Proceedings of the IEEE International Conference on Communications (ICC 2006)*, Istanbul, Turkey, 11-15 June 2006, vol. 6, pp. 2489-2494.

[LiRu04] C. Liu, L. Ruan, "Finding good candidate cycles for efficient p-cycle network design," *Proceedings of the International Conference on Computer Communications and Networks (ICCCN 2004)*, Chicago, IL, USA, 11-13 October 2004, pp. 321-326.

[LiYa03] H. Liu, O. Yang, S. Shah-Heydari, "A Tree-Based Link Protection Algorithm," *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE 2003)*, 4-7 May 2003, vol. 2, pp. 939-942.

[MaGr97] M. MacGregor, W. Grover, K. Ryhorchuk, "Optimal Spare Capacity Preconfiguration for Faster Restoration of Mesh Networks," *Journal of Network and Systems Management*, vol. 5, no. 2, June 1997, pp. 159-171.

[Maed98] M. W. Maeda, "Management and Control of Transparent Optical Networks," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 16, no. 7, September 1998, pp. 1005-1023.

[MaHa07] E. D. Manley, H. S. Hamza, J. S. Deogun, "On The Bandwidth Efficiency of Pre-Crossconnected Trails", *Proceedings of the IEEE International Conference on Communications (ICC 2007)*, Glasgow, Scotland, 24-28 June 2007, pp. 2294-2299.

[MeFi99] M. Médard, S. G. Finn, R. A. Barry, "Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs," *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, October 1999, pp. 641-652.

[MoGr98] G. D. Morley and W. D. Grover, "A Comparative Survey of Methods for Automated Design of Ring-based Transport Networks," *Technical Report TR-97-04*, TRLabs (1998).

[Mukh06] B. Mukherjee, *Optical WDM Networks (Optical Networks)*. Springer-Verlag New York, Inc., Secaucus, NJ, 2006.

[OnGr08] D. P. Onguetou, W. D. Grover, "Approaches to *p*-cycle network design with controlled optical path lengths in the restored network state", *Journal of Optical Networking*, vol. 7, no. 7, July 2008, pp. 673-691.

[Poul06] K. Poulsen. (2006, Jan. 19). The Backhoe: A Real Cyberthreat. Wired News. [Online]. Available: http://wired.com/news/technology/0,70040-0.html?tw=wn_tophead_1

[RaMu99] S. Ramamurthy, B. Mukherjee, "Survivable WDM mesh networks, Part I - Protection," *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99)*, New York, USA, 21-25 March 1999, vol. 2, pp. 744-751.

[ScGr04] D. A. Schupke, W. D. Grover, M. Clouqueur, "Strategies for enhanced dual failure restorability with static or reconfigurable *p*-cycle networks," *Proceedings of the IEEE International Conference on Communications (ICC 2004)*, Paris, France, 20-24 June 2004, pp. 1628-1633.

[ScGr02] D. A. Schupke, C. G. Gruber, A. Autenrieth, "Optimal Configuration of *p*-Cycles in WDM Networks," *Proceedings of IEEE the International Conference on Communications (ICC 2002),* New York, NY, April-May 2002, pp. 2761-2765, .

[ShYa01] S. Shah-Heydari, O. Yang, "A Tree-Based Algorithm for Protection/Restoration in Optical Mesh Networks," *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE 2001)*, Toronto, Canada, May 2001, pp. 1169-1173.

[ShYa04] S. Shah-Heydari, O. Yang, "Hierarchical protection tree scheme for failure recovery in mesh networks," *J. Photon. Netw. Commun.*, vol. 7, no. 2, March 2004, pp. 145-159.

[StCo08] D. Staessens, D. Colle, M. Pickavet, P. Demeester, "Path protection in WSXC switched networks," *Proceedings of the 34th European Conference on Optical Communication (ECOC 2008)*, Brussels, Belgium, 21-25 September 2008, pp.1-2.

[Stam97] D. Stamatelakis, "Theory and Algorithms for Preconfiguration of Spare Capacity in Mesh Restorable Networks", M. Sc. Thesis, University of Alberta, Edmonton, Canada, Spring 1997.

[StGr00] D. Stamatelakis, W.D. Grover, "Theoretical Underpinnings for the Efficiency of Restorable Networks Using Pre-configured Cycles ("*p*-cycles")," *IEEE Transactions on Communications*, vol. 48, no. 8, August 2000, pp. 1262-1265.

[ToDo08] B. Todd, J. Doucette, "Use of Network Families in Survivable Network Design and Optimization," *Proceedings of the IEEE International Conference on Communications (ICC 2008)*, Beijing, China, 19-23 May 2008, pp. 151-157.

[ToSc02] P. Tomsu, C. Schmutzer, *Next Generation Optical Networks: The Convergence of IP Intelligence and Optical Technologies*. Prentice Hall, Upper Saddle River, NJ, 2002.

[VePo02] A. J. Vernon, J. D. Portier, "Protection of Optical Channels in All-Optical Networks," *Proceedings of the 18th Annual National Fiber Optic Engineers Conference (NFOEC 2002)*, Dallas, TX, 15-19 September 2002, pp. 1695-1706.

[WuYe07a] B. Wu, K. L. Yeung, K. Lui, S. Xu, "A New ILP-Based *p*-Cycle Construction Algorithm without Candidate Cycle Enumeration", *Proceedings of the IEEE International Conference on Communications (ICC 2007)*, Glasgow, Scotland, 24-28 June 2007, pp. 2236-2241.

[WuYe07b] B. Wu, K. L. Yeung, S. Xu, "ILP Formulation for *p*-Cycle Construction Based on Flow Conservation", *Proceedings of the Global Telecommunications Conference (GLOBECOM 2007)*, Washington, DC, USA, 26-30 November 2007, pp. 2310-2314.

[WuYe08] B. Wu, K. L. Yeung, P. Ho, "A Comparative Study of Fast Protection Schemes in WDM Mesh Networks," *Proceedings of the IEEE International Conference on Communications (ICC 2008)*, Beijing, China, 19-23 May 2008, pp.5160-5164.

[Wu92] T. S. Wu, *Fiber Network Service Survivability*, Artech House, 1992.

[XuCh02] G. Xue, L. Chen, K. Thulasiraman, "Delay reduction in redundant trees for preplanned protection against single link/Node failure in 2-connected graphs," *Proceedings of the Global Telecommunications Conference (GLOBECOM 2002)*, Taipei, Taiwan, November 2002, vol. 3, pp. 2691-2695.

[XuCh03] G. Xue, L. Chen, K. Thulasiraman, "Quality of service and quality of protection issues in preplanned recovery schemes using redundant trees," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 8, October 2003, pp. 1332-1345.

[YaZh02] O. Yang, Y. Zhang, "A distributed tree algorithms for WDM network protection/restoration," *Proceedings of the High Speed Networks and Multimedia Communications 5th IEEE International Conference*, Jeju Island, Korea, July 2002, pp. 289-294.

[Zeng07] H. Zeng, *Monitoring-Cycle Based Fault Detection and Localization in Mesh All-Optical Networks*, Ph.D. Dissertation, Carleton University, Ottawa, Canada, Spring 2007

[ZeHu06] H. Zeng, C. Huang, A.Vukovic, "A Novel Fault Detection and Localization Scheme for Mesh All-Optical Networks Based on Monitoring-Cycles", *Photonic Network Communications*, vol. 11, No. 3, May 2006, pp. 277-286.

[ZhZh06] F. Zhang, W. Zhong, "A Novel Path-Protecting p-Cycle Heuristic Algorithm," *Proceedings of the 2006 International Conference on Transparent Optical Networks (ICTON 2006)*, Nottingham, United Kingdom, 18-22 June 2006, vol. 3, pp. 203-206.

[ZhYa02] H. Zhang, O. Yang, "Finding protection cycles in DWDM networks," *Proceedings of the IEEE International Conference on Communications (ICC 2002)*, New York City, NY, USA, 28 April-2 May 2002, vol. 5, pp. 2756-2760.

[ZhXu08] W. Zhang, G. Xue, J. Tang, K. Thulasiraman, "Faster Algorithms for Construction of Recovery Trees Enhancing QoP and QoS," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, June 2008, pp. 642-655.

Appendix A

Test Network Topology Details

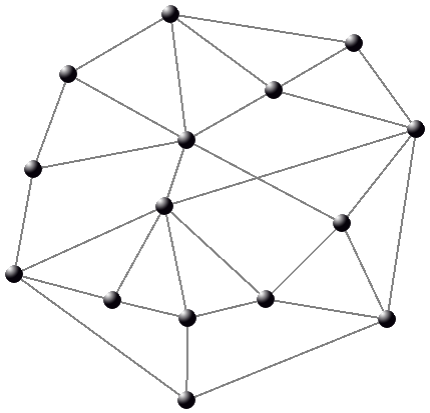
This Appendix gives the topological details of all the test networks used in this thesis. Details for the network families (with 15 node, 20 node, and 25 node master networks) are given first in a compressed form. Details for other networks that are not part of network families follow afterward.

A.1 Network Families

For network families, a diagram and complete span and node listing is provided for only the master (most highly connected) networks. For the subsequent, lower degree networks, we provide a list of spans only (the number and placement of nodes remains constant). For each network, the span that is to be removed to generate the next member of the family is shown in **bold** text.

A.1.1 15 Node Network Family (15n30s1)

A.1.1.1 Master Network Diagram



A.1.1.2 Node Listing

Node	X Coordinate	Y Coordinate
N01	125	140
N02	268	55
N03	413	162
N04	525	97
N05	290	233
N06	612	218
N07	508	349
N08	572	485
N09	402	456
N10	259	325
N11	291	598
N12	292	483
N13	186	458
N14	49	421
N15	75	274

A.1.1.3 Master Network Span Listing

Span	Origin	Destination	Length
S01	N01	N02	166.355
S02	N01	N05	189.404
S03	N01	N15	143.024
S04	N02	N03	180.205
S05	N02	N04	260.409
S06	N02	N05	179.354
S07	N03	N04	129.495
S08	N03	N05	142.021
S09	N03	N06	206.729
S10	N04	N06	149.03
S11	N05	N07	246.941
S12	N05	N10	97.082
S13	N05	N15	218.874
S14	N06	N07	167.263
S15	N07	N08	150.306

Span	Origin	Destination	Length
S16	N07	N09	150.615
S17	N08	N06	269.98
S18	N08	N11	302.87
S19	N09	N08	172.456
S20	N09	N10	193.933
S21	N10	N06	368.86
S22	N10	N14	230.903
S23	N11	N12	115.004
S24	N11	N14	299.822
S25	N12	N09	113.265
S26	N12	N10	161.409
S27	N13	N10	151.717
S28	N13	N12	108.908
S29	N14	N13	141.908
S30	N15	N14	149.282

A.1.1.4 Family Member Span Listings

15n30s-30s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 S21 **S22** S23 S24 S25 S26 S27 S28 S29 S30

15n30s-29s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 **S12** S13 S14 S15 S16 S17

S18 S19 S20 S21 S23 S24 S25 S26 S27 S28 S29 S30

15n30s-28s S01 S02 S03 S04 S05 S06 S07 **S08** S09 S10 S11 S13 S14 S15 S16 S17 S18 S19 S20 S21 S23 S24 S25 S26 S27 S28 S29 S30

15n30s-27s S01 S02 S03 S04 S05 S06 S07 S09 S10 S11 S13 **S14** S15 S16 S17 S18 S19 S20 S21 S23 S24 S25 S26 S27 S28 S29 S30

15n30s-26s S01 S02 S03 S04 S05 **S06** S07 S09 S10 S11 S13 S15 S16 S17 S18 S19 S20 S21 S23 S24 S25 S26 S27 S28 S29 S30

15n30s-25s S01 S02 S03 S04 S05 S07 S09 S10 S11 S13 **S15** S16 S17 S18 S19 S20 S21 S23 S24 S25 S26 S27 S28 S29 S30

15n30s-24s S01 S02 S03 S04 **S05** S07 S09 S10 S11 S13 S16 S17 S18 S19 S20 S21 S23 S24 S25 S26 S27 S28 S29 S30

15n30s-23s S01 S02 S03 S04 S07 **S09** S10 S11 S13 S16 S17 S18 S19 S20 S21 S23 S24 S25 S26 S27 S28 S29 S30

15n30s-22s S01 S02 S03 S04 S07 S10 S11 S13 S16 S17 S18 S19 S20 S21 S23 **S24** S25 S26 S27 S28 S29 S30

15n30s-21s S01 S02 S03 S04 S07 S10 S11 S13 S16 S17 S18 **S19** S20 S21 S23 S25 S26 S27 S28 S29 S30

15n30s-20s S01 S02 S03 S04 S07 S10 S11 S13 S16 S17 S18 S20 S21 S23 S25 S26 S27 **S28** S29 S30

15n30s-19s S01 S02 S03 S04 S07 S10 S11 S13 S16 S17 S18 S20 S21 S23 S25 **S26** S27 S29 S30

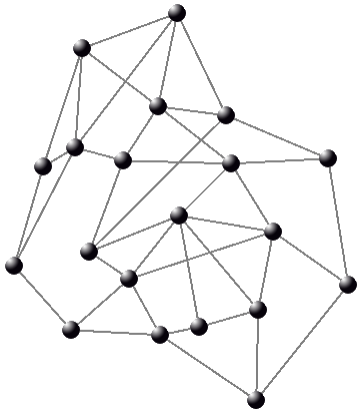
15n30s-18s S01 **S02** S03 S04 S07 S10 S11 S13 S16 S17 S18 S20 S21 S23 S25 S27 S29 S30

15n30s-17s S01 S03 S04 S07 S10 S11 S13 S16 S17 S18 S20 **S21** S23 S25 S27 S29 S30

15n30s-16s S01 S03 S04 S07 S10 S11 S13 S16 S17 S18 S20 S23 S25 S27 S29 S30

A.1.2 20 Node Network Family (20n40s1)

A.1.2.1 Master Network Diagram



A.1.2.2 Node Listing

Node	X Coordinate	Y Coordinate
N01	183	456
N02	222	322
N03	275	163
N04	266	297
N05	403	116
N06	470	253
N07	378	241
N08	331	314
N09	476	318
N10	607	311
N11	533	410
N12	634	482
N13	513	516
N14	406	389
N15	285	437
N16	338	473
N17	433	538
N18	510	637
N19	380	549
N20	260	543

A.1.2.3 Master Network Span Listing

Span	Origin	Destination	Length
S01	N01	N02	139.56
S02	N01	N04	179.36
S03	N01	N20	116.181
S04	N02	N03	167.601
S05	N02	N04	50.606
S06	N03	N07	129.201
S07	N04	N03	134.302
S08	N04	N05	227.002
S09	N04	N08	67.186
S10	N05	N03	136.356
S11	N05	N06	152.506
S12	N05	N07	127.475
S13	N06	N10	148.772
S14	N06	N15	260.923
S15	N07	N06	92.779
S16	N07	N08	86.822
S17	N07	N09	124.631
S18	N08	N09	145.055
S19	N08	N15	131.32
S20	N09	N11	108.227

