uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Sadrul Habib Chowdhury
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Performance Comparisons of Selected Protection Algorithms for Optical Networks

TITRE DE LA THÈSE / TITLE OF THESIS

Oliver Young
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Nejib Zaguia

Pat Morin

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Performance Comparisons of Selected Protection Algorithms for Optical Networks

Sadrul Habib Chowdhury

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa

January 19, 2009

Your file Votre référence
ISBN: 978-0-494-59897-9
Our file Notre référence
ISBN: 978-0-494-59897-9

Abstract

Optical networking is a promising solution to support the ever increasing Internet traffic. This thesis aims to investigate the important issue of network protection after a failure occurs in the network. We propose a framework for measuring the performances of network protection algorithms for various protection metrics. The framework is designed to easily and quickly collect protection measures for an algorithm that can be effectively compared to the same protection measure of another algorithm. We then use this framework to compute and compare the performances of various network protection algorithms, mainly between algorithms that use p-cycles and ones that use protection trees. Different performance measures including restorability, sharability and change in data-path length are used. We use the framework on randomly generated mesh networks with varying topology parameters such as network size, network order, average nodal degree etc. We use the result to study the effect of the variation in different network parameters on the performances of the protection algorithms. Finally, commercial networks are used as specific application examples.

Acknowledgement

I would like to express my sincere thanks to my supervisor, Professor Oliver W W Yang for his research supervision and guidance. I would also like to thank the members in the Computer Communication Network Research (CCNR) group in the School of Information Technology and Engineering (SITE) of the University of Ottawa for their continuous cooperation and encouragement.

I must also express my deepest gratitude to my parents for their endless love and encouragement. This work would not have succeeded without their continuous moral support.

Table of Contents

List of Figures

List of Tables

Table of Acronyms, Notations and Symbols

		Page of 1st appearance
AND	Average Nodal Degree	17
ATM	Asynchronous Transfer Mode	1
CIDA	Capacitated Iterative Design Algorithm	35
DWDM	Dense Wavelength Division Multiplexing	1
GMPLS	Generalized Multi-protocol Label Switching	2
IP	Internet Protocol	1
MM	Metric Module	31
MPLS	Multi-protocol Label Switching	2
MSCC	Min-Sum Cycle Cover Detection/Protection	45
MSCT	Maximum Spare Capacity Tree	6
MST	Minimum Spanning Tree	8
MST2SP	Maximum Spare-capacity Tree and Two Shortest Path	48
p-Cycle	Protection Cycle	5
PM	Protection Module	31
QoS	Quality of Service	4
RWA	Routing and Wavelength Assignment	20
SLA	Straddling Link Algorithm	4
SONET	Synchronous Optical Network	1
TDM	Time Division Multiplexing	1
TT	Token Tree Algorithm	4

Chapter 1
Introduction

High-speed networks are used for various purposes in modern world, including streaming media, e-commerce etc. To ensure the reliability of such networks, it is very important that they are capable of continuing high-speed data transmission even if some failure occurs. In fact, maximum acceptable downtimes have been specified for various services by standard organizations [T1A193]. Based on that reference, a voice call may be dropped after 200 milliseconds of interruption, while a data session typically times out within after a few seconds, or even a few minutes. Interruptions of 50 milliseconds or lower are usually transparent to network services.

Dense Wavelength Division Multiplexing (DWDM) [Kart99] is the process of multiplexing signals of different wavelengths onto a single fiber. This technique increases the transmission capacity dramatically without the need for expensive re-cabling, this cutting down the cost of bandwidth. These developments and the advances in DWDM for optical networking lead to an IP (Internet Protocol) over optics paradigm.

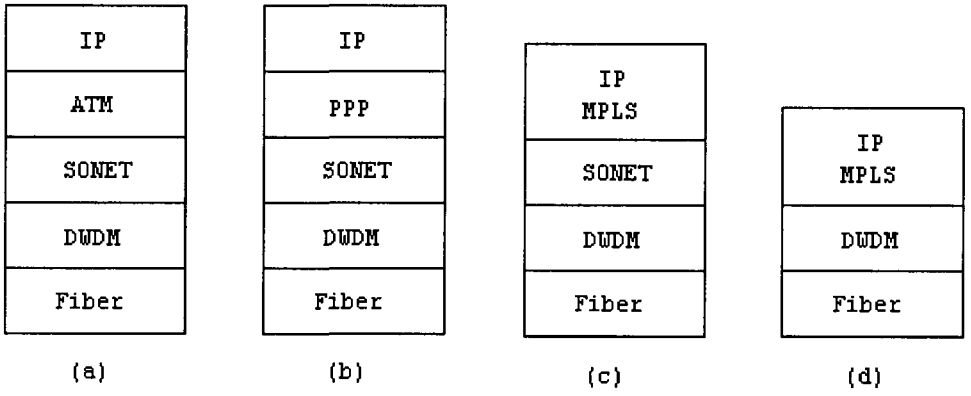


Figure 1.1: Evolution of the protocol stack of DWDM Networks

Multiple protocol layers co-exist during the early days of DWDM networking, e.g. SONET (Synchronous Optical Network), ATM (Asynchronous Transfer Mode), IP, DWDM. Such multi-layer approach requires the deployment of many expensive ATM and SONET devices. Each layer also tends to have its own management systems. And

any one layer can act as a bottleneck to throughput and speed. Therefore, a model with a reduced number of protocol layers is attractive because of the reduction in cost and complexity. A two layer model: an IP-based services layer with MPLS, and a DWDM optical switched transport layer, is very attractive. In this reduced protocol stack, many of the functionalities of the eloped layers are performed by the existing layers. For example, the role of the ATM layer is assumed by the IP layer, with MPLS and GMPLS to replace ATM. Similarly, many tasks of the SONET layer are done by the optical cross-connecting and transport layers. Some functions of SONET, e.g. framing, error monitoring etc. cannot be eliminated, but they can be performed by lightweight implementation of the full SONET standards, or by digital wrappers [Grov03]. This reduced layer of protocols also needs to provide a mechanism to guard against network failures, and a fast restoration strategy. Implementation of such network restoration strategy in the DWDM layer attracts a lot of attention because the restoration process can be performed very fast and efficiently in this layer.

The level of service restoration offered by network service provides depends on the type of the service. Dedicated protection [LeGr02] is referred to as the *gold* class of service, and it offers restoration times of 60 milliseconds or less from the time of the failure. This guarantees against interruptions for virtually any network services. Likewise, a shared link restoration [LeGr02] falls in the *silver* class of service, and offers restoration times of 100 to 500 milliseconds. A *bronze* class offers restoration times of 2-10 seconds as is typical for IP layer path re-routing. Cheaper classes of service may include unprotected services or services where the traffic can be interrupted to accommodate higher priority traffic. In order to achieve restoration time targets for these classes of services, various methods have been researched and proposed. We review a number of such methods.

1.1 Classifications of Protection/Restoration algorithms

The basic idea to make a network fault tolerant is to find an alternate route in order to send data to the destination after a failure occurs in a network. There can be various classification criteria:

1.1.1 Location of Protection/Restoration

A failure in a network can happen in either a node or a link connecting two nodes. The algorithms can be classified into two classes based on what kind of failure they can protect against: (1) Node failure, and (2) Link failure.

Node Failure

This class of algorithms can provide protection against a failure in a node. A node failure can happen if a node gets disconnected from the entire network, quite possibly by some natural disaster. Algorithms that can protect against node failures are more resilient because they can essentially handle simultaneous multiple link failures. The "Dual Tree" scheme [FeCu01], for example, creates an additional secondary tree as a complement to the primary tree. This secondary tree provides the backup path that can be used when a node in the primary tree fails. The "Redundant Tree" algorithm [MeFi99] also creates node-redundant protection trees to provide backup paths. However, the approach in this algorithm is to create two directional trees for each node such that any common edge between the two trees has opposite directions in them.

Link Failure

This class of algorithms can provide protection against only a failure in a link. A link failure can happen if a link connecting two nodes is severed, by natural disasters, for example, or a fabric failure in a node. The Redundant Tree algorithm [MeFi99] discussed earlier can also protection against link failures by creating link-redundant trees, instead of node-redundant trees. The Token Tree algorithm [ZhYa02a] creates a heuristic based spanning tree. The algorithm uses the spanning tree to provide the backup path for any link failure. In the event that a link on the protection tree itself fails, the algorithm finds a shortest path between the nodes and uses that to provide the backup path. The Multiple Shared Backup Cycles algorithm [HwAh01] also protects against link failures. However, it uses cycles to protect the network. It creates m cycles for each link, where each cycle protects $1/m$ of the bandwidth of the link.

1.1.2 Traffic Diversion of Protection/Restoration

The algorithms can also be classified into two classes depending on how they reroute the affected traffic: (1) Link protection, and (2) Path protection.

Link Protection

An algorithm in this class finds an alternate path that connects the incident nodes of the failed link. As a result, only one protection path is used for one link failure, and only the nodes in that protection path need to be aware of the failure. However, since a link is replaced by a path, the length of the end-to-end path can increase, which can have an adverse effect on the quality of service of the network. The Token Tree (TT) algorithm [ZhYa02a] discussed in an earlier section provides link protection. The Straddling Link Algorithm (SLA) [ZhYa02b] creates a protection cycle for each link, and uses that protection cycle to reroute the traffic on the link if the link fails. The Self Healing Ring architecture [WuLa90] also restores all the data of a failed link along the same protection path. The Hierarchical Protection Tree Scheme [ShYa04] also provides link protection by rerouting the data of the failed link through the backup path along the rooted protection tree.

Path Protection

An algorithm that finds an alternate end-to-end path for the disrupted traffic falls into this class. Alternate source-to-destination paths are calculated for each different path that sent traffic through the failed link. This requires all the affected sources, the destinations and the nodes on all the alternate paths to be aware of the failure. However, because there are multiple protection paths, it is more likely that all the affected traffic can be successfully restored after the failure. The Redundant Tree algorithm [MeFi99] creates multiple rooted tree for each node. One of the trees is used to route the primary data. If a link on the tree fails, then the data is rerouted over the backup tree, thus changing the entire data-path of the traffic. The QoS Redundant Tree algorithms [XuCh02, ZhXu06] also use the same algorithm, but enhance it by providing an assured quality of service (QoS). Failure-Independent Path Protecting cycles [KoGr05] provide alternate end-to-end backup paths using pre-connected protection paths.

1.1.3 Timing of Protection/Restoration

The algorithms can be classified into two classes based on when this alternate path is calculated: (1) Dynamic (Restoration), and (2) Pre-determined (Protection).

Dynamic (Restoration)

The algorithms in this class determine the restoration paths after the failure occurs in the network. Such algorithms have the knowledge of the current state of the network, such as the loads on the network nodes and the links. This knowledge enables the algorithms to find the best alternate path for the disrupted traffic. However, because this calculation happens after the failure happens, such algorithms need to be very fast, and a moderately fast algorithm can still be too slow for the desired quality of service of the network. There are very few algorithms that provide dynamic protection. The Highly Efficient Path-Restoration algorithm [IrGr00] is a fast algorithm that can be used in real-time for dynamic protection. The algorithm uses a heuristic algorithm to provide a backup path with an assured restoration level. The hierarchical tree algorithm [ShYa04] can also be used to provide dynamic backup paths. This algorithm provides a mechanism to reconfigure the protection tree after a failure happens. The reconfiguration process is very fast, and can provide backup path for subsequent failures. Also, due to the improvement in processing capabilities, a number of algorithms that were originally proposed to provide pre-determined protection can be used to dynamic restoration.

Pre-determined (Protection)

This class of algorithms calculates the protection path before the failure occurs in the network. As a result, these protection paths may not be the best choice among the available alternate paths at the moment of failure. However, because the paths are pre-determined, the traffic affected by the failure can be very quickly rerouted to the destination. Most of the algorithms in the literature fall in this category. For example, the Heuristic Method p-cycle algorithm [ZhZh03] creates p-Cycles beforehand, and uses the cycles to provide the backup paths for a network failure when it happens. The Optimum route algorithm [Fris63] also computes the protection nets before the failures happen. These pre-computed protection nets provide the backup paths for the affected traffic after

a network failure.

1.1.4 Logical Topology

A physical network topology is the layout of physical network facilities, including physical fiber spans and optical ADMs (Add-Drop Multiplexers), switches, routers etc. A logical network topology is built on top of a physical topology and seen as a graph representing all nodes and logical inter-nodal connections, which may be a WDM channel, an ATM virtual circuit, a TCP connection etc. More details about logical topologies are provided in Chapter 2. Almost all of the algorithms can be classified into two classes, based on the logical topology used by the algorithms to protect a network: (1) Tree-based algorithms, and (2) Cycle-based algorithms.

Tree-based Algorithms

There are several protection algorithms that use logical tree topologies to protect the network against failures. A tree algorithm creates one or more spanning trees for the network, and uses the tree to find the backup path for a data-path when the path is severed due to a network failure. For example, the Token Tree (TT) algorithm [ZhYa02a], as discussed before, creates a spanning tree using a heuristic algorithm which provides backup paths for failed links. If a link on the tree itself fails, the TT algorithm finds the shortest path between the nodes adjacent to the failed link, and uses the shortest path for backup. The Maximum Spare Capacity Tree (MSCT) based 2-Shortest Path algorithm [LiYa03] also uses a similar approach. However, this algorithm creates two edge-disjoint shortest paths connecting each pair of nodes adjacent to the links on the protection tree. The Hierarchical Protection Tree [ShYa04] creates a hierarchical protection tree based on a heuristic regarding the local density of the nodes in the network. The protection tree is constructed in a way that it can reconfigure itself after a failure happens to protect against a second failure. There are also tree algorithms that create multiple protection trees. For example, the Dual Tree algorithm [FeCu01] creates an additional backup tree for the network to use for protection. The Redundant Tree algorithms [MeFi99, XuCh02, ZhXu06] also create multiple protection trees. For example, [MeFi99] creates a different backup tree for each node.

Cycle-based Algorithms

Using rings, or cycles, to make networks fault tolerant is a very widely used idea. A Self Healing Ring (SHR) consists of a ring of bidirectional links between a set of network nodes. In normal dispatch, traffic is routed over the shortest path towards the destination node along the ring. In the event of a failure in the data-path, the data is rerouted over the only existing path between the nodes along the ring. Two commonly used SHR rings in SONET are Unidirectional Path Switched Ring (UPSR) and Bidirectional Line Switched Ring (BLSR) [Bell95a, Bell95b, GeRa00a, GeRa00b, RaSi02]. In UPSR, each pair of nodes is connected by two link and node disjoint paths. Each link between the nodes contains two fibers. Traffic from the source node is directed simultaneously in both directions along the ring, but on different fibers. In the case of a failure in the working path, the destination node starts receiving signal from the protecting path, thus restoring the network. In BLSR, the nodes are connected by four fibers. Two fibers are used as working fibers, and two are used for protection. Unlike in UPSR, traffic in BLSR can be carried on both directions along the ring. SHR rings can also protect WDM networks [KiLe98].

A p-cycle, or a protection cycle, is a ring with straddling links [Grov98]. A straddling link is an off-ring link that connects two nodes in the ring. For a failure on an on-cycle link, the protection works like a typical self-healing ring by rerouting the traffic along the only available path along the cycle. However, for a failure in a straddling link, data can be routed over both the available paths along the cycle. The initial step in the solution requires all simple cycles to be enumerated from the network graph. Various cycle enumeration algorithms have been used for this purpose, e.g. [HwAh01, AmLi03, DePr82, Hort87] etc. Finding the optimal set of p-cycles from the generated cycles to protect all the links can be done jointly or non-jointly with the demand routing [GrDo02, GrSc02]. There have been a few approaches to solve this NP-complete problem in a reasonable fashion. For example, The Straddling Link Algorithm [ZhYa02b] provides a simple and fast procedure to generate a set of elemental p-cycles, each with one straddling link. More complex cycles can be formed by performing some operations on the generated primary p-cycles [DoHe03]. The resulting p-cycles can have multiple straddling links, thus reducing the complexity of the restoration process. In [GrSh03] the

authors extend the idea of p-cycles to path-segment protection rather than only on-cycle and straddling links protection. This implies some improvement in the spare capacity usage, although the complexity of the problem is rather increased.

1.1.5 Decision Making for Protection/Restoration

The algorithms can be classified as distributed, or centralized depending on how and where the decision is made to determine the protection path after a network failure happens [WuTs92].

Distributed Algorithm

In a distributed algorithm, no central repository is maintained to store information about the state of the network. The nodes affected by the network failure use the knowledge regarding its neighbors to compute the necessary backup paths. The nodes may be required to corroborate connectivity information among its neighbor nodes in the process. For example, the Pick-up-Parent Algorithm (PUP) [ZhYa02c] is a distributed heuristic based MST algorithm. In this algorithm, a node initiates the restoration process upon receiving a special message *(pick-up-parent message)*. The restoration process is carried out by the nodes constructing a spanning tree through passing messages. Only one node is *active* at any given time, and the *active* node adds its incident links to the protection tree, and sends a *pick-up-parent message* to one of its neighboring *passive* nodes, which then becomes the active node and carries out the same process until a spanning tree is constructed. Thus the entire spanning tree is constructed without any centralized server or node controlling the entire process. The Hierarchical Protection Tree Scheme [ShYa04] and the SLA algorithm [ZhYa02b] are also distributed algorithm because these algorithms do not require the information of the entire network configuration. Rather, these algorithms use message passing mechanisms to learn about the local network configuration and create the protection paths.

Centralized Algorithm

For a centralized algorithm, a central controller is responsible for selecting the protection path for a disrupted data-path due to the network failure. Thus, the controller needs to

maintain the updated information about the entire network it wants to protect, and every time the state of the network changes, the information in the controller needs to be notified and updated accordingly. For example, the Redundant Tree for Pre-planned Recovery [MeFi99], the MSCT algorithm [LiYa03], the TT algorithm [ZhYa02a] are centralized algorithm. The algorithms use the knowledge of the entire network configuration to provide the protection paths. The centralized algorithms can usually be modified to work in a distributed fashion. However, this is often not feasible for most algorithms.

1.2 Motivation

As discussed, there are many protection algorithms for network failures of various classes. Due to the differences, it is very difficult to compare the performances of different algorithms effectively. Not much work has been done in the literature in this regard. Whenever a new algorithm is proposed, it is usually compared to other algorithms of the same topologies. However, there is no standard model for doing this comparison. There have been some studies comparing between two classes of algorithms. Mathematical formulas have been used in [ChJa05] to compute and compare the reliability of the ring-based protection algorithms with that of the p-cycle based protection algorithms. The comparison presented shows that the reliability of UPSR and BLSR rings are better than that of p-cycle based algorithms. There have also been some studies comparing particular algorithms. For example, in [GrGr07], the authors present a model to compare capacity efficiency of the Hierarchical protection scheme [ShYa04] with generic p-cycle algorithms. However, the particular model cannot be used to compare algorithms of other classes, or for other performance measures. For this reason, it is necessary to devise a model that can be used to compare the performances of different algorithms for various performance metrics. It is also unknown how different network parameters, e.g., network size, redundancy etc. affect the performances of the protection algorithms. While an algorithm can be most suitable for a particular network, there may be better alternatives for a different network. Therefore, it is important that the performances of protection algorithms are measured with networks of various configurations and the effects of changes in the configuration on the performance are

analyzed. This can also be very critical when selecting a protection algorithm for a network that is expanded to grow larger over time.

1.3 Objectives

We are interested in general to study how various network parameters affect the performance of the protection algorithms. For this reason, we want to do the following:

1) Design a generic framework for measuring performances of various protection algorithms for different performance metrics in an easy and efficient way.
2) Study the effect of varying network configurations on the performances of the algorithms.
3) Make comparisons between several tree-based and cycle-based algorithms on specific performance measures.

1.4 Methodologies

In order to compare the performances of different algorithms on various performance metrics under various network parameters, we design and implement a framework for measuring the performances of various protection algorithms. The framework provides a convenient and efficient way of measuring and comparing the performances of different algorithms for different performance measures. We also implement protection modules for a number of algorithms and metric modules for a number of performance measures. C was used as the implementation language for the modules and the framework, because it is a fairly low-level programming language and it can provide the required optimized computation that can be very useful when measuring performances of complex algorithms. Although there are also existing simulation tools, e.g. NS-2, OPNET etc. for their implementation, we chose to develop the model as a standalone tool primarily because a standalone tool does not have to be restricted by the model of any existing simulation tool, thus providing greater ease of development, especially in the primary stages. Also, it helps mitigate the effect of other existing modules of a simulation tool that may inadvertently interact with our module. It should be noted that the implementation of the model was designed with easy integration with afore mentioned simulation tools in mind. We also generate random networks with different values for the

network parameters, e.g., network order, density etc. The random network generation process is detailed in section 2.3. For each network configuration, we generate 10 random networks, and use the framework to measure the performances of the algorithms on different metrics. We then compute the average value for the performance from these random networks, and use that as a representative value of the performance for a network of that particular configuration. In addition to the generated random networks, we also use several real networks of different representative topologies for our simulations with some changes to meet our requirement of 2-edge connectivity. For some of these real networks, we compute the link redundancy from the traffic matrix when available, while for some others we generate the redundancies using probabilistic methods. We use our framework to measure the performances of algorithms that use p-cycles for protection. We also measure the performances of algorithms that use spanning trees to protect the network. We then investigate the effect of various network parameters on the performances of the protection algorithms. The network parameters include number of nodes in the network, average degree of the nodes in the network etc. Finally, we compare the performances of the algorithms from the results of our measurements. We also show the differences in how the network parameters affect the performances of different algorithms.

1.5 Contributions

The main contributions of this research are:

1) A novel framework for computing performance of network protection algorithms on different protection metrics.
2) Experiment results of the effect of network parameters on the performances of the algorithms.
3) Provide a generic and new way of comparing network protection algorithms.
4) Comparison of several cycle-based and tree-based algorithms.

1.6 Publication

Sadrul Habib Chowdhury, Oliver W W Yang, "*A Generic Framework for Measuring Performance Metrics of Network Protection Algorithms*", 11th Communications and Networking Simulation Symposium, 2008 Spring Simulation Multiconference, April 2008.

1.7 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 includes necessary definitions and assumptions for this thesis, and describes the network topology model for the simulations. Chapter 3 proposes a novel framework for measuring performance of protection algorithms. Chapter 4 and 5 presents and analyzes the simulation results of p-cycle algorithms and tree-algorithms, respectively. Chapter 6 compares the performances of the algorithms for networks with varying network parameters. Chapter 7 concludes our research work. Some supplemental materials are provided in the appendices.

Chapter 2
Network Operations, Models, Definitions and Assumptions

This chapter states some definitions and assumptions used throughout this thesis, reviews various network models and defines our representation of a network and various network parameters.

2.1 Network Operations and Model

2.1.1 Network Operations

We consider a mesh WDM network of N nodes interconnected by M links. Each node is a plain optical switch or a wavelength router. The key elements required to realize these nodes are wavelength multiplexers and demultiplexers, switches and/or wavelength converters. A wavelength converter is a device that takes in data at one wavelength and can output it on a different wavelength. Figure 2.1 shows the use of wavelength converters. The data coming from wavelength λ_1 can be sent over λ_2 or λ_3 with the help of a wavelength converter.

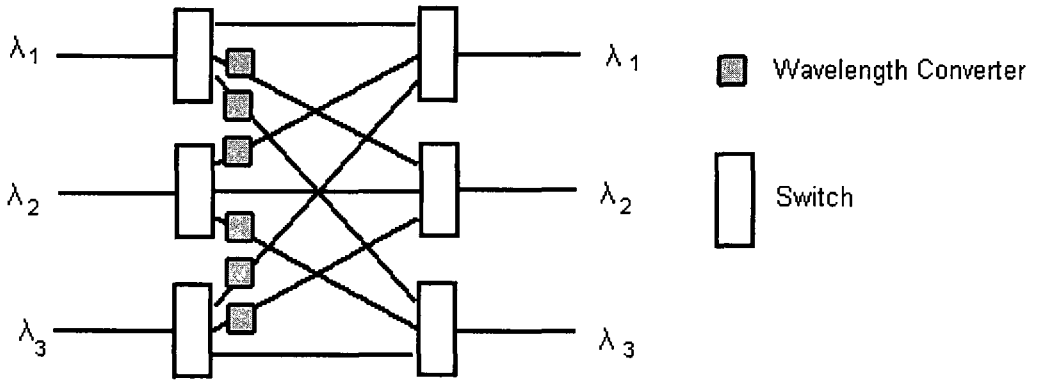


Figure 2.1: Using Wavelength Converters

Figure 2.2 shows an all-optical switch with wavelength conversion capability. This device is capable of transferring data from an incoming fiber to an outgoing fiber. For example, data coming in from Fiber 1 is transferred over to Fiber 2 .The optical switch

uses the elements in Fig.2.1 as building blocks. A multiplexer (mux) combines signals at

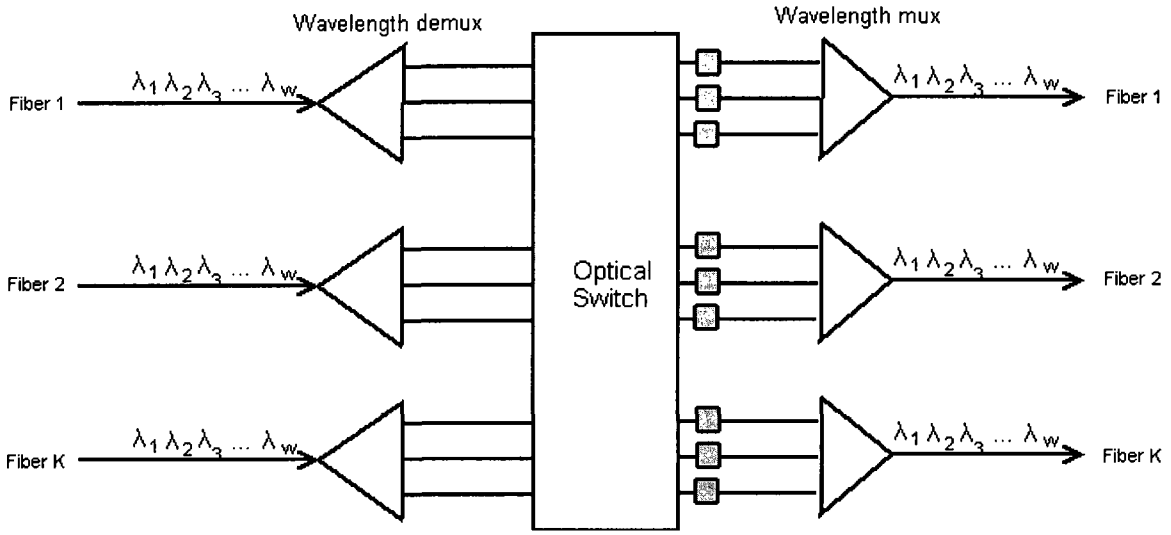
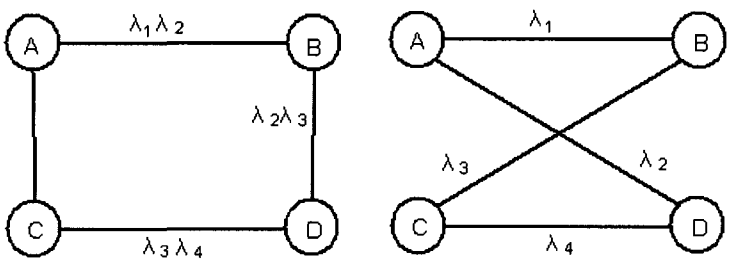


Figure 2.2: Optical Switch with wavelength conversion

different wavelengths from its input ports to a common output port, and a demultiplexer (demux) performs the opposite function. For example, data coming over Fiber 1 in wavelength λ_1 can be sent over wavelength λ_2 in Fiber 2.

A physical network topology is the layout of physical network resources, including physical fiber spans, optical ADMs, switches or routers. On the other hand, a logical network topology is built on top of a physical topology and seen as a graph representing all the nodes and their logical inter-nodal connections, which may be WDM channels, ATM virtual circuits or TCP connections.



(a) Physical topology (b) WDM Layer Logical Topology

Figure 2.3: Physical Topology vs. Logical Topology

In Figure 2.3, the nodes A, B, C and D represent optical switches. There is no physical link between nodes A and D (Fig. 2.3 (a)). However, a logical topology built on this physical topology can have a logical link AD. This link will be consisted of the λ_2 light-

wave routed over the AB and BD physical links through the optical switch at node B. Similarly, a logical link BC can be constructed by the light-wave λ_3 going over physical spans BD and CD and through physical node D.

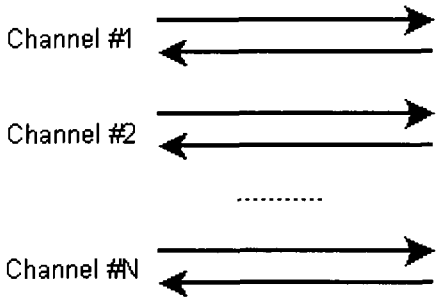


Figure 2.4: Link Model

A 'link' in our network graph is actually a physical span, within which there may be multiple fibers running in parallel between two end nodes. We assume that each end-to-end traffic demand is served by a pair of bi-directional light paths, and those two light paths take the same route. We also assume that on each link, there are N pairs of bi-directional wavelengths, as shown in Fig 2.4.

