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

Dong He

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)

GRADE / DEGREE

Systems Science

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Efficient Cycle Algorithms for Capacitated Optical Network Design

TITRE DE LA THÈSE / TITLE OF THESIS

Oliver Yang

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

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

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Jianping Yao

Jiying Zhao

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

# Efficient Cycle Algorithms for Capacitated Optical Network Design


By

**Dong He**


A Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirement for the degree of

**Master of Science**


Systems Science Program
Faculty of Administration
University of Ottawa


September 6, 2005

*Your file Votre référence*
*ISBN: 0-494-11288-3*
*Our file Notre référence*
*ISBN: 0-494-11288-3*

# Abstract

This thesis studies the cycle-based protection schemes for DWDM (Dense Wavelength Division Multiplexing) networks. We first present the *MSCC* (Min-Sum Cycle Cover) Algorithm for simultaneous detection and protection using the Eulerian graph theory, and then analyze the performance of the algorithm through making comparison of it with other algorithms.

By extending the features of algorithm *SLA* (Straddling Link Algorithm) and initial ideas for *p*-cycle candidate generation (operations called *Add* and *Join*), we formulate more advanced cycle-generation algorithms: *SP-Add, SP-Join* and *SP-Merge*. We then use the developed heuristic spare capacity planning algorithms based on weighted or unity capacity efficiency of *p*-cycles, *WCIDA,* to test the fully restorable *p*-cycle network design. In addition, two capacitated algorithms will be presented and compared with each other.

# Acknowledgements

I am deeply grateful to my supervisor, Dr. Oliver W. W. Yang, for his consistently knowledgeable guidance, and his persistent but helpful comments and suggestions for improvements to this work. I would also like to thank Dr. Wayne Grover, who contributes to the guidance of my research in Chapter 4.

I would also like to thank everyone from the CCNR (Computer Communication Network Research) Lab for their cooperation and for always being available to help during these years of my studies. I also appreciated the helpful working environment.

Finally, I would like to thank my husband, Antai Ning, and my daughter, Yu Ning, who keep on supporting and showing their understanding during my research.

# Content

# List of Figures

# List of Tables

# Table of Acronyms and Abbreviations

| | | Section of 1st Appearance |
|---|---|---|
| ADM | Add Drop Multiplexer | 1.1 |
| AON | All Optical Netrwork | 1.5 |
| APS | Automatic Protection switching | 1.2.2.3 |
| ATM | Asynchronous Transfer Mode | 1.1 |
| BLSR | Bi-directional Link-protection Switch Ring | 1.2.2.3 |
| DCS | Digital Cross-connect System | 1.2.2.3 |
| DWDM | Dense Wavelength Division Multiplexing | 1.1 |
| JCA | Joint Capacity Assignment | 1.4.3 |
| KSP | Kth Shortest Path | 1.7 |
| LP | Linear Programming | 1.2.2.2 |
| ILP | Integer Linear Programming | 1.4.2 |
| MPLS | Multi Protocol Label Switching | 1.1 |
| MSCC | Min-Sum Cycle Cover | 1.4.4 |
| OA(&)M | Operation, Administration and Maintenance | 1.2.2.4 |
| OEO | Optical-Electrical-Optical | 1.5.3 |
| OSPF | Open Shortest Path First | 1.2.2.5 |
| *p*-Cycle | Pre-configured Cycle | 1.2.2.2 |
| *p*-Tree | Pre-configured Tree | 1.2.2.2 |
| QoS | Quality of Service | 1.1 |
| RWA | Routing and Wavelength Assignment | 1.2.2.2 |
| SCA | Separate Capacity Assignment | 1.4.3 |
| SCP | *p*-Cycle Spare Capacity Placement | 4.5.3 |
| SHR | Self-Healing Ring | 1.2.2.3 |
| SLA | Straddling Link Algorithm | 5.1 |
| SONET | Synchronous Optical Network | 1.1 |
| SP | Shortest Path | 4.3 |
| TDM | Time Division Multiplexing | 1.1 |
| UPSR | Unidirectional Link-protection Switch Ring | 1.2.2.3 |
| VCI | Virtual Circuit Identifier | 1.2.2.4 |
| VPI | Virtual Path Identifier | 1.2.2.4 |
| WCIDA | Working Capacity based Iterative *p*-cycle network Design Algorithm | 5.2 |
| WDM | Wavelength Division Multiplexing | 1.1 |

# Table of Notations and Symbols

| | | Section of first Appearance |
|---|---|---|
| A | Adjacency Matrix | 3.1.2 |
| $a_{xy}$ | *link adjacency* | 3.1.2 |
| *AE(p)* | a priori efficiency metric of *p*-cycle *p* | 2.3 |
| *d(x,y)* | distance from node-*x* to node-*y* | 3.2.1 |
| *d or d(x)* | nodal degree for node -*x* | 2.3 |
| $E_w$ | weighted capacity efficiency | 5.1 |
| $E_u$ | unity capacity efficiency | 5.1 |
| *(G, w)* | a weighted graph with weight function *w* | 3.1.2 |
| *G(**N**,**S**)* | a network with node set ***N*** and span set ***S*** | 2.3 |
| *L* | total number of effective links in a network | 3.3.2 |
| *M* | *a* matching in a graph | 3.1 |
| *n* | number of nodes in a network | 1.4.4 |
| *P(G)* | *p*-cycle candidate set for network *G* | 5.1 |
| $\boldsymbol{R}_0^+$ | the non-negative real number set | 3.2.1 |
| *R* | total number of links in a network | 3.4.1 |
| $R_{link}$ | relative costs for fault detection | 3.3.2 |
| *w* | weight function *S* -> $\boldsymbol{R}_0^+$ | 3.1.2 |
| $w_{ij}$ | weight assigning between the node i and node j | 3.1.2 |
| *s* | number of spans in a network | 1.4.4 |
| $s_j$ | spare capacity on the span j | 5.1 |
| *S(e)* | the spare capacity in span *e* | 2.3 |
| *S(G)* | spare capacity in network G | 5.1 |
| *S(e)* | spare capacity in span *e* | 2.3 |
| *T* | the number of required transcievers for monitoring | 3.3.2 |
| $w_j$ | working capacity on the span j | 5.1 |
| *W(G)* | working capacity set in network *G* | 5.1 |
| *Weight(M)* | weight of the matching *M* | 3.2 |
| *Z* | the integer number set | 3.1.2 |

# Chapter 1
# Introduction

## 1.1 Developments in DWDM and DWDM Networking

Dense Wavelength Division Multiplexing (DWDM) is the process of multiplexing signals of different wavelengths onto a single fiber. The Internet and all its applications use Internet Protocol (IP) packets, making IP the dominant form of all traffic. A few years ago, the only way to send IP packets over a DWDM fiber was to connect the IP routers to ATM switches and then send ATM cells over SONET devices that were connected to a DWDM transport system. This resulted in the five-layer protocol architecture (see Fig 1.1). ATM switches were required for multi-service integration (integrating voice and data). In addition, routers were usually limited in speed compared with ATM switches. SONET was required for aggregation – combining 155Mb/s ATM streams to OC-48 SONET streams – and also for protection. Each layer is designed for a particular function.

- IP – for applications and LAN environments;
- ATM – for virtual circuit/virtual path capacity engineering, flow control, performance monitoring, virtual networking and "QoS guarantees" in the data networking layer;
- SONET – for high quality transport of payload over the physical fiber medium, error monitoring, "OA&M", TDM synchronization and protection switching;
- DWDM – for increasing capacity, that is, effectively multiplying the capacity of the fibers currently in the ground.

Over the years, IP routers have become significantly faster. With the introduction of quality of service (QoS) in IP the need for ATMs was reduced. Beginning in 1996, packet over SONET or IP over "PPP" over SONET started becoming more popular. In 1999, several router manufacturers announced fast OC-192 interfaces and therefore, the need for traffic aggregation using SONET add-drop multiplexers (ADMs) was questioned [JaDh01].

Routers with SONET interfaces that can fill an entire wavelength have started appearing. The protection and restoration function that is provided by SONET ADMs can

be subdivided between IP and DWDM equipment. In 2000, Ethernet framing seemed to be gaining a foothold with the evolution of 10 Gigabit Ethernet. Some are predicting that eventually, SONET will not be needed and Ethernet will be running end-to-end. Regardless of what data link layer framing (SONET/PPP/Ethernet) is used, the reduced architecture is called IP over DWDM. Thus, IP and DWDM are the only two layers that are required.

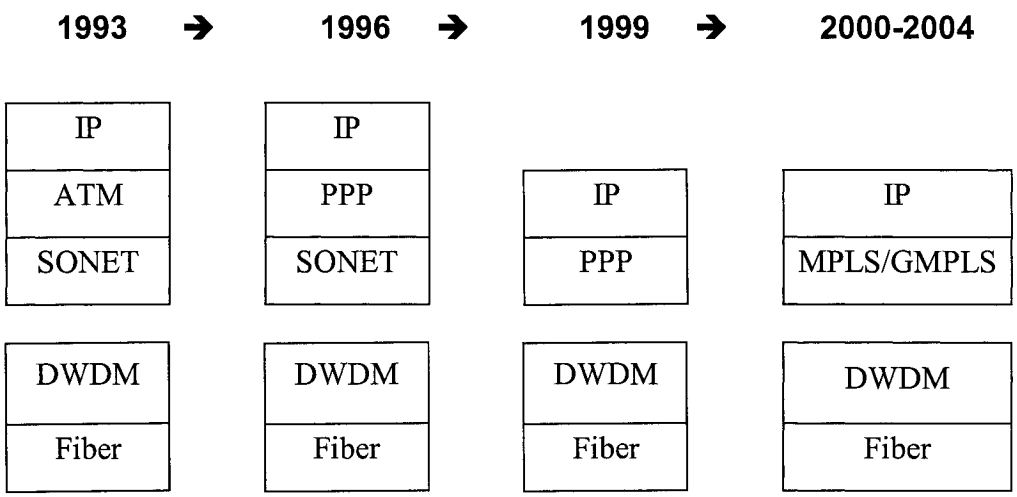



**Figure 1.1: Evolution of Layer Merging/Modifications**

IP is used as the generator for data traffic that provides multiplexing, routing, traffic engineering, and restoration mechanisms. DWDM on the other hand, is used as an inexpensive bandwidth facilitator. By eliminating the more costly SONET and ATM equipment, the IP/DWDM combination actually captures all of the required features for end-to-end application support. In such a combination, a coordinated effort should still be implemented for restoration and for path determination mechanisms in the IP and optical layers. (See Figure 1.1)

## 1.2 Survivability in Optical Networks

In this section, we shall discus the causes and impact of transport network failures, then survivability in optical networks in different layers.