Span	Origin	Destination	Length
S21	N09	N14	99.705
S22	N10	N09	131.187
S23	N10	N12	173.118
S24	N11	N12	124.036
S25	N11	N13	107.87
S26	N12	N18	198.497
S27	N13	N17	82.97
S28	N14	N11	128.725
S29	N14	N13	166.066
S30	N14	N17	151.427
S31	N15	N14	130.173
S32	N15	N16	64.07
S33	N16	N11	204.924
S34	N16	N14	108.074
S35	N16	N20	104.805
S36	N17	N19	54.129
S37	N18	N13	121.037
S38	N19	N16	86.833
S39	N19	N18	156.984
S40	N20	N19	120.15

A.1.2.4 Family Member Span Listings

20n40s-40s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 S21 S22 S23 S24 **S25** S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S36 S37 S38 S39 S40

20n40s-39s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 **S20** S21 S22 S23 S24 S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S36 S37 S38 S39 S40

20n40s-38s S01 S02 S03 S04 S05 S06 S07 S08 **S09** S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 S21 S22 S23 S24 S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S36 S37 S38 S39 S40

20n40s-37s S01 S02 S03 S04 S05 **S06** S07 S08 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 S21 S22 S23 S24 S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S36 S37 S38 S39 S40

20n40s-36s S01 S02 S03 S04 S05 S07 S08 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 S21 S22 S23 **S24** S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S36 S37 S38 S39 S40

20n40s-35s S01 S02 S03 S04 S05 S07 **S08** S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 S21 S22 S23 S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S36 S37 S38 S39 S40

20n40s-34s S01 S02 S03 S04 S05 S07 S10 **S11** S12 S13 S14 S15 S16 S17 S18 S19 S21 S22 S23 S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S36 S37 S38 S39 S40

20n40s-33s S01 S02 S03 S04 S05 S07 S10 S12 S13 S14 S15 S16 S17 S18 S19 S21 S22 S23 S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S36 S37 S38 **S39** S40

20n40s-32s S01 S02 S03 S04 S05 S07 S10 S12 **S13** S14 S15 S16 S17 S18 S19 S21 S22 S23 S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S36 S37 S38 S40

20n40s-31s S01 S02 S03 S04 S05 S07 S10 S12 S14 S15 S16 S17 S18 S19 S21 S22 S23 S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 **S36** S37 S38 S40

20n40s-30s S01 S02 S03 S04 S05 S07 S10 S12 S14 S15 S16 **S17** S18 S19 S21 S22 S23 S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S37 S38 S40

20n40s-29s S01 S02 S03 S04 S05 **S07** S10 S12 S14 S15 S16 S18 S19 S21 S22 S23 S26 S27 S28 S29 S30 S31 S32 S33 S34 S35 S37 S38 S40

20n40s-28s S01 S02 S03 S04 S05 S10 S12 S14 S15 S16 S18 S19 S21 S22 S23 S26 S27 S28 S29 **S30** S31 S32 S33 S34 S35 S37 S38 S40

20n40s-27s S01 S02 S03 S04 S05 S10 S12 S14 S15 S16 S18 S19 S21 S22 S23 S26 S27 S28 S30 S31 S32 S33 **S34** S35 S37 S38 S40

20n40s-26s S01 S02 S03 S04 S05 S10 S12 S14 S15 S16 S18 **S19** S21 S22 S23 S26 S27 S28 S30 S31 S32 S33 S35 S37 S38 S40

20n40s-25s S01 S02 S03 S04 S05 S10 S12 S14 S15 S16 S18 S21 S22 S23 S26 S27 S28 S30 **S31** S32 S33 S35 S37 S38 S40

20n40s-24s **S01** S02 S03 S04 S05 S10 S12 S14 S15 S16 S18 S21 S22 S23 S26 S27 S28 S30 S32 S33 S35 S37 S38 S40

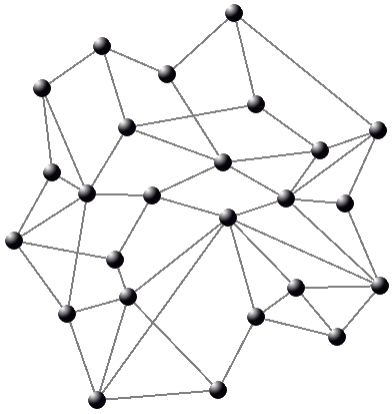
20n40s-23s S02 S03 S04 S05 S10 S12 S14 S15 S16 S18 S21 S22 S23 S26 S27 S28 S30 S32 S33 **S35** S37 S38 S40

20n40s-22s S02 S03 S04 S05 S10 S12 S14 S15 S16 S18 **S21** S22 S23 S26 S27 S28 S30 S32 S33 S37 S38 S40

20n40s-21s S02 S03 S04 S05 S10 S12 S14 S15 S16 S18 S22 S23 S26 S27 S28 S30 S32 S33 S37 S38 S40

A.1.3 25 Node Network Family (25n50s1)

A.1.3.1 Master Network Diagram



A.1.3.2 Node Listing

Node	X Coordinate	Y Coordinate
N01	92	136
N02	175	78
N03	266	117
N04	359	32
N05	390	159
N06	344	239
N07	480	223
N08	561	195
N09	515	297
N10	432	290
N11	564	411

Node	X Coordinate	Y Coordinate
N12	446	414
N13	504	482
N14	390	454
N15	351	316
N16	337	556
N17	168	571
N18	212	427
N19	127	451
N20	193	375
N21	155	283
N22	52	349
N23	105	254
N24	245	286
N25	210	190

A.1.3.3 Master Network Span Listing

Span	Origin	Destination	Length
S01	N01	N02	101.257
S02	N01	N21	159.931
S03	N02	N03	99.005
S04	N02	N25	117.341
S05	N03	N04	125.992
S06	N03	N06	144.803
S07	N05	N04	130.729
S08	N05	N07	110.436
S09	N06	N24	109.59
S10	N07	N06	136.938
S11	N07	N08	85.703
S12	N07	N10	82.42
S13	N08	N04	259.563
S14	N08	N09	111.893
S15	N08	N10	160.206
S16	N09	N10	83.295
S17	N09	N11	124.085
S18	N10	N06	101.71
S19	N11	N10	179.067
S20	N11	N13	92.957
S21	N11	N15	233.225
S22	N12	N11	118.038
S23	N12	N13	89.376
S24	N13	N14	117.388
S25	N14	N12	68.819

Span	Origin	Destination	Length
S26	N15	N10	85.071
S27	N15	N12	136.488
S28	N15	N14	143.405
S29	N15	N17	313.869
S30	N16	N14	114.948
S31	N16	N18	179.627
S32	N17	N16	169.664
S33	N17	N19	126.811
S34	N18	N15	177.882
S35	N18	N17	150.572
S36	N19	N18	88.323
S37	N20	N18	55.362
S38	N21	N19	170.317
S39	N21	N22	122.332
S40	N21	N24	90.05
S41	N22	N19	126.606
S42	N22	N20	143.377
S43	N23	N01	118.714
S44	N23	N21	57.801
S45	N23	N22	108.784
S46	N24	N15	110.164
S47	N24	N20	103.078
S48	N25	N05	182.65
S49	N25	N06	142.678
S50	N25	N21	108.046

A.1.3.4 Family Member Span Listings

25n50s-50s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 S21 S22 S23 S24 S25 S26 S27 S28 S29 S30 S31 S32 S33 S34 **S35** S36 S37 S38 S39 S40 S41 S42 S43 S44 S45 S46 S47 S48 S49 S50

25n50s-49s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 S21 S22 S23 S24 S25 S26 S27 **S28** S29 S30 S31 S32 S33 S34 S36 S37 S38 S39 S40 S41 S42 S43 S44 S45 S46 S47 S48 S49 S50

25n50s-48s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 S21 S22 S23 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 **S36**

S37 S38 S39 S40 S41 S42 S43 S44 S45 S46 S47 S48 S49 S50

25n50s-47s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 **S15** S16 S17 S18 S19 S20 S21 S22 S23 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S37 S38 S39 S40 S41 S42 S43 S44 S45 S46 S47 S48 S49 S50

25n50s-46s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S16 S17 S18 S19 S20 S21 S22 S23 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S37 **S38** S39 S40 S41 S42 S43 S44 S45 S46 S47 S48 S49 S50

25n50s-45s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S16 S17 S18 S19 S20 S21 S22 S23 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S37 S39 S40 S41 S42 S43 S44 **S45** S46 S47 S48 S49 S50

25n50s-44s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S16 S17 S18 S19 S20 S21 S22 **S23** S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S37 S39 S40 S41 S42 S43 S44 S46 S47 S48 S49 S50

25n50s-43s S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S16 S17 S18 S19 S20 S21 S22 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 **S37** S39 S40 S41 S42 S43 S44 S46 S47 S48 S49 S50

25n50s-42s S01 S02 S03 S04 S05 S06 **S07** S08 S09 S10 S11 S12 S13 S14 S16 S17 S18 S19 S20 S21 S22 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S39 S40 S41 S42 S43 S44 S46 S47 S48 S49 S50

25n50s-41s S01 S02 S03 S04 S05 S06 S08 S09 S10 S11 S12 S13 S14 S16 S17 S18 S19 S20 S21 S22 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S39 S40 S41 S42 S43 S44 **S46** S47 S48 S49 S50

25n50s-40s S01 S02 S03 S04 S05 S06 S08 S09 S10 S11 S12 S13 S14 S16 S17 S18 S19 S20 S21 S22 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 **S39** S40 S41 S42 S43 S44 S47 S48 S49 S50

25n50s-39s S01 S02 S03 S04 S05 S06 S08 S09 S10 S11 **S12** S13 S14 S16 S17 S18 S19 S20 S21 S22 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S40 S41 S42 S43

S44 S47 S48 S49 S50

25n50s-38s S01 S02 S03 S04 S05 S06 S08 S09 S10 S11 S13 S14 S16 S17 S18 S19 S20 S21 S22 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 **S49** S50

25n50s-37s S01 S02 **S03** S04 S05 S06 S08 S09 S10 S11 S13 S14 S16 S17 S18 S19 S20 S21 S22 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 S50

25n50s-36s S01 S02 S04 S05 S06 S08 S09 S10 **S11** S13 S14 S16 S17 S18 S19 S20 S21 S22 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 S50

25n50s-35s S01 S02 S04 S05 S06 S08 S09 S10 S13 S14 S16 S17 **S18** S19 S20 S21 S22 S24 S25 S26 S27 S29 S30 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 S50

25n50s-34s S01 S02 S04 S05 S06 S08 S09 S10 S13 S14 S16 S17 S19 S20 S21 S22 S24 S25 **S26** S27 S29 S30 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 S50

25n50s-33s S01 S02 S04 S05 S06 S08 S09 S10 S13 S14 S16 S17 S19 S20 S21 S22 S24 S25 S27 **S29** S30 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 S50

25n50s-32s S01 S02 S04 S05 S06 S08 S09 S10 S13 S14 S16 S17 S19 S20 S21 S22 S24 S25 S27 S30 **S31** S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 S50

25n50s-31s S01 S02 S04 S05 S06 S08 S09 S10 S13 S14 S16 **S17** S19 S20 S21 S22 S24 S25 S27 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 S50

25n50s-30s S01 S02 S04 S05 S06 S08 S09 S10 S13 S14 S16 S19 S20 S21 **S22** S24 S25 S27 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 S50

25n50s-29s S01 S02 S04 S05 S06 S08 S09 S10 S13 S14 S16 S19 S20 **S21** S24 S25 S27 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 S50

25n50s-28s S01 S02 S04 S05 S06 S08 **S09** S10 S13 S14 S16 S19 S20 S24 S25 S27 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 S50

25n50s-27s S01 S02 S04 S05 S06 S08 S10 S13 S14 S16 S19 S20 S24 S25 S27 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48 **S50**

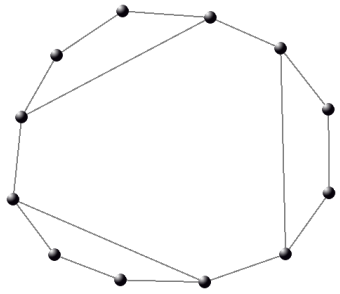
25n50s-26s S01 **S02** S04 S05 S06 S08 S10 S13 S14 S16 S19 S20 S24 S25 S27 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48

25n50s-25s S01 S04 S05 S06 S08 S10 S13 S14 S16 S19 S20 S24 S25 S27 S31 S32 S33 S34 S40 S41 S42 S43 S44 S47 S48

A.2 Other Networks

A.2.1 12-cycle + 3 edges Network

A.2.1.1 Network Diagram



A.2.1.2 Node Listing

Node	X Coordinate	Y Coordinate
Node1	100	225
Node2	146	144
Node3	232	86
Node4	347	94
Node5	439	135
Node6	501	215
Node7	502	324
Node8	445	404
Node9	340	441
Node10	229	438
Node11	143	405
Node12	89	332

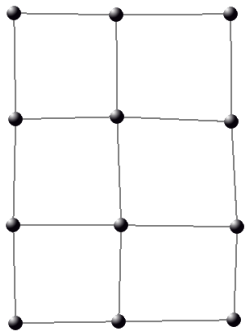
A.2.1.3 Span Listing

Span	Origin	Destination	Length
S01	Node1	Node2	1
S02	Node2	Node3	1
S03	Node3	Node4	1
S04	Node4	Node5	1
S05	Node5	Node6	1
S06	Node6	Node7	1
S07	Node7	Node8	1
S08	Node8	Node9	1

Span	Origin	Destination	Length
S09	Node9	Node10	1
S10	Node10	Node11	1
S11	Node11	Node12	1
S12	Node12	Node1	1
S13	Node1	Node4	1
S14	Node5	Node8	1
S15	Node9	Node12	1
-	-	-	-

A.2.2 3 x 4 grid Network

A.2.2.1 Network Diagram



A.2.2.2 Node Listing

Node	X Coordinate	Y Coordinate
Node1	120	90
Node2	255	92
Node3	405	91
Node4	122	230
Node5	256	227
Node6	407	233
Node7	118	370
Node8	261	370
Node9	414	372
Node10	122	500
Node11	258	498
Node12	410	496

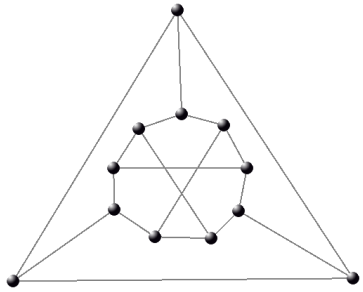
A.2.2.3 Span Listing

Span	Origin	Destination	Length
S01	Node1	Node2	1
S02	Node2	Node3	1
S03	Node3	Node6	1
S04	Node6	Node5	1
S05	Node5	Node4	1
S06	Node4	Node7	1
S07	Node7	Node8	1
S08	Node8	Node9	1
S09	Node9	Node12	1

Span	Origin	Destination	Length
S10	Node12	Node11	1
S11	Node11	Node10	1
S12	Node10	Node7	1
S13	Node4	Node1	1
S14	Node2	Node5	1
S15	Node5	Node8	1
S16	Node8	Node11	1
S17	Node9	Node6	1
-	-	-	-