2.1.2 Modeling of Wavelength Routing and Distribution

Random demand models are used to specify the end-to-end connection requests between each pair of network nodes. Some studies consider a dynamic demand model where the demands arrive at and depart nodes at a rate based on Poisson distribution (e.g. [Homo04]). Such model could also include connection time lengths for tearing down the connection after a random period of activity. In this case, typically an on-off model could be used [Schu04]. Dynamic demand models are more appropriate for studying demand blocking probabilities. In studying restorability and network redundancies, static demand models are often used, a number of them reviewed in [Douc05]. The most common model that was also used in [Douc05] and [Shah07] is the uniform random model that assumes connections between pairs of nodes with no bias. An *NxN* demand matrix is constructed and populated by random integer numbers between one (or zero if no demand case is allowed) and a maximum demand intensity that indicates the scale of difference

between demand nodes. For real-life commercial networks, we use traffic matrix to compute link redundancy when available. We also use the Bayesian estimation traffic matrix model discussed in [MaTa03] to create the network demands for the real networks.

We use a homogeneous link load model where the ratio of working-to-total capacity on every link is constant. This model requires a balanced routing algorithm that would result in a constant load across the network. This model has been used in [Shah07]. We also use a randomly distributed capacity model based on a uniform distribution of link capacities within a given range, or a normal distribution around a mean value. This model assumes that the routing of individual demands in a fixed capacity network based on various parameters could be approximated by randomly distributed link capacities. This model has also been used in [Shah07].

In the architecture of dynamic wavelength routing networks there are two key protocols: wavelength routing protocols, and wavelength distribution protocols. With the wavelength routing protocols, any change in optical wavelength resource in each fiber link is flooded through the whole network, and triggers each node to update its network resource database accordingly. On the other hand, the wavelength distribution protocol signals the setup and teardown of an end-to-end light path. These two protocols are derived from mature architectures (such as Open Shortest Path First etc.).

2.1.3 Modeling of Network failures

In a real world WDM network, various failures may occur, such as node failure, physical span failure, interface card failure and single wavelength channel failure. For critical equipments like backbone WDM switches/routers, there are usually built-in backup modules as well as backup power supply in order to prevent a whole node failure from happening. An interface card failure is equivalent to single or multiple wavelength channel failures, which is a subset of a physical span failure. Therefore we shall focus on the protection against physical span failures, which happen most often in real world networks.

2.2 Network Representation

We represent a network as a connected undirected graph $G = (V, E, C_w, C_s)$, where $V = \{v_1, v_2 .. v_N\}$ represents the set of N nodes in the network, $E = \{e_1, e_2 \ldots e_M\}$ represents the set of M edges, where $e_i = \{v_j, v_k\}$. C_W and C_S denote the working capacity and spare capacity allocations in the network edges, respectively. The parameters C_{wi} and C_{si}, for $i = 1 \ldots M$, represent the working capacity and spare capacity of edge e_i, respectively. Throughout the thesis, we use the terms *'node'* and *'vertex'* interchangeably. We also use the terms *'edge'* and *'link'* interchangeably.

2.2.1 Definition

We shall define below some terminology that we will use in the thesis.

A network with N nodes is said to be of *order* N. The network *size* is M if it has M edges. A *nodal degree* d_i is the number of edges incident to node v_i. The *average nodal degree* (AND) of a network is measured by $2M/N$, and is represented by D. A graph G is said to be *connected* if there is at least one path between any pair of nodes in the graph. If there is no path between a pair of nodes in the graph, then it is *disconnected.* An edge is said to be a *bridge* if removing that edge disconnects the graph. A graph G is *k-connected* if at least k vertices need to be removed to make the graph disconnected. Thus, a connected graph is 1-connected. A graph G is *k-edge-connected* if at least k edges need to be removed from the graph to make it disconnected. The maximum edge connectivity of a graph is the minimum nodal degree in the graph, since removing the edges of that node will disconnect the graph. A connected graph with a bridge is a 1-edge connected graph, while the edge-connectivity of a disconnected graph is 0.

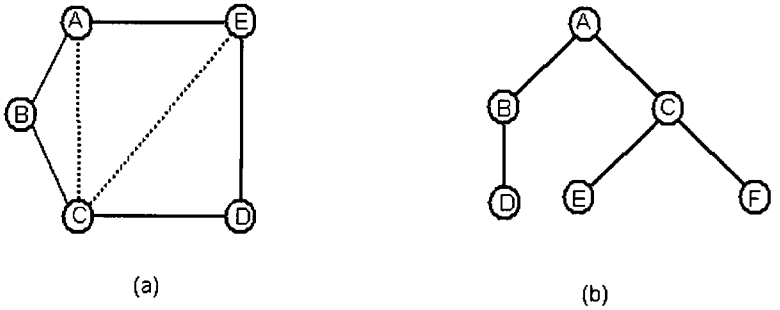


Figure 2.5: (a) p-cycle (b) Tree

P-Cycle

A p-cycle is a ring with one or more *straddling links*. A straddling link is a link that is not part of the cycle itself, but it connects two nodes in the cycle. In Figure 2.5(a), the p-cycle ABCDE straddles two links, AC and CE.

Tree

A tree is a connected network that has no cycles. In a tree, there is exactly one path between any pair of nodes. Every edge in a tree is a *bridge*.

2.3 Network Topology Model

In this section, we first discuss the common network topology models used in current literature. We then propose a different model that we use for our research.

2.3.1 Existing Approaches

A common approach used in current literature is to generate random networks using the Erdos-Renyi model [ErRe59]. In this model, an edge is added between a pair of nodes with probability p. As p is increased, the model is more likely to generate a graph with more edges.

Another common approach used is the Waxman model [Waxm88]. In this model, the locality of the nodes is taken into account when deciding whether an edge should be added to the network. This is very similar to the Erdos-Renyi model, where the probability p is a function of the distance between a pair of nodes. More specifically, an edge between nodes u and v is added with a probability of p_{uv}, where $p_{uv} = \alpha * e^{-\beta Luv / Lmax}$, where L_{uv} denotes the distance between nodes u and v, α and β are constants for a model ($0 \leqslant \alpha, \beta \leqslant 1$) and L_{max} denotes the maximum distance between any pair of nodes in the network. A value of 0 for β reduces the Waxman model into the Erdos-Renyi model. There have been several other models derived off the Waxman model, e.g. [DuGr94], [CaDo97], [ZeCa97] etc.

We used the Waxman model to generate some random networks. However, because this model does not ensure two-connectedness of the generated networks, we found out that generating a lot of networks with a desired number of nodes, edges and two-

connectedness property is a long and time-consuming process. For this reason, we used a different algorithm, proposed in the next sub-section, to generate the random networks used in this thesis. We had compared a number of networks generated by the Waxman model with our algorithm. As shown in Appendix B of this thesis, the results can be different, but the general observations are consistent with those using our algorithm.

2.3.2 Proposed Network Generation Algorithm

While generating random networks for measuring the performance of protection algorithms, we need to ensure that the network is at least 2-edge connected. A disconnected network (0-edge connected) simply cannot be restored, since the network is not fully operational in the first place. A network with a bridge (1-edge connected network) cannot be fully protected either, since a link failure in the bridge will disconnect the network. For this reason, we only use 2-edge connected networks for our simulations.

The network models discussed above do not guarantee 2-edge connectivity of the generated networks. For this reason, the generated networks require additional processing to discard the networks that are not 2-edge connected. To avoid this, we propose a new algorithm for generating random networks. This algorithm guarantees that the generated

Table 2.1: Network Generation Algorithm

1. Initialize the network by assigning an edge between every pair of nodes. This makes the network completely connected.
2. Remove an edge from the network with probability p only if:
   a) The network remains 2-edge connected after removing the edge.
   b) Removing the edge does not reduce the degree of either incident node below the minimum nodal degree.
3. Continue with step 2 until the desired average nodal degree is reached.

network will be 2-edge connected. Also, the algorithm can efficiently handle farther restrictions, for example: constraints on average nodal degree and minimum nodal degree of the network. The algorithm is described in Table 2.1.

The difference between the proposed algorithm and the existing algorithms is, instead of adding edges to an empty graph, we keep removing edges from a complete graph. This allows us to ensure that the generated network fulfills all the desired constraints.

2.4 Network Parameters

In this section, we discuss some network parameters that affect the performance of a protection algorithm.

2.4.1 Redundancy

We define *redundancy* as the ratio between spare capacity and working capacity in the links. The amount of spare capacity determines the amount of traffic that can be restored after a failure. Thus, it can have a huge effect on the restorability of the network. The higher the ratio is, the more likely that the restorability of the network will be higher. Depending on the redundancy, the routing and wavelength assignment (RWA) algorithms can dictate the restorability of the networks. For our simulations, we generate networks with redundancy of 50%, 60%, 70%, 80%, 90% and 100%.

For a network with redundancy X, we can have two different models:

Link-wise redundancy: the ratio of spare to working capacity in each link is approximately X.

Network-wide redundancy: the ratio of total amount of spare capacity across the network and the total amount of working capacity in the network is X. In this scenario, there is no constraint on the working capacity of the individual links so long as the ratio is met network-wide.

For our simulations, we consider both of these scenarios.

We also use a number of real networks for our simulations. The link-redundancies for some of these networks are computed from the traffic matrix, while for other networks it is generated using the Bayesian estimation traffic matrix model. In the traffic matrix model, link redundancy is computed from the result of doing shortest path routing of the traffic from a given traffic matrix to compute the total working capacity and then deducting it from the total link capacity.

2.4.2 Order

The order of the network can have an effect on the performances of the algorithms. For example, a smaller network can use an algorithm with a higher time-complexity but

better restorability. On the other hand, a large network may be required to use a lesser algorithm in terms of restorability, but which has a smaller time-complexity in order to minimize the restoration time after a failure in the network. For our simulations, we generate networks of order 20, 25, 30, 35, 40, 45 and 50.

2.4.3 Density

The *average nodal degree* (AND) of a network determines the connectivity of the network. A higher AND indicates that there will be more alternative paths to choose from. This can have an adverse effect on the time complexity of the algorithms, however a robust algorithm can take advantage of the greater number of available alternate paths and find a protection path that increases the restorability of the network the most. For our simulations, we generate networks with average nodal degree of 3.5 to 6.5, with 0.1 increments.

2.5 Assumptions

Following is a list of the assumptions we make for our simulations, and throughout this thesis:

1) All links are bi-directional, unless stated otherwise. This is because there are algorithms that require a network to be bi-directional, and making this assumption allows us to include such algorithms in the scope of our research.
2) A network is a simple graph, i.e., there is at most one link between a pair of nodes, and there is no link connecting a node to itself. However, some algorithms and results are equally applicable to cases where there are multiple links between the same pair of nodes. These cases will be specifically mentioned where appropriate.
3) Full wavelength conversion is available on each WDM switch/router whenever needed. This allows us the flexibility to consider algorithms that may specifically depend on full optical wavelength conversion for performance issues.
4) Each network is 2-edge connected, as discussed in Section 2.3.2.
5) The protection algorithms are pre-determined, i.e., the backup paths are determined before the failure happens.

Chapter 3

Performance Evaluation Framework

This chapter presents a performance evaluation framework and defines the performance measures used to compare the performances of different algorithms.

Different protection and restoration algorithms use different methods to measure the performance of the algorithm in various networks under different constraints. For our simulations, we use a unified simulation model for all the algorithms we use for our research. This allows us to collect comparable results from the performances of the algorithms. Because different algorithms aim to optimize performance in different criteria, we have designed our model so that it can be used to collect result on any performance criterion desired. This flexibility allows us to compare performances of algorithms using a range of metrics, as will be discussed in later sections.

3.1 Performance Measures

Different algorithms aim to optimize performance of protection/restoration in different criterion. For our research, we measure the performance of the algorithms in the following metrics:

1) The restorability of a network is defined as the ratio between the restorable capacity and the total working capacity of the network. It measures the percentage of the network traffic that can be successfully restored after a failure happens. Restorability is a good indicator of the reliability of the network. In fact, 'Restorability' and 'Reliability' are often used interchangeably [Fris63, XuSu99].

2) The length of the protection path is the number of edges in the protection path from the source node to the destination node. It can have a significant effect on the quality of service of the restored network. So we measure the lengths of the protection paths as a performance metric. However, we calculate the *change* of length in the data-path after restoration, rather than the overall path length, because it is necessary to take into account the path length before the restoration to accurately measure the performance of the algorithm.

3) The sharability of a network is defined as the average number of protection paths sharing an edge. Sharability is used to measure the amount of resource shared by the protection paths. Resource sharing is critical to efficiently protect against network failures. Increased sharability means an edge can be used by many protection paths, which is useful for single failures. However, if an edge is used by too many protection paths, then it can have a negative effect on the performance of the algorithm for multiple failures, especially if one of the failures happens on a link shared by a lot of protection paths.

3.2 Layout of the Framework

To create a generic way of measuring performance metrics of protection algorithms, we design the framework to have three separate modules: a *protection module* to implement a protection algorithm, a *metric module* to compute the performance measure of interest, and the *core module* (or just *core*), which communicates and synchronizes the operations

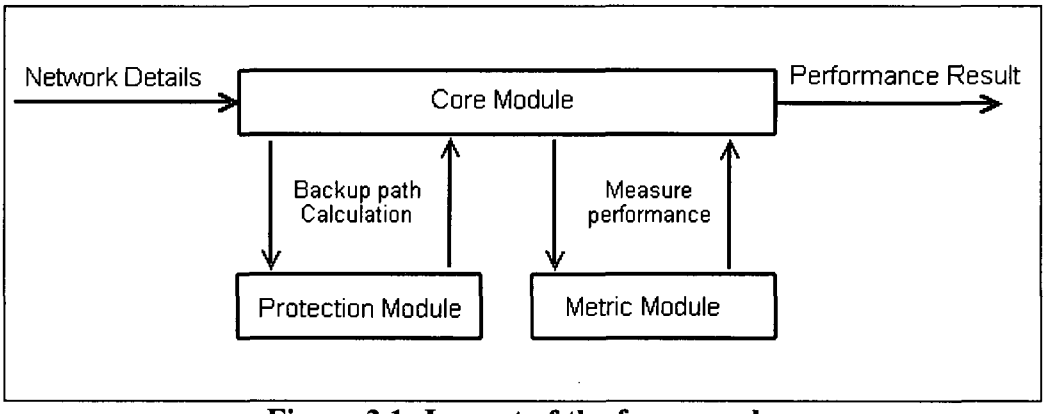


Figure 3.1: Layout of the framework

between the protection module and the metric module. This is illustrated in Figure 1. The *protection module* and the *metric module* are completely independent of each other. This separation between the two types of modules makes it very easy to reuse the same *protection module* with different *metric modules* and compute the performance results, rather than having to re-implement the protection algorithm for each metric. Similarly, we can use the same *metric modules* for another *protection module* (which implements a different protection algorithm). This can vastly simplify the process of measuring performance of a new algorithm, or a new performance metric, and compare them with

the performances of other existing algorithms.

3.3 Protection Module

A *protection module* implements methods that can compute a backup path for any failed link. The framework does not depend on any specific implementation of the *protection module*, i.e., the module can implement either a preconfigured protection cycle (*p-cycle*) algorithm or a tree protection algorithm, or a hybrid algorithm, without requiring extra adjustments in the framework. For example, to compute the performance of the Hierarchical Protection Tree Scheme [ShYa04], the module needs to only implement the protection algorithm itself. The core module communicates with a protection module in the following steps:

1) Initialization

The core module sends the description of the network to the protection module. The description of the network should include the nodes and links of the network. It can also include additional information about the network, for example, the working and spare capacity of each link, the reliability of each link etc. The protection module can initialize its data structure at this stage. It can also verify the prerequisites of the network during initialization. For example, if an algorithm requires the network to be planar (e.g. [MaYa05]), then it can check for planarity of the network. If the prerequisites of the algorithm are not met, then the protection module returns *failure* message to the core. Otherwise, after the initialization is complete, and the prerequisites are confirmed, a protection module implementing a preconfigured protection algorithm uses the protection algorithm to compute information required to protect the network. For example, a p-cycle algorithm should construct the p-cycles, or a tree algorithm should construct the protection tree(s) at this stage. Once completed, the module sends *success* message to the core.

2) Backup Path

The core can send a request to the protection module for a backup path for a failed link. The core sends information about the failed link to the protection module. If the module implements a pre-configured protection algorithm, then the module looks up the backup path for the failed link from the data structure constructed during the initialization phase

of the module. On the other hand, if the algorithm implements a dynamic protection algorithm, then the module should compute the backup path for the failed link at this stage. When the backup path is available, the protection module sends the backup path to the core. If a backup path cannot be constructed, then the module sends a *failure* message to the core.

3) Reconfigure

This step is only useful if the framework is used to measure performance after multiple failures, and allows the algorithm to reconfigure the network after the first failure and before the second one. Some algorithms are able to reconfigure the network to protect against a second failure. For example, a p-cycle algorithm can construct some new p-cycles, or a tree-algorithm can modify the protection tree, to compensate for the first failure. However, for algorithms that are unable to do so, this step is equivalent to the *Initialization* step, where the algorithm will recreate all the data structure it uses for protection. For example, a p-cycle algorithm will recreate the all the p-cycles required to protect the network, or a tree algorithm will recreate the protection tree(s).

3.4 Metric Module

A *metric module* implements methods required to compute the value of a metric on a primary path and on a backup path. The implementation also provides a method to process the values of the measurement. For example, to calculate restorability, a *restorability module* will provide methods to compute the working capacity of a given link, compute the spare capacity of a backup path, and compute the restorability of the network from the working and protected capacities. Similarly, to compute the average change in path-length, the *path-length module* will implement methods to calculate the path-lengths of primary and backup paths, and a method to calculate the average change in path-length from the available data. The core module communicates with a metric module in the following steps:

1) Initialization

The core sends the description of the network to the metric module, same as the protection module. The metric module should initialize its internal data structure during initialization. If there is an error, the module should send a *failure* message to the core.

Otherwise, the module responds with a *success* message to the core.

2) Primary Measurement

The core can send a primary data-path to the metric module and request for the value of the measurement for that path. For example, to calculate *restorability*, the core can request for the working capacity of a path to the module. The module should respond with the requested value to the framework.

3) Backup Measurement

The core can send a primary data-path and the backup data-path to a metric module and request for the value of the measurement for the backup path. It is important to send information about the primary data-path at this step, because the measurement of the backup path can depend on the primary path. For example, for measuring *restorability*, the spare capacity of the backup path is the minimum value between the working capacity of the primary path and the spare capacity of the backup path. When the value of the measurement is available, the module sends the value to the core.

4) Process Measurement

During this communication stage, the core sends measurement values X and Y to the module, where X is the measurement value for the primary path and Y is the value for its backup path. For example, for the *restorability module*, the values imply that for some data-path with working capacity X, the backup path has a spare capacity of Y. The metric module should process the values appropriately. The processing of the data solely depends on the module itself. It should be noted that the values X and Y were retrieved by the core during the *Primary Measurement* and *Backup Measurement* stages.

5) Finalization

At this stage, the core notifies the metric module that the processing of the network is complete. The metric module then computes the measurement value for the entire network, using the accumulated data it received during the *Process Measurement* stage, and sends the result to the core. For the *restorability module*, this return value is the restorability of the network.

3.4.1 Restorability

As discussed in earlier sections, restorability is the percentage of traffic successfully

restored by the protection algorithm after a network failure, and it is measured by the ratio between the total spare capacity to the total working capacity of the network. Therefore, to measure restorability, we need to cumulate the total capacity that can be restored after a failure, and divide that with the cumulated working capacity of the network. So, for a primary path, we need to measure the working capacity of the path. And for a backup path, we need to measure the spare capacity of the path, since the backup path can restore capacity it can provide by utilizing its spare capacity.

1) RestorabilityModule.Initialization

For initialization, we set the initial total working and spare capacities to zero.

Table 3.1: Initialization

```
CumulatedWorkingCapacity = 0
CumulatedProtectedCapacity = 0
```

2) RestorabilityModule.PrimaryMeasurement (Path)

The primary measurement value for a path is its working capacity. The working capacity of a path is the amount of data that can be sent from the source to the destination along the path. In our implementation, the working capacity of a path is the minimum working capacity of all the edges on the path.

Table 3.2: Primary Measurement

```
minCapacity = Infinity
For each link edge in Path:
     edgeCapacity = edge.workingCapacity
     if minCapacity > edgeCapacity:
          minCapacity = edgeCapacity
Return minCapacity
```

3) RestorabilityModule.BackupMeasurement (Path, edge)

The backup measurement value for a path is the spare capacity along the path. The spare capacity of a path is calculated from the minimum of the spare capacities among all the edges in the path. Also, we assert that a backup path cannot protect more than the working capacity of the failed link.

Table 3.3: Backup Measurement

```
minSpare = Infinity
For each link e in Path:
     edgeSpare = e.spareCapacity
     if minSpare > edgeSpare:
          minSpare = edgeSpare
Return MIN(minSpare, edge.workingCapacity)
```

4) RestorabilityModule.ProcessMeasurement(X, Y)

At each step, the module cumulates the working and spare capacities. These cumulated values are later used to calculate the restorability.

Table 3.4: Process Measurement

```
CumulatedWorkingCapacity = CumulatedWorkingCapacity + X
CumulatedProtectedCapacity = CumulatedProtectedCapacity + Y
```

5) RestorabilityModule.Finalize

Finally, once all the computation is over, the restorability module computes the restorability by dividing the total protected capacity by the total working capacity.

Table 3.5: Finalize

```
Restorability = CumulatedProtectedCapacity/ CumulatedWorkingCapacity
Return Restorability
```

3.4.2 Change in Path Length

Change in data-path length is measured as the average difference between the lengths of the primary data-path and the backup path. We measure the amount of increase in path length due to the network failure and subsequent restoration. So, the change in path length for one data-path is computed by subtracting the path-length of the primary path from the length of the backup path. For link restoration, the length of the primary data-path is always 1, because all the data of only the failed link is rerouted over the backup path. This change in data-path is cumulated for each link failure, and then divided by the number of failures to measure the average change in data-path length. The larger the change in path length is, the more adverse effect the algorithm will have on the performance of the algorithm because of the additional transmission time caused by the

increase. The algorithm for this module is fairly trivial. So we provide only the pseudocode in Table 3.6.

Table 3.6: Path Module

```
Initialization():
   CumulatedPathChange = 0
   NumberOfFailures = 0

PrimaryMeasurement(Path):
   Return PathLength(Path)

BackupMeasurement(Path, edge):
   Return PathLength(Path)

ProcessMeasurement(X, Y):
   PathLengthChange = Y - X
   CumulatedPathChange = CumulatedPathchange + PathLengthChange
   NumberOfFailures = NumberOfFailures + 1

Finalization():
   AverageChange = CumulatedPathChange / NumberOfFailures
   Return AverageChange
```

3.4.3 Sharability

As discussed in earlier sections, sharability is measured by the average number of protection paths using an edge. Therefore, we need to count the number of protection paths an edge is a part of, and divide the cumulated count of all the edges with the total number of distinct edges used by all protection paths.

In the *Initialization* step, the module initializes the usage-count for each edge and the total number of distinct edges used by protection paths to zero. Because sharability does not depend on the working path, the *PrimaryMeasurement* and *ProcessMeasurement* steps for this module have no operations (NOP). In the *BackupMeasurement* step, the module increase the usage-count for each edge on the backup path by one, and for each new edge being used, increases the unique edges used by protection paths by one. In the Finalization step, the module computes the sharability and returns the value.

Table 3.7: Sharability Module

```
Initialization():
   For each link e in the Network:
      e.usageCount = 0
   NumberOfEdges = 0

PrimaryMeasurement(Path):
   Return 0

BackupMeasurement(Path, edge):
   For each edge e in Path:
      If e.usageCount = 0:
           NumerOfEdges = NumberOfEdges + 1
      e.usageCount = e.usageCount + 1
   Return 0

ProcessMeasurement(X, Y):
   Return

Finalization():
   totalUsage = 0
   For each edge e in the Network:
      totalUsage = totalUsage + e.usageCount
   Sharability = totalUsage / NumberOfEdges
   Return Sharability
```

3.5 Core Module

The core module of the framework communicates with the other modules and synchronizes the entire simulation process. The same core module can be used with any combination of protection modules and metric modules.

1) Initialization

The core module is initialized after getting information about a network. The initialization of the framework involves initializing the *protection module* and the *performance module.* This step allows the modules to perform any necessary initialization required for the protection of the network. The *protection module* should check for its prerequisites for the network in this step, and terminate the process if the requirements are not met. For example, if the protection algorithm requires the network

to have k-connectivity, and the provided network is not k-connected, then the *protection algorithm* should request the core module to terminate the process.

In the pseudo-codes in the following tables, PM represents the *protection module* and MM represents the *metric module.*

Table 3.8: Initialization

```
1. Read the Network structure.
2. Initialize the Protection Module: PM.Initialization().
3. Initialize the Metric Module: MM.Initialization().
4. If either of the modules in steps 2 and 3 fails, then abort.
```

2) Measure Performance

Once initialized, the framework is ready to process a network and compute the performance of the algorithm. To measure the performance of a metric, the core causes each edge in the network to fail one at a time. The exact process of failing an edge depends on the implementation of the core. For example, the implementation can remove the edge from the network, or assign the working and spare capacities of the link to zero. After failing each edge, the core uses the method provided by the *protection module* to compute the backup path for the failed link. The core then uses the methods provided by the *metric module* to compute the value of the metric of the failed link and the value of the metric of the backup path. It then sends the values back to the *metric module.* Once all the data is collected, the core uses the method provided by the *metric module* to calculate the value of the metric (e.g., restorability) of the entire network.

It should be noted that the framework does not operate on the measurement values (**X** and **Y**). This allows the metric modules to use any data structure necessary for its own implementation. This is a very important feature for the flexibility of the framework.

Multiple Failures

The framework can be used to measure performance of an algorithm after multiple failures with only minor adjustments to the core module. No changes are required to the protection or metric modules. For example, to measure a performance after two

Table 3.9: Measure Performance

```
1. For each link e:
     a. Fail the link e
     b. P = PM.BackupPath(e)
     c. X = MM.PrimaryMeasurement(e)
     d. Y = MM.BackupMeasurement(P, e)
     e. MM.ProcessMeasurement(X, Y)
     f. Restore link e
2. Result = MM.Finalize()
```

simultaneous network failures (i.e., the second network failure happens before the network is reconfigured after the first failure), the core module measures the performances after failing every pair of links,

Table 3.10: Multiple Failures without reconfiguration

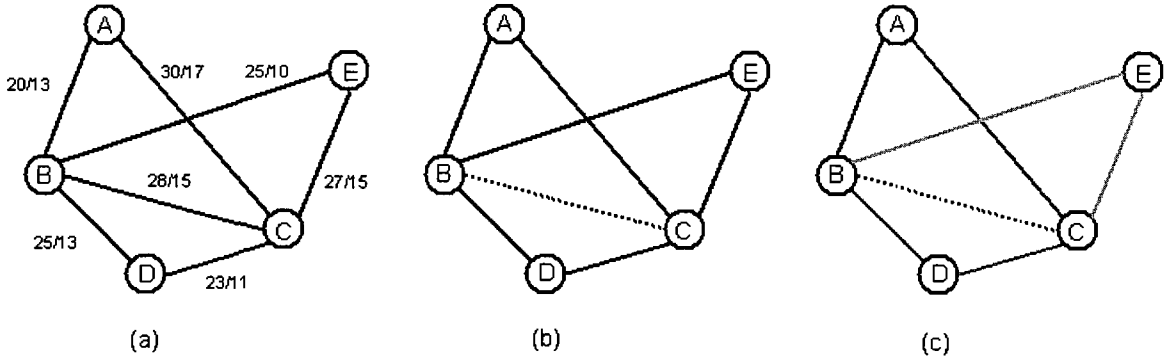
```
1. For each link f:
     a. Fail the link f
     b. For each link e that is not f:
          i.  P = PM.BackupPath(e)
         ii.  X = MM.PrimaryMeasurement(e)
        iii.  Y = MM.BackupMeasurement(P, e)
         iv.  MM.ProcessMeasurement(X, Y)
          v.  Restore link e
     c.  Restore link f
2. Result = MM.Finalize()
```

If we want to measure the performance after multiple failures where the second failure happens after the network is reconfigured after the first, then we re-initialize the protection module after the first failure before we cause the second failure to happen. This only adds an additional step where the core module requests the protection module to reconfigure the network.

Although the framework can be used to measure performance of any metric after multiple failures, we only measure the restorability, and use it for both the reconfigured and the non-reconfigured network.