### 1.2.1 Transport Network Failures and Their Impact [ToNe94, VePo02, Grov97]

There are many reasons for the failure of a transport network. One frequent occurrence is cable cut. Although it is estimated that any given mile of cable will operate for about 228 years before it is damaged [ToNe94], studies also indicate that, on average, more than one cut occurs each

day for every 100,000 miles of installed cable. According to the FCC, in 2002, metro networks annually experienced 13 cuts for every 1,000 miles of fiber, and long haul networks experienced 3 cuts for 1,000 miles of fiber [VePo02].

Typically, a cable cut causes the immediate loss of 100,000 or more telephone calls and all data connections and other services that are in progress. Indirect effects may include overloads on distant switching machines, the loss of 911 services, the loss of credit card verification service, etc. The preceding explains why network survivability design is very important to the telecommunications industry.

### 1.2.2 Network Survivability by Layers

*Survivability* is the ability of a network to continue providing service if a network failure occurs, i.e. a network may be described as survivable if it is possible to construct a path between any two-end nodes in such a way that the path will always remain connected. There are different protection methods for the different layers.

#### 1.2.2.1 Survivability on the Physical Layer [Grov03]

The physical layer, sometimes called Layer 1, is the infrastructure of physical resources on which the network is based: buildings, rights-of-way, cable ducts, cables, underground vaults, and so on. In this layer, survivability considerations are primarily aimed at physical protection of signal-bearing assets and ensuring that the physical layer topology has a basic spatial diversity so as to enable higher layer survivability techniques to function.

#### 1.2.2.2 Survivability on DWDM Layers [MaLe02, PuKu02, Mapa02]

In DWDM layers, different logical topologies can be built up so that there is spare capacity in the fibers on the top of different physical networks, such as a single logical ring, logical ring/cycle covers, *p*-cycles, and *p*-trees. All of these will be discussed in depth in Section 1.4. Routing and Wavelength Assignment (RWA) methods can also be used to assign an end-to-end traffic demand and are related to the protection path in DWDM networks in which a survivable network can be obtained (there are numerous references to this in the literature such as [AsSh01], [ShBo00]). This can be modeled as an LP (Linear Programming) problem, which can obtain an optimal solution, except for large networks, where it is time consuming and not feasible in practice. In [GrSh03], the *p*-cycle concept is extended for path-segment protection.

#### 1.2.2.3 Survivability on SONET Layers [StBa99, Wula90, WuTs95]

SONET network architecture is a well-developed and widely used protection mechanism. For both point-to-point and SHR (Self-Healing Ring) systems, Automatic Protection Switching is used, enabling the network to perform failure restoration in milliseconds (50 ms to detect the failure and to complete the switching process).

APS (Automatic Protection Switching) is used to improve the reliability and the availability performance of SONET transport systems by switching to standby equipment when failures occur. SONET linear APS is defined for (1+1), (1:1), or (1:N) protection architectures in point-to-point systems.

Two typical ring protection schemes are 2-fiber Unidirectional Path Switched Ring (UPSR) and 2 or 4 fiber Bi-directional Line Switched Ring (BLSR). In UPSR, the traffic is only routed in one direction (usually clockwise) unless failure occurs. No communication is required between two end nodes. The protection switching time is less than 50 ms, which is not affected by the number of nodes in the ring. A UPSR only requires two fibers but requests 100% redundancy, which is not efficient.

Unlike UPSR, BLSR can be constructed with either a 2-fiber or a 4-fiber ring. A 2-fiber BLSR is similar to a UPSR, except that traffic is routed in both directions around the ring, which is good for load balancing. In this way the deployed equipment and fiber resources are best utilized. BLSR reroutes the traffic by looping the entire working line signal back onto the protection fiber at both nodes adjacent to a failure [Grov03]. So signaling is required to coordinate at both ends of the failure. The protection switching time is about 50 ms but is restricted to maximum 16 nodes. With BLSR, the protection capacity has to be equal to the working capacity to achieve 100% restorability. In a 4-fiber BLSR, a separate pair of bi-directional fibers is used for the working and the protection path. An advantage of BLSRs over UPSRs is that the channels can be reused around the ring and the protection bandwidth is shared throughout the working span [Grov03].

If the networks have mesh topologies, we can use a ring/cycle cover [MoGr97] for survivability. There also exist DCS (Digital Cross-connect System)-based centralized and DCS-based distributed restoration techniques.

#### 1.2.2.4 Survivability on ATM Layers [AnDo94, SlLa00]

Most of the existing ATM restoration schemes focus on permanent virtual connections. For each permanent virtual connection established, identifiers of the pre-planned backup virtual connections need to be assigned during the connection establishment even though the bandwidth does not need to be reserved.

Further, standard new protocols using OAM cells are needed to accommodate the message passing required by these restoration schemes. In an environment where virtual connections are established on demand, one of the concerns associated with the existing approaches is the potential for VPI/VCI exhaustion that may cause inefficiency due to unusable bandwidth in the network. The slow process of standardization that has prevented the timely implementation of these schemes is another concern. The restoration methods we could use in ATM networks include local rerouting, source rerouting and local-destination rerouting which are similar to link restoration and path restoration.

#### 1.2.2.5 Survivability on IP Layers [StGr99, YeDi00, RaLu00, ThSo01, LaBo02]

Traditionally, when a path fails in IP layers, there are mature routing protocols and algorithms that would find an appropriate alternate route to replace the failed path. However, all these take a minute or more to accomplish the task. The newest protection/restoration method in IP layers is the IP-based pre-configured cycle (*p*-cycle) that can be viewed as the combination of IP protocols and *p*-cycle topologies [StGr99].

*p*-Cycles are implemented as closed-loop virtual circuits. In the event of failure, packets that would normally have been disrupted are encapsulated and immediately diverted by an alternate routing table entry onto a *p*-cycle. They travel through the *p*-cycle until the failure has been cleared. The normal re-routing protocol, such as OSPF, is also triggered, developing a longer-term global update routing table network-wide. The *p*-cycle provides an immediate detour for the packets, and prevents their loss until conventional global routing re-convergence occurs. Thus, the affected traffic will be re-routed around the failure more rapidly than could be accomplished using normal IP protocols.

#### 1.2.2.6 Survivability on MPLS (Multi Protocol Label Switching) Layers [YeDi00]

Since MPLS only uses fixed length labels matched to an index of interfaces on a router, the path routing complexity is significantly simplified. Routers cache all labels from all their neighbors,

and in the event of a path failure, the next available cached label can be retrieved quickly to provide an alternate path.

MPLS protection switching refers to the MPLS layers' ability to conduct a quick and complete restoration of traffic during any changes in the status of the MPLS layer. According to how the protection entities are set up to restore the working traffic upon failure, there is either a dynamic or a pre-negotiated protection. According to how the repairs are affected upon the occurrence of a failure on the working path, there is either an end-to-end (global, centralized) or a local (distributed) repair. Also, according to whether or not the traffic is switched back from the protection path to the working path (once the working path is repaired or restored), there is either a revertible or a non-revertible mode. Finally, according to the relationship between the active/working paths and backup paths, one can have 1+1, 1:1, 1:N protection schemes.

Network protection/restoration is such an important issue in communication networks that this thesis is focused on network protection/restoration problems.

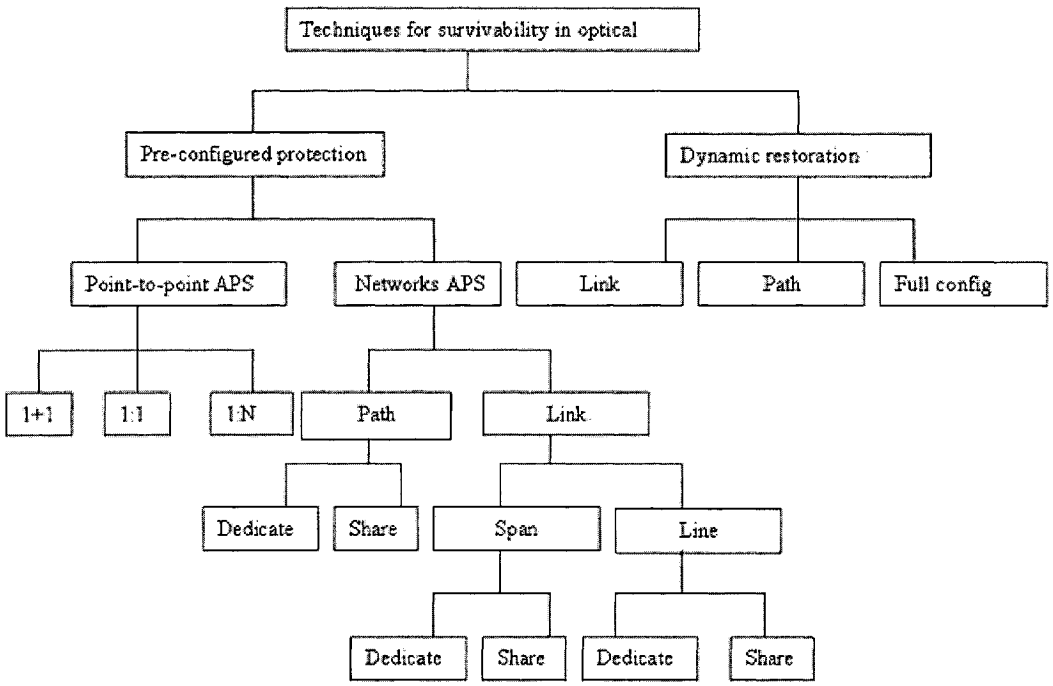



**Figure 1.2: Classification of Survivability**

## 1.3 Network Survivability Classification

Survivability techniques consist of restoration which is a dynamic operation and protection that is pre-planned (or pre-configured) [RaMu99a]. In this thesis, we mainly discuss the protection

techniques. In Figure 1.2, we categorize the protection schemes by point-to-point and end-to-end (for networks) ([RaMu99a], [RaMu99b], [MaPa02], etc).

There are 4 levels of protection at which various survivability methods can be employed in the context of WDM networks.: physical layer, transmission system layer, logical layer and service layer [T1A193]. Each layer has a generic type of demand unit that it provides to the next higher level.

### 1.3.1 Path Protection vs. Link Protection

In path protection/restoration, when a link fails, the node and the destination node of each connection that traverses the failed link are informed from the nodes adjacent to the failed link [RaMu99a]. The mechanism of finding the protection path and establishing the connection is based on shortest path calculation and end-to-end signaling.

Path protection can be further classified as dedicated path protection (1+1 or 1:1) and shared path protection (1:N or M: N). On the other hand, for link protection/restoration all the connections that traverse the failed link are rerouted around that link [RaMu99b]. In link protection, it may not be feasible to find a dedicated backup path around each link of the primary path. So link protection only considers shared-link protection schemes. The restoration path in both path and link protection schemes can be either pre-computed or dynamically discovered when a failure occurs.