A.2.3 Tietze's graph Network

A.2.3.1 Network Diagram



A.2.3.2 Node Listing

Node	X Coordinate	Y Coordinate
Node1	280	93
Node2	518	456
Node3	58	460
Node4	228	254
Node5	286	234
Node6	343	249
Node7	374	307
Node8	363	365
Node9	326	402
Node10	250	400
Node11	195	363
Node12	194	307

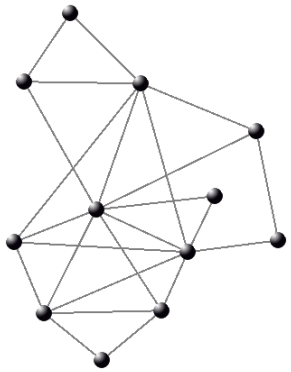
A.2.3.3 Span Listing

Span	Origin	Destination	Length
S01	Node4	Node5	1
S02	Node5	Node6	1
S03	Node6	Node7	1
S04	Node7	Node8	1
S05	Node8	Node9	1
S06	Node9	Node10	1
S07	Node10	Node11	1
S08	Node11	Node12	1
S09	Node12	Node4	1

Span	Origin	Destination	Length
S10	Node4	Node9	1
S11	Node10	Node6	1
S12	Node7	Node12	1
S13	Node5	Node1	1
S14	Node8	Node2	1
S15	Node11	Node3	1
S16	Node3	Node2	1
S17	Node2	Node1	1
S18	Node1	Node3	1

A.2.4 Murakami & Kim Network

A.2.4.1 Network Diagram



A.2.4.2 Node Listing

Node	X Coordinate	Y Coordinate
Node1	94	195
Node2	162	93
Node3	267	198
Node4	439	269
Node5	470	431
Node6	377	365
Node7	201	385
Node8	337	448
Node9	298	535
Node10	124	539
Node11	79	433
Node12	210	609

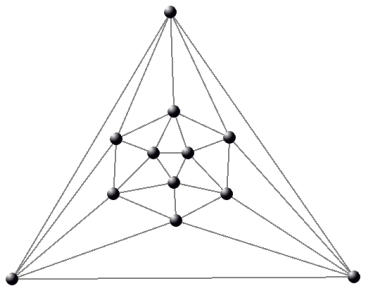
A.2.4.3 Span Listing

Span	Origin	Destination	Length
S01	Node1	Node2	1
S02	Node2	Node3	1
S03	Node3	Node1	1
S04	Node3	Node4	1
S05	Node4	Node5	1
S06	Node5	Node8	1
S07	Node8	Node6	1
S08	Node6	Node7	1
S09	Node7	Node8	1
S10	Node8	Node3	1
S11	Node3	Node7	1
S12	Node7	Node4	1

Span	Origin	Destination	Length
S13	Node1	Node7	1
S14	Node11	Node3	1
S15	Node11	Node7	1
S16	Node11	Node8	1
S17	Node8	Node9	1
S18	Node9	Node10	1
S19	Node10	Node7	1
S20	Node11	Node10	1
S21	Node10	Node8	1
S22	Node9	Node12	1
S23	Node12	Node10	1
S24	Node9	Node7	1

A.2.5 Icosahedron Network

A.2.5.1 Network Diagram



A.2.5.2 Node Listing

Node	X Coordinate	Y Coordinate
Node1	294	25
Node2	564	415
Node3	61	418
Node4	299	172
Node5	381	210
Node6	377	293
Node7	302	332
Node8	210	293
Node9	214	212
Node10	269	233
Node11	320	233
Node12	299	276

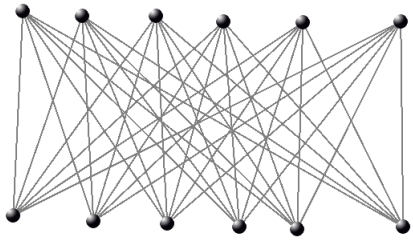
A.2.5.3 Span Listing

Span	Origin	Destination	Length
S01	Node1	Node2	1
S02	Node2	Node3	1
S03	Node3	Node1	1
S04	Node4	Node5	1
S05	Node5	Node6	1
S06	Node6	Node7	1
S07	Node7	Node8	1
S08	Node8	Node9	1
S09	Node9	Node4	1
S10	Node10	Node11	1
S11	Node11	Node12	1
S12	Node12	Node10	1
S13	Node10	Node9	1
S14	Node10	Node4	1
S15	Node10	Node8	1

Span	Origin	Destination	Length
S16	Node11	Node4	1
S17	Node11	Node5	1
S18	Node11	Node6	1
S19	Node12	Node8	1
S20	Node12	Node6	1
S21	Node12	Node7	1
S22	Node4	Node1	1
S23	Node6	Node2	1
S24	Node8	Node3	1
S25	Node9	Node1	1
S26	Node1	Node5	1
S27	Node5	Node2	1
S28	Node2	Node7	1
S29	Node7	Node3	1
S30	Node3	Node9	1

A.2.6 K6,6 Network

A.2.6.1 Master Network Diagram



A.2.6.2 Node Listing

Node	X Coordinate	Y Coordinate
Node1	69	136
Node2	122	140
Node3	187	140
Node4	247	145
Node5	317	146
Node6	404	145
Node7	61	317
Node8	132	322
Node9	197	324
Node10	261	327
Node11	312	329
Node12	406	327

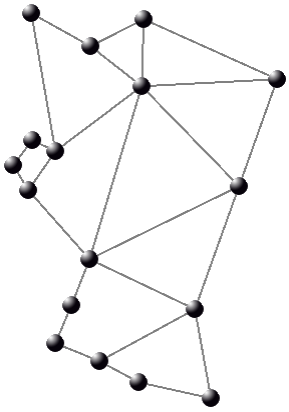
A.2.6.3 Span Listing

Span	Origin	Destination	Length
S01	Node1	Node7	1
S02	Node1	Node8	1
S03	Node1	Node9	1
S04	Node1	Node10	1
S05	Node1	Node11	1
S06	Node1	Node12	1
S07	Node2	Node7	1
S08	Node2	Node8	1
S09	Node2	Node9	1
S10	Node2	Node10	1
S11	Node2	Node11	1
S12	Node2	Node12	1
S13	Node3	Node7	1
S14	Node3	Node8	1
S15	Node3	Node9	1
S16	Node3	Node10	1
S17	Node3	Node11	1
S18	Node3	Node12	1

Span	Origin	Destination	Length
S19	Node4	Node7	1
S20	Node4	Node8	1
S21	Node4	Node9	1
S22	Node4	Node10	1
S23	Node4	Node11	1
S24	Node4	Node12	1
S25	Node5	Node7	1
S26	Node5	Node8	1
S27	Node5	Node9	1
S28	Node5	Node10	1
S29	Node5	Node11	1
S30	Node5	Node12	1
S31	Node6	Node7	1
S32	Node6	Node8	1
S33	Node6	Node9	1
S34	Node6	Node10	1
S35	Node6	Node11	1
S36	Node6	Node12	1

A.2.7 Germany Network

A.2.7.1 Master Network Diagram



A.2.7.2 Node Listing

Node	X Coordinate	Y Coordinate
N01	108	75
N02	196	124
N03	276	84
N04	273	184
N05	476	174
N06	109	266
N07	143	282
N08	81	303
N09	103	340
N10	419	335
N11	194	445
N12	167	513
N13	144	571
N14	210	597
N15	269	630
N16	377	654
N17	352	520

A.2.7.3 Span Listing

Span	Origin	Destination	Length
S01	N01	N02	120
S02	N01	N07	232
S03	N02	N03	95
S04	N02	N04	100
S05	N03	N04	133
S06	N03	N05	255
S07	N04	N05	246
S08	N04	N07	183
S09	N04	N10	215
S10	N04	N11	262
S11	N05	N10	145
S12	N06	N07	31
S13	N06	N08	30

Span	Origin	Destination	Length
S14	N07	N09	73
S15	N08	N09	34
S16	N09	N11	152
S17	N10	N11	294
S18	N10	N17	229
S19	N11	N12	71
S20	N11	N17	187
S21	N12	N13	53
S22	N13	N14	62
S23	N14	N15	72
S24	N14	N17	156
S25	N15	N16	119
S26	N16	N17	149

Appendix B

Test Network Demand Pattern Details

This Appendix gives the details of the demand patterns used for the experiments throughout this thesis. Demand patterns are given both for network families and individual networks. For a given network family, the demand pattern is constant across all members of the family.

B.1 15 Node Network Family (15n30s1)

Demand	Origin	Destination	Size
D1	N01	N02	8
D2	N01	N03	4
D3	N01	N04	5
D4	N01	N05	6
D5	N01	N06	10
D6	N01	N07	3
D7	N01	N08	9
D8	N01	N09	9
D9	N01	N10	5
D10	N01	N11	10
D11	N01	N12	1
D12	N01	N13	1
D13	N01	N14	4
D14	N01	N15	4
D15	N02	N03	8
D16	N02	N04	2
D17	N02	N05	5
D18	N02	N06	1

Demand	Origin	Destination	Size
D19	N02	N07	6
D20	N02	N08	1
D21	N02	N09	1
D22	N02	N10	3
D23	N02	N11	9
D24	N02	N12	6
D25	N02	N13	5
D26	N02	N14	5
D27	N02	N15	7
D28	N03	N04	2
D29	N03	N05	9
D30	N03	N06	2
D31	N03	N07	3
D32	N03	N08	6
D33	N03	N09	5
D34	N03	N10	9
D35	N03	N11	9
D36	N03	N12	6

Demand	Origin	Destination	Size
D37	N03	N13	1
D38	N03	N14	4
D39	N03	N15	2
D40	N04	N05	3
D41	N04	N06	2
D42	N04	N07	4
D43	N04	N08	7
D44	N04	N09	9
D45	N04	N10	3
D46	N04	N11	5
D47	N04	N12	6
D48	N04	N13	4
D49	N04	N14	2
D50	N04	N15	1
D51	N05	N06	10
D52	N05	N07	7
D53	N05	N08	9
D54	N05	N09	6
D55	N05	N10	3
D56	N05	N11	3
D57	N05	N12	1
D58	N05	N13	1
D59	N05	N14	7
D60	N05	N15	3
D61	N06	N07	4
D62	N06	N08	6
D63	N06	N09	4
D64	N06	N10	5
D65	N06	N11	2
D66	N06	N12	5
D67	N06	N13	8
D68	N06	N14	9
D69	N06	N15	3
D70	N07	N08	5
D71	N07	N09	5

Demand	Origin	Destination	Size
D72	N07	N10	7
D73	N07	N11	7
D74	N07	N12	1
D75	N07	N13	9
D76	N07	N14	4
D77	N07	N15	7
D78	N08	N09	2
D79	N08	N10	3
D80	N08	N11	6
D81	N08	N12	8
D82	N08	N13	8
D83	N08	N14	2
D84	N08	N15	10
D85	N09	N10	8
D86	N09	N11	6
D87	N09	N12	8
D88	N09	N13	10
D89	N09	N14	1
D90	N09	N15	1
D91	N10	N11	10
D92	N10	N12	2
D93	N10	N13	1
D94	N10	N14	7
D95	N10	N15	1
D96	N11	N12	5
D97	N11	N13	7
D98	N11	N14	1
D99	N11	N15	3
D100	N12	N13	3
D101	N12	N14	5
D102	N12	N15	1
D103	N13	N14	10
D104	N13	N15	3
D105	N14	N15	3
-	-	-	-

B.2 20 Node Network Family (20n40s1)

Demand	Origin	Destination	Size
D1	N01	N02	9
D2	N01	N03	1
D3	N01	N04	7
D4	N01	N05	3
D5	N01	N06	3
D6	N01	N07	5
D7	N01	N08	9
D8	N01	N09	4
D9	N01	N10	7
D10	N01	N11	2
D11	N01	N12	6
D12	N01	N13	2
D13	N01	N14	3
D14	N01	N15	1
D15	N01	N16	2
D16	N01	N17	6
D17	N01	N18	1
D18	N01	N19	3
D19	N01	N20	10
D20	N02	N03	6
D21	N02	N04	7
D22	N02	N05	3
D23	N02	N06	6
D24	N02	N07	3
D25	N02	N08	2
D26	N02	N09	9
D27	N02	N10	1
D28	N02	N11	10
D29	N02	N12	2
D30	N02	N13	8
D31	N02	N14	6
D32	N02	N15	3
D33	N02	N16	10
D34	N02	N17	10
D35	N02	N18	4
D36	N02	N19	2
D37	N02	N20	3
D38	N03	N04	2
D39	N03	N05	4

Demand	Origin	Destination	Size
D40	N03	N06	5
D41	N03	N07	8
D42	N03	N08	4
D43	N03	N09	9
D44	N03	N10	5
D45	N03	N11	8
D46	N03	N12	7
D47	N03	N13	2
D48	N03	N14	3
D49	N03	N15	8
D50	N03	N16	2
D51	N03	N17	1
D52	N03	N18	3
D53	N03	N19	5
D54	N03	N20	8
D55	N04	N05	5
D56	N04	N06	2
D57	N04	N07	5
D58	N04	N08	8
D59	N04	N09	6
D60	N04	N10	5
D61	N04	N11	3
D62	N04	N12	7
D63	N04	N13	10
D64	N04	N14	6
D65	N04	N15	2
D66	N04	N16	9
D67	N04	N17	5
D68	N04	N18	9
D69	N04	N19	9
D70	N04	N20	6
D71	N05	N06	9
D72	N05	N07	5
D73	N05	N08	7
D74	N05	N09	6
D75	N05	N10	8
D76	N05	N11	5
D77	N05	N12	5
D78	N05	N13	5

Demand	Origin	Destination	Size
D79	N05	N14	8
D80	N05	N15	1
D81	N05	N16	7
D82	N05	N17	10
D83	N05	N18	9
D84	N05	N19	8
D85	N05	N20	8
D86	N06	N07	10
D87	N06	N08	5
D88	N06	N09	2
D89	N06	N10	3
D90	N06	N11	3
D91	N06	N12	4
D92	N06	N13	7
D93	N06	N14	10
D94	N06	N15	6
D95	N06	N16	1
D96	N06	N17	1
D97	N06	N18	10
D98	N06	N19	7
D99	N06	N20	2
D100	N07	N08	4
D101	N07	N09	4
D102	N07	N10	9
D103	N07	N11	3
D104	N07	N12	4
D105	N07	N13	8
D106	N07	N14	4
D107	N07	N15	1
D108	N07	N16	6
D109	N07	N17	9
D110	N07	N18	3
D111	N07	N19	10
D112	N07	N20	1
D113	N08	N09	4
D114	N08	N10	3
D115	N08	N11	6
D116	N08	N12	10
D117	N08	N13	5
D118	N08	N14	8
D119	N08	N15	8
D120	N08	N16	5