Table 3.11: Multiple Failures with reconfiguration

```
1. For each link f:
      a. Fail the link f
      b. Reconfigure the network: PM.Reconfigure()
      c. For each link e that is not f:
            i.   P = PM.BackupPath(e)
           ii.   X = MM.PrimaryMeasurement(e)
          iii.   Y = MM.BackupMeasurement(P, e)
           iv.   MM.ProcessMeasurement(X, Y)
            v.   Restore link e
      d. Restore link f
2. Result = MM.Finalize()
```



(the edges in (a) are labeled with '*working capacity/spare capacity*')

Figure 3.2: Illustration of the framework

3.6 Example

To illustrate how the framework works, we consider the simple network in Figure 3.2 (a). To measure the performance of an algorithm for some metric, the framework causes each link in the network to fail. After the link failure, the framework requests the protection module to provide the backup path for the failed link. For example, if the link BC fails (Figure 3.2 (b)), the framework requests the protection module for the backup path for the failed link BC. The protection module will select a backup path from the available paths BAC, BDC and BEC. Different protection module can select different backup paths for the same failed link, depending on how the algorithm selects the backup path. The performance of the algorithm is affected by the selection of the backup path. For

example, when measuring restorability, the framework will request the metric module for restorability to measure the amount of data that can be rerouted after the failure happens. From Figure 3.2 (a), we note that the working capacity on the link BC is 28 units. The restorability module calculates the spare capacities of the BAC, BDC and BEC paths, which are 13, 11 and 10 respectively. Therefore, an algorithm that selects BAC as the backup path will have a better performance measure than an algorithm that selects BEC as the backup path, since BAC can restore a greater amount of data. The framework performs the same steps for each of the other links AB, AC, BD, DC, BE and EC, and measures the total restorability of the network.

The framework uses the same procedure to measure other performance metrics by using the appropriate metric module.

3.7 Concluding Remarks

We have proposed a framework to provide an easy way of collecting performance measurements of algorithms that can be compared to the measurements of other algorithms. The methodology has been detailed. We have provided examples for various modules to demonstrate the functionality. As mentioned in earlier sections, such framework simplifies the comparison of different combinations of protection algorithms and their measures.

We have also implemented several modules for measuring several performance metrics, including restorability, change in path length and sharability. Other performance criteria used in current literature include reliability [ChJa05], modularity [Grov03] etc, which we have not implemented. However, the performance metrics we use are the most widely used measures, and represent the overall usefulness of the algorithms.

Chapter 4

Performance of Cycle-Based Algorithms

In this chapter, we discuss the cycle based algorithm we use for our simulation and present the simulation results. We also discuss the effect of various network parameters on the performance of the algorithms. Among the cycle based protection techniques, p-cycle based techniques are generally regarded as a better system than the others. A comparative study of p-cycle and ring technologies is presented in [Grov03]. We first study the CIDA algorithm and compare it with other candidates later on.

4.1 The Capacitated Iterative Design Algorithm

As discussed before, the basic concept behind p-cycle protection is that it protects the on-cycle links by redirecting the traffic along the path of the cycle that did not fail, and it protects the straddling links by routing the data along either or both of the paths along the cycle. The main difference among different p-cycle algorithms is the way the cycles are created. We have chosen the Capacitated Iterative Design Algorithm (CIDA) [DoHe03] because it provides an easy way of generating a set of cycles based on an existing cycle set by performing any number of operations on the cycles. Because of the simplicity and efficiency, we use the CIDA algorithm to create a set of good p-cycles by performing the *Grow* operation on the initial set of cycles generated by SLA (Straddling Link Algorithm) [ZhYa02b].

The SLA algorithm is used to produce a suitable initial set of primary cycles. The cycles generated are generally inefficient for an overall p-cycle network design because each of these cycles have only one straddling link. A more efficient set of cycles is generated from this set of basic cycles by using the *Grow* operation. The *Grow* operation takes an existing primary cycle, and creates a new cycle from it by replacing a link on the cycle with the shortest path between the nodes adjacent to the link such that the path is node-disjoint to the cycle, if such a path exists. The link removed becomes a straddling link of the new cycle. It is ensured that the new path added to the cycle is node-disjoint to the original primary cycle, and to all the other paths added to primary cycle. Every time a

new p-cycle is created by replacing an on-cycle link with a path, thus turning the link into a straddling link, the computation starts anew for the entire set. In this way, the *Grow* operation finds the shortest cycle for each link such that the link is an on-cycle link, and adds the cycle to the set of cycles. So the set of cycles generated after the *Grow* operation include both very small cycles and very large cycles (the maximally expanded cycles). From this set of candidate cycles, the efficiency of each cycle is computed using the following equation:

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

Here, S is the set of links in the network, c_i is the cost of a unit of capacity on link i, w_i is the amount of unprotected working capacity on link i, and $X_{p,i}$ is the number of protection relationships available to span i from cycle p, i.e., $X_{p,i}$ is 1 if i is an on-cycle link, or 2 if i is a straddling link. The cycle with the highest efficiency is added to the design. The working capacity values of the network are updated, and the efficiency of each remaining candidate p-cycles are computed again. This process continues until there remains no unprotected working capacity on any link. The algorithm is described in pseudocode in Table 4.1.

The CIDA algorithm first creates M candidate p-cycles using the SLA algorithm. It then creates new candidate p-cycles using the *Grow* operation. In the worst case, the total number of candidate p-cycles is M^2N. During the creation of every p-cycle, it uses Dijkstra's shortest path algorithm at least once. Thus, the algorithmic complexity of the CIDA algorithm is $O(M^2N \,.\, M \log N)$, i.e. $O(M^3N \log N)$.

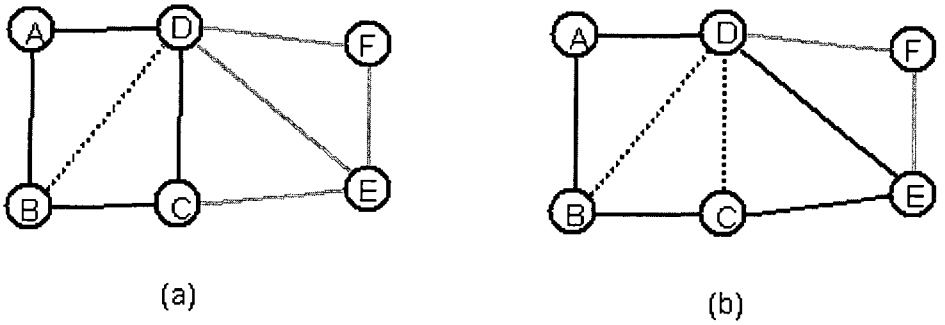


Figure 4.1: Grow Operation

Table 4.1: CIDA Algorithm [DoHe03]

1. Initialize *CandidateCycles*, the set of candidate p-cycles
2. For each link e = <u, v> in the network:
   i. Use Dijkstra's shortest path algorithm to find the shortest path P_1 between u and v that does not contain e.
   ii. If P_1 is computed, then use Dijkstra's shortest path algorithm again to find the shortest path P_2 between u and v that does not contain e, and is node and link disjoint to P_1.
   iii. If both P_1 and P_2 are computed, then create a cycle with all the links in both P_1 and P_2. Add the cycle to *CandidateCycles*
3. Initialize *SPAddCycles*, another set of p-cycles
4. For each cycle C in *CandidateCycles*:
   i. For each link e = <u, v> in C:
      a) Use Dijkstra's algorithm to find the shortest path P between u and v so that P is node and link disjoint to the cycle.
      b) If P is computed, then create a new cycle by removing link e from C, and adding path P to C. Add this new cycle to *SPAddCycles*.
5. For each cycle C in *SPAddCycles*:
   i. For each link e = <u, v> in C:
      a) Use Dijkstra's algorithm to find the shortest path P between u and v where P is node and link disjoint to C.
      b) If P can be computed, then create a new cycle by removing link e from C and adding path P to C. Add this new cycle to *CandidateCycles*.
      c) Start again from step a) with the first link e in C.
6. Add all cycles in *SPAddCycles* in *CandidateCycles*.
7. For each link e = <u, v> in the network:
   i. Use Dijkstra's algorithm to find the shortest path P between u and v.
   ii. If P can be computed, then create a cycle by adding link e to path P. Add this cycle to *CandidateCycles*.
8. Initialize *CycleSet*, the set of p-cycles used for protection.
9. While there is unprotected working capacity:
   i. Compute efficiency for each p-cycle
   ii. Find cycle C with the highest efficiency value
   iii. Add C to *CycleSet* and remove C from CandidateCycles
   iv. Update the working capacity values of the network

Example

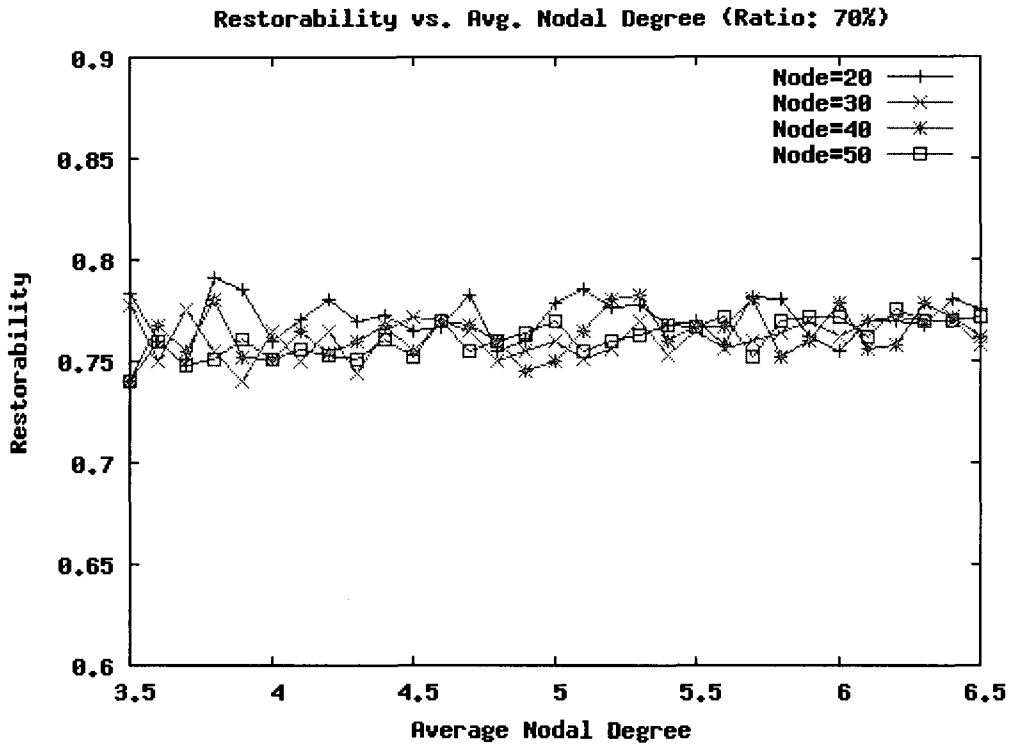
The creation of the candidate p-cycles is illustrated in Figure 4.1. For link BD, the link disjoint shortest paths BAD and BCD between B and D are computed using Dijkstra's shortest path algorithm. These two paths are combined to form a p-cycle ABCD which straddles the link BD. Links CE, DE, DF and EF are off-cycle links. Performing the *Grow* operation on the link CD means to remove the link from the cycle, and add CED, the shortest path connecting C and D which is node-disjoint to the original cycle ABCD. The resultant p-cycle ABCED straddles both links BD and CD.

4.2 Restorability

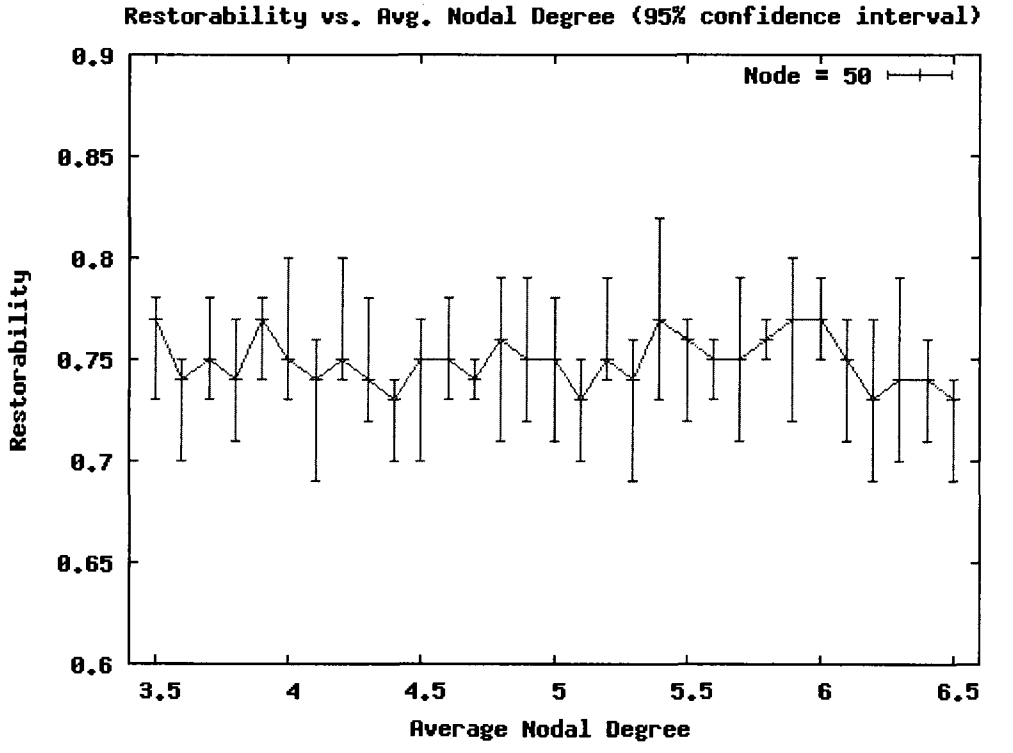
The restorability of a network measures the percentage of the network traffic that can be successfully restored after a failure happens. It is defined as the ratio between the restorable capacity and the total working capacity of the network. In this section, we measure the restorability of a network after a single failure in varying network size, network density and spare capacity when a p-cycle algorithm is used for protection. We analyze the effect of these network parameters in the following subsections.

4.2.1 Effect of Network Order and Density

In Figure 4.2, we can see the effect of changes in network size and density on the restorability of the network provided by a p-cycle algorithm. For clarity, we separate the curves into two graphs. Although the performance seems to vary unpredictably, it remains within a very small range, and hence can be considered stable. For example, a network with 40 nodes has restorability hovering within the interval (0.74, 0.79), which is about 3% of the average. As the graph suggests, neither the network size nor the network density has any clear effect on the restorability of the network. For example, for a network of average nodal degree 4.0, the restorability for a network with 20 nodes is 76%, while the restorability for a network with 50 nodes and the same average nodal degree is 75%. On the other hand, to see the effect of varying nodal density, if we consider a network with 40 nodes, then the restorability is 74% when average nodal degree is 3.5, and the restorability changes to 76%. We also observe in Figure 4.2(b) that



(a)



(b)

Figure 4.2: Effect of Network size and density on Restorability for p-cycle algorithm

the 95% confidence intervals for the simulation results are small, which indicates that the simulation result from the simulation is consistent and reliable. From this, we can conclude that the restorability provided by the p-cycle algorithm is not affected greatly by neither the network order nor the network density. We get similar results for the random networks for both the link-wise and network-wide redundancy models (defined in Section 2.4.1). This is also true for the other performance measures in this chapter, and for the tree-based algorithms discussed in Chapter 5. For this reason, we do not distinguish between the two models from hereon.

4.2.2 Effect of Spare Capacity

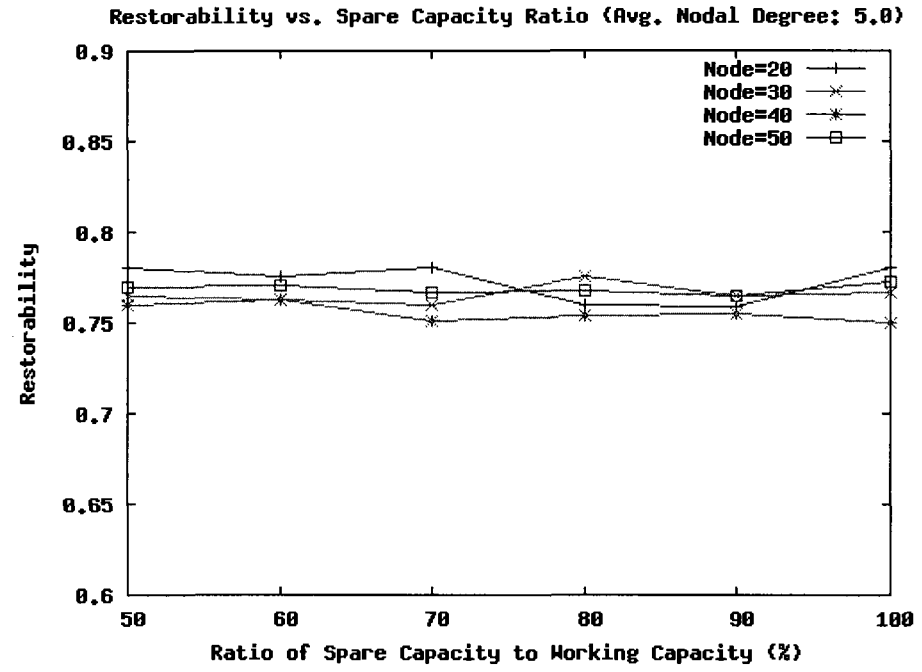
Figure 4.3 shows the effect of spare capacity allocation on restorability for a p-cycle algorithm. As with network size and density, we notice that the restorability provided by a p-cycle algorithm remains within a very small range even if the ratio of spare capacity to working capacity of the network is varied greatly. For example, for a network of 40 nodes, the restorability changes from 79% to 80% as the ratio of spare capacity to working capacity is changed from 50% to 100%. From Figures 4.2 and 4.3, we can conclude that, for p-cycle algorithms, none of network size, density or spare capacity allocation has significant effect on the restorability.

4.3 Change in Path Length

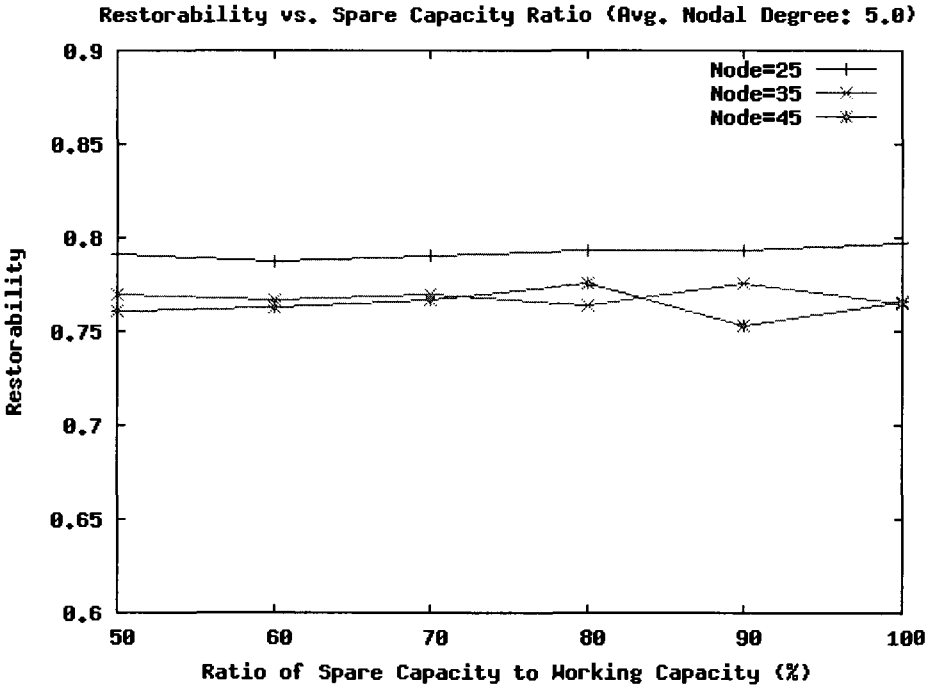
The average change in the backup path length for a network can help to determine the change in the quality of service in a network restored by a protection algorithm when a failure occurs. After a network failure, if the restored backup paths are significantly longer than the original data-paths, then it will have an adverse effect on the end-to-end delay of the network. The smaller the increase in the data-path length after the network is restored, the better the algorithm will perform since it will reduce the end-to-end delay in the data transfer. Thus this measurement helps us ascertain the quality of a service of a restored network.

We measure the change in path-length of the restored network after a single link failure. We have considered networks of varying size and density. We do not, however, measure the performance for networks with varying redundancy, because the protection

algorithms we implement do not take into account the redundancy of the network when determining the backup path. This, in fact, is the case for most protection algorithms, because protection algorithms are pre-determined, i.e., the backup paths are determined



(a)



(b)

Figure 4.3: Effect of spare capacity allocation on Restorability for p-cycle algorithm

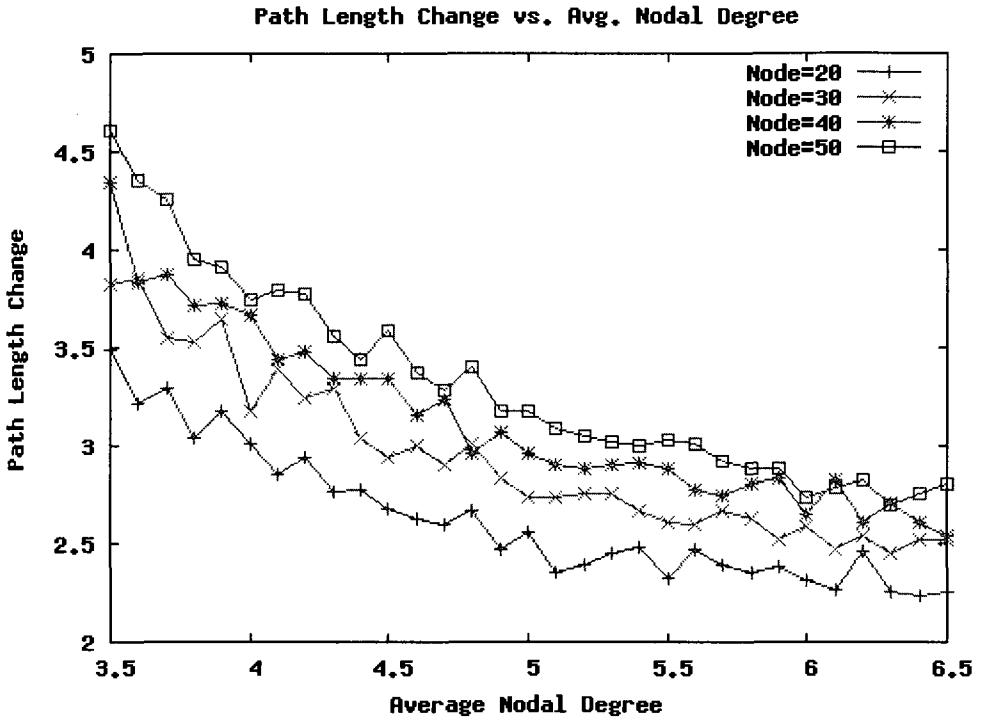
before the network failure happens. Thus the redundancy of the network during the construction of the backup paths and the redundancy of the network after a failure occurs may not be the same. We would like to point out, however, that our simulation setup can comfortably accommodate algorithms that do take the redundancy into account and measure the performance of such algorithms on networks with various redundancies.

As Figure 4.4 indicates, the average change in data-path length in a network protected by a p-cycle algorithm is affected by both the network order and the network density. The change in data-path length increases with the network order, but decreases as the nodal density is increased. For example, for a network of order 20 and AND of 3.5, the measure is 3.5, and for a network of order 50 with the same AND, the measure is 4.6. However, the measure decreases from 3.5 to 2.4 for a network of order 20 if the AND is decreased from 3.5 to 6.5.

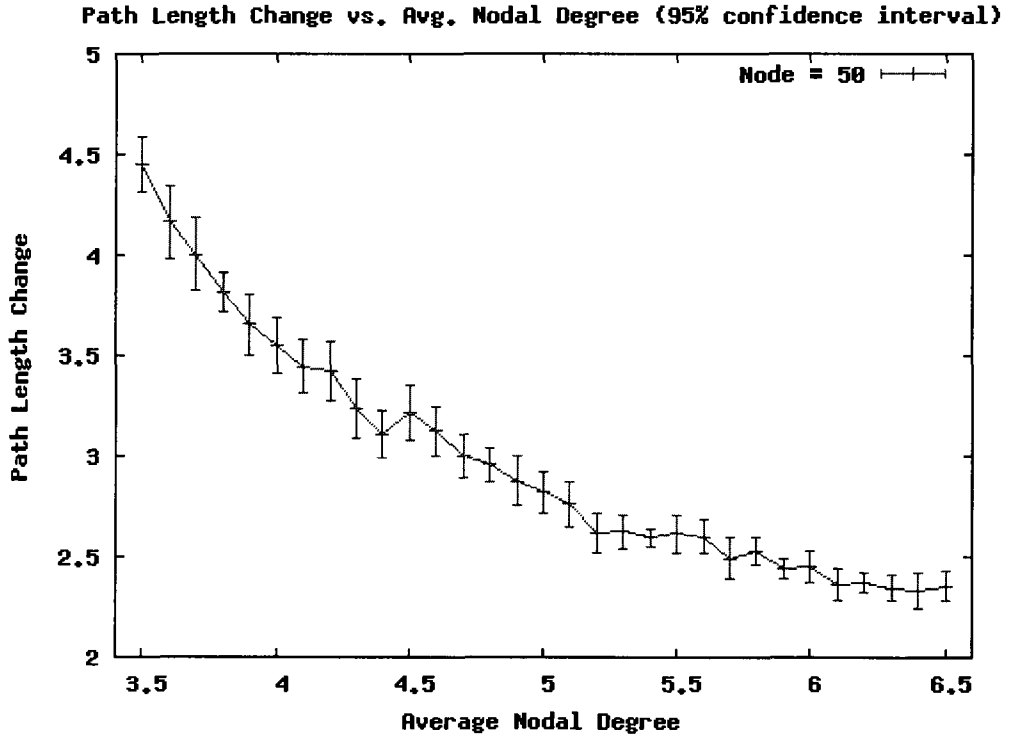
4.4 Sharability

Sharability of a network can be used to measure the resource sharing by the protection algorithm. Sharability is the average number of protection paths using an edge. A higher value for sharability means that a lot of protection paths are utilizing the edges, which means the protection algorithm tries to find the best protection path by reusing the resource. However, if sharability is too high, i.e., if too many protection paths share the same edges, then it can have an adverse effect in the case of a second failure before the network can reconfigure for a new protection topology after the first failure. Because once an edge is used by a protection path, another protection path that uses the same edge cannot be used for the second failure, if necessary. Here we focus on the sharability of a network for a single link failure.