### 1.3.2 Shared vs. Dedicated Bandwidth

When using dedicated bandwidth, the backup wavelength reserved on the links of the backup path are dedicated to that call, and are not shared with other backup paths, such as 1+1 automatic protection switching. On the other hand, two or more backup paths can use the shared bandwidth on links. One example is that in 1:N automatic protection switching, one unit of protection bandwidth is shared among N working channels. Shared bandwidth is more economic and efficient. Under acceptable signaling overhead, we prefer to have shared protection bandwidth.

### 1.3.3 Pre-configured vs. Dynamic

A backup path is configured before any failure occurs in the pre-configured method, but in the dynamic method, one has to find a backup path after any failure happens. Notice that the

dynamic method has the advantage of recovery of multiple failures that can occur in a short time if such an algorithm can be handled in real-time.

## 1.4 Topological Protection Methods

Topology is one of the two key factors for network protection/restoration, the other one is protection bandwidth planning. Topology is concerned with the layout of restoration paths, while protection bandwidth determines how much restoration traffic can go through each of the restoration paths. When talking about network topologies, one may refer to either physical or logical (virtual) topologies.

### 1.4.1 Physical Topologies in Network Survivability

There are three major types of physical topologies in common use for transport networks: ring, tree, and mesh. Since there are many singly-connected nodes (leaf nodes) in a simple physical tree, and one single physical span cut can completely isolate a sub-tree from the rest of the network, ring and mesh are the two common physical topologies used in survivable transport networks.

BLSR/4 and UPSR rings [WuLa90, MoGr99a] are examples of a physical ring network. Many of the long haul transport networks are becoming physical mesh topologies, such as the USA long haul network and the France Telecom network [MuKi97, Zhan02a]. These networks have an average nodal degree between 3 and 4, and are therefore considered to be sparse mesh networks. Mesh networks have higher bandwidth efficiency than ring networks, but are generally slower in restoration speed [GrSt98].

### 1.4.2 Logical Topologies in Network Protection

There are many logical topologies for network protection to choose from because, in theory, any logical topology can be overlaid on a physical mesh topology. Some common logical topologies are the following: single logical ring, logical ring/cycle cover, *p*-cycle, and *p*-tree.

#### 1.4.2.1 Single Logical Ring [Zhan01a]

BLSR/2 ring is one example of single logical ring networks. Only two fibers are deployed in BLSR/2, and in each fiber, the total bandwidth is cut into halves for working and protection. The logical operation is the same as BLSR/4.

#### 1.4.2.2 Logical Ring/Cycle Cover [MoGr99a]

In most transport networks of real interest, multiple rings/cycles are usually required for the following reasons: (1) the number of active nodes may exceed the limit for a single ring, (2) the volume of demand may be greater than the maximum ring/cycle size (capacity), and/or (3) the network topology does not have a single cycle that connects all of the nodes in the network.

The method of logical ring/cycle cover is to choose multiple logical rings to cover all the nodes and links in a mesh network. Each of the rings is configured as a self-healing ring based on either TDM or WDM. Thus, fast restoration in physical mesh networks can be achieved.

To find all possible rings/cycles in a network we usually use a depth first search algorithm. The problem is then to find which rings/cycles to include in the cover. Heuristics are usually used since the problem is NP-complete [GaHe94].

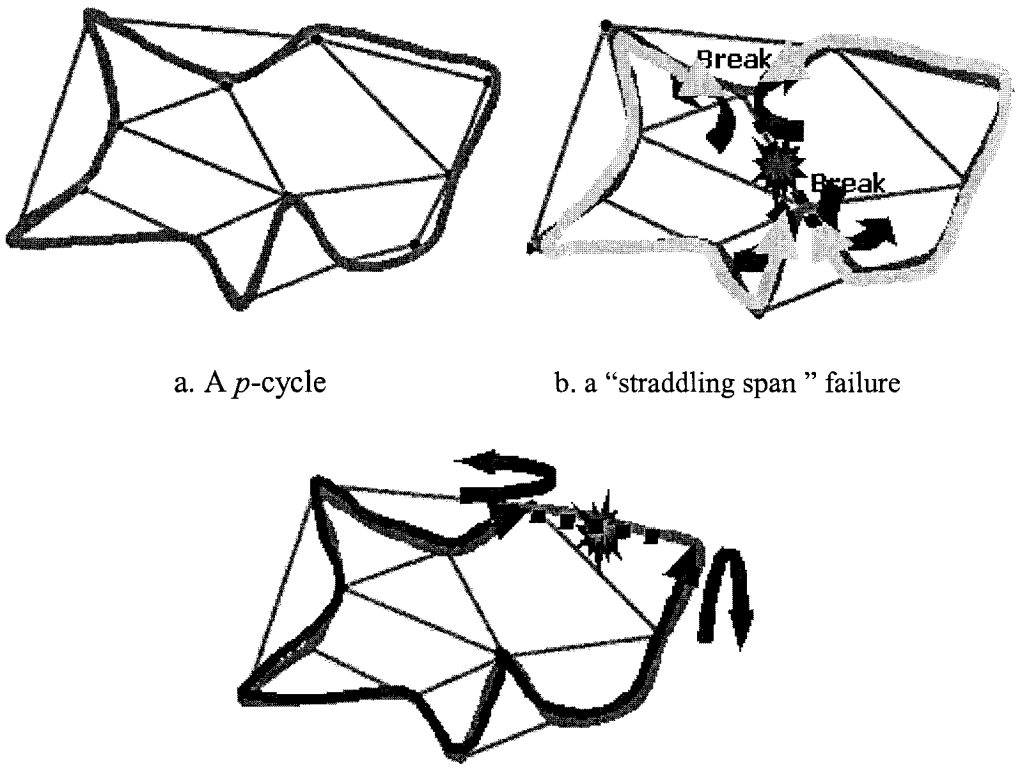
The algorithms for the ring/cycle finding include the classical algorithms, such as using the circuit vector space [MaDe76] [MaRe65] [HsHo72], backtracking algorithms [MaDe76], [John75], using the powers of adjacency matrix [MaDe76], and edge-digraph [MaDe76], etc. The heuristic algorithms, with the goal of minimizing costs, include ring node routing [Wase91], Eulerian graph decomposition [GaHe94], and K-shortest-path-based method [Zhan01a], etc.

After solving the ring/cycle finding problem, we have to determine the capacity allocation of each ring, the location of inter-ring transitions, etc. All of these are referred to as ring/cycle planning problems, which can be solved by using linear programming (LP) or heuristic-based algorithms [MoGr99a] [MoGr99b] [ArGr00]. In most cases the redundancy required is more than 100%.

#### 1.4.2.3 *p*-Tree [Zhan01b]

*p*-Tree (pre-configured protection tree) is formed by using spare capacity of a mesh network. Several tree-based algorithms have been proposed for pre-configured shared link protection in mesh networks [ShYa04] [ShYa01] [ZhYa02a] [LiYa03].

*p*-Tree can protect both on-tree links and non-tree links. Unlike *p*-cycle (see the next section), *p*-tree can have only one protection path per working link. There are several reasons for developing *p*-tree schemes: (1) to use in dynamic networks; (2) Trees are localized and scalable; (3) to build in a distributed manner.




a. A *p*-cycle b. a "straddling span " failure

c. an "on-cycle" failure

**Figure 1.3: Use of *p*-Cycles in Restoration**

### 1.4.2.4 *p*-Cycle

*p*-Cycle scheme was originally proposed for fault restoration in transport networks (WDM and SONET) [GrSt98]. It is found that the *p*-cycle scheme can achieve 100% restorability with little or no increase of the spare capacity in mesh-based restorable networks while maintaining the BLSR-like restoration speed, which is summed up by the notion of "ring like speed with mesh-like efficiency" [GrSt98].

Unlike ring and other cycle-based protection schemes, *p*-cycles protect not only on-cycle link but also straddling link. Figure 1.3 illustrates the way in which an individual *p*-cycle may be used for failure recovery. Figure 1.3a shows an example of *p*-cycle, Figure 1.3b shows a case where a link on the cycle fails, and *p*-cycle X provides one restoration path. Figure 1.3c shows a scenario where a straddling link fails, *p*-cycle X offers two restoration paths.

It has been shown [StGr00a, StGr00b] that *p*-cycles have the best performance of the restorability among simple trees and linear segments, closed cycles and arbitrary mixtures of all these patterns.

### 1.4.3 *p*-Cycle Design Issues

Recently the design of *p*-cycle based network is mainly focused on two open and inter-related issues [GrDo02]. The first is the requirement to reduce the complexity of solving optimal *p*-cycle design problems, making it practical to continually re-optimize a *p*-cycle based network in service, adapting to changing demand patterns. The second is to study how to increases the efficiency of a *p*-cycle network with joint optimization of the working path routes and *p*-cycle placement. Based on this consideration, we can group these issues into two kinds of *p*-cycle design in a capacitated network: the "joint" and the "non-joint" designs as follows

#### 1.4.3.1 Non-joint Design

The "non-joint" design can also be considered as spare capacity only design, of which the optimized capacity design is known as the Separate Capacity Assignment (SCA). In non-joint design, the route for the working path is first determined independently. This is followed by minimization of the spare capacity to support 100% restorability.

This design is reasonable in practice but may not be optimal from the resource utilization point of view. The input of it is the entire lightpath requirement, and the output is the *p*-cycles and spare capacity for the network. There are two approaches that can implement the non-joint design, one is heuristic, and the other one is optimal (SCA). The two steps for heuristic one are *p*-cycle pre-selection and heuristic spare capacity planning, which for SCA are *p*-cycle enumerating and ILP solution for spare capacity planning.

As mentioned before it is an NP-complete problem to find all possible cycles. Fortunately there is a practical heuristic algorithm called SLA (Straddling Link Algorithm) [ZhYa02] to pre-select the *p*-cycle candidate. This can be implemented by two calls of Dijkstra's algorithm for at most each span in the network, it is very scalable with network size $(O(S \cdot N \cdot \log N))$. For spare capacity planning, the two solutions can be found in [StGr00b, ZhZh04, and KaRe03] for the optimal solution, and in [GrDo02, ZhZh04, and DoHe03] for the heuristic.

#### 1.4.3.2 Joint Design

In this design, joint optimization of route choices (in each working design) and spare capacity assignment is required. It is also known as Jointly optimized Capacity Allocation (JCA) where the choice of route taken by each working demand is optimally coordinated with the spare

capacity assignment decisions so as to minimize total working and spare capacity. In short, with JCA, the working route assignments are chosen in conjunction with survivability considerations. The input of the design is the enumerating eligible working routes or the graph cycles (all possible cycles), while the output of it including not only the p-cycles and spare capacity distribution, but also the working capacity and routes. Basically there are extra capacity savings arise over SCA primarily from working-flow leveling effects, which is discussed in detail in Chapter 5 of [Grov03]. JCA is an "all-in-one" ILP problem and is too complicate to solve. An approach is proposed [GrDo02] to use ILP for *p*-cycle network design.