Demand	Origin	Destination	Size
D121	N08	N17	9
D122	N08	N18	6
D123	N08	N19	4
D124	N08	N20	3
D125	N09	N10	8
D126	N09	N11	8
D127	N09	N12	8
D128	N09	N13	3
D129	N09	N14	2
D130	N09	N15	10
D131	N09	N16	10
D132	N09	N17	7
D133	N09	N18	5
D134	N09	N19	4
D135	N09	N20	10
D136	N10	N11	8
D137	N10	N12	5
D138	N10	N13	4
D139	N10	N14	10
D140	N10	N15	1
D141	N10	N16	8
D142	N10	N17	9
D143	N10	N18	5
D144	N10	N19	1
D145	N10	N20	1
D146	N11	N12	2
D147	N11	N13	2
D148	N11	N14	7
D149	N11	N15	8
D150	N11	N16	10
D151	N11	N17	5
D152	N11	N18	1
D153	N11	N19	10
D154	N11	N20	2
D155	N12	N13	8
D156	N12	N14	6
D157	N12	N15	7
D158	N12	N16	9
D159	N12	N17	2
D160	N12	N18	4
D161	N12	N19	7
D162	N12	N20	9

Demand	Origin	Destination	Size
D163	N13	N14	9
D164	N13	N15	2
D165	N13	N16	3
D166	N13	N17	8
D167	N13	N18	7
D168	N13	N19	5
D169	N13	N20	8
D170	N14	N15	3
D171	N14	N16	9
D172	N14	N17	3
D173	N14	N18	8
D174	N14	N19	6
D175	N14	N20	1
D176	N15	N16	4

Demand	Origin	Destination	Size
D177	N15	N17	6
D178	N15	N18	10
D179	N15	N19	1
D180	N15	N20	6
D181	N16	N17	1
D182	N16	N18	9
D183	N16	N19	6
D184	N16	N20	5
D185	N17	N18	4
D186	N17	N19	7
D187	N17	N20	1
D188	N18	N19	2
D189	N18	N20	1
D190	N19	N20	9

B.3 25 Node Network Family (25n50s1)

Demand	Origin	Destination	Size
D1	N01	N02	2
D2	N01	N03	1
D3	N01	N04	9
D4	N01	N05	7
D5	N01	N06	1
D6	N01	N07	2
D7	N01	N08	4
D8	N01	N09	10
D9	N01	N10	1
D10	N01	N11	3
D11	N01	N12	4
D12	N01	N13	3
D13	N01	N14	4
D14	N01	N15	3
D15	N01	N16	10
D16	N01	N17	1
D17	N01	N18	6
D18	N01	N19	1
D19	N01	N20	10
D20	N01	N21	3
D21	N01	N22	9
D22	N01	N23	10
D23	N01	N24	9

Demand	Origin	Destination	Size
D24	N01	N25	4
D25	N02	N03	8
D26	N02	N04	1
D27	N02	N05	5
D28	N02	N06	8
D29	N02	N07	4
D30	N02	N08	2
D31	N02	N09	6
D32	N02	N10	3
D33	N02	N11	9
D34	N02	N12	6
D35	N02	N13	3
D36	N02	N14	10
D37	N02	N15	4
D38	N02	N16	6
D39	N02	N17	3
D40	N02	N18	4
D41	N02	N19	10
D42	N02	N20	6
D43	N02	N21	3
D44	N02	N22	7
D45	N02	N23	7
D46	N02	N24	1

Demand	Origin	Destination	Size
D47	N02	N25	1
D48	N03	N04	6
D49	N03	N05	4
D50	N03	N06	3
D51	N03	N07	6
D52	N03	N08	3
D53	N03	N09	5
D54	N03	N10	9
D55	N03	N11	1
D56	N03	N12	3
D57	N03	N13	1
D58	N03	N14	2
D59	N03	N15	6
D60	N03	N16	6
D61	N03	N17	9
D62	N03	N18	3
D63	N03	N19	5
D64	N03	N20	1
D65	N03	N21	5
D66	N03	N22	10
D67	N03	N23	6
D68	N03	N24	8
D69	N03	N25	3
D70	N04	N05	1
D71	N04	N06	5
D72	N04	N07	7
D73	N04	N08	6
D74	N04	N09	5
D75	N04	N10	4
D76	N04	N11	5
D77	N04	N12	3
D78	N04	N13	1
D79	N04	N14	3
D80	N04	N15	3
D81	N04	N16	8
D82	N04	N17	2
D83	N04	N18	7
D84	N04	N19	9
D85	N04	N20	9
D86	N04	N21	8
D87	N04	N22	5
D88	N04	N23	3
D89	N04	N24	8
D90	N04	N25	1
D91	N05	N06	5
D92	N05	N07	4
D93	N05	N08	4
D94	N05	N09	10
D95	N05	N10	8
D96	N05	N11	8
D97	N05	N12	4
D98	N05	N13	2
D99	N05	N14	8
D100	N05	N15	9
D101	N05	N16	8
D102	N05	N17	8
D103	N05	N18	6
D104	N05	N19	7
D105	N05	N20	2
D106	N05	N21	4
D107	N05	N22	6
D108	N05	N23	1
D109	N05	N24	4
D110	N05	N25	6
D111	N06	N07	5
D112	N06	N08	4
D113	N06	N09	1
D114	N06	N10	4
D115	N06	N11	2
D116	N06	N12	6
D117	N06	N13	6
D118	N06	N14	9
D119	N06	N15	1
D120	N06	N16	9
D121	N06	N17	3
D122	N06	N18	3
D123	N06	N19	5
D124	N06	N20	5
D125	N06	N21	8
D126	N06	N22	3

Demand	Origin	Destination	Size
D127	N06	N23	10
D128	N06	N24	8
D129	N06	N25	2
D130	N07	N08	9
D131	N07	N09	4
D132	N07	N10	1
D133	N07	N11	7
D134	N07	N12	2
D135	N07	N13	8
D136	N07	N14	3
D137	N07	N15	7
D138	N07	N16	6
D139	N07	N17	3
D140	N07	N18	9
D141	N07	N19	5
D142	N07	N20	1
D143	N07	N21	2
D144	N07	N22	5
D145	N07	N23	7
D146	N07	N24	4
D147	N07	N25	1
D148	N08	N09	8
D149	N08	N10	5
D150	N08	N11	3
D151	N08	N12	1
D152	N08	N13	5
D153	N08	N14	7
D154	N08	N15	10
D155	N08	N16	7
D156	N08	N17	5
D157	N08	N18	6
D158	N08	N19	9
D159	N08	N20	5
D160	N08	N21	10
D161	N08	N22	7
D162	N08	N23	4
D163	N08	N24	6
D164	N08	N25	10
D165	N09	N10	2
D166	N09	N11	10

Demand	Origin	Destination	Size
D167	N09	N12	5
D168	N09	N13	5
D169	N09	N14	10
D170	N09	N15	1
D171	N09	N16	4
D172	N09	N17	8
D173	N09	N18	2
D174	N09	N19	4
D175	N09	N20	5
D176	N09	N21	5
D177	N09	N22	3
D178	N09	N23	5
D179	N09	N24	9
D180	N09	N25	3
D181	N10	N11	7
D182	N10	N12	5
D183	N10	N13	4
D184	N10	N14	8
D185	N10	N15	2
D186	N10	N16	10
D187	N10	N17	2
D188	N10	N18	5
D189	N10	N19	3
D190	N10	N20	10
D191	N10	N21	6
D192	N10	N22	7
D193	N10	N23	9
D194	N10	N24	9
D195	N10	N25	9
D196	N11	N12	9
D197	N11	N13	2
D198	N11	N14	4
D199	N11	N15	7
D200	N11	N16	10
D201	N11	N17	4
D202	N11	N18	2
D203	N11	N19	8
D204	N11	N20	8
D205	N11	N21	7
D206	N11	N22	10

Demand	Origin	Destination	Size
D207	N11	N23	2
D208	N11	N24	2
D209	N11	N25	5
D210	N12	N13	4
D211	N12	N14	4
D212	N12	N15	10
D213	N12	N16	2
D214	N12	N17	7
D215	N12	N18	3
D216	N12	N19	4
D217	N12	N20	3
D218	N12	N21	9
D219	N12	N22	8
D220	N12	N23	8
D221	N12	N24	6
D222	N12	N25	10
D223	N13	N14	1
D224	N13	N15	4
D225	N13	N16	4
D226	N13	N17	8
D227	N13	N18	9
D228	N13	N19	5
D229	N13	N20	9
D230	N13	N21	3
D231	N13	N22	6
D232	N13	N23	4
D233	N13	N24	8
D234	N13	N25	6
D235	N14	N15	5
D236	N14	N16	9
D237	N14	N17	3
D238	N14	N18	9
D239	N14	N19	8
D240	N14	N20	6
D241	N14	N21	5
D242	N14	N22	9
D243	N14	N23	8
D244	N14	N24	5
D245	N14	N25	9
D246	N15	N16	4

Demand	Origin	Destination	Size
D247	N15	N17	1
D248	N15	N18	5
D249	N15	N19	4
D250	N15	N20	5
D251	N15	N21	9
D252	N15	N22	6
D253	N15	N23	1
D254	N15	N24	8
D255	N15	N25	5
D256	N16	N17	4
D257	N16	N18	10
D258	N16	N19	8
D259	N16	N20	1
D260	N16	N21	6
D261	N16	N22	5
D262	N16	N23	7
D263	N16	N24	1
D264	N16	N25	7
D265	N17	N18	1
D266	N17	N19	3
D267	N17	N20	3
D268	N17	N21	9
D269	N17	N22	10
D270	N17	N23	2
D271	N17	N24	10
D272	N17	N25	3
D273	N18	N19	3
D274	N18	N20	9
D275	N18	N21	1
D276	N18	N22	4
D277	N18	N23	5
D278	N18	N24	9
D279	N18	N25	10
D280	N19	N20	8
D281	N19	N21	4
D282	N19	N22	5
D283	N19	N23	9
D284	N19	N24	10
D285	N19	N25	4
D286	N20	N21	1

Demand	Origin	Destination	Size
D287	N20	N22	1
D288	N20	N23	1
D289	N20	N24	6
D290	N20	N25	10
D291	N21	N22	3
D292	N21	N23	6
D293	N21	N24	10

Demand	Origin	Destination	Size
D294	N21	N25	8
D295	N22	N23	2
D296	N22	N24	5
D297	N22	N25	8
D298	N23	N24	7
D299	N23	N25	1
D300	N24	N25	7

B.4 12-cycle + 3 edges Neighbor Pattern

Demand	Origin	Destination	Size
D1	Node1	Node2	10
D2	Node2	Node3	10
D3	Node3	Node4	10
D4	Node4	Node5	10
D5	Node5	Node6	10
D6	Node6	Node7	10
D7	Node7	Node8	10
D8	Node8	Node9	10

Demand	Origin	Destination	Size
D9	Node9	Node10	10
D10	Node10	Node11	10
D11	Node11	Node12	10
D12	Node1	Node12	10
D13	Node1	Node4	10
D14	Node5	Node8	10
D15	Node9	Node12	10
-	-	-	-

B.5 3 x 4 grid Neighbor Pattern

Demand	Origin	Destination	Size
D1	Node1	Node2	10
D2	Node2	Node3	10
D3	Node3	Node6	10
D4	Node5	Node6	10
D5	Node4	Node5	10
D6	Node4	Node7	10
D7	Node7	Node8	10
D8	Node8	Node9	10
D9	Node9	Node12	10

Demand	Origin	Destination	Size
D10	Node11	Node12	10
D11	Node10	Node11	10
D12	Node7	Node10	10
D13	Node1	Node4	10
D14	Node2	Node5	10
D15	Node5	Node8	10
D16	Node8	Node11	10
D17	Node6	Node9	10
-	-	-	-

B.6 Tietze's graph Neighbor Pattern

Demand	Origin	Destination	Size
D1	Node4	Node5	10
D2	Node5	Node6	10
D3	Node6	Node7	10
D4	Node7	Node8	10
D5	Node8	Node9	10
D6	Node9	Node10	10
D7	Node10	Node11	10
D8	Node11	Node12	10
D9	Node4	Node12	10

Demand	Origin	Destination	Size
D10	Node4	Node9	10
D11	Node6	Node10	10
D12	Node7	Node12	10
D13	Node1	Node5	10
D14	Node2	Node8	10
D15	Node3	Node11	10
D16	Node2	Node3	10
D17	Node1	Node2	10
D18	Node1	Node3	10

B.7 Murakami & Kim Neighbor Pattern

Demand	Origin	Destination	Size
D1	Node1	Node2	5
D2	Node1	Node3	5
D3	Node1	Node4	5
D4	Node1	Node5	5
D5	Node1	Node6	5
D6	Node1	Node7	5
D7	Node1	Node8	5
D8	Node1	Node9	5
D9	Node1	Node10	5
D10	Node1	Node11	5
D11	Node1	Node12	5
D12	Node2	Node3	5
D13	Node2	Node4	5
D14	Node2	Node5	5
D15	Node2	Node6	5
D16	Node2	Node7	5
D17	Node2	Node8	5
D18	Node2	Node9	5
D19	Node2	Node10	5
D20	Node2	Node11	5
D21	Node2	Node12	5
D22	Node3	Node4	5
D23	Node3	Node5	5
D24	Node3	Node6	5
D25	Node3	Node7	5
D26	Node3	Node8	5
D27	Node3	Node9	5
D28	Node3	Node10	5
D29	Node3	Node11	5
D30	Node3	Node12	5
D31	Node4	Node5	5
D32	Node4	Node6	5
D33	Node4	Node7	5
D34	Node4	Node8	5
D35	Node4	Node9	5
D36	Node4	Node10	5
D37	Node4	Node11	5
D38	Node4	Node12	5
D39	Node5	Node6	5
D40	Node5	Node7	5
D41	Node5	Node8	5
D42	Node5	Node9	5
D43	Node5	Node10	5
D44	Node5	Node11	5
D45	Node5	Node12	5
D46	Node6	Node7	5
D47	Node6	Node8	5
D48	Node6	Node9	5
D49	Node6	Node10	5
D50	Node6	Node11	5
D51	Node6	Node12	5
D52	Node7	Node8	5
D53	Node7	Node9	5
D54	Node7	Node10	5
D55	Node7	Node11	5
D56	Node7	Node12	5
D57	Node8	Node9	5
D58	Node8	Node10	5
D59	Node8	Node11	5
D60	Node8	Node12	5
D61	Node9	Node10	5
D62	Node9	Node11	5
D63	Node9	Node12	5
D64	Node10	Node11	5
D65	Node10	Node12	5
D66	Node11	Node12	5

B.8 Icosahedron Neighbor Pattern

Demand	Origin	Destination	Size
D1	Node1	Node2	10
D2	Node2	Node3	10
D3	Node1	Node3	10
D4	Node4	Node5	10
D5	Node5	Node6	10
D6	Node6	Node7	10
D7	Node7	Node8	10
D8	Node8	Node9	10
D9	Node4	Node9	10
D10	Node10	Node11	10
D11	Node11	Node12	10
D12	Node10	Node12	10
D13	Node9	Node10	10
D14	Node4	Node10	10
D15	Node8	Node10	10

Demand	Origin	Destination	Size
D16	Node4	Node11	10
D17	Node5	Node11	10
D18	Node6	Node11	10
D19	Node8	Node12	10
D20	Node6	Node12	10
D21	Node7	Node12	10
D22	Node1	Node4	10
D23	Node2	Node6	10
D24	Node3	Node8	10
D25	Node1	Node9	10
D26	Node1	Node5	10
D27	Node2	Node5	10
D28	Node2	Node7	10
D29	Node3	Node7	10
D30	Node3	Node9	10