As is evident from Figure 4.5, network order and density affect the sharability of a network when it is protected by a p-cycle algorithm too. However, in this case, the sharability increases with increasing network order, but decreases with increasing AND. This is because as the number of nodes in the network increases, the number of required protection paths also increases, which leads to the same edges being used for more

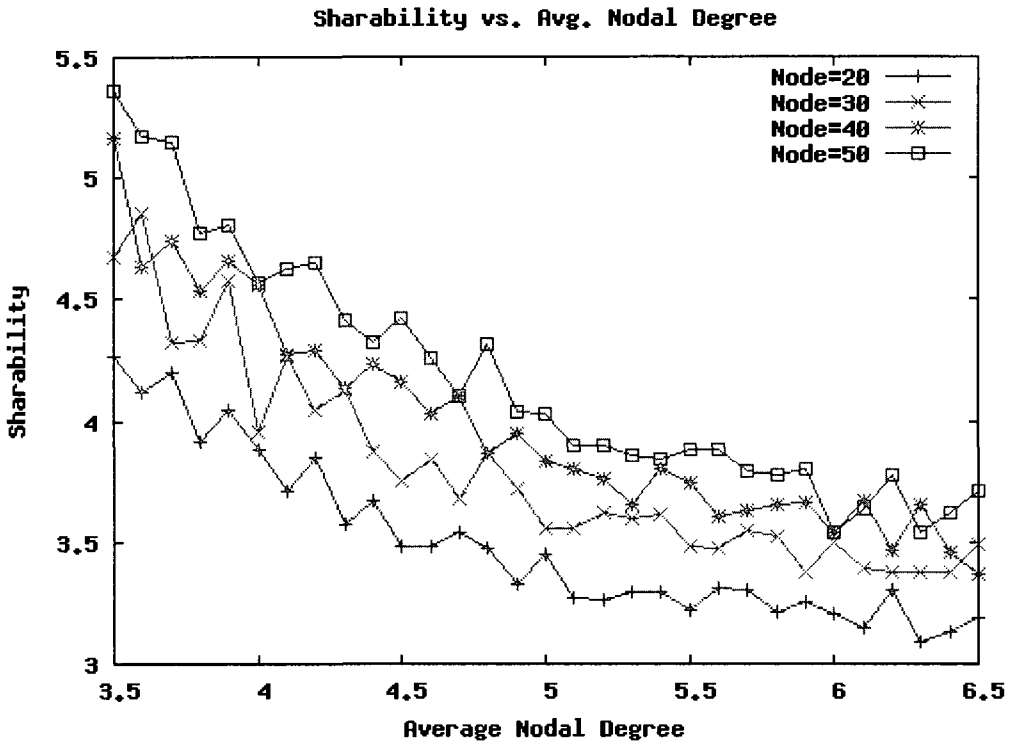


(a)

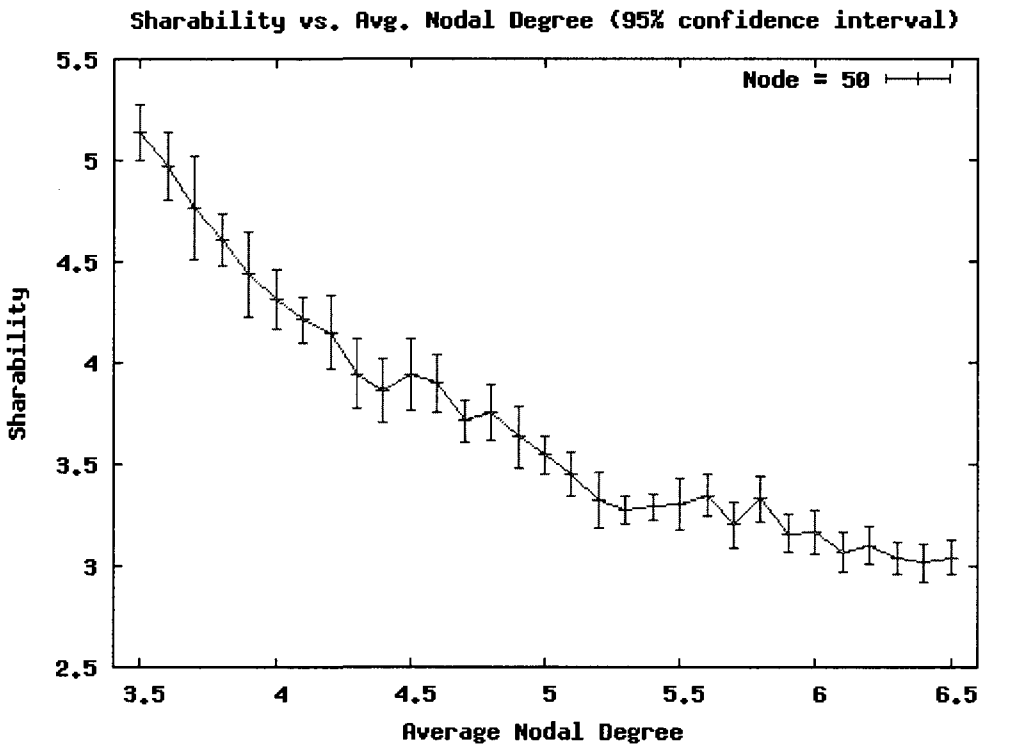


(b)

Figure 4.4: Effect of network order and density on Average Change in path-length for p-cycle algorithm



(a)



(b)

Figure 4.5: Effect of network order and density on Sharability for p-cycle algorithm

protection paths. This causes an increase in sharability. However, as the AND in the network increases, there are more alternative paths available for network protection, which reduces the sharability in the network. For example, for a network of order 20 and AND 3.5, the sharability is 4.3, while sharability is 5.4 for a network of order 50 and the same AND. Also, the sharability decreases from 4.3 down to 3.3 as the AND is increased from 3.5 to 6.5 in a network of order 20. The small values for 95% confidence interval in Figure 4.5(b) indicate that this result is consistent and reliable.

4.5 Comparison with other cycle algorithms

In this section, we compare the performance results of the CIDA algorithm with the results of another p-cycle algorithm, namely the Min-Sum Cycle Cover Detection/Protection Algorithm (MSCC) [DoYa06]. The MSCC algorithm finds a family of cycles in which each node and edge of the graph appears at least in one of these cycles in order to minimize the total cost of the network. The algorithm utilizes algorithms of Eulerian Graph, Minimum Perfect Matching algorithm [PaSt82] and Dijkstra's shortest path algorithm [Dijk59]. The algorithmic complexity of the MSCC algorithm is $max\{O[N^3 (M + N \log N)], O(N^6)]\}$. Following is a performance table comparing the results of the MSCC algorithm with the CIDA algorithm [DoYa06]:

Table 4.2: Comparison between MSCC and CIDA algorithms

Nodes	Edges	Cycles Required		Redundancy	
		CIDA	MSCC	CIDA	MSCC
13	23	1	7	56.5%	108.7%
11	26	1	7	42.3%	111.5%
20	33	1	7	60.6%	118.2%
14	21	1	3	66.7%	123.8%

From the comparison result presented in Table 4.2, we notice that the CIDA algorithm outperforms the MSCC algorithm in both the number of cycles required to protect the network, and the redundancy required in the network for rerouting the affected data after a network failure. For this reason, we select the CIDA algorithm as the representative for cycle-based algorithms when comparing between a p-cycle algorithm and a tree-based algorithm in Chapter 6.

Table 4.3: Commercial Networks

Name	Nodes	Edges	AND	Map Path Length
ATT2	32	66	4.1	11.2
BBN	11	19	3.5	5.6
Brazil RNP	11	19	3.5	5.1
DataXChange	8	24	6	3.4
GridNet	10	25	5	4.2
AARNet	13	14	2.2	7.4
XONet	9	14	3.1	5.4
CanadaNET	24	41	3.4	9.6

4.6 Commercial Networks

We used several commercial networks for our simulation. The networks are of small to medium order, and are of a varying range of density. The parameters of some commercial networks we used are shown in Table 4.3. Detailed topological information about these networks are provided in Appendix A. We present our measurement of the performances

Table 4.4: Performance of p-cycle algorithm for Commercial Networks

Performance Measure	Networks with Redundancy from Traffic Matrix			Networks with Redundancy from Probability Distribution							
	ATT2	BRNP	XO Net	AT T2	BR NP	XO Net	Data Xcha nge	Grid Net	AAR Net	BBN	C NE T
Restorability	0.65	0.42	0.48	0.7	0.48	0.56	0.64	0.62	0.30	0.71	0.64
Change in Path Length	2.81	1.05	3.57	2.81	1.05	3.57	2.75	3.68	1.67	4.26	4.08
Sharability	3.80	1.80	5.56	3.80	1.80	5.56	4.13	5.11	1.0	5.40	4.94

for some of the commercial networks in Table 4.4. The traffic matrices we use have zero diagonal elements (no sending traffic back to a node itself), but with unity traffic in the rest of a matrix (i.e., unity traffic is sent from each node to each other node i8n the network). From the results, we can see that for networks where redundancy was computed using a probability distribution, the restorability in some commercial networks stay within a small range, e.g. in ATT2, BBN and GridNet, with exceptions in some other networks, notably in AARNet, where the density is very low. We note that the change in path length and the sharability also do not follow any specific patterns. Interestingly, although BBN

and Brazil RNP networks have the same network order and AND, the performance of the p-cycle algorithm for these two networks are very different. This indicates that although the network parameters of our interest are the same for both of these networks, there are other network parameters that affect the performance of the algorithm. Additionally, for networks where redundancy is computed from traffic matrix, the performance measures stay within a comparable range of values, and the observations we make for networks with redundancy computed from probability distributions also apply to these networks.

Table 4.5: Effects of Network Parameters on Performances of p-cycle Algorithms

Performance Measure	**Network Order**	**Network Density**	**Redundancy**
Restorability	Minimal	Minimal	Minimal
Change in Path Length	Directly Proportional	Inversely Proportional	N/A
Sharability	Directly Proportional	Inversely Proportional	N/A

4.7 Summary

Table 4.5 summarizes the findings in this chapter. As we can see, neither network order nor network density affects the restorability of the CIDA protection algorithm. However, change in path length and sharability both increase as the network order increases. On the other hand, both change in path length and sharability decrease as network density is increased. Network redundancy does not affect restorability greatly. We do not compute the effect of redundancy on change in path length or sharability, because the redundancy does not affect the protection paths computed, and thus does not affect either of these performance metrics.

Chapter 5

Performance of Tree-Based Algorithms

In this chapter, we discuss the tree based algorithm we use for our simulation and present the simulation results. We shall use the MST2SP, the Red-Blue Tree Algorithm and the Hierarchical Tree algorithm for our investigation. We also discuss the effect of various network parameters on the performance of the algorithms.

5.1 Maximum Spare-capacity Tree and Two Shortest Paths

We shall use the Maximum Spare-capacity Tree and Two Shortest Path algorithm (MST2SP) in our simulation. The MST2SP algorithm creates only one spanning tree, as opposed to many algorithms that use a primary tree and another secondary tree to provide a backup path in such cases [MeFi99, FeCu01]. Instead of creating a redundant tree to provide protection for the on-tree links, this algorithm uses the shortest paths between the nodes adjacent to each link on the protection tree as the backup paths for the on-tree links.

The MST2SP algorithm first reduces the chains in a network into logical links. A chain is a path in the network where all the non-terminating nodes in the path has a degree of two. By doing this, the algorithm reduces the complexity of the later steps of the protection process without affecting the performance of the generated protection tree, since any protection mechanism that can protect a link in the chain can also protect the rest of the links in the chain. The algorithm then constructs a maximum spanning tree, using the spare capacities of the links as the weight of the link. We use the commonly used Minimum Spanning Tree algorithm by Kruskal [Krus56] for this purpose. For each of the links in this MSCT, the algorithm now finds a pair of shortest paths that are link-disjoint with each other. Clearly the paths cannot contain the link (or chain) directly connecting the nodes. However, it should be noted that the paths are not required to be link-disjoint with the protection cycle. These shortest paths can be computed by any of the well-known shortest path algorithms. We use Dijkstra's algorithm [Dijk59] using binary heaps for this purpose [AhMa93]. The steps of the algorithm are described in Table 5.1.

Table 5.1: MST2SP Algorithm

```
1. Reduce every chain into a logical link.
2. Using Kruskal's algorithm, create a maximum spanning tree T.
3. For each link e = <u, v> in the network, but not in T:
      a. Find the path P connecting u and v along the tree T.
      b. Assign P as the backup path for the link e.
4. For each link e = <u, v> in T:
      a. Use Dijkstra's algorithm to find the shortest path P
         between u and v that is link disjoint to T.
      b. Assign P as the backup path for the link e.
```

The creation of the MST requires *O(M log N)* processing time. For each of the link on the spanning tree, i.e., for *(N – 1)* links, Dijkstra's algorithm is used at least once. Thus, the algorithmic complexity of the MST2SP algorithm is *O(M log N)* + *O(N M logN)*, i.e., *O(M N log N)*.

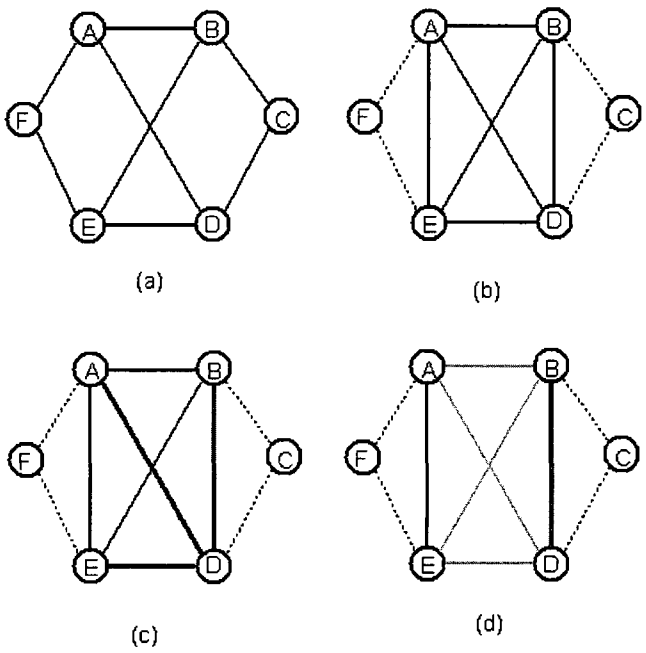
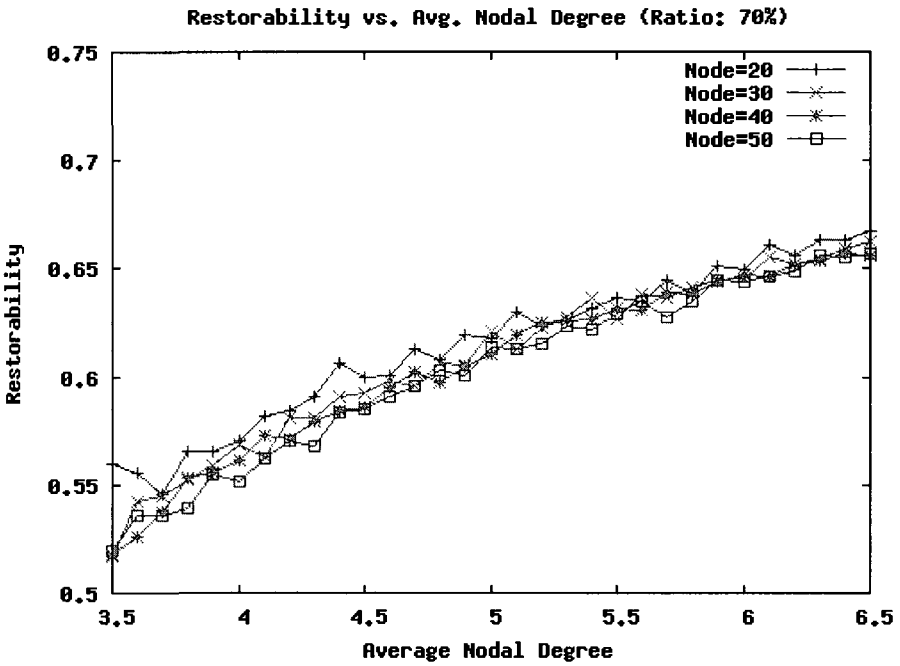


Figure 5.1: Different steps of MST2SP algorithm

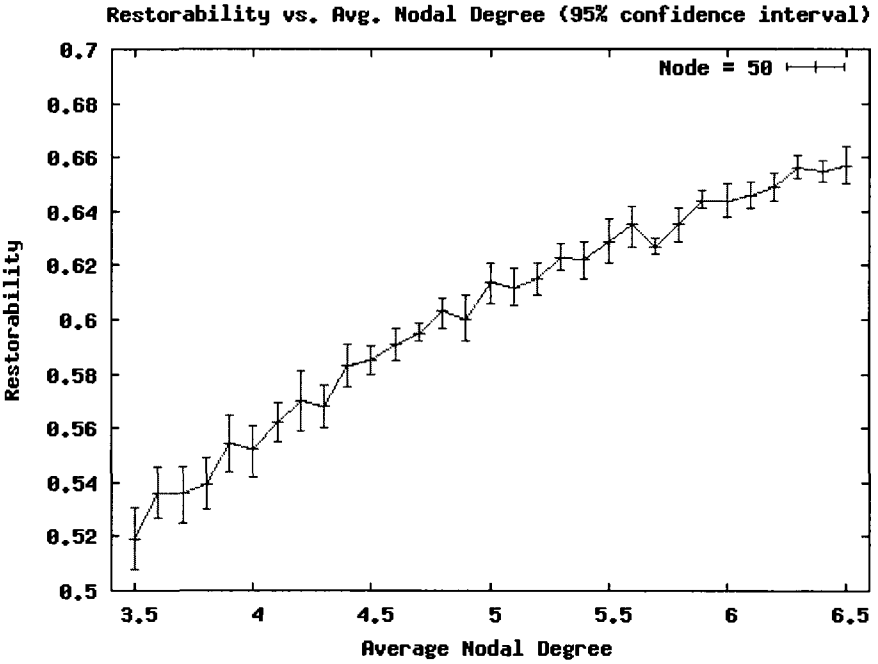
Example

For the network in Figure 5.1(a), the algorithm first reduces the AFE chain into AE link (Figure 5.1(b)). Similarly, the algorithm converts the BCD chain into logical link BD. A maximum spanning tree is then constructed, consisting of links AD, BD and DE (Figure

5.1 (c)). A pair of shortest paths is computed for each of these links in the tree. For example, the shortest paths for the BCD chain are BAD and BED (Figure 5.1 (d)), and both share a link with the protection cycle in Figure 5.1 (c).



(a)



(b)

Figure 5.2: Effect of Network order and density on Restorability for MST2SP

5.2 Performance of MST2SP

In this section, we investigate the performance of the MST2SP algorithm for several performance metrics, such as restorability, change in path length and sharability. We also investigate how various network parameters, such as network order, density and redundancy affect its performance.

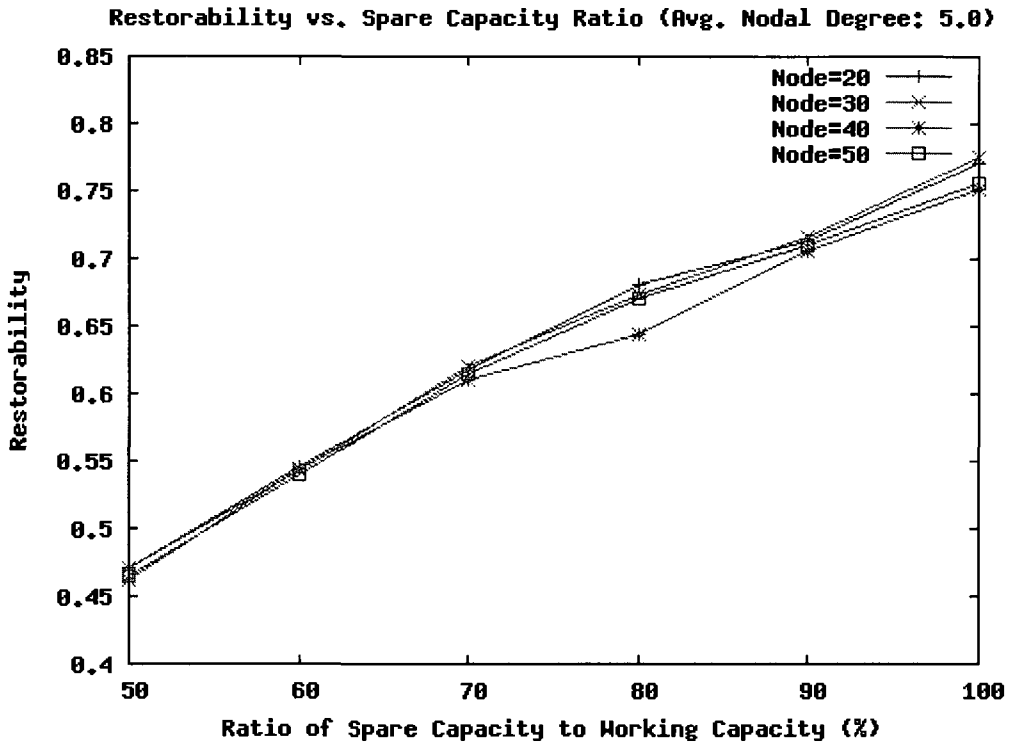
5.2.1 Restorability

We measure the restorability of a network after a single failure in varying network size, network density and spare capacity when a tree-algorithm is used for protection. We analyze the effect of these network parameters in the following subsections.

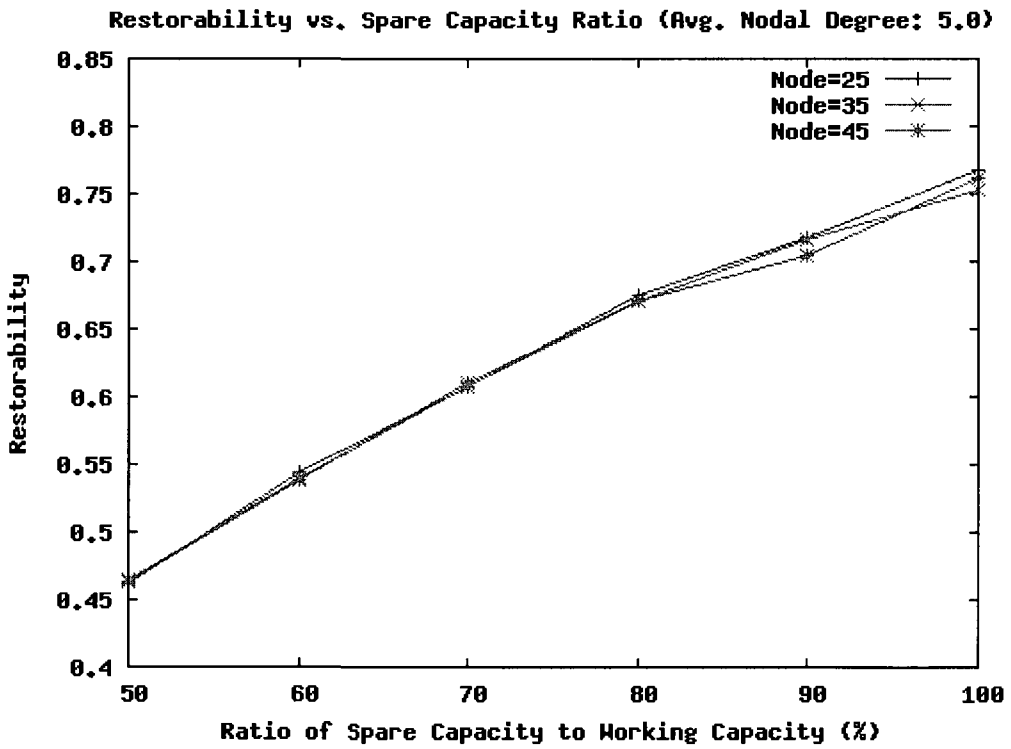
5.2.1.1 Effect of Network Order and Density

Figure 5.2 shows the restorability provided by the MST2SP algorithm in varying network order and AND. The graph indicates that restorability is an increasing function of the AND of the network. We can see that as the AND of the network changes, it has a much greater effect on the restorability of the network. For example, for a network of 30 nodes, the restorability is 57% when average nodal degree in the network is 4. However, for a network of the same size, the restorability increases to 66% as the average nodal degree is increased to 6. This is because as the AND increases, there are more edges to chose from at each step when constructing the spanning tree. The increased number of edges also provides a greater number of possible edges to use when constructing the shortest paths. Both of these can cause the restorability of the network to increase. However, for a fixed network density, the restorability provided by the algorithm varies only very slightly with the network size. For example, for an average nodal degree of 4.5, the restorability of a network of 20 nodes is 59%, while the restorability of a network of 50 nodes is 58%. The small values for 95% confidence intervals in Figure 5.2(b) indicate that these results are reliable and consistent. This clearly demonstrates that, for the MST2SP algorithm:

- The network order has very little effect on the network restorability
- Network restorability is directly proportional to network density, i.e., network restorability increases as the network density is increased; and
- Network density has a greater effect on network restorability than network size.

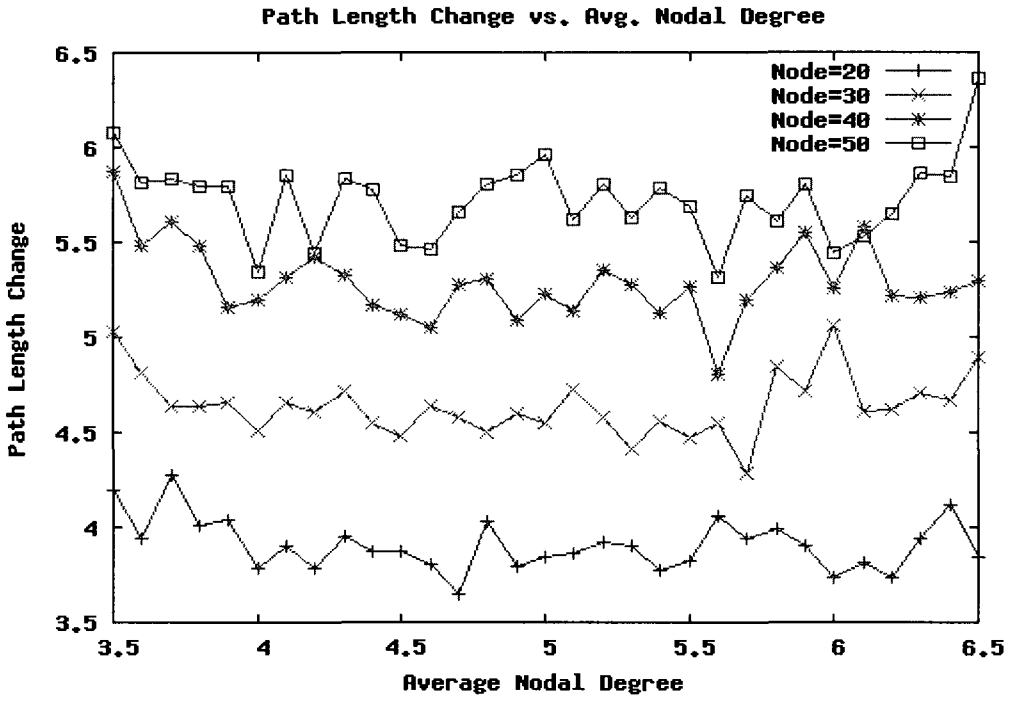


(a)

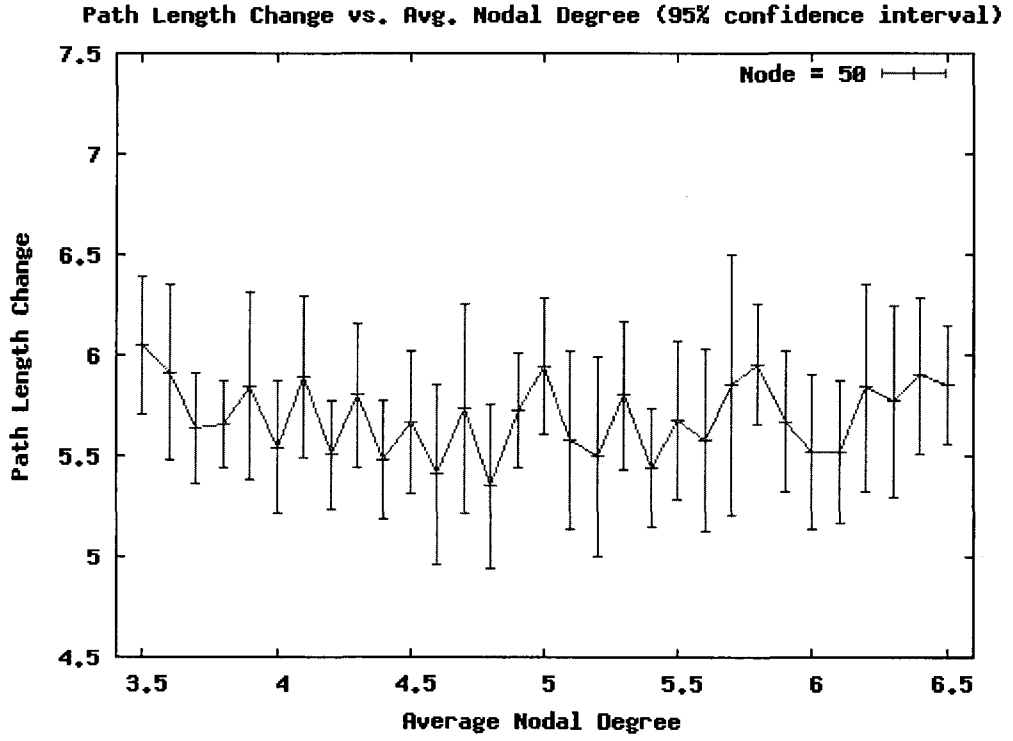


(b)

Figure 5.3: Effect of spare capacity allocation on Restorability for MST2SP



(a)



(b)

Figure 5.4: Effect of network order and density on Average change in path-length for MST2SP algorithm

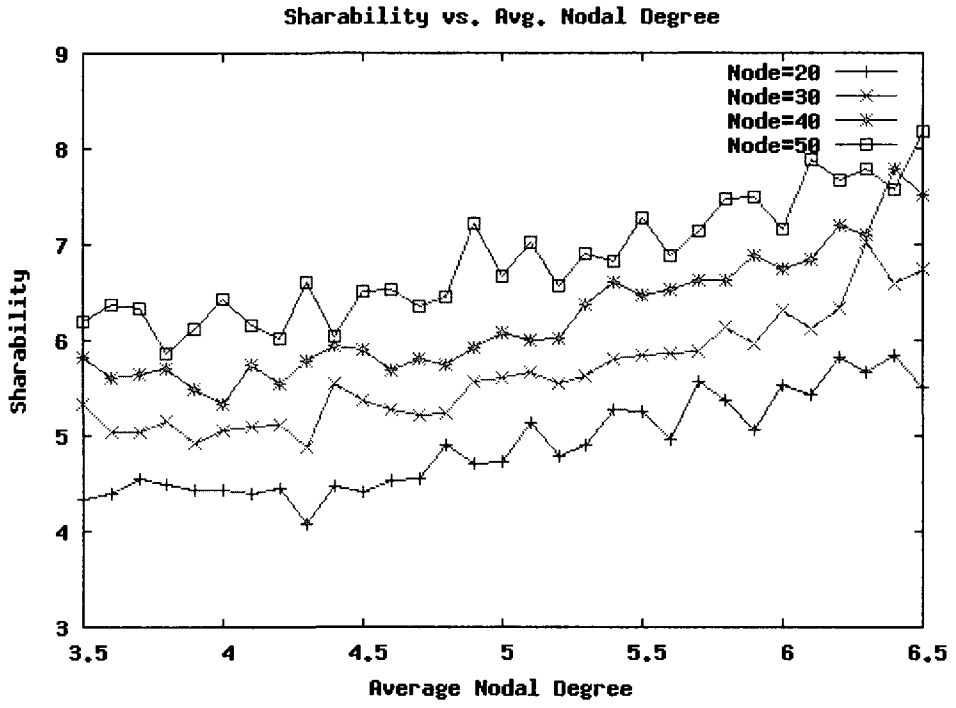
5.2.1.2 Effect of Spare Capacity

Figure 5.3 display the change in network restorability provided by the MST2SP when the spare capacity allocation in the network is changed. For clarity, we separate the overlapping performance curves into two parts: Figure 5.3(a) and Figure 5.3(b). From the graphs, we can see that the restorability of the network changes greatly when there is more redundancy, i.e., when a greater amount of available capacity is reserved for the backup paths. For example, for a network of 20 nodes with an average nodal degree of 5.0, the restorability provided by the MST2SP algorithm is only 46% if the spare capacity in the network is half of the working capacity. However, when there is an equal amount of spare capacity and working capacity in the network, i.e., when the spare capacity to working capacity ratio is 100%, the restorability increases to 75% for the same algorithm. This is because the larger spare capacity allows rerouting a greater amount of data, which means a greater redundancy provides greater restorability.