### 1.4.4 Graph Theory

Graph theory and algorithms are very important for the logical topology design. For example, the properties of Eulerian graphs can be used for failure detection/protection.

Every network shall be either Eulerian or non-Eulerian from the view of its topology. According to its property, non-Eulerian graph, which can be transformed to the Eulerian through augmenting paths in between the nodes with odd degree; Eulerian network can be decomposed into several smaller cycles (corresponding to the graph theory concept Eulerian circuits), which can be seen as the monitoring/protection cycles to restore the failed span. The application can be found in [GaHe94] for protection and in [ZeHu04] for failure monitoring. Related definitions and properties on Eulerian Graph can be found in many books on graph theory [Wils72, Temp81, etc].

Several graph algorithms will be used in the above design including: the SP (Shortest Path) algorithms (Dijkstra's Algorithm [Dijk59] and Floyd's Algorithm [Floy62]). In practice, Dijkstra's algorithm is suitable for most shortest-path finding purposes on networks, while Folyd's is the variation on Dijkstra's algorithm, the complexity of which is $O(N^3)$, but it is reported often to show $O(S)$ behavior when the average nodal degree is small and long paths tend to have more hops than shorter paths, so it is somewhat better than Dijkstra's if the network is decentralized.

Minimum Perfect Matching algorithm ([MoSh91], [CoLe90]) has been used in MSCC algorithm. There are two kinds of matching algorithms, the non-weighted and the weighted. For both cases, there are different algorithms for bipartite and non-bipartite [PaSt82, Wils72]. Depth First Search algorithm (DFS) will be mentioned when we discuss the method of generation of p-cycles [e.g. RaFa99].

## 1.5 Motivation

As discussed before, even a lambda channel failure has the most impact in the network. Therefore, monitoring and protecting/restoring the failure are imperative in the design of DWDM networks. However the techniques considered for transport monitoring and restorations are slow processes, and the methods have no direct regard for capacity congestion effects that may arise from the routing changes. If a lambda channel fails, it is important to monitor and protect or restore it in the first place. Therefore, we are required to address first on the network monitoring and protection/restoration issues in DWDM layer.

### 1.5.1 Need for a Heuristic Algorithm to Enumerate the *p*-Cycle Candidates

Both the ring cover and the *p*-cycle design can guarantee 100% restorability on physical layer for 2-connected networks, while a simple tree cannot in general (e.g. a physical tree can not provide a restoration path to any span which lies on the tree itself because such a failure creates disconnected residual patterns). Compared to ring cover, the significance of *p*-cycle is that it permits ring-like switching speeds and yet the efficient network capacity plans. According to [GrSt00], the optimized network capacity plans of *p*-cycles are virtually as efficient as a span-restorable network. Therefore we would like to study *p*-cycle protection approaches as well.

Previously reported Integer Linear Programming (ILP) methods for *p*-cycle network design require the enumeration of all simple cycles or a representative sample of the set of all cycles as a starting part. However the number of simple cycles in a graph grows exponentially with the number of nodes and edges in the graph. A suitably sized subset of cycles through "pre-selection" has proven effective in reducing the ILP runtime component. Therefore it is our next intention to do further reseach on this.

There have been few approaches to solving this NP-complete problem in a reasonable amount of time. Although SLA is an extremely simple and fast procedure to produce high-coverage initial designs for further use in network planning, the cycles produced are generally inefficient for the purposes of providing overall *p*-cycle network designs for capacitated restorable networks.

The main concern with this algorithm is that the primary *p*-cycles, as a result of duplication and/or overlap, often fail to share *p*-cycle spare capacity to cover more than one

straddling span each. To overcome these shortcomings, we would like to study various ideas of building more efficient *p*-cycles by merging simple cycles with existing *p*-cycles.

### 1.5.2 Need for a Heuristic Algorithm to Plan the Spare Capacity of *p*-Cycle

Although many Integer Linear Programming (ILP) methods for *p*-cycle network design have been presented, they are more suitable for small or medium-sized networks than large ones because of the complexity (NP hard) of the algorithms. Therefore we need to introduce heuristic *p*-cycle algorithm for capacitated network to reduce the complexity of the algorithm.

### 1.5.3 Need for both Failure Location Detection and Protection/Restoration

Roughly speaking, so far restoration/protection schemes have only considered how to protect a single failure once it has been detected without considering how to accurately identify and locate network failures. In transparent optical networks, faults may propagate to various parts of the network from the original site; as a result, multiple alarms can be generated for a single failure. In order to reduce the number of redundant alarms, also to simplify fault localization, as well as to lower financial investment in network monitoring equipment, failure monitor placement should be optimized for a given network.

All known failure detection mechanisms are invented for electronic/electrical network, and can hardly be applied directly to AONs due to the lack of electrical terminations in AONs [ZeHu04]. Even some of these mechanisms might have been deployed in optical networks with Optical-Electrical-Optical (OEO) conversion, they cannot be transplanted to AONs. Therefore new methodologies and mechanisms are necessary for failure detection in AONs. A monitoring cycle concept is proposed [ZeHu04] as a fault detection mechanism based on decomposing AONs into a set of cycles (a cycle cover) through utilizing independent wavelengths as supervisory channels, in which each cycle is defined as a monitoring cycle and one monitor will be assigned to every cycle of the topology network. It will be very interesting and necessary to find ways to reduce the number of monitors and while satisfying the requirements of detection and restoration. This may also help to simultaneously satisfy the requirements of detection and restoration, which has no research done so far.

## 1.6 Objectives

Our general objective is to obtain a feasible and efficient logical topology on the mesh physical

network from the distributions of working and unknown spare capacities in a practical mesh network. We would like to minimize the cost of spare capacity with 100% restorability for network protection/restoration. Specially, we are interested in:

1) Finding an efficient detection/protection approach using cycle cover algorithm;
2) Investigating the protection/restoration of the long haul DWDM networks by using *p*-cycle topologies.

Based on the discussion in the motivation, we need to find efficient detection/protection approaches using cycle-cover algorithm or *p*-cycle algorithm. Therefore our objectives include:

3) to find cycle sets to guarantee 100% covering all the spans of the networks;
4) to find the effective *p*-cycle candidate sets in order to reduce the complexity of computation;
5) to find heuristic algorithm to simplify the step of capacity planning;
6) to provide the corresponding protection approaches to restore the failure.

## 1.7 Methodologies and Approaches

In this thesis, we basically choose the hueristic algorithms instead of the optimatic ones because of the complexity of the latter (up to NP hard).

We first consider the failure detection algorithm, MSCC (Min-Sum Cycle Cover Algorithm), as a good application of Eulerian graph thoery. The MSCC is the simplest among all hueristic algorithms to provide a cycle cover for both detection and 100% protection. Due to the fact that no two cycles have a common edges in Eulerian Graph, the minimum number of monitoring wavelengths incident to each edge can be achieved, so we consider using the Eulerian graph theory on it. The main idea of the algorithm is how to find 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. We shall utilize the properties and basic algorihtms of Eulerian graph, using minimum perfect matching algorithm, Dijstra shortest path algorithm. We simulate our MSCC algorithm in four different networks, from which we can get better performance compared to the two algorithms in [ZeHu04].

Using the family of primary *p*-cycles generated by Straddling Link Algorithm (SLA) as the starting point, Dr. Grover proposes two merging operations [Grov03], *Add* and *Join*, as the basic idea for the *p*-cycle design after the work of SLA [ZhYa02]. The initial idea was to study operations that would transform the set of cycles from SLA into more efficient *p*-cycles and to

develop a fully restorable *p*-cycle network design algorithm based on this approach. The main benefit of such an algorithm is that it avoids any form of explicit cycle-set enumeration. Enlightened by the idea, we composed the heuristic operations to generate more candidates. They are *SP-Add*, *SP-Merge* and *SP-Join*.

When constructing the efficient *p*-cycle candidate set, we also use some graphical concepts and methods such as, the connectivity of a network, the link (node)-disjoint shortest path, KSP (k-Shortest Paths) algorithm, etc. While planning the capacity, we use the heuristic algorithm.

We take redundancy as the efficiency to evaluate the algorithms in the chapter 5, and restorability will be the parameter used to compare the different designs of *p*-cycle and tree in chapter 6.

We shall use the simulations to test our algorithms on the existing networks, such as the US long haul network and the France Telecom network. and we use ILOG CPLEX 7.0 [Ilog00] and AMPL [Ampl00] for ILP solution.

## 1.8 Contribution and Organization of the Thesis

The following are the contributions of this thesis:

1) The formulation of the MSCC algorithm based on the Eulerian graphical theories and performance analysis: We show that the MSCC algorithm can have better performance in terms of number of monitors and $R_{link}$ than the two algorithms in [ZeHu04];

2) The algorithms for merging *p*-cycles: *SP-Add, SP-Join*, and *Merging*: We demonstrate the simulation results of the operations in the terms of *AE(p)*, average cycle length, cycle size evaluation, complexity, and optimal evaluation and make comparisons among the operations;
3) The *WCIDA* heuristic spare capacity planning algorithms based on weighted or unity capacity efficiency of *p*-cycles: the latter one is our contribution. Make the comparison of two capacity planning algorithms.
4) Combining the *p*-cycle pre-selection operation with ILP planning algorithm to form the *Join-ILP* or *Merge-ILP* algorithms, and obtaining the better performance.

The remainder of the thesis is organized as follows. Chapter 2 will give the standards for network operations, models, and assumptions for later discussion. In Chapter 3, we will present

an algorithmic approach to solving detection and protection simultaneously using the Eulerian graph theory, and present a comparison and analysis of the performance. In Chapter 4, we contribute to the more advanced cycle generation algorithms: *SP-Add, SP-Join, SP-Merge.* In Chapter 5 we use the developed capacity-planning algorithm to test the fully restorable *p*-cycle network design. One capacitated algorithm will be presented and compared to other *WCIDA(x)* (Working Capacity based Iterative *p*-cycle network Design Algorithm). In Chapter 6, we provide the design guideline according to different considerations, such as objectives, assumptions and network properties. Lastly, conclusion and future work will be discussed in Chapter 7.

## 1.9 Publication

[DoHe03] J. Doucette, Donna He, W. D. Grover, and Oliver Yang, "Algorithmic Approaches for Efficient Enumeration of Candidate *p*-Cycles and Capacitated *p*-Cycle Network Design," *Proc. Workshop on Design of Reliable Communication Networks (DRCN 2003)*, Banff, AB, Canada, pp. 212-220, 19-22 October 2003.

# Chapter 2
# Network Operations, Models and Assumptions

We shall provide network operations, network model and assumptions, and give details of two different working capacity distribution models that will be used in later chapters.

## 2.1 Network Layout and Operation

We consider a general DWDM network with a general mesh topology in the thesis. The architecture we adopt is the most recent IP/DWDM combination (see the part from 2000 to 2004 in Figure 1.1). It is also important to make it clear that the cycle protection here is a logical scheme that can be implemented at the fiber layer and the WDM level. When we speak of "transport networks", we take the system and logical layers together.