B.9 K6,6 Neighbor Pattern

Demand	Origin	Destination	Size
D1	Node1	Node7	10
D2	Node1	Node8	10
D3	Node1	Node9	10
D4	Node1	Node10	10
D5	Node1	Node11	10
D6	Node1	Node12	10
D7	Node2	Node7	10
D8	Node2	Node8	10
D9	Node2	Node9	10
D10	Node2	Node10	10
D11	Node2	Node11	10
D12	Node2	Node12	10
D13	Node3	Node7	10
D14	Node3	Node8	10
D15	Node3	Node9	10
D16	Node3	Node10	10
D17	Node3	Node11	10
D18	Node3	Node12	10

Demand	Origin	Destination	Size
D19	Node4	Node7	10
D20	Node4	Node8	10
D21	Node4	Node9	10
D22	Node4	Node10	10
D23	Node4	Node11	10
D24	Node4	Node12	10
D25	Node5	Node7	10
D26	Node5	Node8	10
D27	Node5	Node9	10
D28	Node5	Node10	10
D29	Node5	Node11	10
D30	Node5	Node12	10
D31	Node6	Node7	10
D32	Node6	Node8	10
D33	Node6	Node9	10
D34	Node6	Node10	10
D35	Node6	Node11	10
D36	Node6	Node12	10

B.10 Uniform Pattern for All 12 Node Networks (12-cycle + 3 edges, 3 x 4 grid, Tietze's graph, Murakami & Kim, Icosahedron, K6,6)

Demand	Origin	Destination	Size
D1	Node1	Node2	10
D2	Node2	Node3	10
D3	Node1	Node3	10
D4	Node3	Node4	10
D5	Node4	Node5	10
D6	Node5	Node8	10
D7	Node6	Node8	10
D8	Node6	Node7	10
D9	Node7	Node8	10
D10	Node3	Node8	10
D11	Node3	Node7	10
D12	Node4	Node7	10

Demand	Origin	Destination	Size
D13	Node1	Node7	10
D14	Node3	Node11	10
D15	Node7	Node11	10
D16	Node8	Node11	10
D17	Node8	Node9	10
D18	Node9	Node10	10
D19	Node7	Node10	10
D20	Node10	Node11	10
D21	Node8	Node10	10
D22	Node9	Node12	10
D23	Node10	Node12	10
D24	Node7	Node9	10

B.11 Germany Network

Demand	Origin	Destination	Size
D1	N01	N03	2
D2	N01	N11	2
D3	N02	N03	1
D4	N02	N04	1
D5	N03	N04	3
D6	N03	N05	1
D7	N03	N07	1
D8	N03	N08	1
D9	N03	N09	1
D10	N03	N10	2
D11	N03	N11	1
D12	N03	N14	1
D13	N03	N16	1
D14	N04	N05	2
D15	N04	N07	4
D16	N04	N08	1
D17	N04	N09	4
D18	N04	N10	3
D19	N04	N11	5

Demand	Origin	Destination	Size
D20	N04	N14	1
D21	N04	N17	2
D22	N05	N10	2
D23	N05	N11	1
D24	N05	N16	1
D25	N05	N17	1
D26	N06	N07	1
D27	N06	N08	1
D28	N06	N09	1
D29	N07	N08	1
D30	N07	N09	4
D31	N07	N10	3
D32	N07	N11	3
D33	N08	N09	2
D34	N08	N11	1
D35	N08	N14	1
D36	N09	N10	3
D37	N09	N11	5
D38	N09	N14	1

Demand	Origin	Destination	Size
D39	N10	N11	3
D40	N10	N14	1
D41	N10	N15	1
D42	N10	N16	1
D43	N10	N17	1
D44	N11	N12	1
D45	N11	N13	1
D46	N11	N14	2
D47	N11	N15	2
D48	N11	N16	2

Demand	Origin	Destination	Size
D49	N11	N17	1
D50	N12	N13	1
D51	N12	N14	1
D52	N13	N14	1
D53	N13	N16	1
D54	N14	N15	1
D55	N14	N16	1
D56	N14	N17	1
D57	N15	N16	1
D58	N16	N17	1

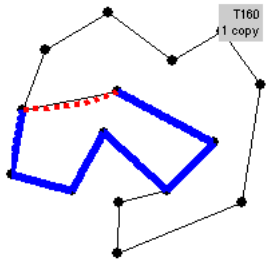
Appendix C

Span *p*-Tree Diagrams for *p*-Tree/*p*-Cycle Hybrid Designs for the 15n30s1 and 20n40s1 Network Families

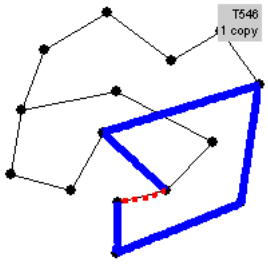
The diagrams on the following pages show the *p*-trees in the hybrid designs that were generated for the 15n30s1 and 20n40s1 network families. Only the trees are shown, as cycles vastly outnumber trees in the designs and therefore there are far too many to be shown here. Each diagram is captioned with the number of spans in the network that it was found in.

Structures are drawn with solid lines. Protected spans are shown as dashed lines.

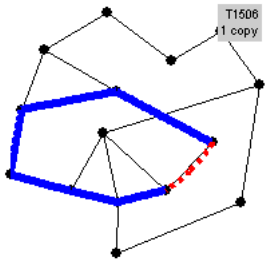
C.1 15n30s1 Family



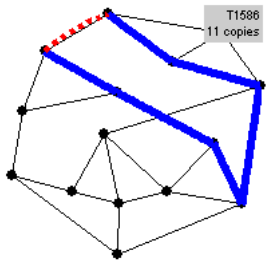
16 span network



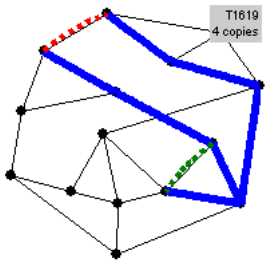
17 span network



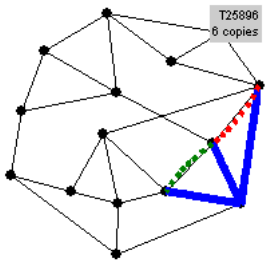
20 span network



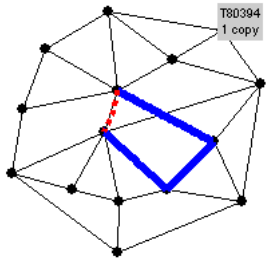
25 span network



25 span network

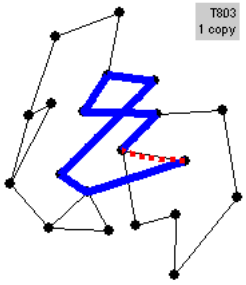


27 span network

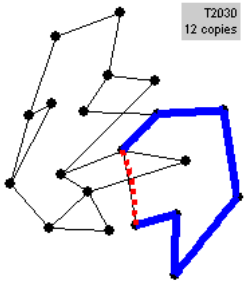


30 span network (master)

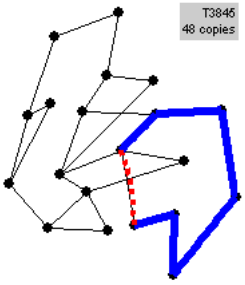
C.2 20n40s1 Family



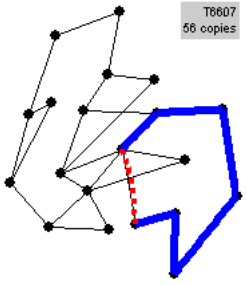
24 span network



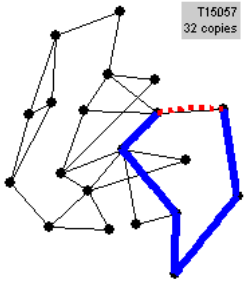
25 span network



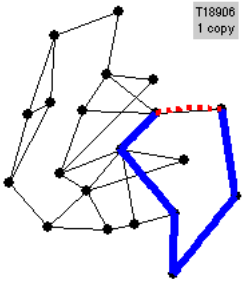
26 span network



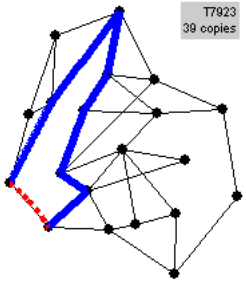
27 span network



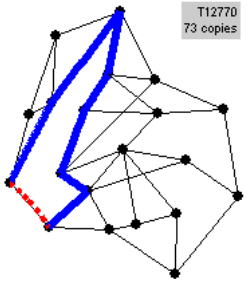
30 span network



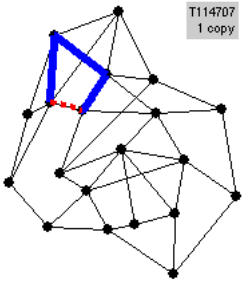
31 span network



35 span network



36 span network



40 span network (master)

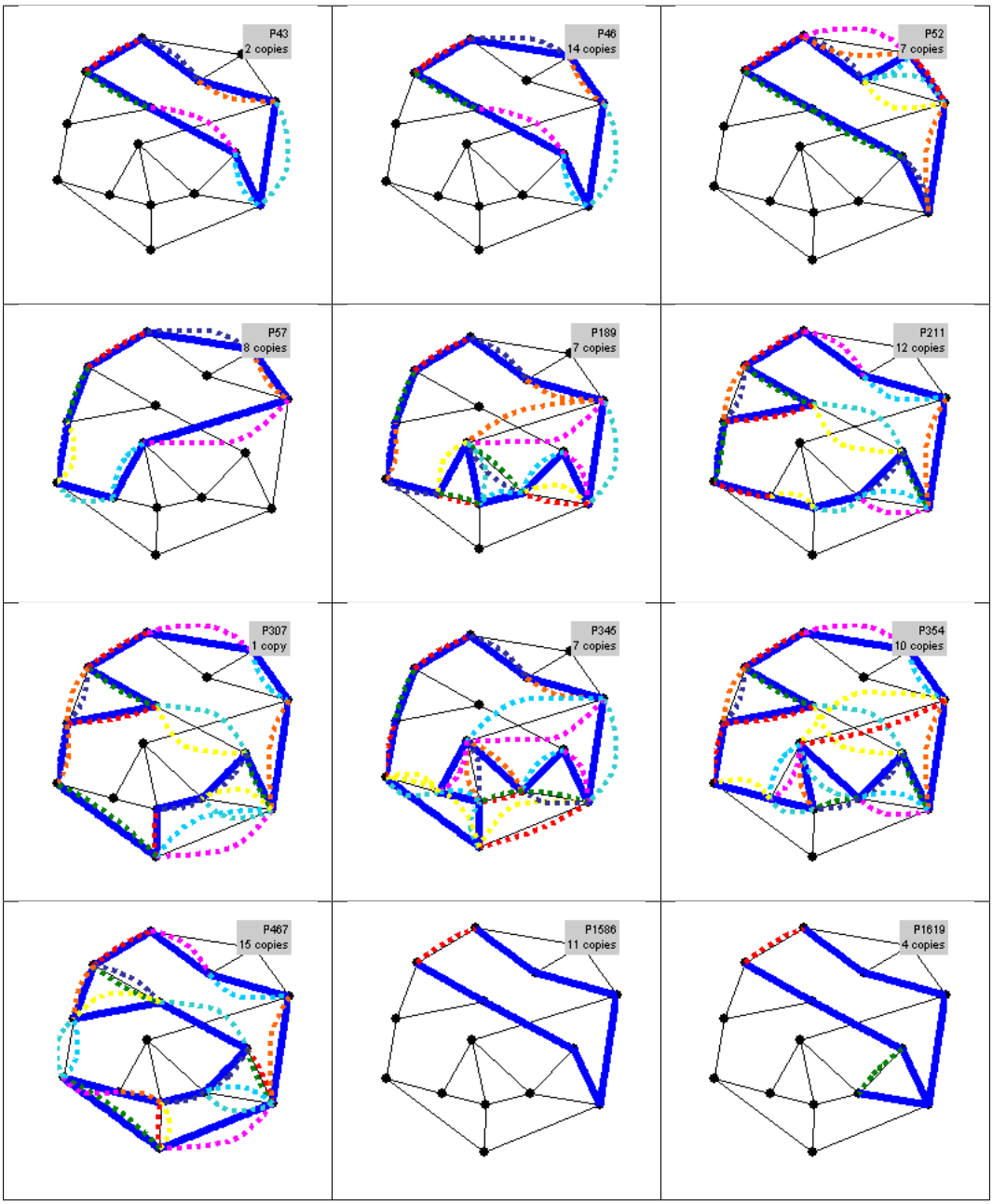
Appendix D

Structure Diagrams: *p*-Tree/*p*-Cycle Hybrid Designs and *p*-Segment/*p*-Cycle Hybrid Designs for the 15n30s1-25s and 15n30s1-27s Networks

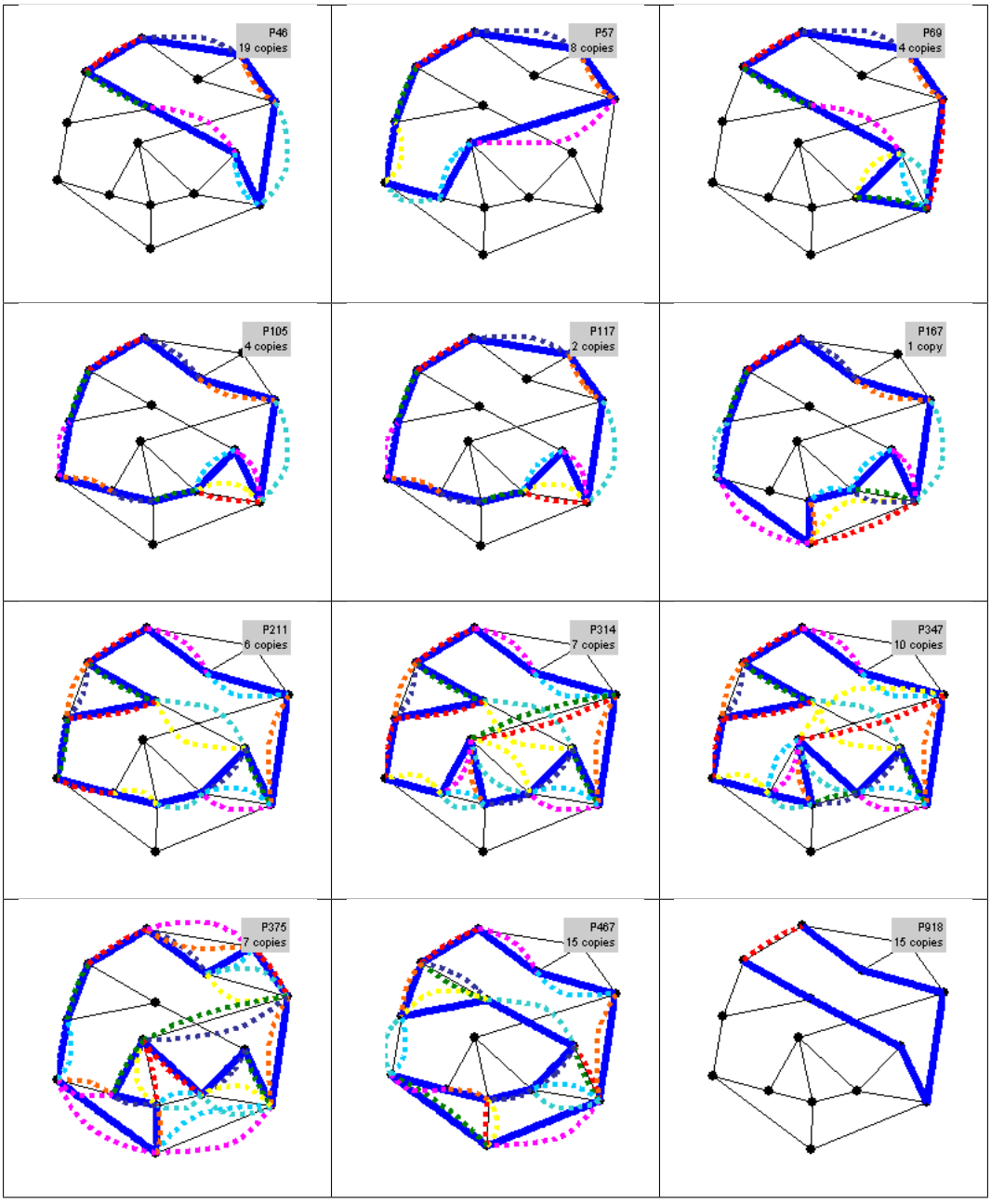
The diagrams on the following pages show all of the structures used in both the *p*-tree/*p*-cycle and *p*-segment/*p*-cycle hybrid designs that were generated for the 25 node and 27 node networks in the 15n30s1 network family. For the *p*-segment/*p*-cycle hybrid designs, the 7-hop limited and the 9/10-hop limited designs were identical, so the diagrams listed here apply to both equally. Structures are drawn with solid lines. Protected spans are shown as dashed lines. Straddling spans on cycles are shown with two dashed lines because they are given two units of protection.