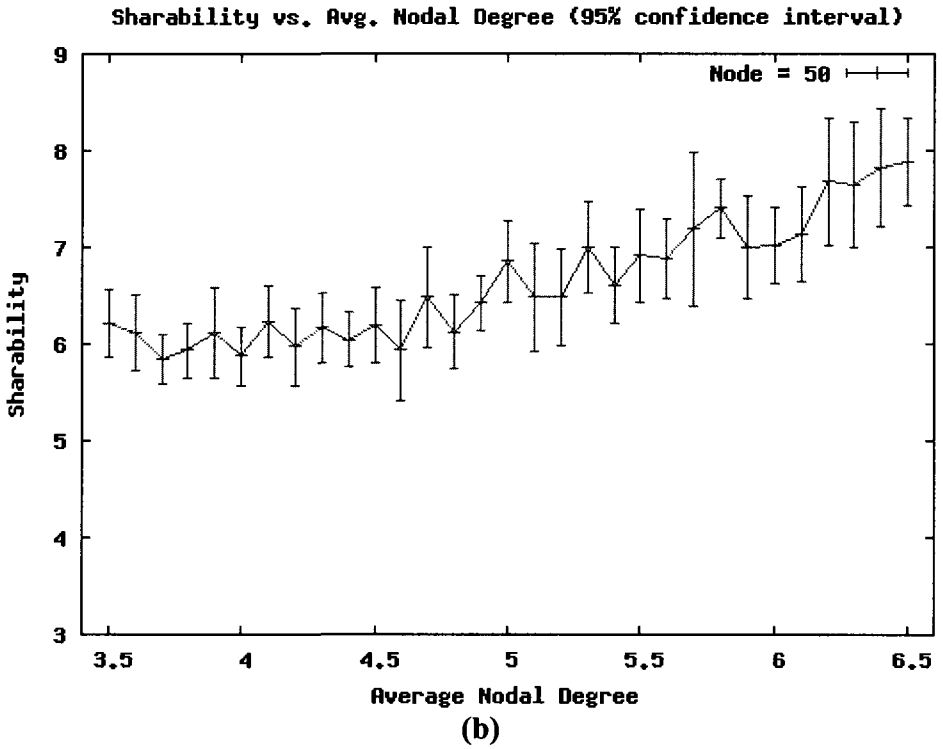
5.2.2 Change in Path Length

We measure the change in path-length of the restored network after a single network failure in networks of varying size and density. As for p-Cycles, we do not measure the performance for networks with varying redundancy, because the protection algorithms we implement do not take into account the redundancy of the network when determining the backup path.

As is evident from Figure 5.4, the average change in path-length for a network protected by the MST2SP algorithm increases as the order of the network increases. For example, for a network with 20 nodes and average nodal degree of 3.5, the average change in data-path length is 4.2. However, for a network of order 50 of the same average nodal degree, the measure is 6.1. On the other hand, for a network of the same size, increasing the average nodal degree has little effect on the change in path length. For example, for a network of 30 nodes, the average change in data-path length in a restored network is 5 when average nodal degree is 3.5. For the same network, the measure is 4.9 when the nodal density is increased to 6.5. We also note that the values for 95% confidence interval in Figure 5.4(b) are small, which indicates that the result we collected are consistent and reliable.



(a)



(b)

Figure 5.5: Effect of network order and density on Sharability for MST2SP

5.2.3 Sharability

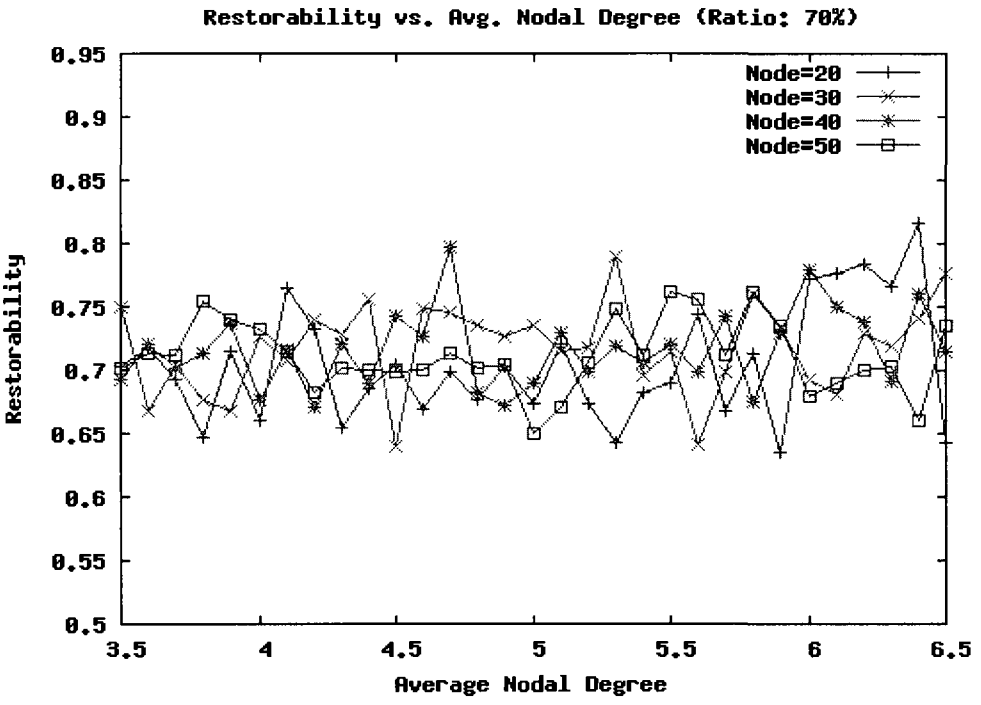
As for p-cycles, we measure the sharability of a network after a single link failure for networks of varying size and density. As Figure 5.5 indicates, both network order and density have an effect on the sharability of a network. As the number of nodes in a network increases, so does the sharability. The sharability also increases with increasing average nodal degree. For example, in a network with 20 nodes and AND of 3.5, the sharability is 4.4, while in a network with 50 nodes with the same AND, the sharability is 6.2. And if the AND is increased from 3.5 to 6.5 for a network of order 20, the sharability increases from 4.4 to 5.6.

5.3 The Red Blue Tree Algorithm

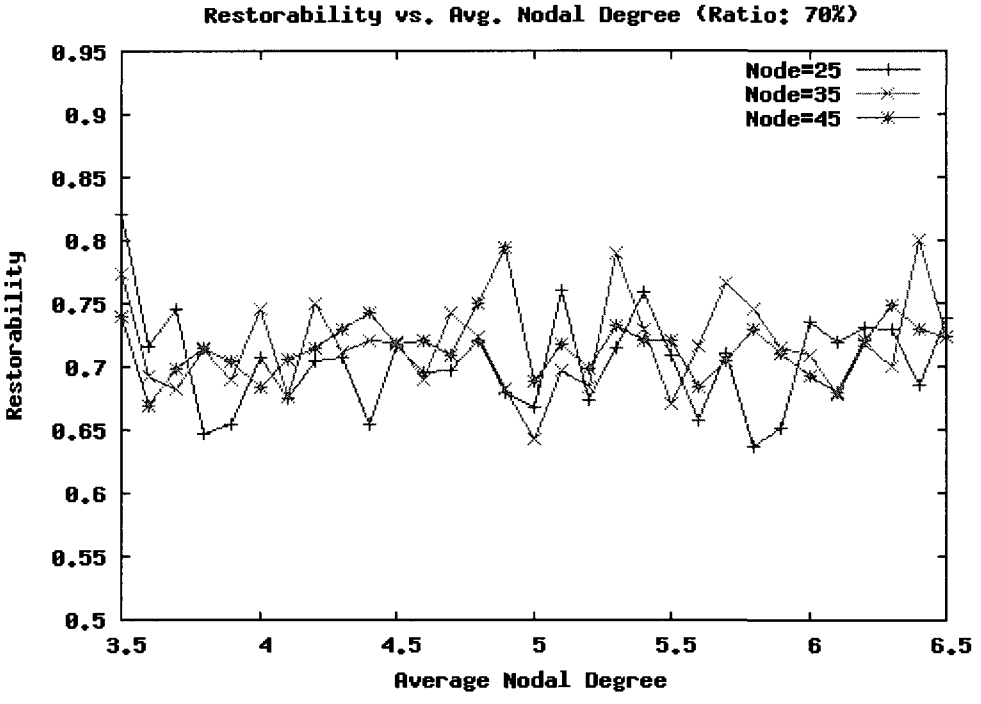
Another tree-based protection algorithm we select for our comparison is the Redundant Tree algorithm presented in [MeFi99]. Due to the way the network works, we name it the Red Blue Tree Algorithm. This algorithm creates two trees for each node in the network: the primary tree, or the Red tree, and the secondary tree, or the Blue tree. The trees are constructed in a way that if any link in the network fails, the source node can still reach all the other nodes in the network through either the Red tree or the Blue tree. The construction of the trees for one source node *s* is described in Table 5.2

Table 5.2: The Red-Blue Algorithm

1. Initialize empty trees TreeRed and TreeBlue.
2. Choose any cycle {s, c_1, …, c_k, s} in the graph with k >= 2.
3. Let Q be the set of vertices in the cycle. Thus Q = {s, c_1, … c_k}.
4. Add links {s, c_1}, {c_1, c_2}, … {c_{k-1}, c_k} to TreeBlue
5. Add links {s, c_k}, {c_k, c_{k-1}}, … {c_2, c_1} to TreeRed
6. If |Q| = N, then terminate.
7. Choose a path P = {a, u_1, …, u_k, b} with at least 3 nodes such that both the end nodes of P (a and b) are in Q, but none of the internal nodes of P are in Q. Or, choose a cycle C = {a, u_1, … u_k, b}, where b = a, with only one node (a) in Q.
8. Add links {a, u_1}, {u_1, u_2}, … {u_{k-1}, u_k} to TreeBlue
9. Add links {b, u_k}, {u_k, u_{k-1}} … {u_2, u_1} to TreeRed.
10. Q = Q U {u_1, u_2, … u_k}
11. If |Q| = N, then terminate.
12. Go to Step 7



(a)



(b)

Figure 5.6: Effect of Network order and density on Restorability for tree algorithm (Red-Blue)

5.4 Performance of Red Blue Tree Algorithm

In this section, we investigate the performance of the Red Blue Tree Algorithm for several performance metrics, such as restorability, change in path length and sharability. We also investigate how various network parameters, such as network order, density and redundancy affect its performance.

5.4.1 Restorability

We measure the restorability of a network after a single failure in varying network size, network density and spare capacity when a tree-algorithm is used for protection. We analyze the effect of these network parameters in the following subsections.

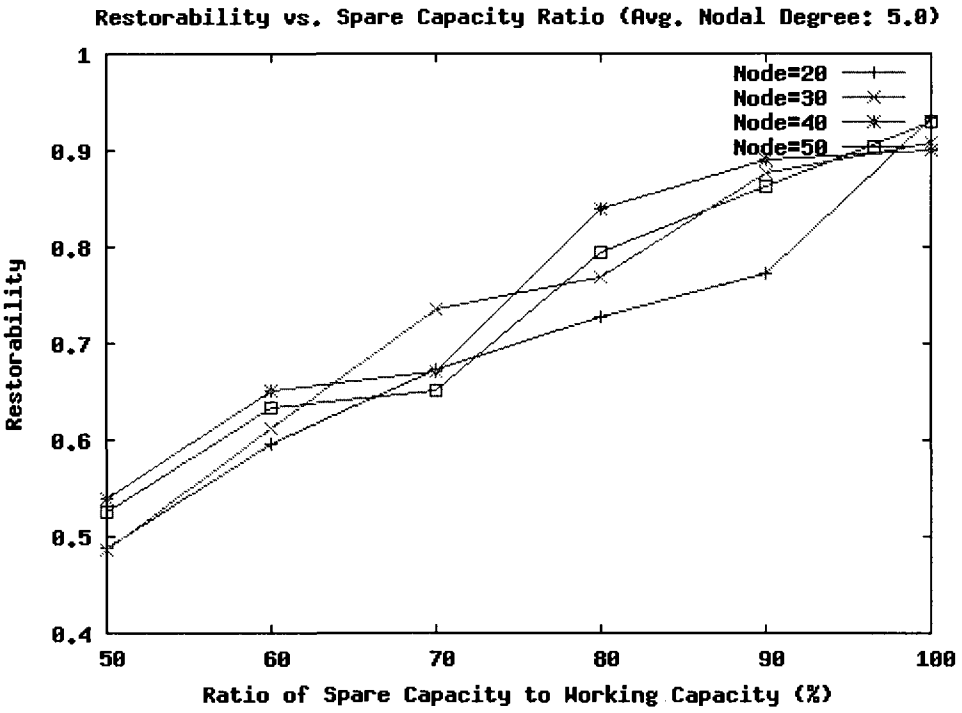
5.4.1.1 Effect of Network Order and Density

Figure 5.6 shows the performance of the Red-Blue tree algorithm. From the graph, we can see that the density and order of the network has less effect on the performance of this algorithm. This algorithm performs more like the p-cycle algorithm we used in Chapter 4. Although for this algorithm, the performance curve is more ragged. We conclude that neither the network order nor the network density has a significant on the performance of the Red-Blue algorithm.

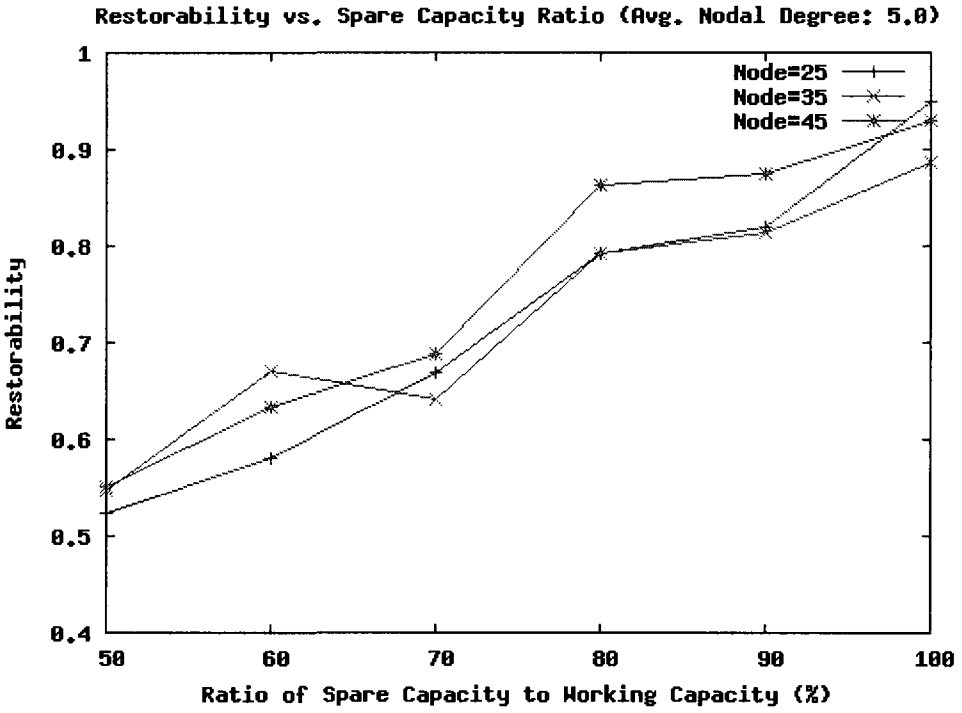
From both of the tree algorithms we use, we can see that the network order does not affect the restorability for either of the algorithms, although the two algorithms use completely different approaches to generate the protection paths. From this, we predict that the network order does not affect the restorability provided by tree algorithms in general. However, we cannot draw a conclusion on the effect of average nodal degree on the restorability.

5.4.1.2 Effect of Spare Capacity

Figure 5.7 displays the change in network restorability provided by the Red Blue Tree algorithm when the spare capacity allocation in the network is changed. We notice that the performance curve for the Red-Blue algorithm displays the same behavior as the performance curve for MST2SP algorithm, i.e., the redundancy (spare capacity to working capacity ratio) greatly affects the restorability of the network. For example, for a

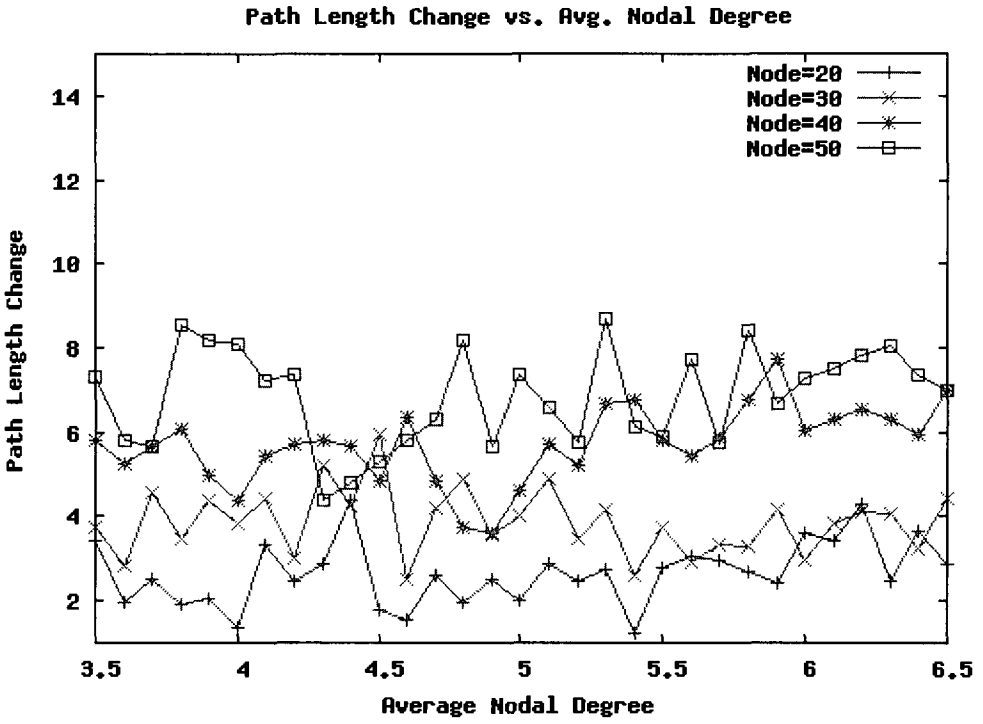


(a)

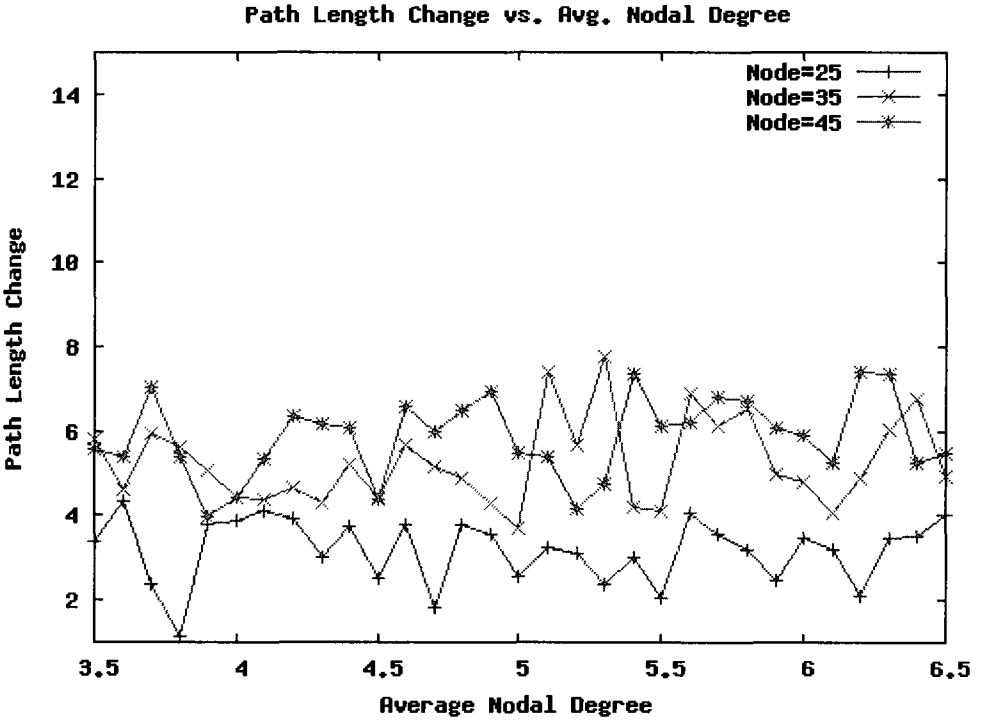


(b)

Figure 5.7: Effect of spare capacity allocation on Restorability for Red-Blue algorithm

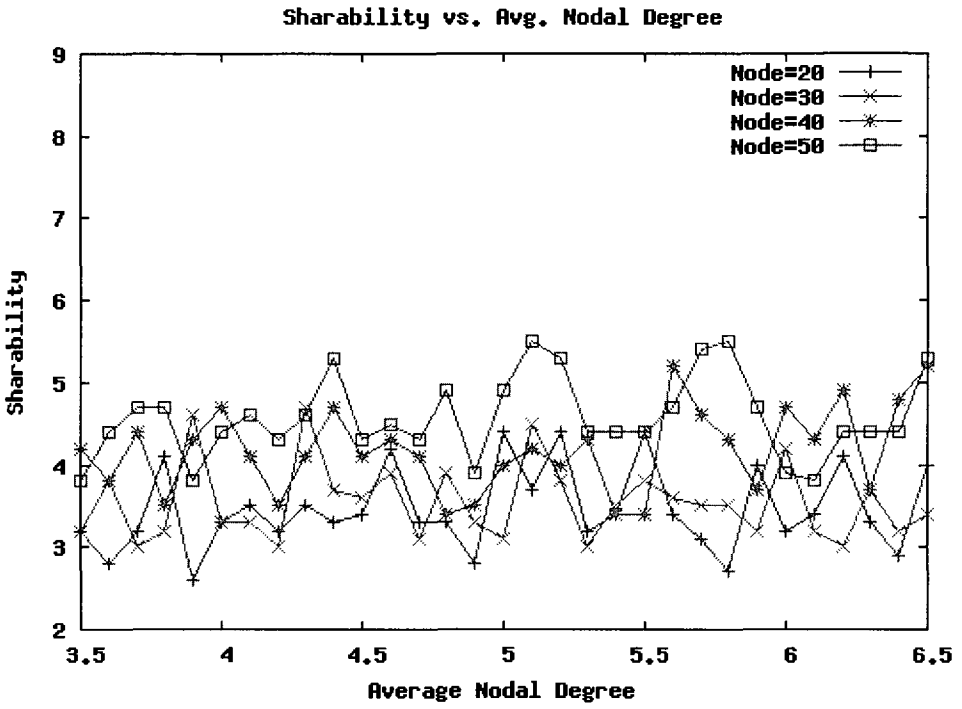


(a)

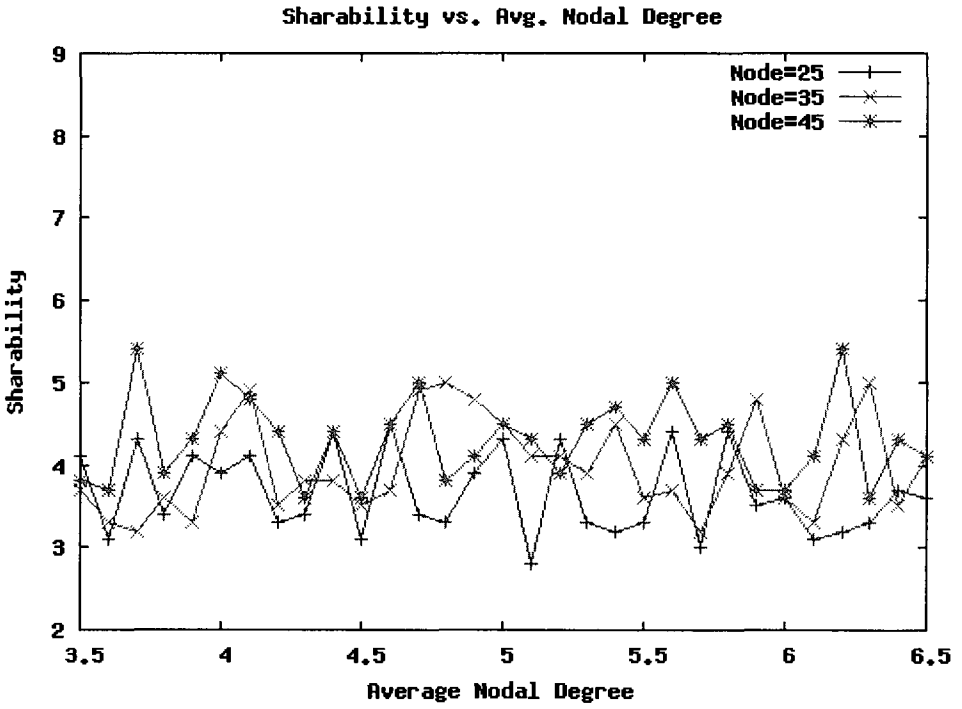


(b)

Figure 5.8: Effect of network order and density on Average Change in path-length for Red-Blue algorithm



(a)



(b)

Figure 5.9: Effect of network order and density on Sharability for Red Blue Algorithm

network of 30 nodes with AND 5.0, the restorability increases from 60% to 87% as the spare to working capacity ratio is increased from 60% to 90%.

These results confirm that the spare capacity allocation in the network has a very significant effect on the restorability of the network. This is the expected because increasing the ratio of the spare capacity allows for larger amount of data to be restored successfully, thus increasing the restorability. On the other hand, if the spare capacity ratio is very low, then the backup path may not be able to restore all the data of the primary data-path, thus reducing the restorability

5.4.2 Change in Path Length

From Figure 5.8, we notice that the change in path length for the Red-Blue algorithm is quite close to the change for the MST2SP algorithm in most cases. For example, for a network of 25 nodes with 5.0 AND, the average change in path length for the MST2SP algorithm is 5.4, and for the Red-Blue algorithm, it is 5.8. We also notice the same affect of network order and density on the performance curves for both the algorithms. This clearly demonstrates that, for both these algorithms:

- The network order significantly affects the average change in data-path length; and
- The average change in data-path length is not greatly affected by the nodal density of the network.

5.4.3 Sharability

As Figure 5.9 indicates, the affect of the network order does not have a significant effect on the sharability of the protection algorithm. We also note that the sharability is not affected a great deal by changing the AND of the network. For example, for a network with 25 nodes and AND 4.5, the sharability is 3.0. If we increase the order of the network to 45 nodes with the same AND, then the sharability is 3.5. On the other hand, if we change AND to 6.5 for the network with 25 nodes, the sharability becomes 3.6. Neither of these changes are significant, and the performance curves also suggest that there is only insignificant change in sharability with changes in nodal degree and AND.