A mesh network contains nodes to originate, route and terminate lightpaths, and spans to connect adjacent nodes and through which network traffic is carried. A link in our network graph is an individual capacity unit between adjacent nodes on which switching devices operate to inter-connect capacity. Each end-to-end traffic demand is considered to be directional and therefore served by a pair of uni-directional lightpaths which take on the same route so that it composes the bi-directional traffic between the source and the destination.

Due to the continuity requirement of a light path, conflicts may occur if two lightpaths share the same physical link. By using wavelength converters at intermediate nodes, our network model actually consider data flowing along semi-lightpaths, each of which formed by the concatenation of lightpath segments of various wavelength.

The wavelength routers at each node collect network layer information either from an in-band signaling channel or an out-of-band signaling channel. This is done both periodically and upon each change in topology/wavelength distribution. Therefore, each wavelength router has a local image of the network topology and the distribution of working/spare wavelengths on each link. The wavelength routers can automatically setup end-to-end semi-lightpaths once the source node, destination node, and bandwidth requirement are specified. Upon a physical link failure, the disrupted traffic is rerouted at the WDM layer, according to the local databases of the network topologies at related nodes.

We assume that full wavelength conversion is available on each WDM switch/router, whenever needed. In other words, we care only about wavelength availability in lightpath setup, protection switching or protection rerouting, instead of the wavelength continuity.

In the architecture of dynamic wavelength routing networks there are two key protocols: wavelength routing protocol and wavelength distribution protocol. With the wavelength routing protocol, any change in optical wavelength resource in each fiber link is flooded through the whole network, and trigger 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 lightpath.

These two protocols are derived from mature architectures (such as OSPF and LDP) and expected to be comparatively straight forward, yet they are still in prototyping stage and not commercially available. For all those discussions we assume that the wavelength routing and distribution protocols are up and running, such that each node knows the topology of the whole network as well as the wavelength resource on each link, and is able to request a lightpath to another node when needed.

### 2.1.1 Protection

All restorable mesh networks shall contain both working and spare links distributed among its

After 3 working links lost, 3 protection paths are found and the failed span is protected.

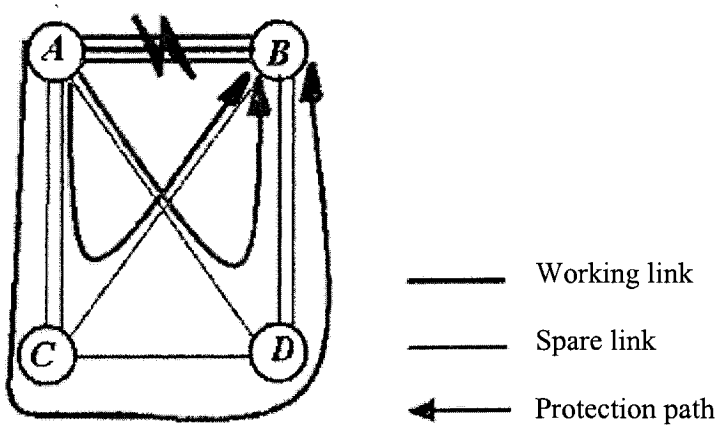



**Figure 2.1: Protection Path Set formed in Response to a Span Failure**

spans. The working links will be configured into working paths, which will carry the traffic between the network nodes. If there are no failures on the spans, the spare links are unconnected and remain in a stand-by state. In the event of a span failure, the lost traffic, originally flowing through the working links, will be rerouted onto protection paths formed from the spare links.

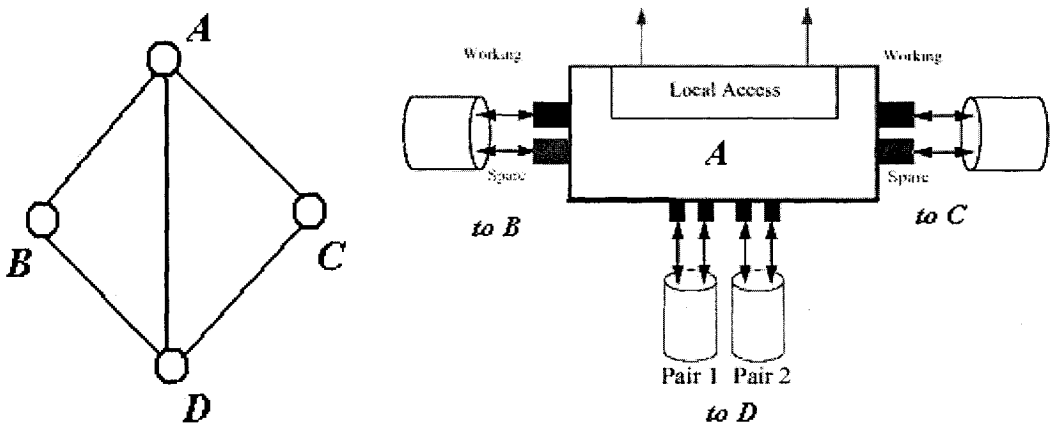
The presence of appropriately placed spare capacity distributed throughout the network can permit the protection of span failures.

Figure 2.1 is an example of a protection path set formed in response to a span failure using the network's distributed spare capacity. There is a span *AB* with three active working demand between *A* and *B*. If the span fails, protection switch will be done along three different paths, *A-C-B*, *A-D-B*, and *A-C-D-B*. Along every path, one spare link between the adjacent nodes will be taken to protect the failure.

## 2.2 Cycle Protection

We consider two types of cycle protections in DWDM layer for link protection: cycle cover and *p*-cycle. *p*-Cycle has been introduced and discussed in Section 1.4.2.4.

In cycle cover design, the entire network is divided into smaller cycles in such a way that each edge comes under at least one cycle. Basically traditional ring network planning is to find a low-cost cycle cover and to have other protection links along the cycle protecting the whole cycle. In this approach, every span may be covered by more than one cycle. Thus the problem for cycle cover is formulated in such a way to minimize the redundancy of protection links. In most cases the redundancy required is more than 100%.




a) *p*-cycle *ABCD* b) ADM-like node device

**Figure 2.2: WDM capacity-slice nodal device for *p*-cycle**

### 2.2.1 Node Device for Cycle Protection in DWDM layer

Figure 2.2 is a nodal device portrayal for a simple p-cycle ABCD, where the span AD is a straddler (a straddling span that is not a span on the p-cycle but its two end nodes are on the

cycle). Fig. 2.2b depicts the implementation of node A that functions like an ADM (Add/Drop Multiplexer). Span AB has two channels, one working and the other spare (for protection) interfaced optically to node A through a port (a port is an optical interface for the two wavelength channels here); likewise for span AC. As for span AD, there can be a number of working links (wavelength channels) routed into spans AB and AC, or terminated locally through a local access tributary add/drop interface. In the figure here, we are only showing four channels in two logical groups (two pairs of straddling links) interfaced to node A through two ports.

If failure occurs on any pair of straddling links (wavelength channels) along AD, it can be restored by switching the two wavelengths into the spares of spans AB and AC. In another word, each straddling port provides shared protection access through the two halves of the respective p-cycle. This is unlike a ring ADM which forms the cycle ACDB using only the working wavelength of spans AB and AC, because a straddling-span interface would carry additional working ports to node D.

In summary, if a node has a nodal degree d =2, only a traditional ADM is needed which is the same as the traditional SONET ring. If a node has a degree d>2, one would need ADM devices to support up to (d-2) physical straddlers.

## 2.3 Graph Models

We consider a general mesh DWDM network described by a graph $G(\boldsymbol{N}, \boldsymbol{S})$, where $\boldsymbol{N}$ represents the set of physical nodes (which are wavelength routers), and $\boldsymbol{S}$ represents the physical spans in the network. We use $W(e)$ to represent the working capacity in span $e$, and $S(e)$ to represent the spare capacity in span $e$.

A graph $G$ is *simple* if it has no parallel spans or self-loops. Two spans are *parallel* if they are incident on the same pair of end nodes, and a *self-loop* is a span that begins and ends on the same node. A graph is called a *multigraph* if it has more than one span in parallel between one or more pairs of vertices. A *complete* graph on all $N$ nodes is a graph where every node is connected by a unique span to all (n-1) other nodes.

A graph in which a number $w_{ij}$ is associate with every span $\{i, j\}$ is called a *weighted graph* and the number $w_{ij}$ is the *weight* of span $\{i, j\}$. In transport networks these weights often represent cost, distance, or capacity. A graph is called a *capacitated* graph if the span weights

represent capacities, and a problem is called *capacitated* if it requires capacity determination or solving a routing problem with assigned capacities.

We use the term *link* to denote an individual capacity unit between adjacent nodes on which switching devices operate to inter-connect capacity. Thus a *span* is a physical entity consisting of the set of all parallel working or/and spare logical links between adjacent connected nodes of the network graph. The so-called "homogeneous network" is thus a generally protecting networks in which all spans are assumed to have identical amounts of installed channel capacity. We will present other terminologies in the text as they arise.

In graph theory, a *walk* is any sequence of adjacent edges in a graph. A *walk (or a path)* is closed if its origin and destination nodes are the same. In general, a closed walk is called a *cycle* or *circuit*, and might figure-8 on itself and pass through the same nodes more than once. A closed simple walk is called a *simple* or *elemental cycle*.

A *cycle cover* is any decomposition of $G$ into elementary cycles such that each edge is present in at least one cycle. A *Hamiltonian cycle* is a cycle that connects all of the vertices in a graph, and passes through each only once. Hamiltonian cycles have a special relationship to $p$-cycle networks because a single Hamiltonian has a potential protection to every span of the graph.

The edges in a planar graph can divide the plane into $S\text{-}N+2$ non-overlapping regions called *faces*. Faces need not be produced explicitly. For example, during the cycle generating cycles in Chapter 4, we only need to find the shortest cycle (in term of the number of hops) for which it is an on-cycle span.[1]

*Average nodal degree* is the mean number of connected links in each node of a network. It shows the connectivity of a network. The *Mean (or Average) protection path length* is defined to be the average number of hops along the protection routes. It is very important in evaluating the performance of a protection scheme because it represents the restoration time for a network during a link failure. Usually, it is preferable to have a shorter path. Likewise, the *Average p-cycle length* (in hops) is defined to be the total number of hops of all $p$-cycles divided by the total number of $p$-cycle.

---

[1] While this will not necessarily produce the set of faces in all cases, it is equivalent, or nearly so, in most real transport networks. The purpose is only to enrich the candidate cycle set.

*Complexity* of an algorithm is defined to be number of steps in computation with respect to the number of nodes and links in a network. It is the direct measure used to evaluate whether the algorithm is feasible or not. Usually, a lower complexity means an algorithm can run faster. Naturally, we prefer to choose the protection scheme with a lower complexity to protect a network.