D.1 15n30s1-25s Network

D.1.1 *p*-Tree/*p*-Cycle Hybrid Design

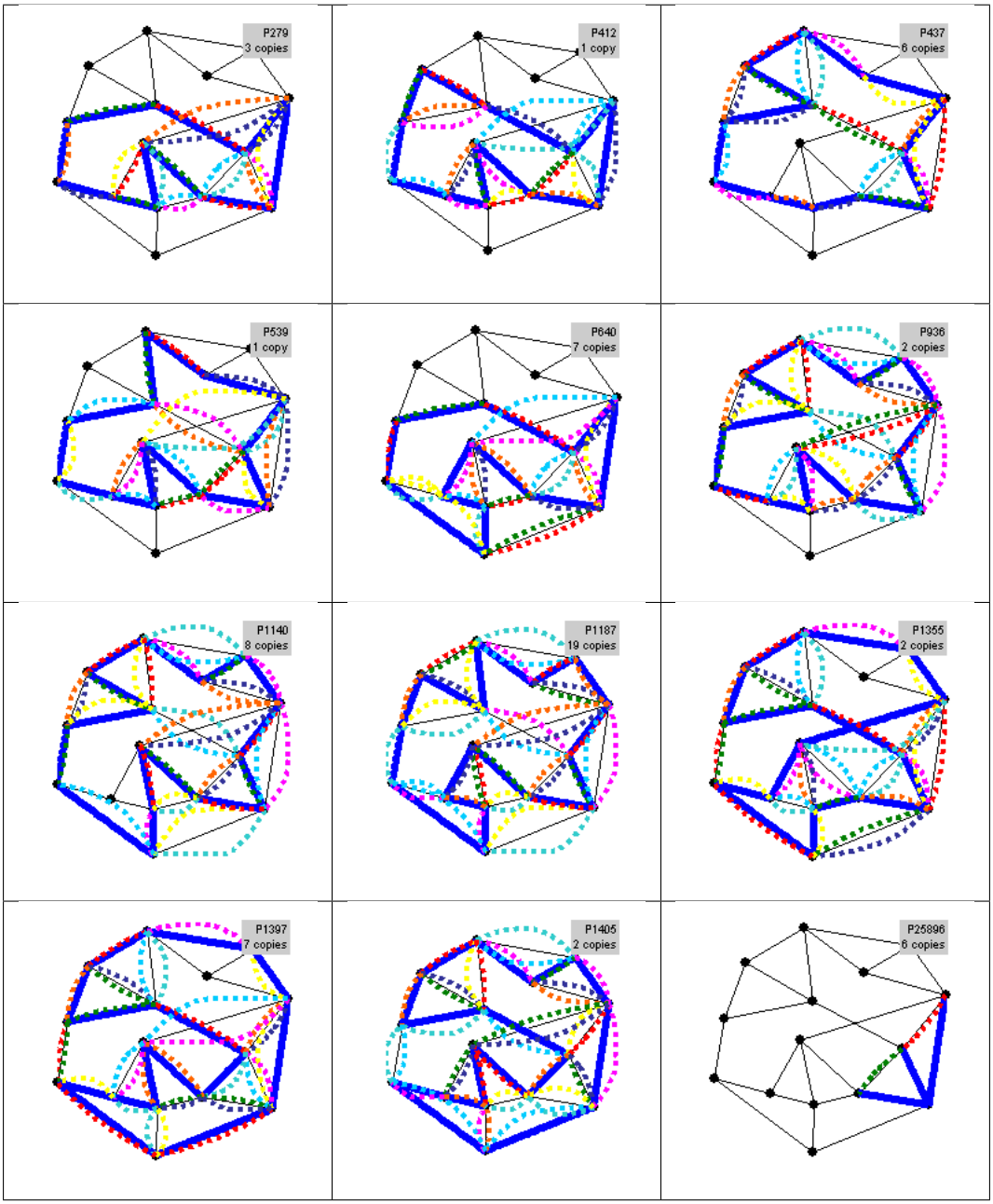


D.1.2 *p*-Segment/*p*-Cycle Hybrid Design

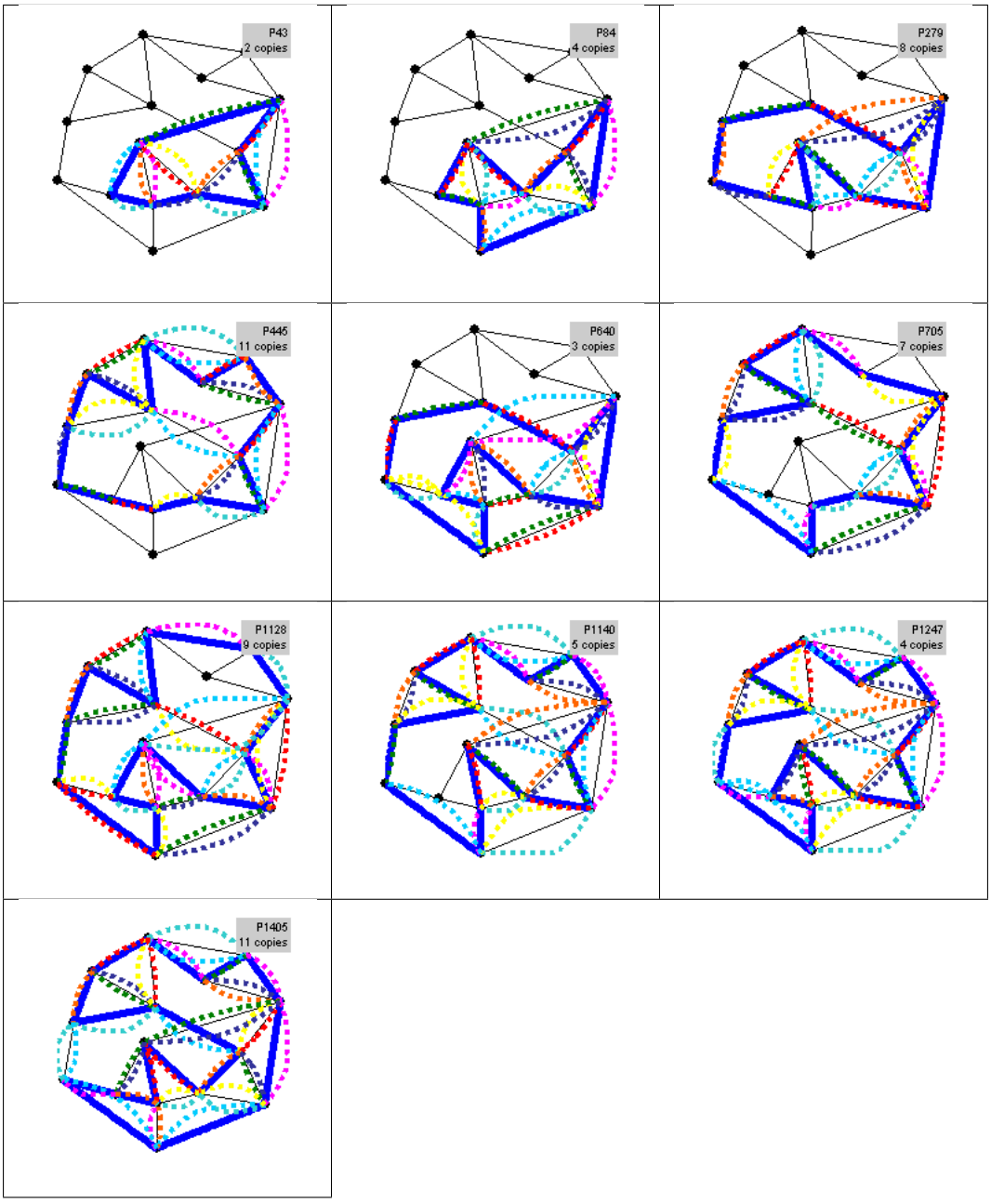


D.2 15n30s1-27s Network

D.2.1 *p*-Tree/*p*-Cycle Hybrid Design



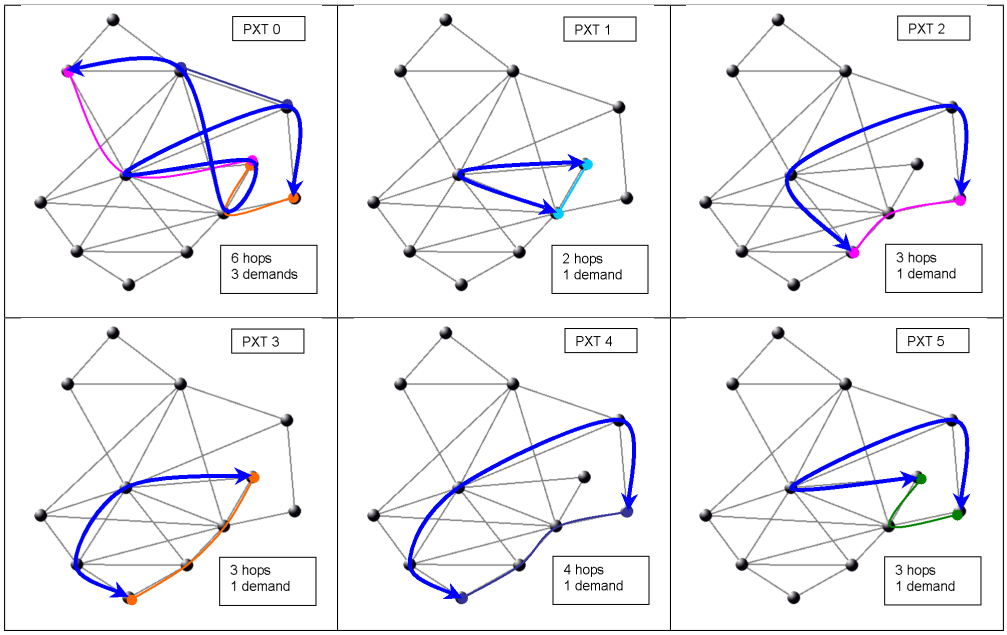
D.2.2 *p*-Segment/*p*-Cycle Hybrid Design