5.5 The Hierarchical Tree Algorithm

Another tree-based protection algorithm we select for our comparison is the Hierarchical Tree algorithm presented in [ShYa04]. The algorithm provides a hierarchical layering of the network by constructing a hierarchical protection tree. The algorithm starts off by selecting a root node. The selection of the root node can depend on several criteria. We use the average adjacent protection capacity, i.e. the average of the protection capacities of all the adjacent edges of a node, as the root selection criterion. The algorithm then uses a distributed signaling mechanism to construct a layered tree, where each node in the network has an identifier that immediately describes the position of the node in the protection tree hierarchy, and also indicates its parent node in the tree. The construction of the trees for one source node s is described in Table 5.3

Table 5.3: The Hierarchical Tree Algorithm

1. Compute average adjacent protection capacity for each node in the network
2. Select the node with the highest average capacity as the root node
3. Add the root node to the Protection Tree
4. For each node *A* in the Protection Tree
    a. For each neighbor node *B* of *A*
        i. If *B* is not in the Protection Tree, or if B is in the protection tree, but as a child of a node with a less average capacity than *A*,
            1. Add *B* in the protection tree as a child node of node *A*

5.6 Performance of Hierarchical Tree Algorithm

In this section, we investigate the performance of the Hierarchical Tree Algorithm for several performance metrics, such as restorability, change in path length and sharability. We also investigate how various network parameters, such as network order, density and redundancy affect its performance.

5.6.1 Restorability

We measure the restorability of a network after a single failure in varying network size,

network density and spare capacity when a tree-algorithm is used for protection. We analyze the effect of these network parameters in the following subsections.

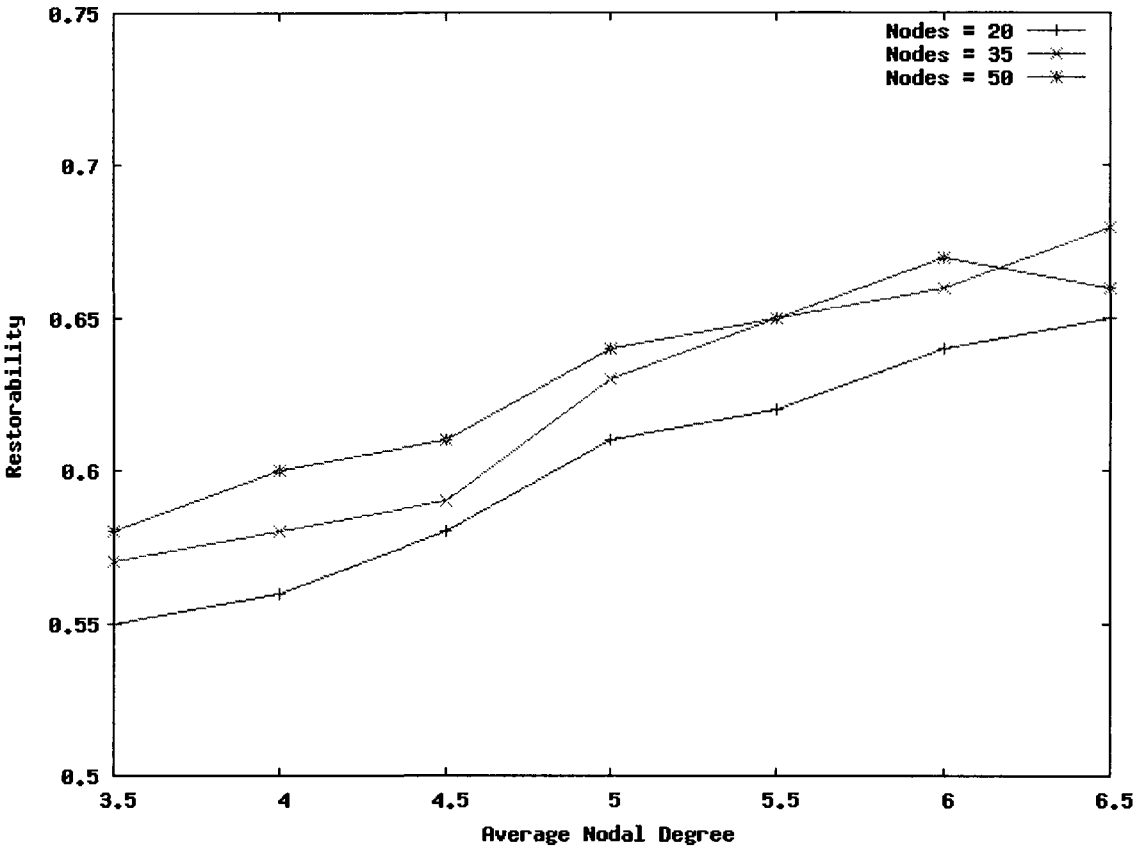


Figure 5.10: Effect of Network order and density on Restorability for tree algorithm (Hierarchical Tree)

5.6.1.1 Effect of Network Order and Density

Figure 5.10 shows the performance of the Hierarchical Tree algorithm. From the graph, we can see that the restorability increases as the average nodal degree is increased. The amount of improvement in performance is comparable with that for the MST2SP algorithm. We also note that the restorability also increases as the number of nodes in the network is increased, although very slightly. This is a different observation than for the MST2SP algorithm, where the network order does not affect the restorability.

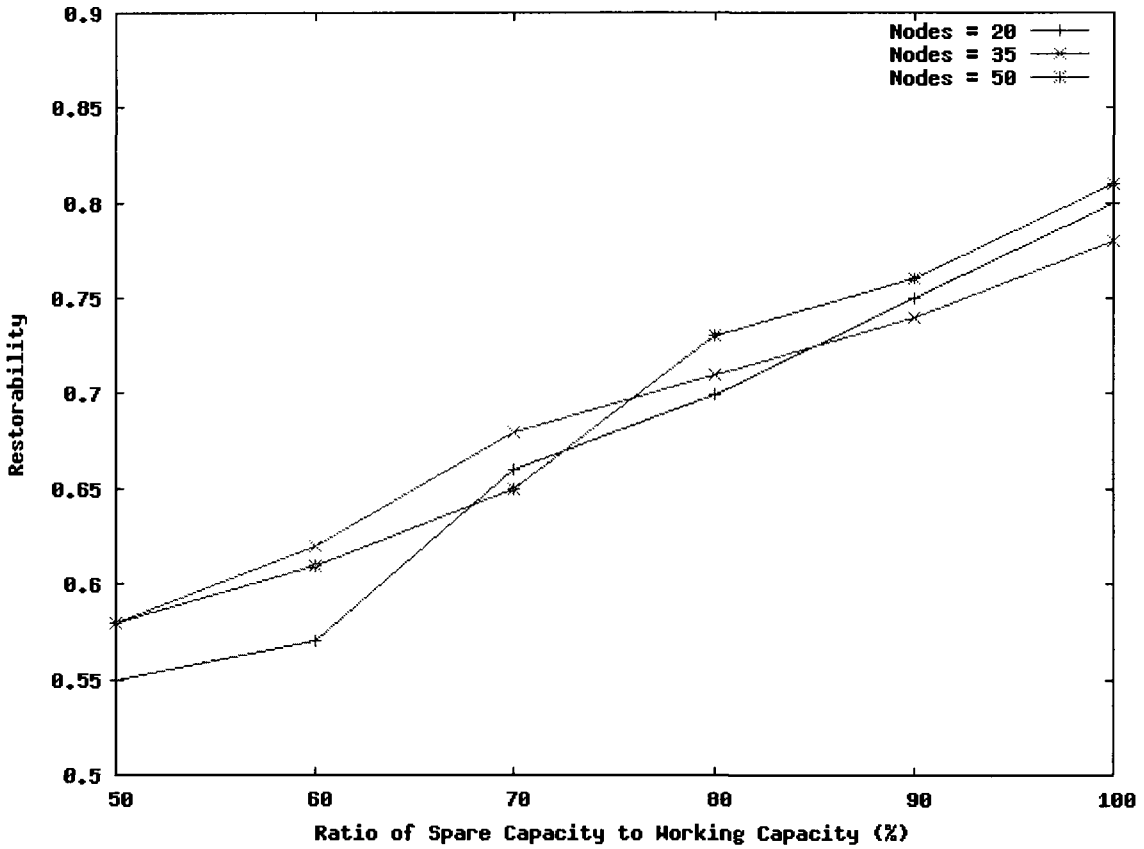


Figure 5.11: Effect of spare capacity allocation on Restorability for Hierarchical Tree algorithm

5.6.1.2 Effect of Spare Capacity

Figure 5.11 displays the change in network restorability provided by the Hierarchical Tree algorithm when the spare capacity allocation in the network is changed. We notice that the performance curve for this algorithm displays the same behavior as the performance curve for the MST2SP or the Red-Blue tree algorithm, i.e., the redundancy (spare capacity to working capacity ratio) greatly affects the restorability of the network. For example, for a network of 35 nodes with AND 5.0, the restorability increases from 62% to 74% as the spare to working capacity ratio is increased from 60% to 90%. We note that it outperforms the MST2SP algorithm when the redundancy is very low (approx 50%-60%), but both algorithms perform very similarly as the redundancy is increased (approx 90%-100%). However, the red-blue tree algorithm outperforms both of these algorithms in this regard.

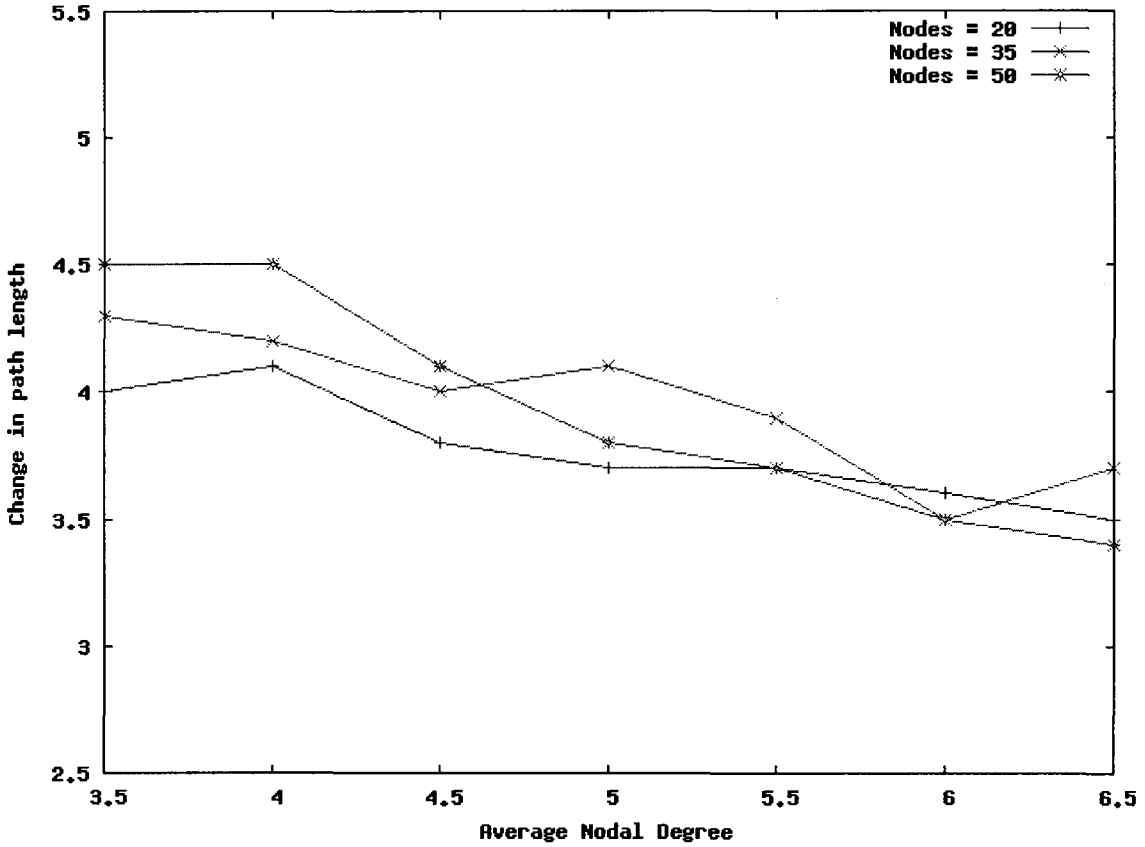


Figure 5.12: Effect of network order and density on Average Change in path-length for Hierarchical Tree algorithm

5.6.2 Change in Path Length

From Figure 5.12, we notice that unlike for the MST2SP algorithm or the Red-Blue tree algorithm, the average change in path length for the Hierarchical tree algorithm behaves differently as the average nodal degree is changed. While the performance measure is not affected by the nodal degree for the MST2SP or the Red-Blue tree algorithms, it does affect the performance for Hierarchical tree algorithm. We note that as the average nodal degree is increased, the change in path length decreases, although not by a great margin. In this regard, this algorithm performs better than the other two tree-based algorithms we use. We also note that like the earlier two tree-based algorithms, increasing the network order causes the average change in path length to increase, although we should note that the increase is much less for the hierarchical tree algorithm than for either of the MST2SP and the Red-Blue tree algorithms.

5.7 Comparison between MST2SP, Red-Blue Tree and Hierarchical Tree Algorithm

From the simulation results presented earlier in this chapter, we notice that the Red-Blue tree algorithm performs better than the MST2SP algorithm in terms of reliability. However, in the rest of the performance criteria we use, MST2SP algorithm performs better. We also note that the performance of hierarchical tree algorithm is very similar to the performance of the MST2SP algorithm. Changes in the order and density of the network affect the performances in a very similar fashion in both the algorithms. For example, as the AND of the network is increased, the restorability is also increased for both the algorithms. For this reason, in the next section, when comparing between a p-cycle algorithm and a tree-based algorithm, we use the MST2SP algorithm as the representative algorithm for the latter class of algorithms.

Table 5.4: Commercial Networks

Name	Nodes	Edges	AND	Map Path Length
ATT2	32	66	4.1	11.2
BBN	11	19	3.5	5.6
Brazil RNP	11	19	3.5	5.1
DataXChange	8	24	6	3.4
GridNet	10	25	5	4.2
AARNet	13	14	2.2	7.4
XONet	9	14	3.1	5.4
CanadaNET	24	41	3.4	9.6

5.8 Commercial Networks

We used several commercial networks for our simulation. The parameters of some commercial networks we used are shown in Table 5.4. Detailed topological information about these networks are provided in Appendix A. We present our measurement of the performances for some of the same commercial networks in Table 5.5, 5.6 and 5.7. For networks where redundancy was computed using probability distribution, we see that the restorability for the commercial networks for both the MST2SP algorithm and the Red-Blue tree algorithm agree with the results from the random networks we used. Note that the change in path length and sharability metrics also confirm the effect of AND on the performance of both these algorithms. However, the effect of network order is not evident from these results. We also note that the results are in a comparable range of

values for networks where redundancy was computed from the traffic matrix.

Table 5.5: Performance of MST2SP algorithm for Commercial Networks

Performance Measure	Networks with Redundancy from Traffic Matrix			Networks with Redundancy from Probability Distribution							
	ATT2	Brazil RNP	XONet	ATT2	Brazil RNP	XO Net	Data Xchange	GridNet	AAR Net	BBN	CNET
Restorability	0.60	0.59	0.60	0.65	0.67	0.65	0.76	0.56	0.45	0.45	0.59
Change in Path Length	3.26	2.74	3.07	3.26	2.74	3.07	2.58	2.76	8.8	4.53	2.88
Sharability	3.42	2.74	3.31	3.42	2.74	3.31	3.65	3.83	2.0	2.82	2.95

Table 5.6: Performance of Red-Blue Tree algorithm for Commercial Networks

Performance Measure	Networks with Redundancy from Traffic Matrix			Networks with Redundancy from Probability Distribution							
	ATT2	Brazil RNP	XONet	ATT2	Brazil RNP	XONet	DataXchange	GridNet	AARNet	BBN	CNET
Restorability	0.64	0.64	0.56	0.58	0.56	0.61	0.79	0.74	0.8	0.61	0.70
Change in Path Length	4.89	1.02	1.43	4.89	1.02	1.43	1.49	1.12	3.0	1.31	1.28
Sharability	3.32	4.2	4.31	3.32	4.2	4.31	3.22	2.92	2.0	3.56	3.15

Table 5.7: Performance of Hierarchical Tree algorithm for Commercial Networks

Performance Measure	Networks with Redundancy from Traffic Matrix			Networks with Redundancy from Probability Distribution							
	ATT2	Brazil RNP	XONet	ATT2	Brazil RNP	XONet	DataXchange	GridNet	AAR Net	BBN	CNET
Restorability	0.61	0.55	0.48	0.60	0.59	0.41	0.72	0.68	0.46	0.59	0.60
Change in Path Length	4.10	3.45	3.76	4.10	3.45	3.76	3.12	3.08	6.87	4.32	3.2

5.9 Summary

From Table 5.8, we can see that although network order does not affect restorability greatly, it does have a significant effect on change in path length and sharability, where both these performance metrics increase as the network order is increased. Network

Table 5.8: Effects of Network Parameters on Performances of Tree-Algorithms

Performance Measure	Network Order	Network Density	Redundancy
Restorability	Minimal	Directly Proportional	Directly Proportional
Change in Path Length	Directly Proportional	Minimal	N/A
Sharability	Directly Proportional	Directly Proportional	N/A

density, on the other hand, does not affect the change in path length. However, both restorability and sharability increase as the network density increases. Network redundancy also affects restorability, where restorability increases as the redundancy in the network is increased. We do not compute the effect of redundancy on change in path length or sharability, because the network redundancy does not affect the creation of the protection trees, and thus does not affect either of these performance metrics.

Chapter 6
Comparison of Tree and Cycle Algorithms

In this section, we compare the performances of the MST2SP algorithm with the CIDA algorithm because they are the better ones from previous comparisons of cycle-based (Section 4.5) and of tree-based algorithms (Section 5.7). Although our previous comparisons are done from a limited number of candidates, we attempt to provide here some general performance comparisons between cycle-based and tree-based algorithms, keeping in mind that many of the comparisons are not absolute, and may require further explorations.

6.1 Restorability

The network parameters, network order, network density and redundancy all can affect the performances of different algorithms in a different way.

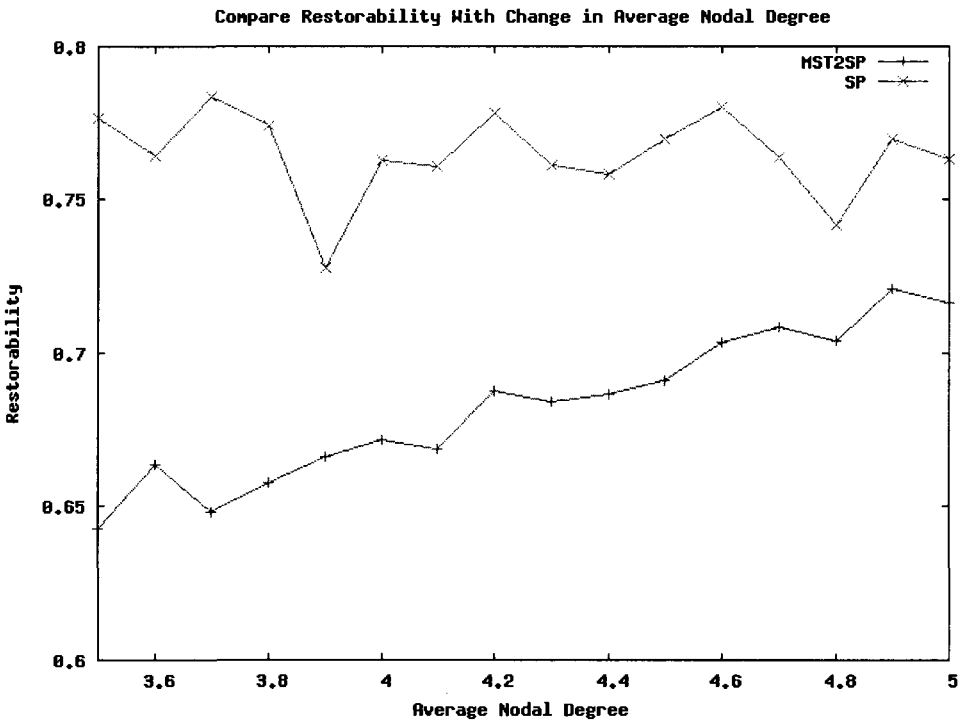
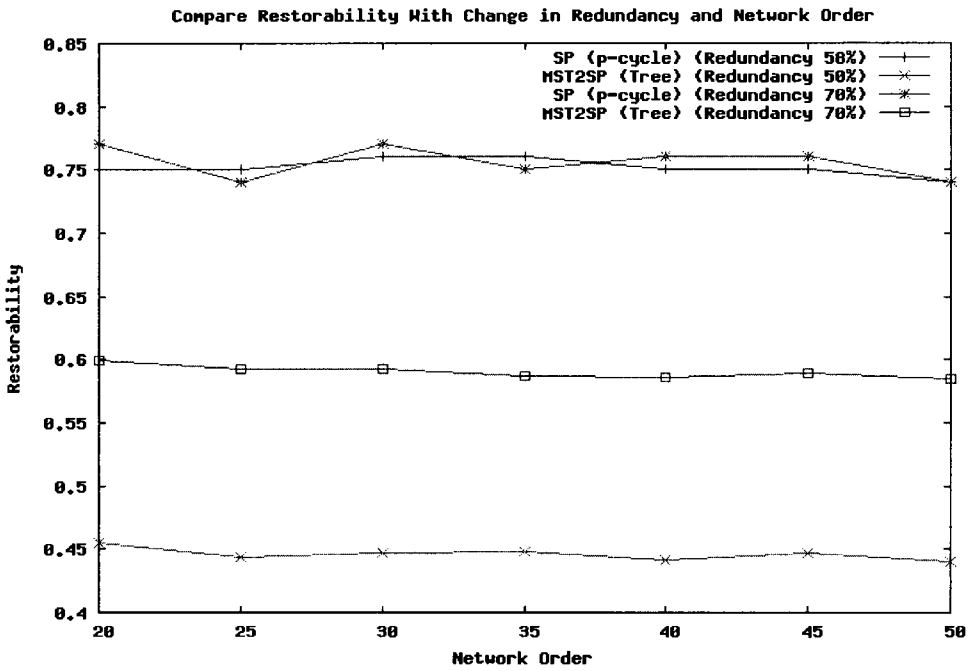
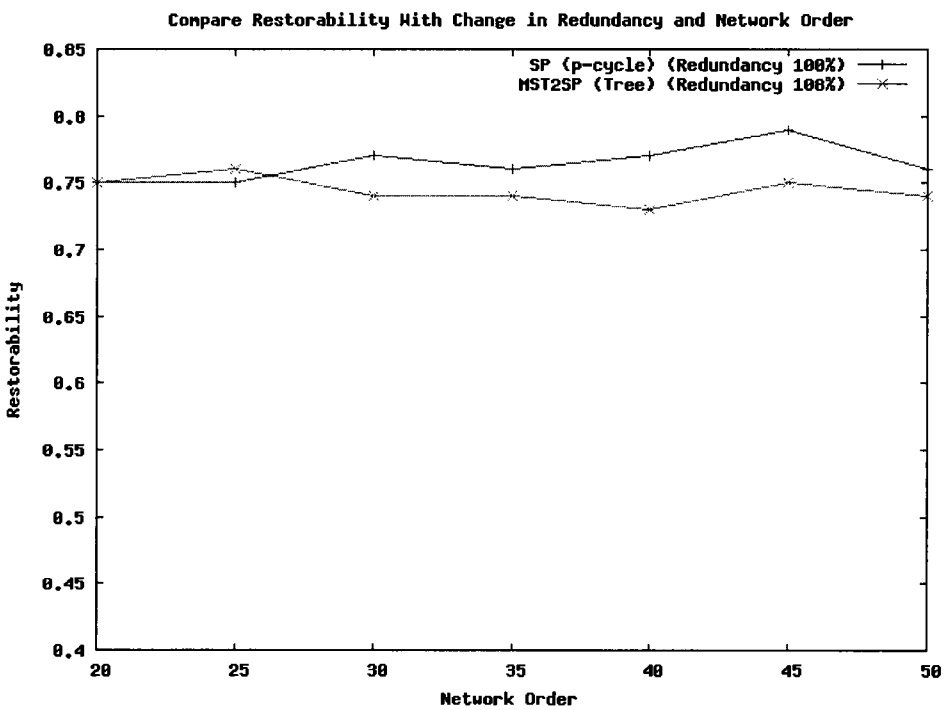


Figure 6.1: Comparison of Restorability with changes in Average Nodal Degree

All the networks in Figure 6.1 had 30 nodes and an average redundancy of 90%. We notice from the figure that the restorability of the tree algorithm has a largely increasing



(a)



(b)

Figure 6.2: Comparison of Restorability with changes in Redundancy and Network Order

tendency as the nodal degree increases. At the same time, however, the restorability of the p-cycle algorithm maintains the same average value irrespective of the change in AND.

For all the networks in Figure 6.2, the average nodal degree (AND) is 4.5. From the figure, we can see that at a low redundancy, the p-cycle algorithm (labeled 'SP') performs better than the tree algorithm (labeled 'MST2SP'). However, as the redundancy increases, the restorability of the p-cycle algorithm remain the same, but the restorability of the tree algorithm is becoming better. At 100% redundancy, we see that both the p-cycle algorithm and the tree algorithm achieve almost the same restorability.

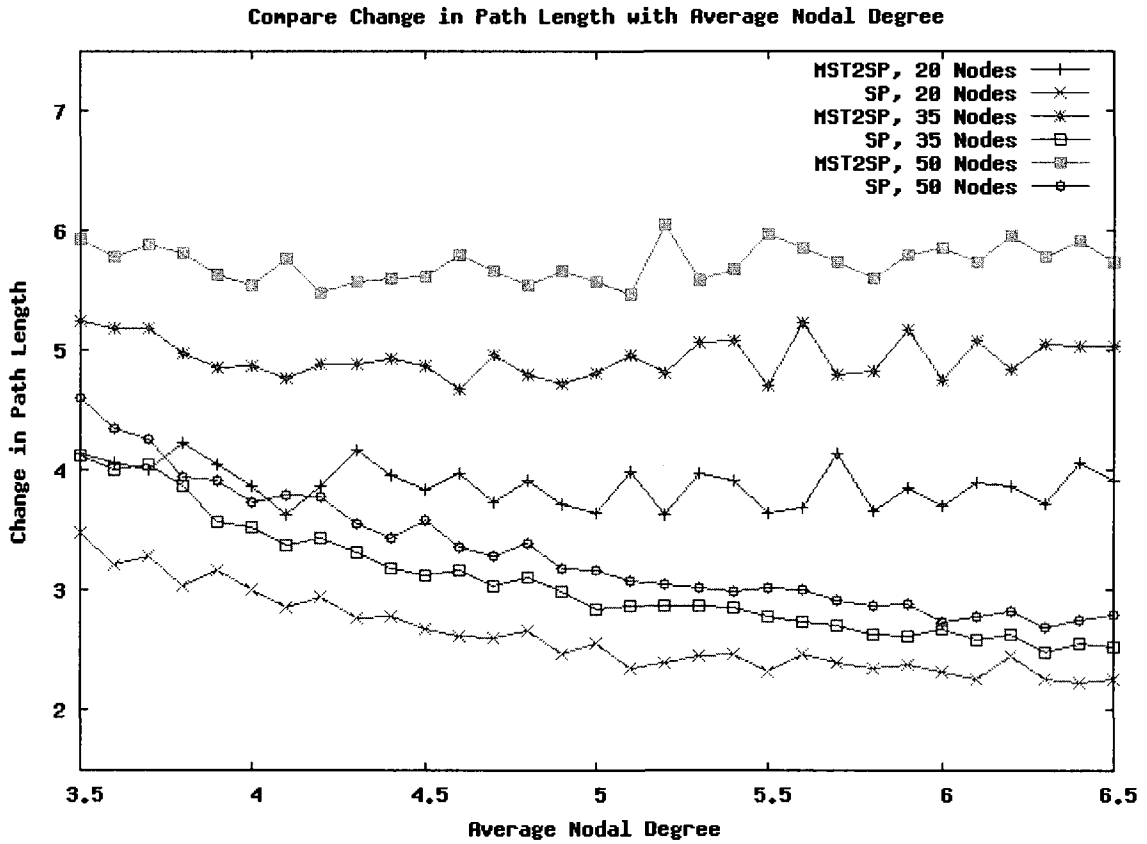


Figure 6.3: Comparison of Change in Path Length with changes in Average Nodal Degree

6.2 Change in Path Length

From Figure 6.3, we observe that the change in path length of the backup path generated by the p-cycle starts to decrease as the nodal degree is increased. The same trend is

observed even if the order of the network is increased. However, for the backup paths that the tree algorithm creates, the change in path length remains the same even as the network density is increased. So the network density has a larger effect on the performance of the p-cycle algorithm than on that of the tree algorithm.