In later chapters, we also use the performance measures of efficiency metric, redundancy, and restorability that are defined as follows:

a) The *a priori efficiency metric AE (p)* measures the potential efficiency of using a particular cycle as a *p*-cycle. Let $S_{C,p}$ = # of on-cycle, $S_{S,p}$ = # of straddlers, $c_i$ = unit cost of span *i*. Then mathematically,

$$AE\ (p) = \frac{\sum_{\forall i \in X} X_{p,i}}{\sum_{\forall i \in X | X_{p,i}=1} c_i} = \frac{2 * |S_{S,p}| + |S_{C,p}|}{|S_{C,p}|}, \qquad (2.1)$$

where $X_{p,j}$ = 1 if span *j* is on cycle, $X_{p,j}$ = 2 if span *j* is a straddler,

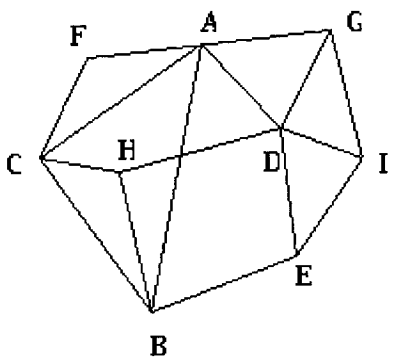



**Figure 2.3: One Example on *p*-cycle Operation**

Using Fig. 2.3 as an example, we know the *AE* of the closed path A-B-C-A is 1, because there are no straddlers for the cycle. If we hypothesize A-B-C-A as a primary cycle as obtained by SLA, we can perform the following operations: find the cycle-disjoint shortest path for the edge AB, which is A-D-E-B. After merging the path with the cycle, and removing the edge AB itself (AB is a straddler and not an on-cycle span), we can obtain another closed path, A-D-E-B-C-A, which has a better AE score: (1*2+5)/5 = 1.4.

We can use this operation along the primary cycle to get one more effective *p*-cycle, A-D-E-B-H-C-F-A, for which the AE is (3*2+7)/7 = 1.857. Therefore we can get the subset without enumerating the whole cycle set.

b) *Redundancy* of a network is a measure of architectural efficiency for a survivable transport network, and is defined as the pure ratio of spare to working channel counts. Mathematically,

$$Redundancy = = \frac{\sum_{i=1}^{S} s_i}{\sum_{i=1}^{S} w_i}, \qquad (2.2)$$

where $s_i$ is equal to the number of spare links present on span *i*, $w_i$ is the number of working links present on the span *i*, and *S* is the total number of spans.

c) *Restorability* of a network is a measure of survivability, and is defined as the pure ratio of protected working to working channel counts. Mathematically,

$$Restorability = \frac{\sum_{i=1}^{S} p_i}{\sum_{i=1}^{S} w_i}, \qquad (2.3)$$

where $p_i$ is equal to the number of protected working capacities on span *i*, $w_i$ is equal to the number of working capacities on the span *i*, and *S* is the total number of spans.

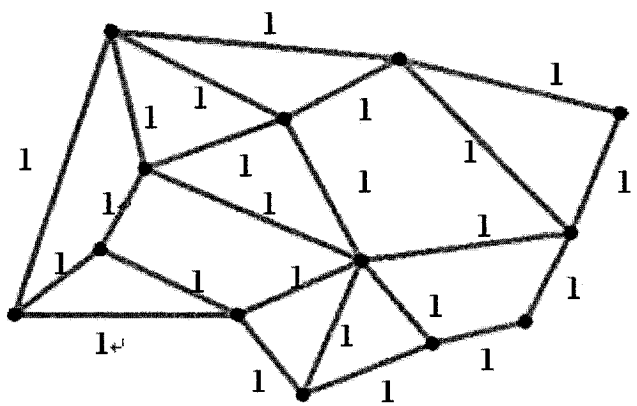


**Figure 2.4: Unit-Working-Capacity Model**

### 2.3.1 Working Capacity Model

There are two primary models we use throughout this thesis: the unit-working-capacity model and the non-unity working capacity model.

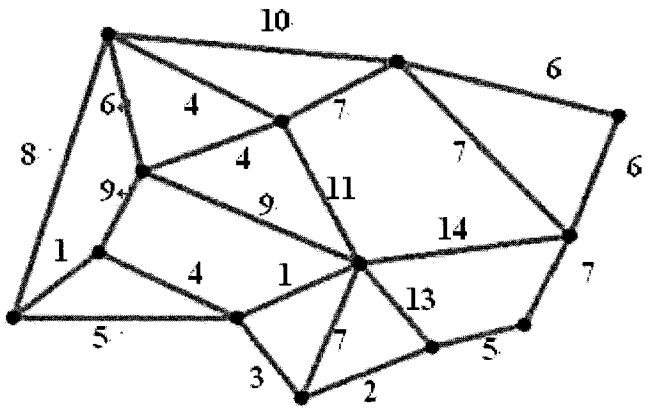



**Figure 2.5: Non-unity Working Capacity Model**

In the unit-working-capacity model, the working capacity in each link of the network is the same. For simplicity, we can assume that this amount of working capacity in the link is equal to one unit, and hence the name "unit-working-capacity model" as shown in Fig.2.4. We are interested in finding a logical topology (using spare capacity) that can provide restoration routes for all the working links with a minimum number of logical links. We observe that any such topology would only need one unit of spare capacity on each link in order to achieve 100% restorability. In other words, the topology in this simple case will not be contrained by the spare capacity (or the total capacity per link) as long as we have a unit-spare-capacity per link as well (one unit for the working, and the second as spare for protection).

The second model is the non-unity working capacity model, in which the working capacity in each link is different (e.g. Fig.2.5). If the total capacity of a span is fixed, then the spare capacity in each physical span is also determined by the working capacity (difference between the total and working capacities).

Assuming the end-to-end traffic demand is 1, we use the shortest path first method to find a working path for this demand. Then we can calculate the working capacity in each link. After that, the problem of building a logical topology based on these spare capacities so as to obtain the best restoration for the working capacities becomes a more challenging one.

**Table 2.1: Statistical data for some networks using the unit working capacity model**

| Item | Canada | NSFNET | ARPA2 | SmallNet | BellCore | Cost239 | ARB |
|---|---|---|---|---|---|---|---|
| Number of Nodes | 13 | 14 | 21 | 10 | 15 | 11 | 20 |
| Number of Links | 23 | 21 | 25 | 22 | 28 | 26 | 33 |
| Average Nodal Degree | 3.54 | 3 | 2.38 | 4.4 | 3.73 | 4.73 | 3.3 |

### 2.3.2 Simulated Networks

The following all the few networks we use in this thesis for testing, some of these are small because it is not feasible to use a long route to cover all the nodes for protection with the performance measure of restoration time in the unit-working capacity model.

Thus, in the unit-working capacity model, the simulated models we use are arbitrarily chosen small mesh networks. The statistical data of the unit-working capacity networks are shown in Table 2.1, while their layouts are provided in the Appendix A.6. The network "ARB" is an arbitrarily generated one with 20 nodes and 33 links and the network is non-Eulerian.

**Table 2.2: Statistical data for some existing networks using in non-identical working capacity models**

| **Item** | **USA** | **France** |
|---|---|---|
| Number of Nodes | 28 | 44 |
| Number of Links | 45 | 71 |
| Average Nodal Degree | 3.21 | 3.18 |

With the non-identical working capacity model, in order to compare the performance of our algorithms with other previously studied protection schemes, we have run our algorithms in the same networks as in [ShYa01, Zhan01a and Zhan01b]. The statistical data of these networks are shown in Table 2.2. The layouts of these networks can also be found in the Appendix A. We use two capacity models for the assignment of spare capacity according to the different objectives.

## 2.4 Assumptions

The following assumptions are used for the rest of the thesis, special assumptions for different algorithms will be mentioned in the context of the thesis:

1) Each unit of capacity is one wavelength. This implies that the total capacity of a link is the total number (an integer) of wavelengths available on this link.
2) The spare capacity is infinite, and therefore the restorability is 100% for analysis the redundancy of the network.
3) Wavelength routers are used in each node, and spare capacity can be shared.
4) Only one physical link may fail at a given time. This can be approximately true because there are usually built-in backup modules as well as a backup power supply for critical equipment like backbone WDM routers. Therefore, we shall mainly focus on protection against a single physical link failure, which happens most often in real world networks.
5) Both the wavelength distribution protocol and the wavelength routing protocol can gather all the required network topology information to guarantee proper network operation.
6) Full wavelength conversion is available on each WDM router. This is because we are only concerned with wavelength availability in the light-path setup and in the protection rerouting, rather than wavelength continuity.
7) In a central algorithm, i.e., a node knows all the information of the network.

# Chapter 3
# The Min-Sum Cycle Cover Detection/Protection Algorithm (MSCC)

In this chapter, we propose and study a heuristic algorithm, called the Min-Sum cost Cycle Cover algorithm (MSCC) for reducing cycle covers that can be used for both detecting and protecting a link failure. To describe the algorithm, we introduce the objective of the algorithm, review some basic graph theory and algorithms, present an analysis of the algorithm and compare the performance with the *p*-cycle later.

## 3.1 Principles of MSCC

MSCC is a graphic algorithm in which we use some basic graph theory and algorithms such as, Dijkstra Shortest Path algorithm and Minimum Weight Perfect Matching (MPM) algorithm, etc. The algorithm is based on the properties of an Eulerian graph. The main idea of MSCC is to decompose a network into a set of cycles so that all nodes and links of the given network are covered by at least one cycle. Before describing the algorithm, we need to summarize the basic definitions.

### 3.1.1 Basic Graph theory

The Eulerian graph properties are essential ingredients of our MSCC algorithm. An *Eulerian trail* can be defined as a walk on the graph edges of a graph, which uses each graph edge exactly once.