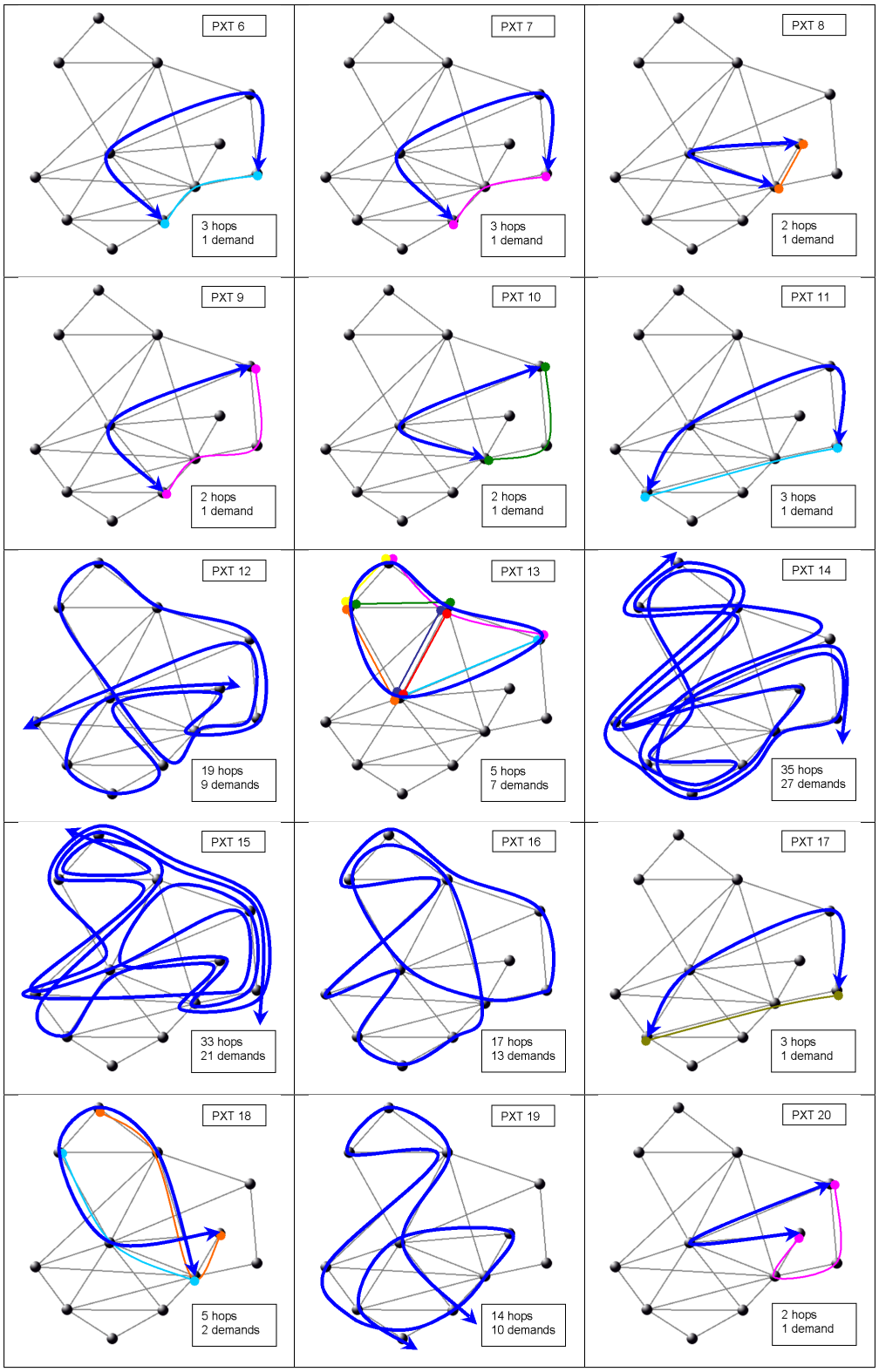


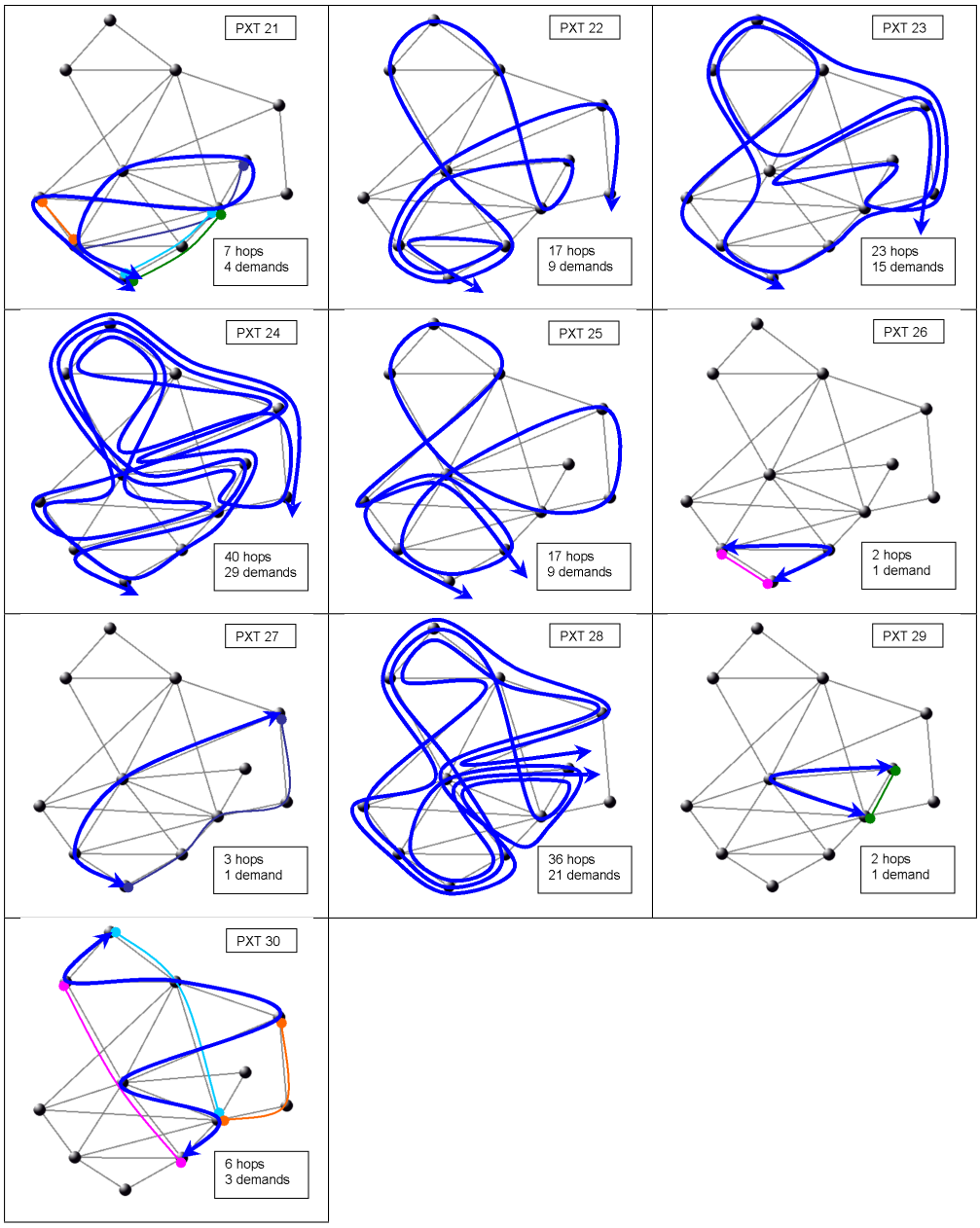
Appendix E

Structure Diagrams: Greedy Heuristic PXT Design

This appendix contains diagrams of all of the PXT structures in the greedy heuristic PXT design for the Murakami & Kim network topology. PXTs are illustrated with thick blue lines terminated with arrowheads, while demands are shown using thinner lines with circular handles at the ends. Each diagram also shows the PXT's distinguishing ID number, the length of the PXT (in hops), and the number of protected demands. Demands are not shown on diagrams where the number of protected demands is too high to illustrate clearly.



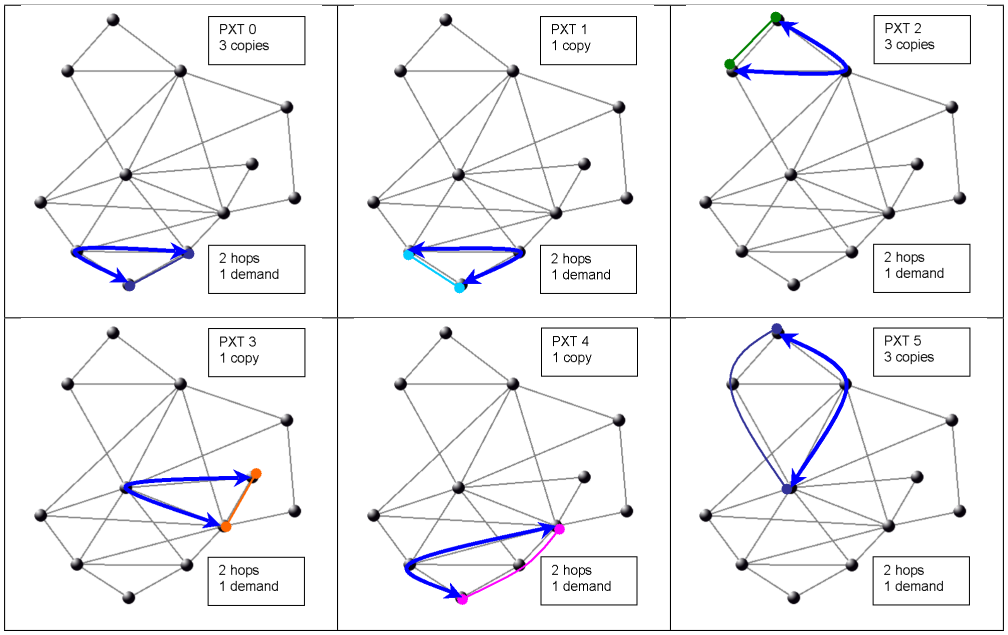

PXT 6
3 hops
1 demand
PXT 7
3 hops
1 demand
PXT 8
2 hops
1 demand
PXT 9
2 hops
1 demand
PXT 10
2 hops
1 demand
PXT 11
3 hops
1 demand
PXT 12
19 hops
9 demands
PXT 13
5 hops
7 demands
PXT 14
35 hops
27 demands
PXT 15
33 hops
21 demands
PXT 16
17 hops
13 demands
PXT 17
3 hops
1 demand
PXT 18
5 hops
2 demands
PXT 19
14 hops
10 demands
PXT 20
2 hops
1 demand

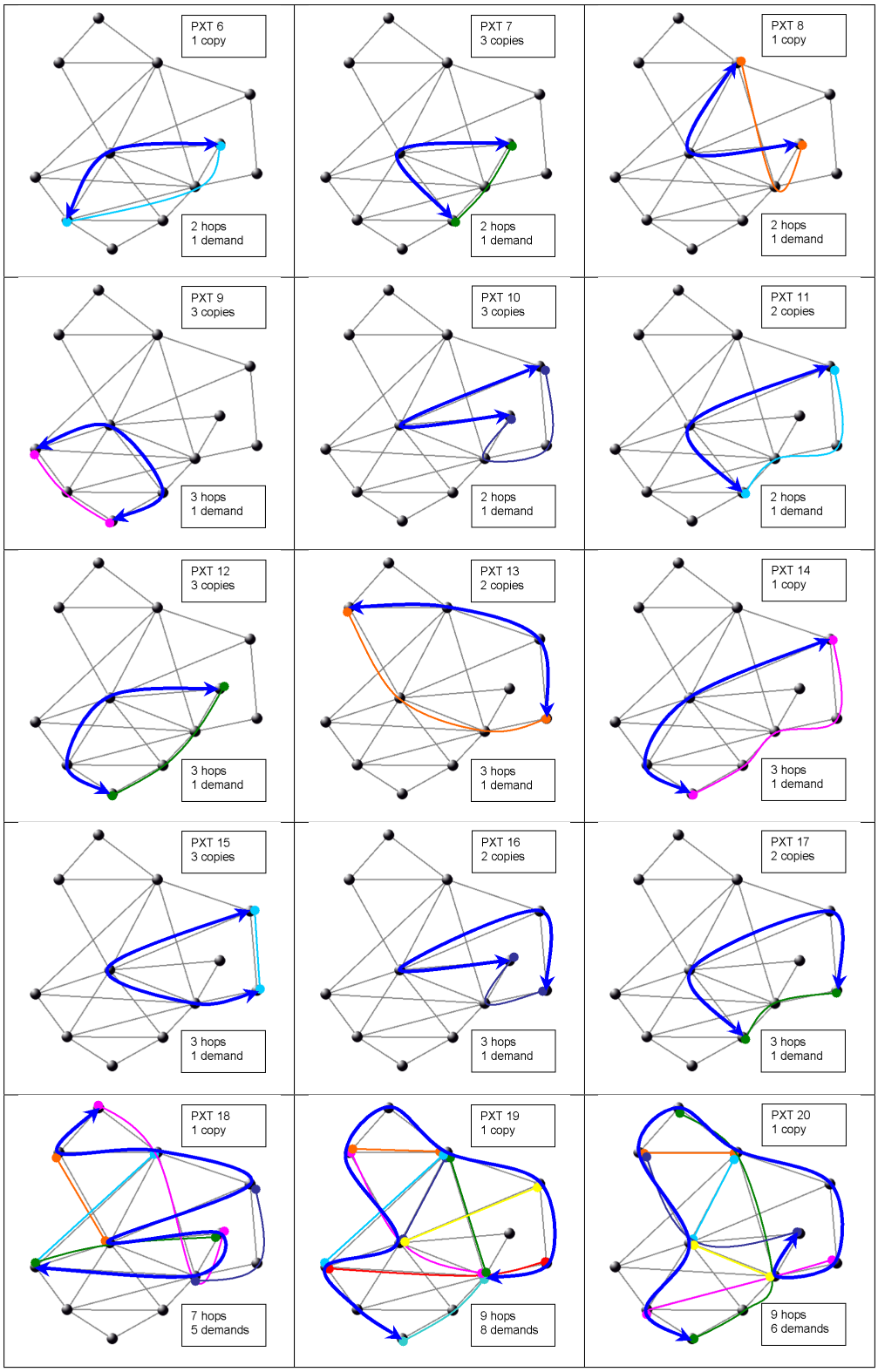

PXT 21
7 hops
4 demands
PXT 22
17 hops
9 demands
PXT 23
23 hops
15 demands
PXT 24
40 hops
29 demands
PXT 25
17 hops
9 demands
PXT 26
2 hops
1 demand
PXT 27
3 hops
1 demand
PXT 28
36 hops
21 demands
PXT 29
2 hops
1 demand
PXT 30
6 hops
3 demands