We also note that for a constant density, the change in path length increases if the network order is increased. This holds for the backup paths generated by both the p-cycle and the tree algorithm. However, the rate of change in the p-cycle algorithm is very small, where the rate of change for the tree algorithm is larger. For example, at a constant AND of 5.0, if we increase the network order from 20 to 50, then the change in path length for the p-cycle algorithm increases from 2.5 to 3.4, which is only 36%, whereas the for the tree algorithm, the change in path length increases from 3.9 to 5.8, which is a 50% increment. We can explain this result analytically. For a tree-based protection algorithm, the protection paths for most of the failed links, except for the ones that are on the tree itself), are provided by the path between the two nodes affected by the failure along the tree. The length of the path along the tree can be large, especially for graphs of low density, because a spanning tree in sparse graph has more depth. On the other hand, for p-cycles, the backup path along the cycle is usually not very large, because most algorithms impose a constraint on the maximum length of the p-cycle. For example, the SLA algorithm we use to generate the candidate p-cycles create the p-cycle by joining the two shortest paths between a pair of nodes. This ensures that the backup path along the cycle is most often better than the backup path along a protection tree in a tree-based algorithm.

6.3 Sharability

As is evident from Figure 6.4, the AND of the network also have different effects on the sharability of different protection algorithms. For the p-cycle algorithm, the sharability starts decreasing very slowly as the density is increased. However, increasing the density of the network has a completely different effect on the tree algorithm, where the sharability starts increasing with the density. For example, for a network of 50 nodes, the sharability *decreases* from 5.4 to 3.8 (approx. 30%) as the density is increased from 3.5

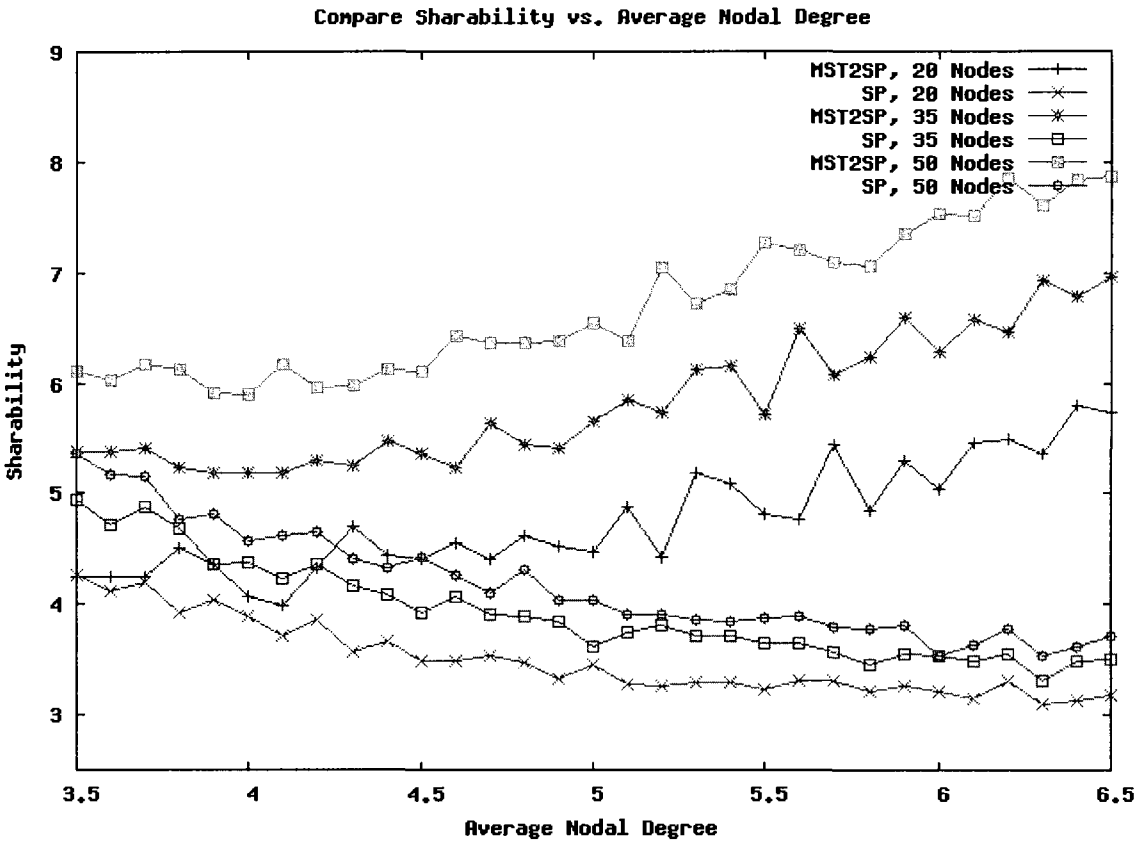


Figure 6.4: Comparison of change in Sharability with changes in Average Nodal Degree

AND to 6.5 AND, whereas the sharability for the tree algorithm *increases* from 6.1 to 7.9 (approx. 30%) for the same change in network density. This is a quite interesting result, and it demonstrates a clearer difference between the two algorithms. From these results, we claim that the tree algorithm provides a higher sharability than the p-cycle algorithm.

6.4 Complexity

As discussed in Section 4.1, the algorithmic complexity of the CIDA algorithm is $O(M^3N \log N)$. However, the complexity of the MST2SP algorithm, as discussed in Section 5.1, is only $O(MN \log N)$, which indicates that the latter algorithm is better suited for large networks, especially for dense networks where there are a lot of edges. Other p-cycle algorithms also require construction of candidate p-cycles first. This imposes an additional cost of computing and then rejecting some candidate p-cycles. This is not the case for most tree-based algorithms. There are, however, some tree-based algorithms

where it is possible to create multiple trees, and the algorithm uses a heuristic to select one. To find the optimum tree, the algorithm would need to construct the candidate trees and select the best one. For example, the Hierarchical Tree algorithm [ShYa04] selects a root node for the protection tree based on a heuristic that uses the nodal degree. However, to guarantee an optimum solution, the algorithm would need to construct a rooted protection tree for all the nodes, and then compute the best one. Any algorithm, cycle-based or tree-based, that require creating data-structures and rejecting some of them add additional computation and complexity to the performance, which is not desirable.

6.5 Commercial Networks

Comparing our performance measurements for the algorithms for real life commercial networks, we observe that the restorability for both the p-cycle algorithm and the tree-algorithm are very close, with the p-cycle slightly outperforming the tree algorithm. The change in path-length for the tree algorithm is also larger than for the p-cycle algorithm. In case of sharability, however, no clear victor emerges, as the tree algorithm achieves higher sharability than the p-cycle algorithms for some commercial networks, while vice versa for the others. Although the orders of most of the commercial networks are smaller than those of the generated networks, these results do agree with the trend of the comparative performances of the algorithms.

Table 6.1: Restorability in Commercial Networks

Name		P-cycle Alg. (CIDA)	Tree Alg. (MST2SP)
Networks with Redundancy from Traffic Matrix	ATT2	0.65	0.60
	BBN	0.71	0.45
	Brazil RNP	0.42	0.59
Networks with Redundancy from Probability Distribution	DataXChange	0.64	0.76
	GridNet	0.62	0.56
	AARNet	0.30	0.45
	XONet	0.48	0.60
	CanadaNET	0.64	0.59

Table 6.2: Change in Path Length in Commercial Networks

Name		P-cycle Alg. (CIDA)	Tree Alg. (MST2SP)
Networks with Redundancy from Traffic Matrix	ATT2	2.81	3.26
	Brazil RNP	1.05	2.74
	XONet	3.57	3.07
Networks with Redundancy from Probability Distribution	DataXChange	2.75	2.58
	GridNet	3.68	2.76
	AARNet	1.67	8.8
	BBN	4.26	4.53
	CanadaNET	4.08	2.88

Table 6.3: Sharability in Commercial Networks

Name		P-cycle Alg. (CIDA)	Tree Alg. (MST2SP)
Networks with Redundancy from Traffic Matrix	ATT2	3.80	3.42
	Brazil RNP	1.80	2.74
	XONet	5.56	3.31
Networks with Redundancy from Probability Distribution	DataXChange	4.13	3.65
	GridNet	5.11	3.83
	AARNet	1.0	2.0
	BBN	5.40	2.82
	CanadaNET	4.94	2.95

6.6 Concluding Remarks

As we have demonstrated through our simulation results, the p-cycle algorithm and tree algorithm perform differently under the same network conditions. Changes in various network parameters also affect the performances of the algorithms differently. As a consequence, this demonstrates the importance of properly comparing performances of various algorithms under variable network configurations before deploying a particular network protection algorithm. The performances for more protection algorithms and for other performance metrics should also be measured using the framework proposed in Chapter 3.

Chapter 7
Conclusions

In this research, we have proposed and implemented a novel simulation framework to measure performances of network protection algorithms for any metric. This framework is very flexible and scalable and can be used with any protection algorithm to measure the performance for any metric and compare the result with the performance of any other algorithm. We have used the framework on various network protection algorithms, and have presented and analyzed the effect of different network parameters on the performances of the protection algorithms for different metrics. We have also compared the performances of the tree-based algorithms and the p-cycle based algorithms. Our performance comparison demonstrated that in some performance criteria such as restorability, change in path length, the p-cycle algorithm performs better than the tree-based algorithm. However, for sharability, the tree-based algorithm performs better. The tree algorithms discussed are also of smaller algorithmic complexity, thus requiring less computation resources. We can decidedly conclude that there is no one algorithm, or one class of algorithm, that will outperform other algorithms for all performance measures. Rather, an algorithm can perform better than the others for certain performance criteria, but it may not be a good algorithm to use when a different performance criteria is desired. The selection of protection algorithms for a network should thus be decided based on the network parameters, and the desired performance metrics.

Our study suggests that the comparison could have been more elaborate and complete had we established a database recording the performances of various protection and restoration algorithms for various metrics. This database can work as a benchmark for new protection algorithms. It can also help administrators of large optical networks to select an appropriate protection algorithm suitable for that particular network.

7.1 Future Work

The proposed simulation framework can be improved to measure performance on various metrics simultaneously, utilizing parallel computing abilities. Also, performance can be measured for the algorithms on various other performance metrics, and the effect of more network parameters on the performances of the algorithms can be measured.

References

[AhMa93] K Ahuja, T L Magnanti, "Network Flows: Theory, Algorithms, and Applications," *Prentice Hall*, ISBN: 978-0136175490

[AmLi03] E Amaldi, L Liberti, N Maculan, F Maffioli, "The Minimum Fundamental Cycle Basis Problem: A New Heuristic Based On Edge Swaps", *Proceedings of Workshop on Approximation and Online Algorithms,* 2003, pp 315-317.

[AmLi04] E Amaldi, L Liberti, F Maffoli, "Algorithms for finding minimum fundamental cycle bases in graphs", *Proceedings of Workshop on Graphs and Combinatorial Optimization*, 2004, vol. 17, pp. 29-33.

[Bell95a] Bellcore, "SONET Dual-Fed Unidirectional Path Switched Ring (UPSR) Equipment Generic Criteria", *GR-1400-Core,* 1995. Issue 1.

[Bell95b] Bellcore, "SONET Bidirectional Line-Switched Ring Equipment Generic Criteria", *GR-1230-Core,* 1995. Issue 2.

[CaDo97] K Calvert, M Doar, E Zegura, "Modeling Internet Topology," *IEEE Communication Magazine*, 1997. pp. 160-163.

[ChJa05] P. Cholda, A. Jajszczyk, "Reliability Assessment of Rings and p-Cycles in DWDM Networks", *IEEE 1st EuroNGI Conference on Next Generation Internet, Networks - Traffic Engineering*, Rome, Italy, April 18-20, 2005.

[DePr82] N Deo, GM Prabhu, "Algorithms for Generating Fundamental Cycles in a Graph", *ACM Transactions on Mathematical Software*, March 1982, vol. 8, pp 26-42.

[Dijk59] E. W. Dijkstra, "A note on two problems in connexion with graphs," *In Numerische Mathematik, 1*, 1959, pp. 269–271.

[DoHe03] J Doucette, D He, W D Grover, O Yang, "Algorithmic Approaches for Efficient Enumeration of Candidate p-Cycles and Capacitated p-Cycle Network Design", *Proceedings of Workshop on Design of Reliable Communication Networks*, 2003, pp. 212-220.

[Douc05] J Doucete, "Advances on Design and Analysis of Mesh-Restorable Networks", PhD Thesis, University of Alberta, Spring 2005.

[DoYa06] D He, O Yang, "A Heuristic Detection/Protection Algorithm on DWDM Layers", *Canadian Conference on Electrical and Computer Engineering,* 2006, pp 859-862

[DuGr94] D A Dunn, W D Grover, M H MacGregor, "Comparison of k-Shortest Paths and Maximum Flow Routing for Network Facility Restoration," *IEEE Journal on Selected Areas in Communications*, 1994, vol. 12, pp. 88-99.

[ErRe59] P Erdős, A Rényi, "On Random Graphs. I," *Publicationes Mathematicae*, 1959. pp. 290-297.

[FeCu01] A Fei, J Cui, M Gerla, D Cavendish, "A "Dual-Tree" Scheme for Fault-Tolerant Multicast", *IEEE International Conference on Communications, 2001.* ICC 2001, vol. 3, pp. 690-694.

[Fris63] I T Frisch, "Optimum Routes in Communication Systems with Channel Capacities and Channel Reliabilities", *IEEE Transactions on Communicatons Systems, June 1963,* pp 241-245.

[GeRa00a] O Gerstel, R Ramaswami, "Optical Layer Survivability: a services Perspective", *IEEE Communications Magazine*, 2000. vol 38, pp. 104-113.

[GeRa00b] O Gerstel, R Ramaswami, "Optical Layer Survivability: an implementation Perspective", *IEEE Journal on Selected Areas in Communications*, 2000. vol 18, pp. 1885-1899.

[GrDo02] W D Grover, J D Doucette, "Advances in Optical Network Design with p-Cycles: Joint Optimization and Pre-selection of Candidate p-Cycles," *Proceedings of IEEE LEOS Summer, Topical Meeting on All Optical Networking*, 2002. pp. 49-50.

[GrGr07] A Grue, W D Grover, "Comparison of p -cycles and p -trees in a unified mathematical framework", *Photonic Network Communications,* 2007, vol 14, pp. 123-133

[Grov98] W D Grover, "Cycle Oriented Distributed Preconfiguration: Ring-like Speed with Mesh-like Capacity for Self-planning Network Restoration", *Proceedings of IEEE International Conference on Communications*, 1998, pp. 537-543.

[Grov03] W D Grover, "Mesh-Based Survivable Networks," *Prentice Hall*, 2003. ISBN: 0-13-494576-X.

[GrSc02] C G Gruber, D A Schupke, "Capacity Efficient Planning of Resilient Networks with p-cycles," *Proceedings of 10th International Telecommunication Network Strategy and Planning Symposium*, 2002.

[GrSh03] W D Grover, G Shen, "Extending the p-Cycle concept to Path-segment Protection," *Proceedings of IEEE International Conference on Communications*, 2003, vol. 2, pp. 1314-1319.

[GrSt00] W. D. Grover, D. Stamatelakis, "Bridging the ring-mesh dichotomy with p-cycles," *Proceedings of IEEE/VDE Workshop on Design of Reliable Communication Networks (DRCN 2000)*, 2000, pp. 92-104.

[Homo04] P H Ho, H T Mouftah, "Reconfiguration of Spare Capacity for MPLS-Based Recovery in the Internet Backbone Networks", *IEEE/ACM Transactions On Networking*, 2004. vol 12. pp. 73-84.

[Hort87] J D Horton, "A polynomial-time algorithm to find the shortest cycle basis of a graph", *SIAM Journal on Computing*, 1987, vol.16, pp. 358–366.

[HwAh01] H Hwang, S Ahn, Y Yoo, C Sang Kim, "Multiple Shared Backup Cycles for Survivable Optical Mesh Networks", *Proceedings of ICCCN* 2001, Oct. 2001, pp. 284-289.

[IrGr00] R R Iraschko, W D Grover, "A Highly Efficient Path-Restoration Protocol for Management of Optical Network Transport Integrity", *IEEE Journal on Selected Areas of Communications* 2000, vol. 18, pp. 779-793.

[Kart99] S V Kartalopoulos, "Introduction to DWDM Technology", *IEEE Press,* 1999. ISBN: 0-7803-4745-5.

[KiLe98] C H Kim, C Lee, Y C Chung, "Bidirectional WDM Self-Healing Ring Network Based on Simple Bidirectional Add/Drop Amplifier Modules", *IEEE Photonics Technology Letters*, 1998. vol. 10, pp. 1340-1342.

[KoGr05] A Kodian, W D. Grover, "Failure-Independent Path-Protecting p-Cycles: Efficient and Simple Fully Preconnected Optical-Path Protection", *Journal of Lightwave Technology*, 2005, vol. 23, Issue 10, pp. 3241.

[Krus56] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, 1956, vol. 7, pp. 48-50.

[LeGr02] S. K. Lee, D. Griffith and N. O. Song, "An Analytical Approach to Shared Backup Path Provisioning in GMPLS Networks", *Proceedings of IEEE MILCOM 2002, 7-10 Oct.* 2002, vol. 1, pp. 75-80.

[LiYa03] H Liu, O Yang, S Heydari, "A Tree-based Link Protection Algorithm", *IEEE Electrical and Computer Engineering 2003 Canadian Conference*, 2003, vol. 2, pp. 939-942.

[Mapnet] MAPNET web site, *http://www.caida.org/tools/visualization/mapnet/*

[MaTa03] A Madina, N Taft, K Salamation, S Battacharya, C Diot, "Traffic Matrix Estimation: Existing Techniques and New Directions", ACM Special Interest Group on Data Communications 2002, pp. 161-174

[MaYa05] W Mardini, O Yang, Y Zhai, "Using MCG to Find PP-Cycles in Planar Graphs," *Proceedings of OFC/NFOEC*, 2005, vol. 3.

[MeFi99] M Medard, S G Finn, R A Barry, R G Gallager, "Redundant Trees for Preplanned Recovery in Arbitrary Vertex-Redundant or Edge-Redundant Graphs", *IEEE/ACM Transactions on Networking*, 1999. vol. 7, pp. 641-652.

[PaSt82] C Papadimitriou, K Steiglitz, "Combinatorial Optimization Algorithms and Complexity", Prentice Hall, 1982.

[Prim57] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, 1957, vol. 36, pp. 1389-1401.

[RaSi02] R Ramaswami, K N Sivarajan, "Optical Networks: a Practical Perspective", *Morgan Kaufmann*, 2002.

[ScGr02] D. A. Schupke, C. G. Gruber, A. Autenrieth, "Optimal Configuration of p-Cycles in WDM Networks," *Proceedings of IEEE International Conference on Communications (ICC 2002)*, 2002, vol. 5, pp. 2761 –2765..

[Schu04] D A Schupke, "Comparison of p-cycle Configuration Methods for Dynamic Networks", *IFIP Optical Networks & Technologies Conference (OpNeTec)*, 2004.

[ShYa04] S Shah-Heydari, O Yang, "Hierarchical Protection Tree Scheme for Failure Recovery in Mesh Networks," *Photonic Networks Communication Journal*, 2004, vol. 7, pp. 145-159.

[Shah07] S Shah-Heydari, "Self-Repairing Hierarchical Tree-based Link Restoration Scheme for Mesh Networks", PhD Thesis, School of Information Technology and Engineering, University of Ottawa, 2007.

[T1A193] "A Technical Report on Network Survivability Performance", prepared by T1A1.2 Working group on Network Survivability Performance, Report No. 24, November 1993.

[TaRu05] F Tang, L Ruan, "A Protection Tree Scheme for First Failure Protection and Second Failure Restoration in Optical Networks," *Proceedings of ICCNMC* 2005, pp. 620-631.

[Waxm88] B Waxman, "Routing of Multipoint Connections," *IEEE Journal of Selected Areas of Communication*, 1988, vol. 6, pp. 1617-1622.

[WuLa90] T. Wu and R. Lau, "A Class of Self-Healing Ring Architectures for SONET Network Applications", *IEEE Transactions on Communication*, Nov 1992, vol. 40, pg 1746-1756.

[WuTs92] T Wu, "Fiber Network Service Survivability", Artech House, Boston, London, 1992.

[XuCh02] G Xue, L Chen, K Thulasiraman, "QoS issues in redundant trees for protection in vertex-redundant or edge-redundant graphs", *IEEE International Conference on Communications*, 2002, vol 5, pp 2766-2770.

[XuSu99] G Xue, S Sun, "Optimal Multicast Trees in Communication Systems with Channel Capacities and Channel Reliabilities", *IEEE Transactions on Communications,* May 1999, vol 47, pp 662-663.

[ZeCa97] E Zegura, K Calvert, M Donahoo, "A Quantitative Comparison of Graph-Based Models for Internet Topology," *IEEE/ACM Transactions on Networking*, 1997, vol. 5, pp. 770-783.

[ZhXu06] W Zhang, G Xue, J Tang, K Thulasiraman, "Linear Time Construction of Redundant Trees for Recovery Schemes Enhancing QoP and QoS", *INFOCOM* 2005, vol. 4, pp. 2702-2710.

[ZhYa02a] Y Zhang, O Yang, "Different Implementations of Token Tree Algorithm for DWDM Network Protection/Restoration", *Proceedings of IEEE Conference on Computer Communications and Networks*, Oct 2002, pp 290-295.

[ZhYa02b] H Zhang, O Yang, J Wu, J M Savoie, T Shan, G Wang, "A Cycle-Based Rerouting Scheme for Wavelength-Routed WDM Networks", *Proceedings 2004 International Conference on Parallel Processing Workshops* 2004. pp. 427-433.

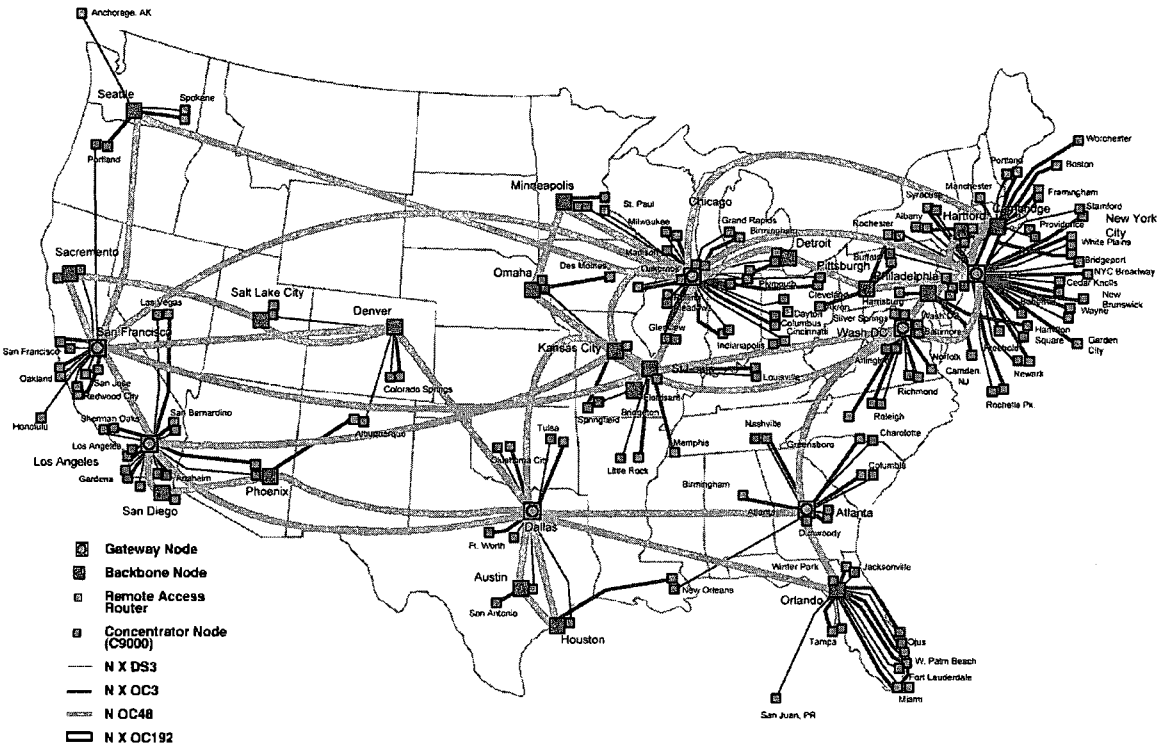
[ZhYa02c] Y Zhang, O Yang, "A Distributed Tree Algorithm for WDM Network Protection/Restoration", *IEEE International Conference on High Speed Networks and Multimedia Communications,* July 2002. pp. 289-294.

[ZhZh03] Z Zhang, W Zhong and B Mukherjee, "A Heuristic Method for Design of Survivable WDM Networks With p-Cycles", *IEEE Communications Letters*, July 2004, vol. 8, pp. 467-469.

Appendix A: Real-life Network Topologies

A.1 Real Network Topologies

The network topologies in this section have been constructed based on real commercial networks from the MAPNET database [Mapnet]. The links to hanging nodes, i.e. nodes with nodal degree of one were removed since such nodes would violate the two-edge connectivity requirements of link restoration. The total link capacities were normalized to DS1 or DS3 channel capacity depending on the granularity of the link capacities.