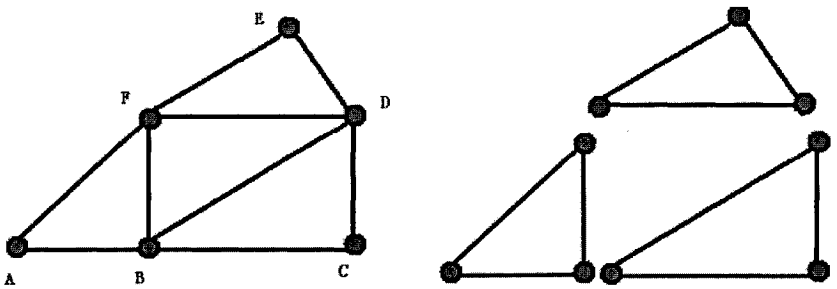


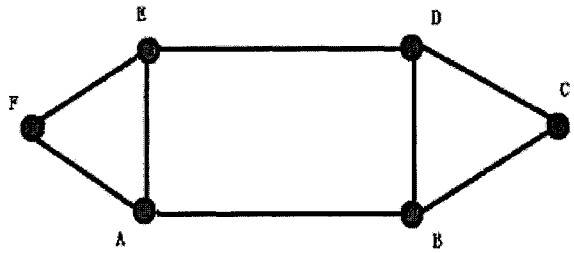

**Figure 3.1: An Eulerian network**

A connected graph has an Eulerian trail if and only if it has at most two graph vertices of odd degree, and an Eulerian trail which starts and ends at the same graph vertex. *Eulerian cycle* (or

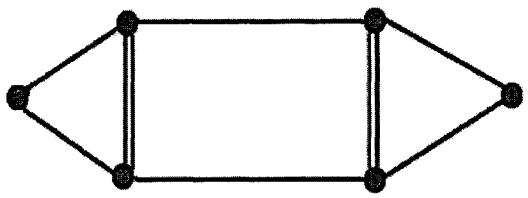
*circuit*) is an Eulerian trail which starts and ends at the same graph vertex. An *Eulerian graph* can therefore be defined as a graph containing an Eulerian circuit. In other words, if we can find a Eulerian cycle in a graph, the graph is Eulerian. There could be more than one Eulerian cycles in an Eulerian graph. *A chain* in a graph is a sequence of vertices from one vertex to another using the edges. The length of a chain is the number of edges used.

From the viewpoint of a topology, a network can be said to be *Eulerian* if there exists a single cycle that covers every link. The famous Euler Theorem proved by Euler is that a graph is Eulerian if and only if every node has an even degree, which means every node is incident to an even number of links. Every Eulerian cycle can be decomposed into a cycle cover.

Here we need to explain the difference of Eulerian cycles and Hamiltonian cycles. A Hamiltonian cycle visits all nodes once, not necessarily traversing all spans, while an Eulerian cycle crosses all spans once, but may revisit nodes. Take Figure 3.1 as an example, the cycle A-B-C-D-E-F-A is a Hamiltonian cycle, and the cycle A-B-F-D-B-C-D-E-F-A is an Eulerian Cycle.




a. Non-Eulerian Graph



b. An Eulerian Graph obtained by adding edges

**Figure 3.2: From Non-Eulerian graph to Eulerian graph**

The network in Figure 3.1 is an Eulerian graph that can be broken into a cycle cover of 3 small cycles: A-B-F-A, B-C-D-B and E-F-D-E. One property of an Eulerian graph is that if all links on a ring/cycle in an Eulerian graph is deleted, the remainder of the graph is still Eulerian. For example, removing any small cycle (A-B-F-A, B-C-D-B, or E-F-D-E), the remainder is still an Eulerian graph.

On the other hand non-Eulerian networks with some nodes with odd degrees can not be decomposed to independent cycles. In Figure 3.2, the nodes A, B, D and E all have odd degrees. Since each link connects only two nodes, the total number of odd-degree nodes is even. Thus one can expand the network by adding new links between pairs of nodes with odd degrees and obtain a new Eulerian network $N'$. We call such graph an *augmented Eulerian* network. Taking Figure 3.2 as an example, by adding a link between A and E, and between B and E, the new graph will be Eulerian.

It should be noted that when one augments a network ***G(N,S)*** by adding a new link to connect a pair of odd degree nodes, the new link can be considered as additional routing along paths of existing links. Any routing through the added path does not change the parity of the degree of the nodes in the middle of the path routed through. That is, if a node has an even degree and an added path is routed through it, then the number of links incident upon the node is increased by two and hence the node still has an even degree. So after adding links that connect the pairs of nodes with odd degree, the network does indeed become the Eulerian one.

If we traverse an Eulerian graph by following its links until a node is re-visited, the traversed part forms a sub-cycle. If we now remove this sub-cycle from the Eulerian graph and repeat this operation by traversing the remainder until all links are removed, this is equivalent to decomposing the Eulerian cycle into a cycle cover $C$ consisting of all sub-cycles. Due to the fact that no two cycles in $C$ have a common link, we can use a minimum number of monitoring wavelengths in each link.

Basically, a network $N$ with $m$ links may have $2^m$ different cycles[2] and, hence $2^{2^m}$ (a doubly exponential number of) different subsets (combinations) of cycles,. To determine whether one of the candidates has the best cycle cover is an NP-hard problem. To solve this problem, an algorithm to find one cycle cover of the network to minimize the cost may be useful. We thus invent the MSCC algorithm in the following section. Before we do that, we need to introduce the MPM standard algorithm first.

---

[2] It is a rough evaluation. The number of fundamental plus extended cycle in *G(N,S)* is of $O(2^{|S|-|N+1|})$.

### 3.1.2 The MPM (Min-weight Perfect Matching) Algorithm

We shall use the *minimum-weight matching* algorithm[PaSt82] to convert the non-Eulerian graph to the Eulerian, so we need to introduce some related definitions and facts.

Let $G$ be a graph with weights assigned to its edges. A *matching* $M$ in $G$ is a subset of edges such that no two edges in $M$ are incident to the same vertex. A matching is *perfect* if every node in $G$ is incident to an edge in $M$. The weight of a matching $M$, denoted by *weight(M),* is the sum of all weights assigned to edges in $M$. *A minimum weight perfect matching* is a perfect matching $M$ such that, for every other perfect matching $M'$ in $G$, weight $(M) \leq$ weight $(M')$.

Our goal is to find a cycle cover $C$ for a network to make the weight minimal. We use the following formulation to find the optimal matchings of $(G, w)$, where $G(N, S)$ is a complete graph $K_{2n}$, and $w$ is the non-negative weight function. We also introduce the adjacency matrix $A=(a_{ij})_{n*n}$ of G, where $a_{ij}=1$ if nodes $i$ and $j$ are adjacent, else $a_{ij} = 0$, as well as the non-negative weight function $w_{ij}$ equal to the number of hops between nodes $i$ and $j$.

Input: The topology of a graph $G$ with $A=(a_{ij})_{n*n}$;

Output: A cycle cover with minimum weight;

Constraint: A perfect (or real complete) matching must exist.

The above formulation can be solved by the Edmonds Algorithm [PaSt82] as summarized in Appendix D. It has been proved that using LP (Linear Programming) always give an integral optimal solution that corresponds to a matching. Unfortunately the LP approach does not give a polynomial time algorithm due to an exponentially large number of constraints. Instead we found out that using a dual-operation will lead us to an efficient algorithm for the general matching problem. This is reformulated as follows.

Consider all subsets of the integer set {1,2,3...n} where n is the number of the nodes which has an odd cardinality greater than one. It is not hard to check that there are Q = $2^{n-1} - n$ such subsets. Let $S_1, S_2, ..., S_Q$ be an enumeration of these subsets, then the integer $s_j$ is selected such that the cardinality $|S_j| = 2s_j + 1$. We also add additional constraint (see D3 in Appendix D.2) in order to guarantee an optimal solution that is an integer. Using $w_{ij}$ be the weight of the edge *(i,j)* as introduced above, and then using the dual D of the equation D1, D2 and D3 in Section D.2 of Appendix D, we now formulate the problem as

Max $\sum_{i=1}^{n} \alpha_i + \sum_{k=1}^{N} s_k \beta_k$

Subject to

$$\alpha_i + \alpha_j + \sum_{i,j \in S_k} \beta_k \leq w_{ij} \qquad \text{for all } i, j \leq n \qquad (3.1)$$

$$\beta_k \leq 0, \qquad \text{for all } k \leq N \qquad (3.2)$$

where $\alpha_i$'s are the dual variables to indicate whether a perfect matching exists (corresponding to the constraints in D1), and $\beta_i$'s are the dual variables to satisfy the odd cardinality variable $s_j$ as discussed above (thus satisfying Constraint D3).

It has been proven [MoSh91, CoLe90] that such a matching can be obtained in time bounded by $O(N^3)$, where $N$ is the number of nodes. The solution given in [PaSt82] has a complexity of $O(N^4)$ and its implementation is provided in Appendix D.

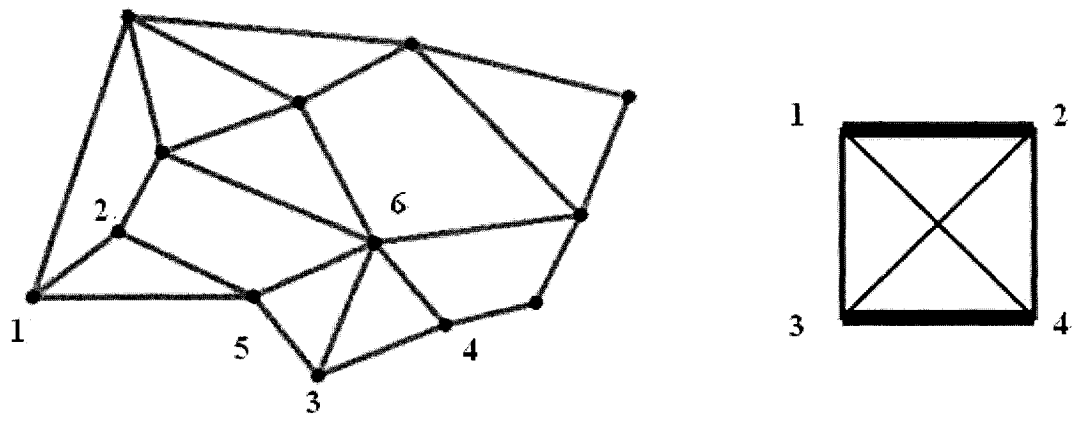
| | 1 | 2 | ... | j | ... | n |
|---|---|---|---|---|---|---|
| 1 | - | $w_{1,2}$ | ... | $w_{1,j}$ | ... | $w_{1,n}$ |
| 2 | $w_{1,2}$ | - | ... | $w_{2,j}$ | ... | $w_{2,n}$ |
| ... | ... | ... | ... | | ... | ... |
| i | $w_{1,i}$ | $w_{2,i}$ | ... | $w_{i,j}$ | | $w_{i,n}$ |
| ... | ... | ... | | | ... | ... |
| n | $w_{1,n}$ | $w_{2,n}$ | ... | $w_{j,n}$ | ... | - |

**Figure 3.3: The Minimal Weight Matrix between the Nodes with Odd Degree (j >i)**

In order to find the minimum weight perfect matching and also the augmented paths, we need to construct another graph $G'$. Let $G'(N', S')$ be a graph such that $N'$ is the set of all nodes with odd degrees in $N$ and $S'$ is the set of all possible edges between nodes in $N'$, i.e. $G'$ is a complete graph on the odd degree nodes of $N$. Let an edge $(x, y)$ in $G'$ have a weight $c$, where $c$ is the weight of the minimum weight path $P$ between $x$ and $y$ in $N$. In other words, a path $P$ between nodes x and y now becomes an edge with a weight c in $G'(N', S')$, where $c$ is the sum of the weights on all links $e_1, e_2, \ldots e_k$ along the path $P$ in $G(N, S)$. Since there exist multiple paths

between $x$ and $y$, $c$ is the minimum sum among all paths $P$ between $x$ and $y$. The desired minimum matching now becomes a complete matching of the nodes in the $G'$.