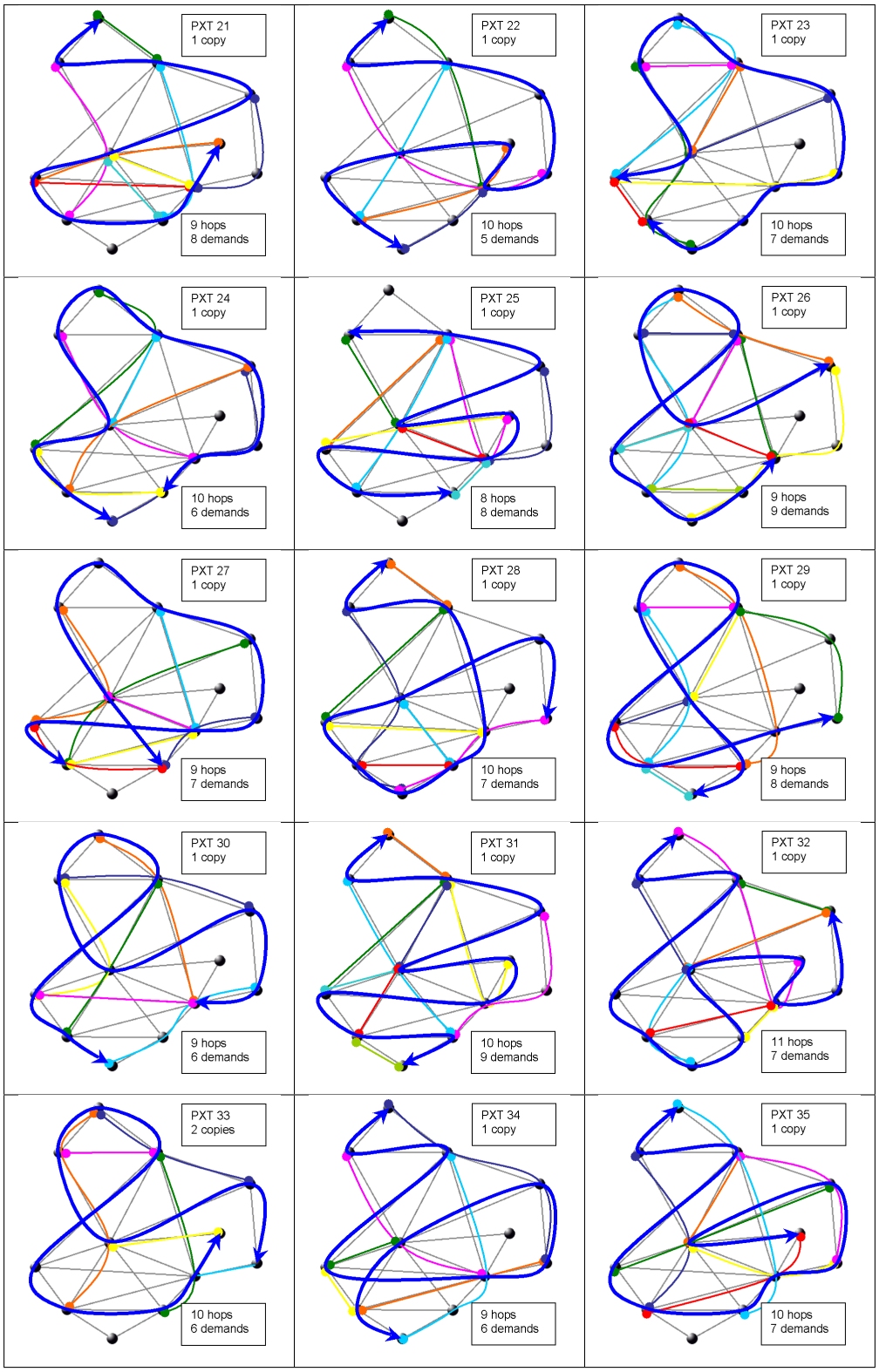
Appendix F

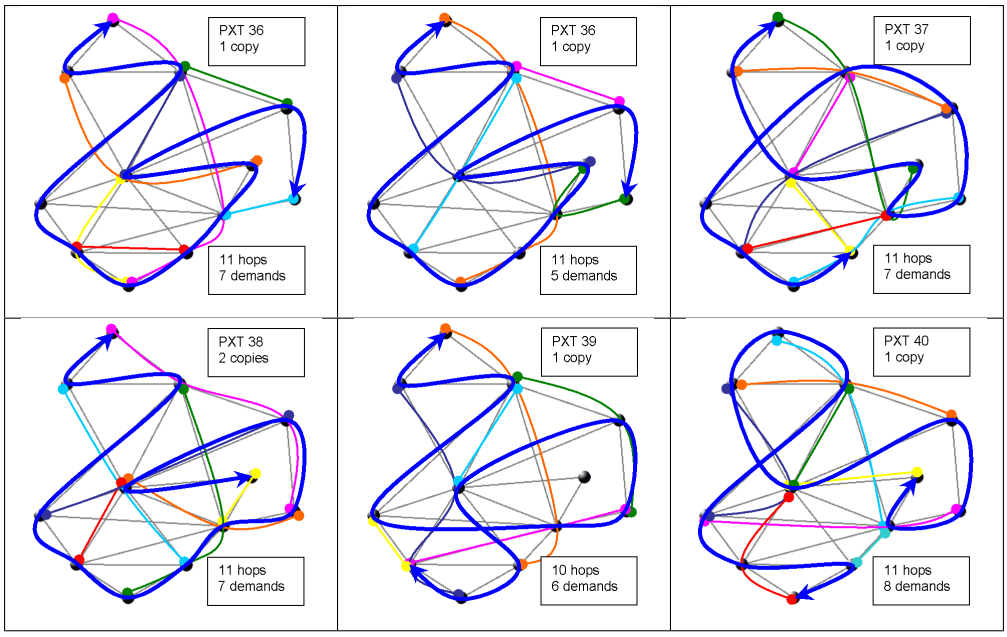
Structure Diagrams: ILP-Based Heuristic PXT Design

This appendix contains diagrams of all of the PXT structures in the PXT design generated by the ILP-based heuristic design method (i.e., the DRS-based ILP model) for the Murakami & Kim network topology. PXTs are illustrated with thick blue lines terminated with arrowheads, while demands are shown using thinner lines with circular handles at the ends. Each diagram also shows the PXT's distinguishing ID number, the length of the PXT (in hops), and the number of protected demands. Demands are not shown on diagrams where the number of protected demands is too high to illustrate clearly. Note that PXT 36 is pictured twice, because it is used to protect two different DRSs.




PXT 6
1 copy
2 hops
1 demand
PXT 7
3 copies
2 hops
1 demand
PXT 8
1 copy
2 hops
1 demand
PXT 9
3 copies
3 hops
1 demand
PXT 10
3 copies
2 hops
1 demand
PXT 11
2 copies
2 hops
1 demand
PXT 12
3 copies
3 hops
1 demand
PXT 13
2 copies
3 hops
1 demand
PXT 14
1 copy
3 hops
1 demand
PXT 15
3 copies
3 hops
1 demand
PXT 16
2 copies
3 hops
1 demand
PXT 17
2 copies
3 hops
1 demand
PXT 18
1 copy
7 hops
5 demands
PXT 19
1 copy
9 hops
8 demands
PXT 20
1 copy
9 hops
6 demands


PXT 21
1 copy
9 hops
8 demands
PXT 22
1 copy
10 hops
5 demands
PXT 23
1 copy
10 hops
7 demands
PXT 24
1 copy
10 hops
6 demands
PXT 25
1 copy
8 hops
8 demands
PXT 26
1 copy
9 hops
9 demands
PXT 27
1 copy
9 hops
7 demands
PXT 28
1 copy
10 hops
7 demands
PXT 29
1 copy
9 hops
8 demands
PXT 30
1 copy
9 hops
6 demands
PXT 31
1 copy
10 hops
9 demands
PXT 32
1 copy
11 hops
7 demands
PXT 33
2 copies
10 hops
6 demands
PXT 34
1 copy
9 hops
6 demands
PXT 35
1 copy
10 hops
7 demands

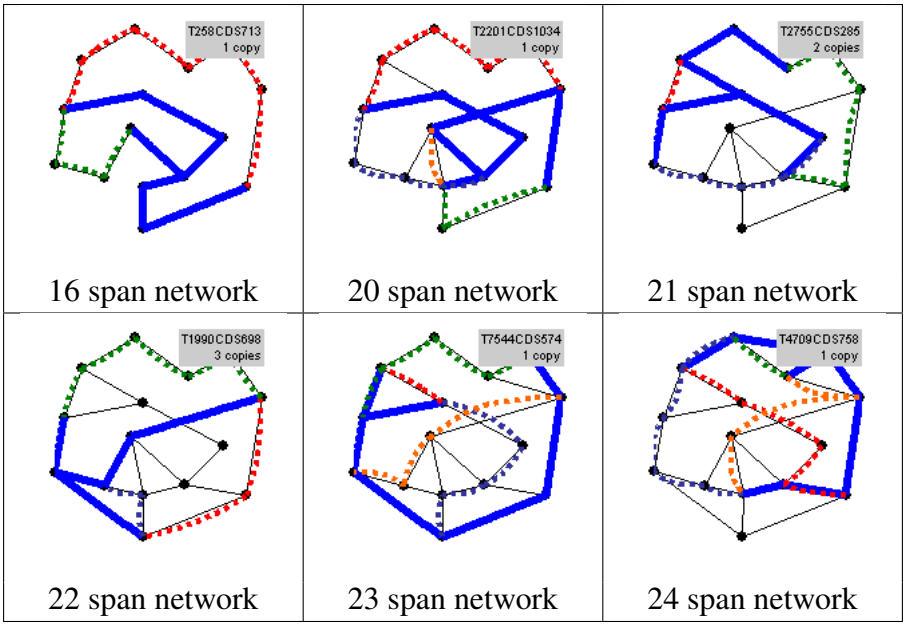

PXT 36
1 copy
11 hops
7 demands
PXT 36
1 copy
11 hops
5 demands
PXT 37
1 copy
11 hops
7 demands
PXT 38
2 copies
11 hops
7 demands
PXT 39
1 copy
10 hops
6 demands
PXT 40
1 copy
11 hops
8 demands

Appendix G

Structure Diagrams: True Tree Diagrams for Path *p*-Tree/FIPP *p*-Cycle Hybrid Designs

G.1 Designs Using ">50% Protection" DRS Protection Criterion

The following diagrams show all of the true tree structures used in the hybrid path *p*-tree/FIPP *p*-cycle designs that were generated for the 15n30s1 network family. These designs used the method of setting $x_c^k = 1$ whenever PXT k could protect *more than half* of the demands in DRS c. Structures are drawn with solid lines. Protected working paths are shown as dashed lines.

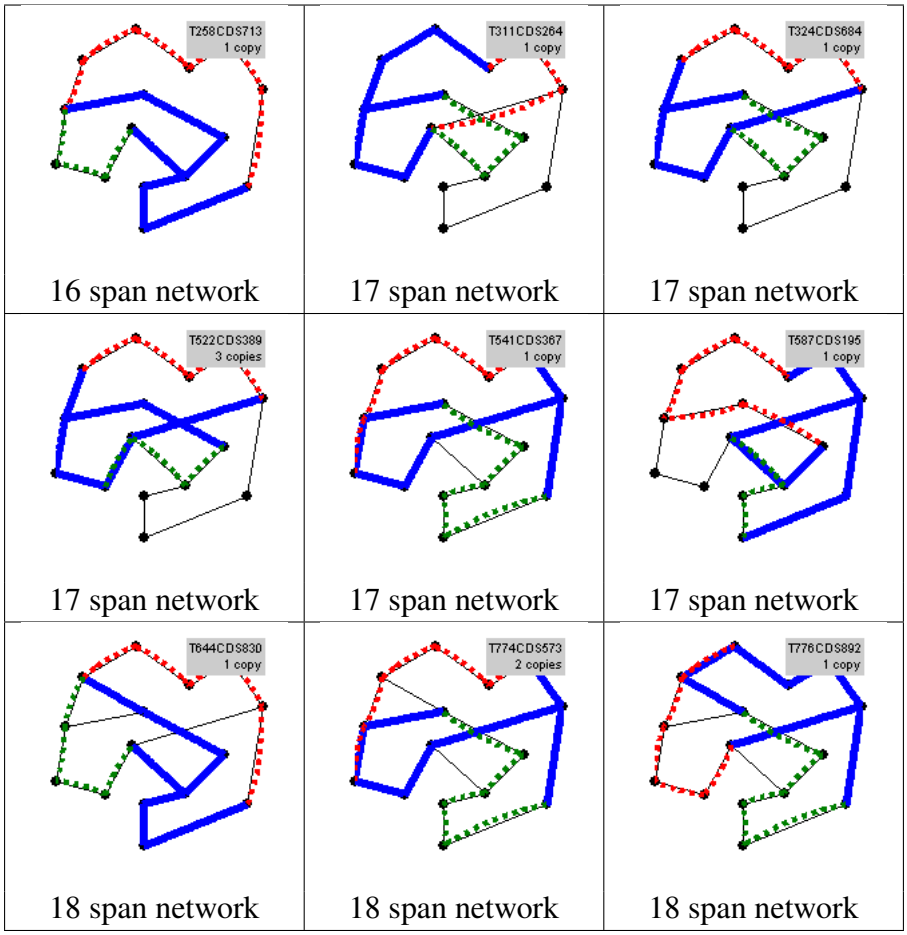


16 span network | 20 span network | 21 span network

22 span network | 23 span network | 24 span network

G.2 Designs Using "Any Protection" DRS Protection Criterion

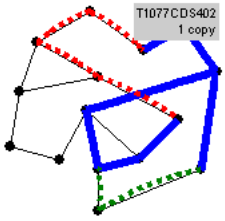
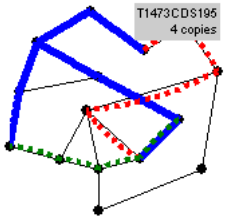
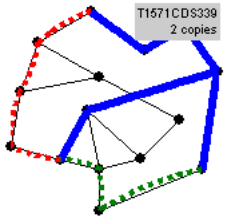
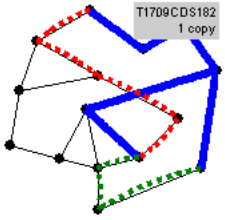
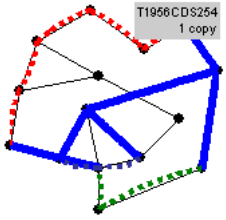
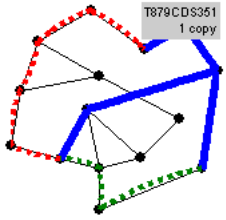
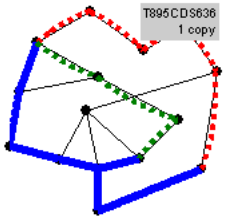
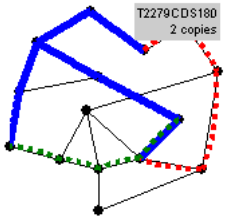
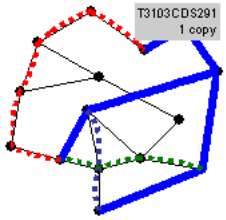
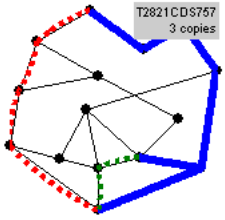
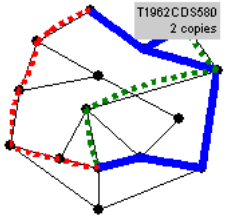
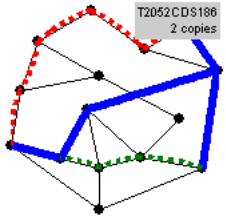
The following diagrams show all of the true tree structures used in the hybrid path *p*-tree/FIPP *p*-cycle designs that were generated for the 15n30s1 network family. Note that designs were only solvable for the *16 through 26 span network*s. These designs used the method of setting $x_c^k = 1$ whenever PXT k could protect *any* of the demands in DRS c. As such, they contain many more true trees than those shown in the previous Section (Appendix G.1). Structures are drawn with solid lines. Protected working paths are shown as dashed lines.

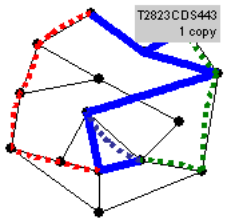
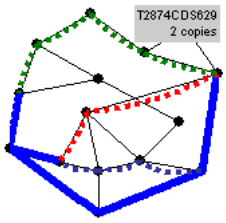
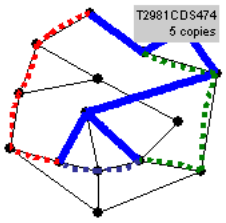
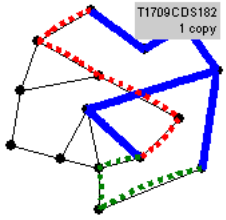
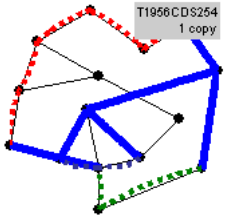
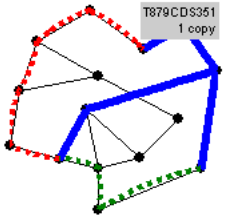
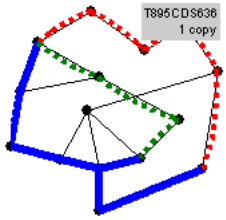
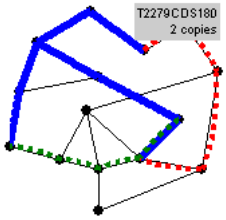
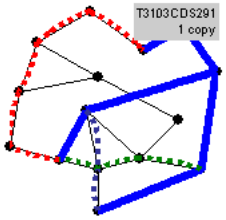
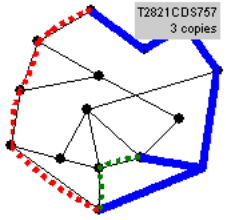
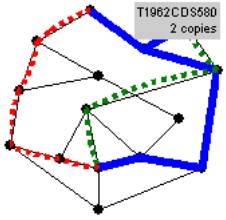


16 span network | 17 span network | 17 span network

17 span network | 17 span network | 17 span network

18 span network | 18 span network | 18 span network

 19 span network	 20 span network	 20 span network
 20 span network	 20 span network	 20 span network
 20 span network	 21 span network	 21 span network
 22 span network	 23 span network	 23 span network

 23 span network	 23 span network	 23 span network
 23 span network	 23 span network	 23 span network
 23 span network	 25 span network	 25 span network
 25 span network	 26 span network	