[Source: http://personalpages.manchester.ac.uk/staff/m.dodge/cybergeography/atlas]

1) ***ATT2***

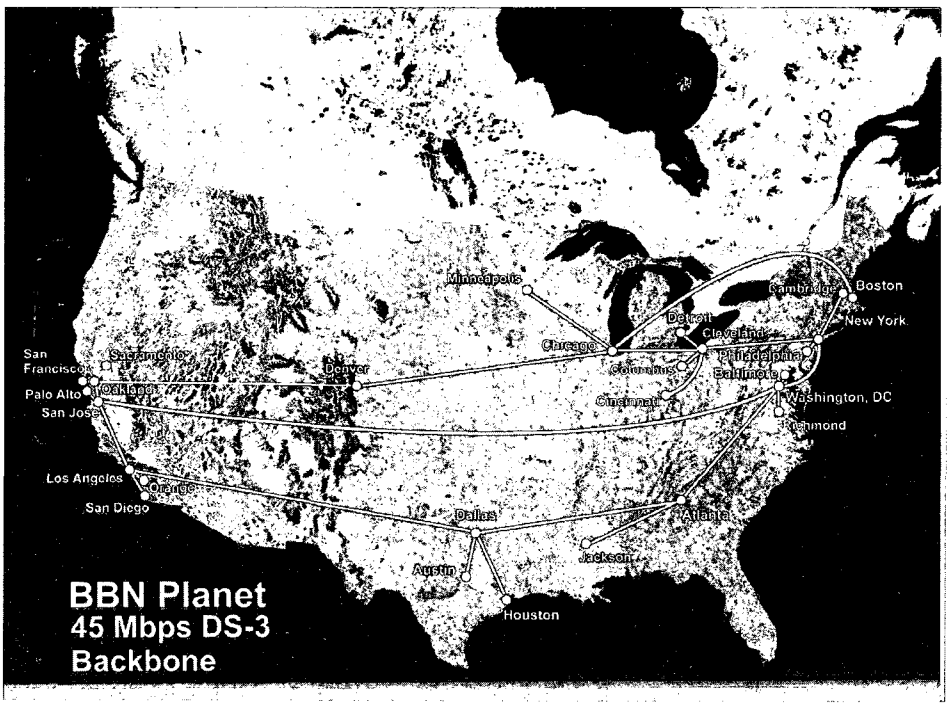
Nodes: 32

Edges: 66

Node 1	Node 2	Total Link Capacity
1	2	242
1	4	50
1	5	2
1	10	2
1	16	1
1	19	50

1	20	48
1	21	48
1	22	48
1	30	4
1	31	1
2	4	48
2	22	48
2	23	1
2	26	3
2	30	3
3	4	50
3	5	48
3	26	3
4	5	50
4	8	2
4	9	50
4	12	48
4	19	2
4	21	1
4	23	48
4	24	49
4	28	4
5	6	48
5	8	50
5	9	48
5	10	2
5	11	2
5	16	2
6	7	48
7	8	49
8	10	96
8	27	1
9	10	2
9	12	48
9	17	2
9	19	48
10	11	2
10	12	96
10	14	48
10	15	48
10	16	48
10	17	48
10	18	48
11	12	48
11	14	4
12	13	48

12	25	49
13	14	48
14	27	3
15	18	48
16	17	49
16	18	2
16	19	1
17	19	50
17	28	3
19	20	48
19	29	3
20	29	3
23	31	1
24	25	48



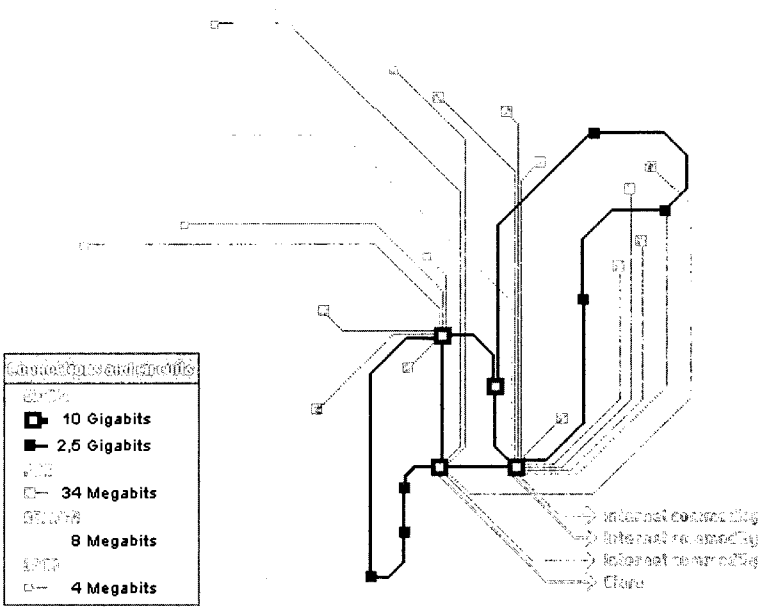
[Source: http://www.nthelp.com/maps.htm]

2) ***BBN***

Nodes: 11

Edges: 19

Node 1	Node 2	Total Link Capacity
1	2	3
1	3	12
1	4	24
1	8	12
2	3	6
3	7	12
3	8	24
3	9	12
4	6	12
4	10	12
5	6	12
5	10	12
5	11	12
6	7	3
8	9	12
8	10	12
8	12	12
9	12	12
10	11	24



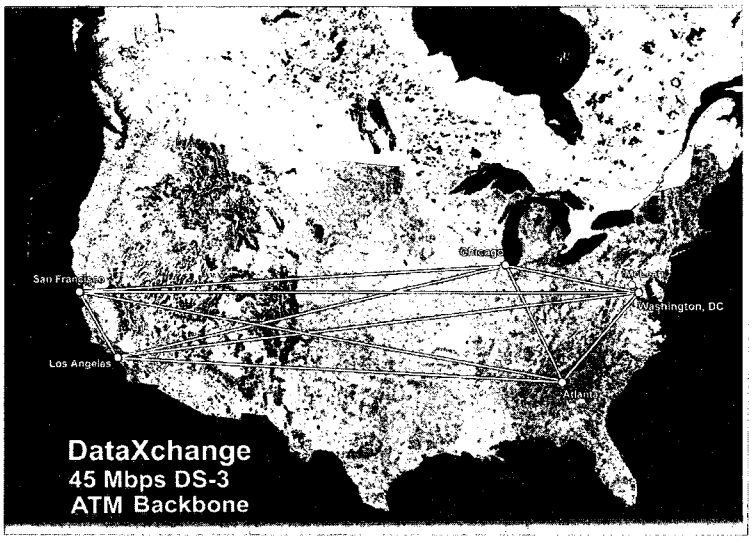
[Source: http://www.rnp.br/en/backbone/index.php]

3) ***Brazil RNP***

Nodes: 11

Edges: 19

Node 1	Node 2	Total Link Capacity
1	2	30
1	3	20
2	3	40
2	4	8
2	5	40
2	6	20
2	7	12
2	8	6
2	9	50
2	10	40
2	11	28
3	4	6
3	5	36
3	6	14
3	7	8
3	8	6
3	9	20
3	10	24
3	11	16



[Source: http://www.nthelp.com/maps.htm]

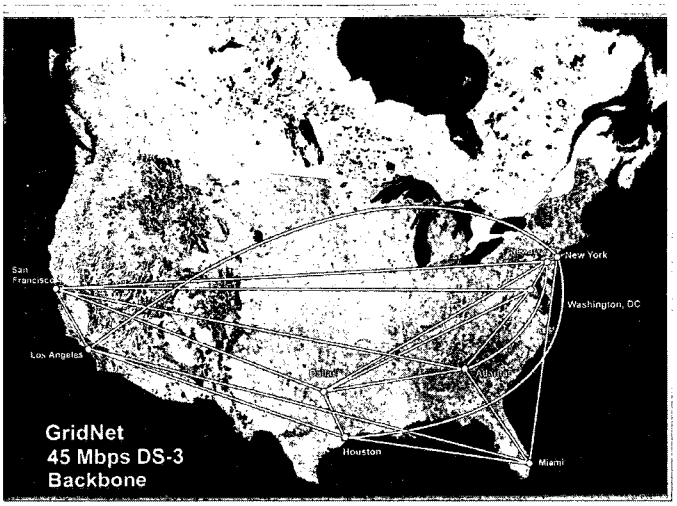
4) ***DataXchange***

Nodes: 8

Edges: 24

Node 1	Node 2	Total Link Capacity
1	2	2
1	3	6
1	4	6
1	5	6
1	6	6
1	7	6
2	3	6
2	4	6
2	5	6
2	6	6
2	7	6
3	4	6
3	5	6
3	6	6
3	7	6
4	5	6
4	6	6
4	7	6
5	6	6
5	7	6
5	8	6
6	7	6
6	8	6
7	8	6

5) ***GridNet***



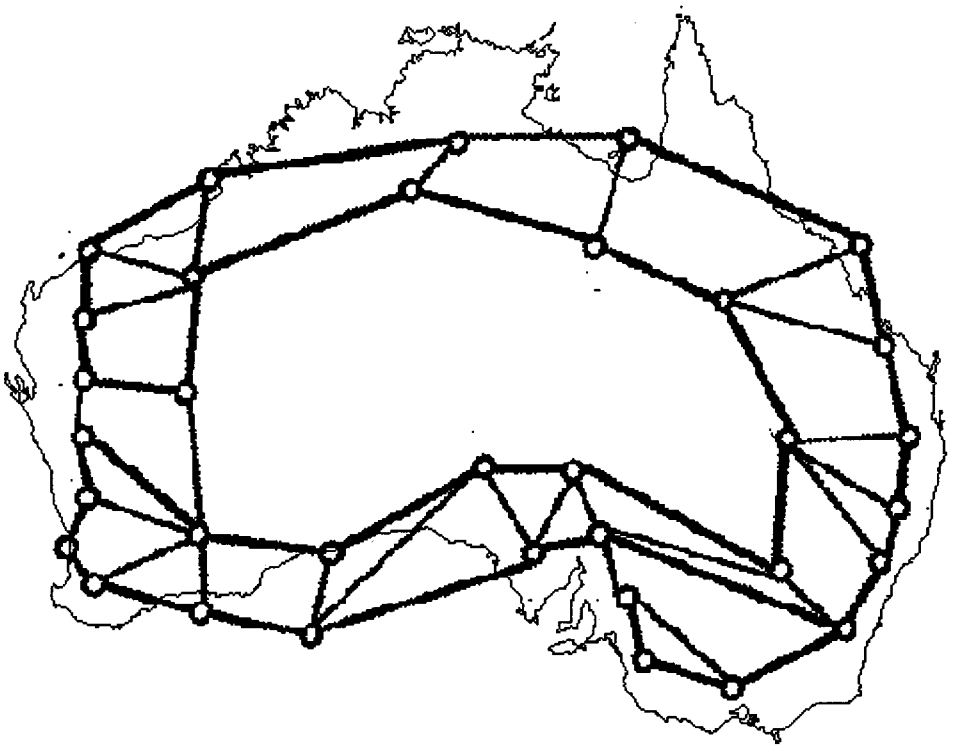
[Source: http://www.nthelp.com/maps.htm]

Nodes: 10

Edges: 25

Node 1	Node 2	Total Link Capacity
1	2	2
1	3	1
1	4	2
1	5	2
1	6	2
1	8	1
2	3	2
2	7	2
2	10	2
3	6	1
3	7	2
3	9	1
3	10	2
4	5	2
4	6	1
4	7	2
4	8	2
4	9	1
5	6	1
5	8	2
5	9	1
6	8	1
7	10	2
8	9	1
8	10	2

6) ***Australian Network***

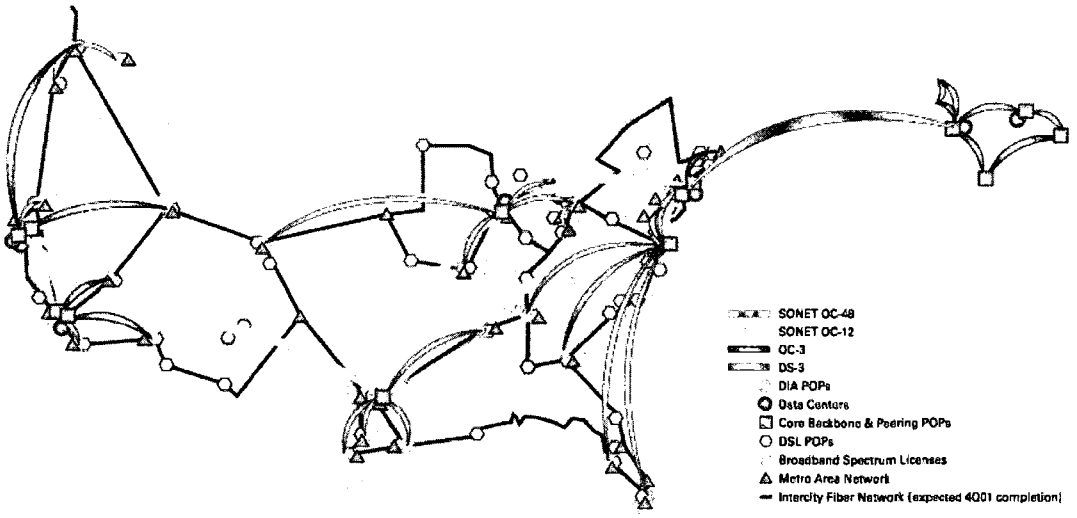


Nodes: 33

Edges: 50

Node 1	Node 2	Total Link Capacity
1	2	10
1	10	10
2	9	10
2	3	10
3	4	10
3	7	10
4	7	10
4	5	10
5	6	10
6	7	10
6	8	10
7	8	10
10	11	10
10	12	10
11	13	10
12	13	10
12	14	10
13	15	10
14	16	10
15	17	10

16	17	10
16	18	10
17	18	10
16	19	10
18	20	10
19	20	10
19	21	10
19	22	10
19	24	10
20	21	10
21	22	10
22	23	10
23	24	10
23	25	10
23	28	10
24	29	10
24	28	10
25	26	10
25	27	10
26	27	10
27	28	10
28	29	10
28	30	10
29	30	10
29	31	10
30	32	10
31	32	10
31	33	10
33	7	10
32	8	10



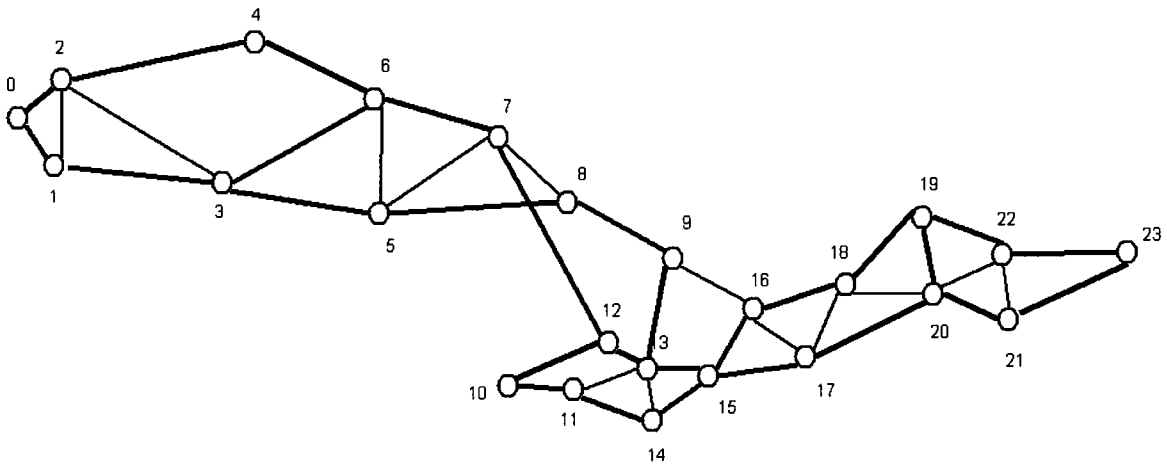
[Source: http://personalpages.manchester.ac.uk/staff/m.dodge/cybergeography/atlas]

7) ***XONet***

Nodes: 9

Edges: 14

Node 1	Node 2	Total Link Capacity
1	2	24
1	3	434
1	4	386
1	5	386
2	3	36
3	6	386
4	7	386
4	8	386
4	9	193
5	6	386
5	7	386
6	7	386
6	8	386
7	9	386



8) CanadaNET

Nodes: 24

Edges: 41

Node 1	Node 2	Total Link Capacity
1	2	10
1	3	10
2	3	10
2	4	10
3	5	10
3	6	10
4	6	10
5	6	10
5	7	10
5	8	10
6	7	10
7	8	10
7	12	10
8	9	10
9	13	10
9	16	10
10	11	10
10	12	10
11	13	10
11	14	10
12	13	10
13	14	10
13	15	10
14	15	10
15	16	10
15	17	10

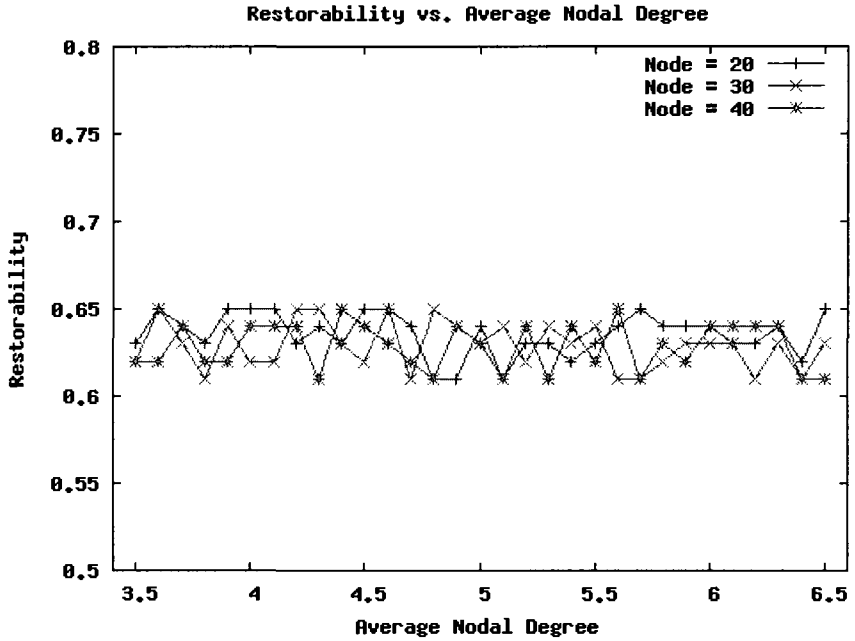
16	17	10
16	18	10
17	18	10
17	20	10
18	19	10
18	20	10
19	20	10
19	22	10
20	21	10
20	22	10
21	22	10
21	23	10
22	23	10
24	1	10
24	2	10

Appendix B: Waxman Random Networks

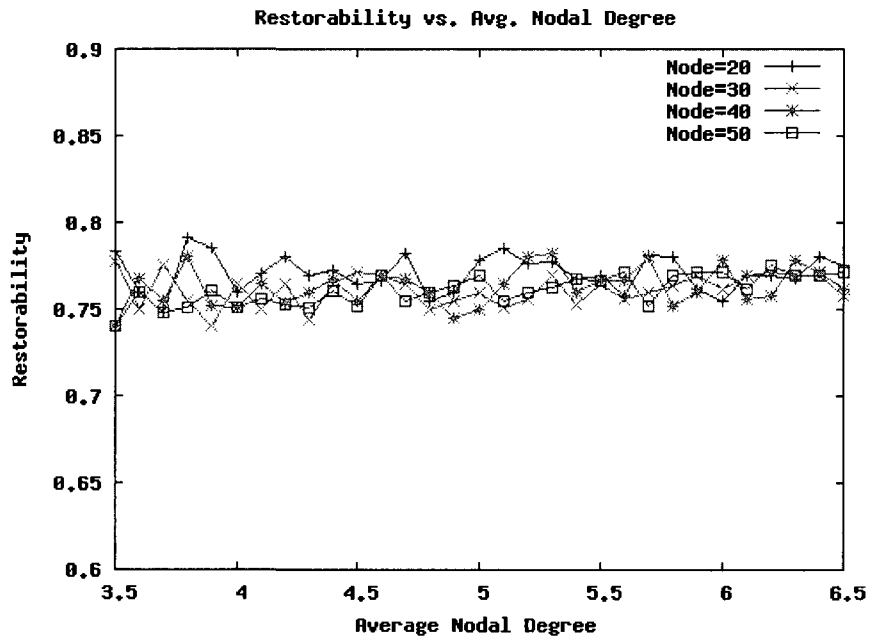
This appendix provides comparison of the performances of the protection algorithms for networks generated by the Waxman model and networks generated by our algorithm proposed in Section 2.3.2, as per comments from the examiner. We show the comparison results for only the CIDA and the MST2SP algorithms. Each data point in all the plots in this appendix represents the average data for 10 randomly generated networks using the respective algorithm.

B.1 CIDA algorithm

B.1.1 Restorability



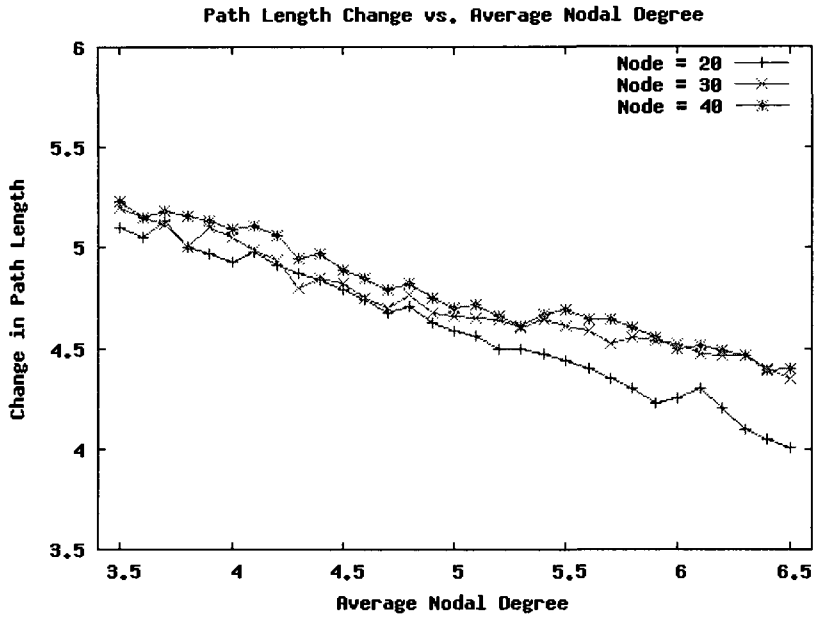
(a)



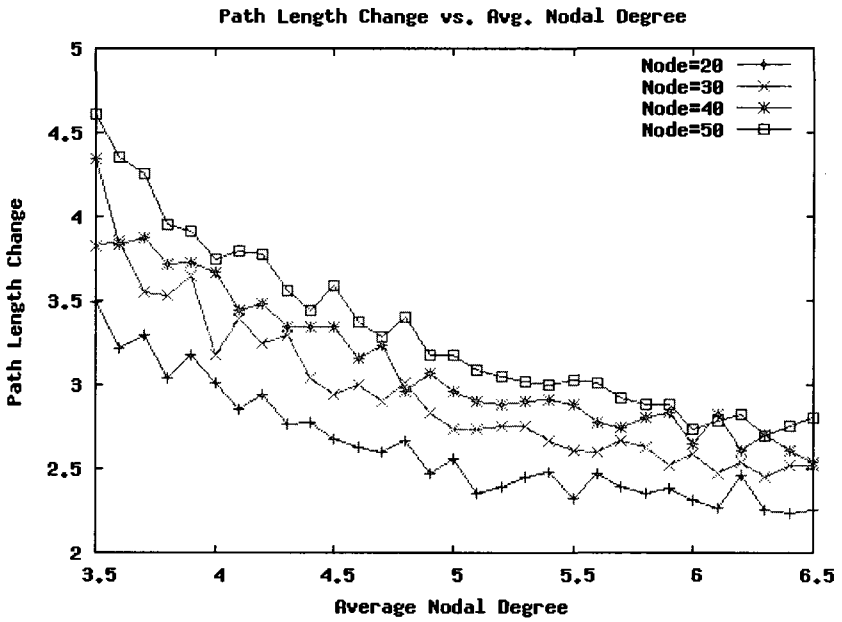
(b)

Figure B.1: Effect of Network order and density on Restorability for CIDA algorithm (a) Waxman algorithm (b) Proposed random network generator

B.1.2 Change in Path Length



(a)

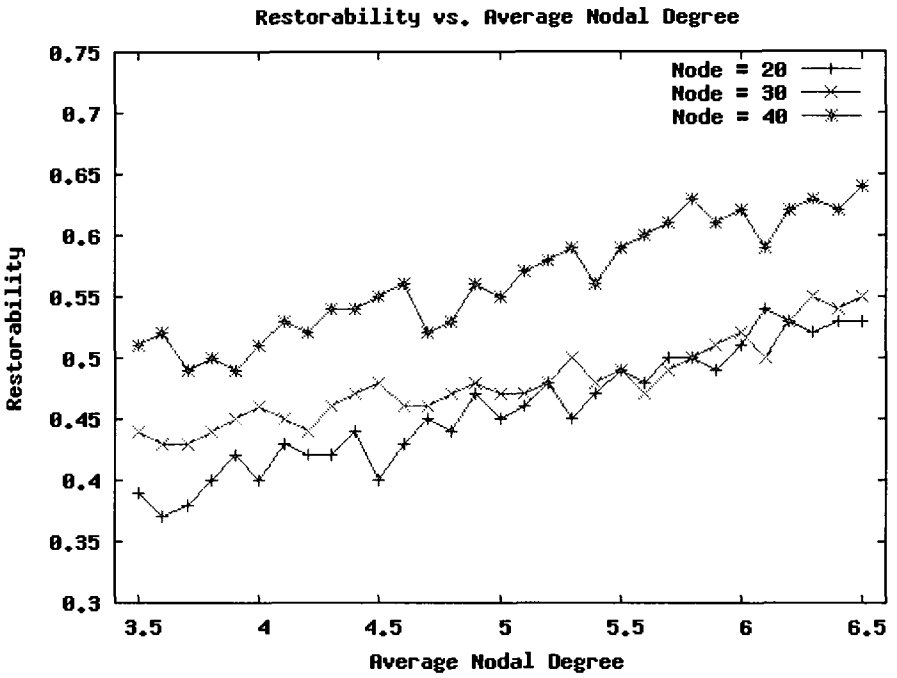


(b)

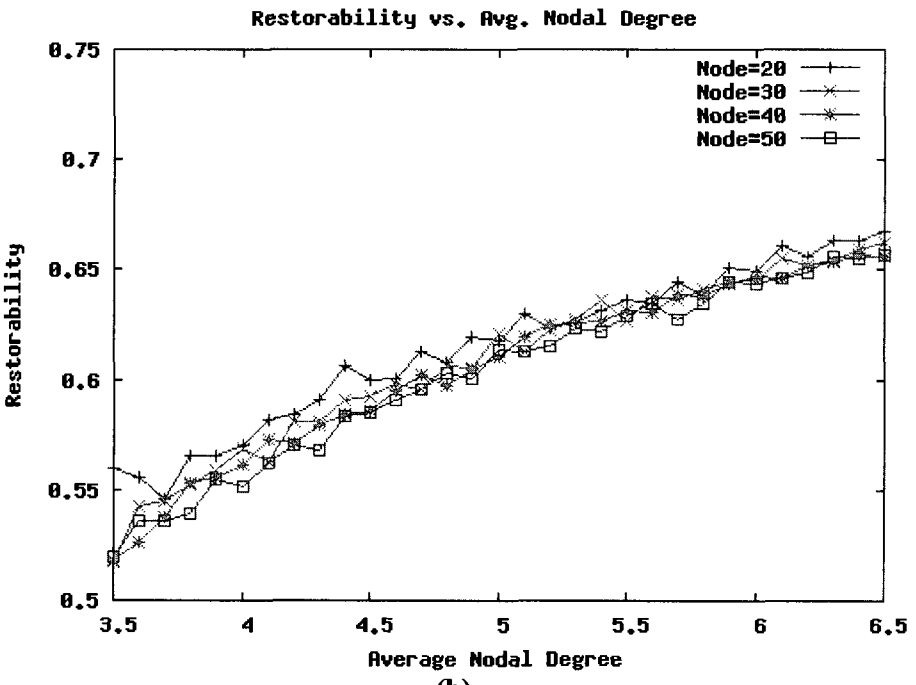
Figure B.2: Effect of Network order and density on Average Change in Path Length for CIDA algorithm
(a) Waxman algorithm (b) Proposed random network generator

B.2 MST2SP Algorithm

B.2.1 Restorability



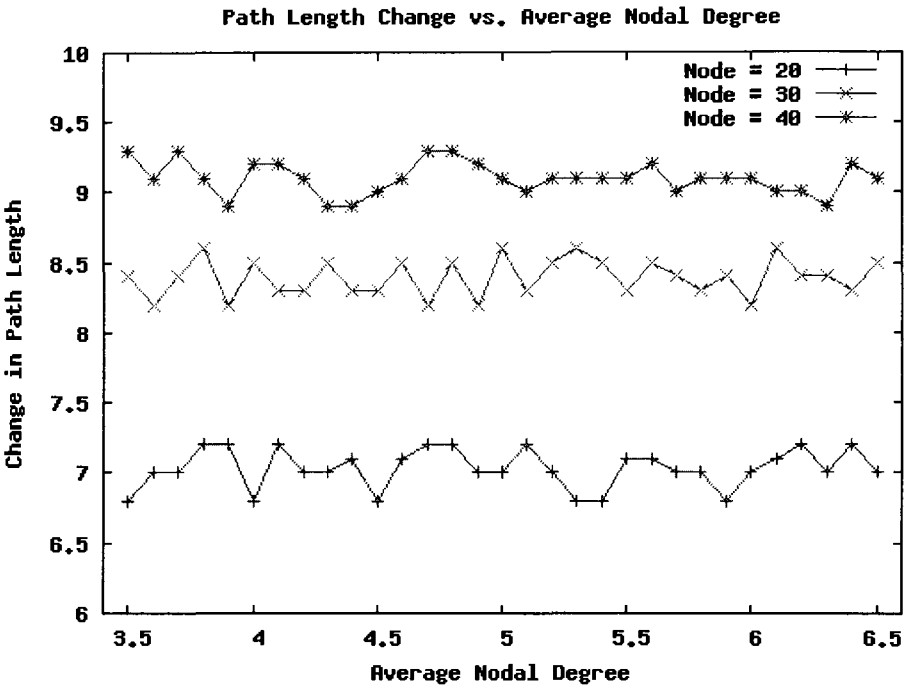
(a)



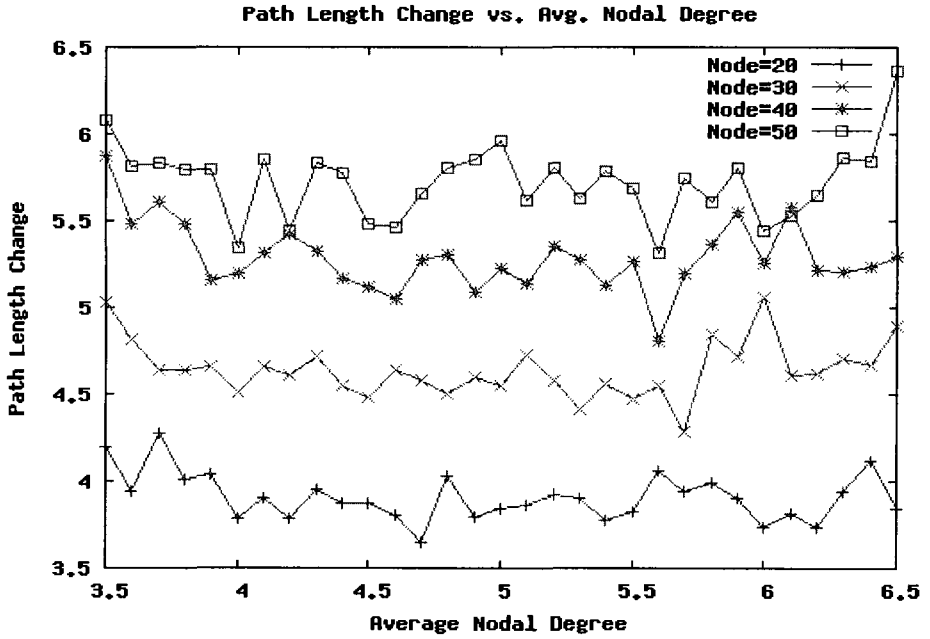
(b)

Figure B.3: Effect of Network order and density on Restorability for MST2SP algorithm (a) Waxman algorithm (b) Proposed random network generator

B.2.2 Change in Path Lengths



(a)



(b)

Figure B.4: Effect of Network order and density on Average Change in Path Length for MST2SP algorithm
(a) Waxman algorithm (b) Proposed random network generator