Let $n$ be the number of nodes in $N'$, then Figure 3.3 depicts the matrix of minimal weight between the nodes with odd degree, where $w_{i,j}$ $(j>i)$ in the $i$th row and $j$th column is the minimal distance between node $i$ and $j$. Since the weight on an edge AB is usually equal to that of the edge BA, the minimum weight matrix is symmetrical, very often we only need to consider the upper right half of the matrix.




a) Node 1, 2, 3 and 4 has odd degree b) The minimum weight matching is {(1,2), (3,4)}

**Figure 3.4: Finding the Minimum Cost Matching among the Odd-degree Nodes in a Canada network**

### 3.1.2 An Example

Figure 3.4 is an example of finding and adding the shortest path (minimum) matching among the odd-degree nodes, by which the network can be converted to an Eulerian graph.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | - | 1 | 2 | 3 |
| 2 | 1 | - | 2 | 3 |
| 3 | 2 | 2 | - | 1 |
| 4 | 3 | 3 | 1 | - |

**Figure 3.5: The minimal weight matrix between the nodes with odd degree in Canada network**

There are n = 4 nodes with odd degree in Figure 3.4a, i.e. node 1, 2, 3 and 4. If we assume the weight for every edge is 1, then we can find the distance between any pair of nodes. For example, there are many paths between nodes 1 and 3, such as 1-2-5-3, 1-2-5-6-4-3, 1-5-3, but the path 1-5-3 is the shortest with path length of 2 hops.

Then all of the possible perfect matchings are: {(1,2), (3,4)), {(1,3), (2,4)), and {(1,4), (2,3)), where $(x,y)$ represents the edge with end nodes $x$ and $y$. Note the perfect matching must be composed by two different edges according to the definition of perfect matching, otherwise it cannot be called perfect according to the definition. For example, the matching {(1,2)}is a matching, but not a perfect matching. Since the minimum weight matrix is symmetrical in this case, and we only need to consider the upper right half of the matrix. According to the weight found on every edge in Figure 3.5, we can determine the weight sum for all of the possible matchings in the following:

1) 1+1= 2 for (1,2) and (3,4);
2) 2+3=5 for (1,3) and (2,4);
3) 3+2=5 for (1,4) and (2,3);

By sorting the above matching according to the sum weight, we determine the matching {(1,2), (3,4)} has the minimum matching. The minimum weight matching delivers the result, (1,2) and (3,4), in shadow as shown in Figure3.6.

In our next step, we need to augment these found shortest paths between node 1 and node 2, and between node 3 and node 4, from which we can obtain a Eularian graph.

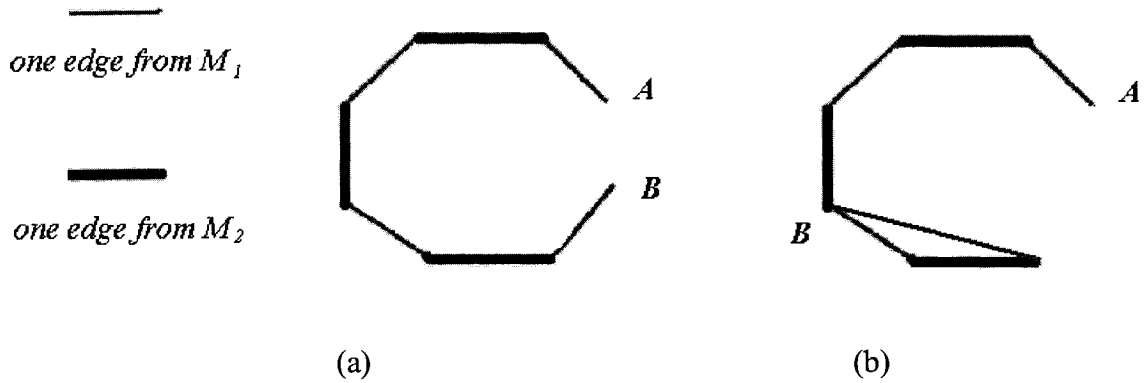
The MPM algorithm is crucial for the MSCC algorithm. After implementing MPM, we can find not only the matching among the nodes with an odd degree, but also the relevant shortest paths between the node pairs in the matching by using SP algorithm. No matter when we augment or delete the shortest paths on the graph, we know it can be converted to an Eulerian graph according to Euler Theorem. Adding the shortest paths can minimize the change in the graph, and make the first-found cycle smaller for MSCC because what we chose is the min-cost cycles. We will provide the detail in the following section.

## 3.2 Finding Cycle Cover

We need to decompose the network into a cycle cover so that we can protect every link using one of the cycles. We first provide a theorem that form the base of our MSCC algorithm aimed at

forming circuits (routing through more than two nodes) in the decomposed network. The following theorem provides the theory base for which two different matching forms cycles in $G'$.

**Theorem 3.1** Consider any two completely different perfect matchings, $M_1$ and $M_2$ (i.e. $M_1 \cap M_2 = \phi$) in the complete graph $G'$, then by concatenating their edges alternatively at a commonly shared node in G, then we obtain cycles.




(a) (b)

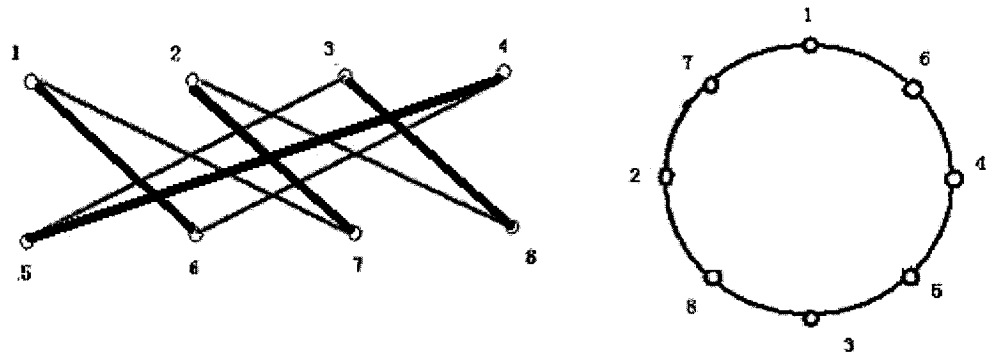
**Figure 3.6: The walk L is not closed**

Proof: The proof is by contradiction. According to the procedure discussed above, we connect the edges alternatively in matchings $M_1$ and $M_2$. We assume there exists a path $L$ that is not closed when we exhaust all of the edges in the two matchings (in $M_1$ Y $M_2$) (see Figure 3.6).

Since the walk is not closed according to the assumption, its origin and its destination nodes can not be same. Therefore there are two possibilities:

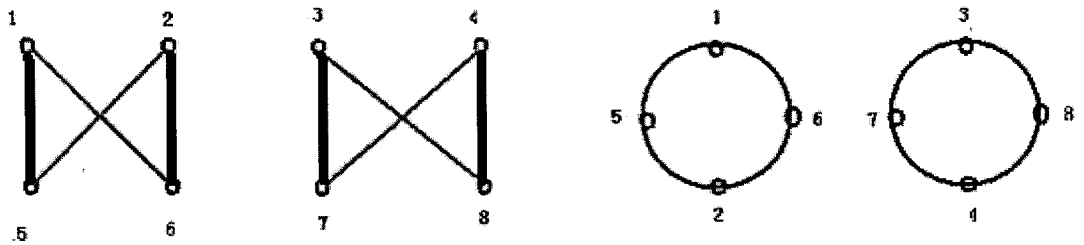
1) The destination $B$ is only incident to one edge (see Figure 3.6(a)), so the nodal degree of $B$, $d$(B)= 1;
2) The destination $B$ is the same with other nodes (but not the origin) on the walk (see Figure 3.6(b)), then degree of $B$, $d$(B) = 3, because $B$ is both the middle node and end nodes on the walk, it incident to 3 different edges.

But both of the possibilities are contradictory to the definition of the perfect matching, every node in $G'$ is incident to one edge in a perfect matching, thus every node in $G'$ is incident to two different edges in $M_1$ Y $M_2$, so $d$(B) = 2. Therefore the assumption is false.




(a) two different perfect matchings (b) forming one cycle

**Figure 3.7: Two Different Perfect Matchings Form One Cycle**




(a) two different perfect matchings (b) forming two cycles

**Figure 3.8: Two Different Perfect Matchings Form More Than One Cycle**

Let us analyze the following examples that illustrate two different cases. In a complete graph $G'$ with eight nodes which are the all nodes with odd degree in a graph $G$ (physical topology not shown here). Figure 3.7(a) shows the first perfect matching $M_1$ with {(1,6), (2,7), (3,8), (4,5)}, while Figure 3.7(b) is the second perfect matching $M_2$ with {(1,7), (2,8), (3,5), (4,6)}. Let us start with one edge from $M_1$, say (1,6). Then we have to choose another from $M_2$, which shares the common node 6. We find the edge (4,6) from $M_2$. Thus the walk is now 1-6-4. In the same way, we can find the edge in $M_1$ to continue the walk. By choosing the nodes from two perfect matchings alternatively, while sharing a common node between two concatenated edges. We can finally obtain one walk that is closed, 1-6-4-5-3-8-2-7-1, as shown in Figure 3.7(b).

Figure 3.8 shows that the number of final walks can be more than one. By merging two different perfect matchings, simple cycles can be formed. In this example, we assume $M_1$ =

{(1,5), (2,6), (3,7), (4,8)} and $M_2$ = {(1,6), (2,5), (3,8), (4,7)} According to the theorem and procedure, we can only get cycles: 1-5-2-6-1, and 3-8-4-7-3 because a circuit was formed before every node is routed.

### 3.2.1 MSCC Algorithm

Based on the discussion above, the general procedure is to convert a non-Eulerian graph to Eulerian by augmenting the first minimum weight perfect matching in *G'*, which corresponds to the shortest paths connecting the same node pairs with odd degree in *G*. The second minimum perfect matching is another one with no common edge with the first one. Generally speaking, we obtain the second minimum perfect matching by setting the weight of the edges in the first minimum perfect matching to infinite.

A cycle cover is then obtained by connecting the first and the second minimum weight perfect matching in *G'* which corresponds to the shortest paths connecting the same node pairs with odd degree in *G* to form one or more cycles. After removing the cycles, the remaining graph is still Eulerian. By starting from one of the remaining edge, we find the shortest path between the end nodes, and remove the edge and the path that form a cycle. We continue the operation until there are no edges and nodes left.

Let the weight function $w$: $S \rightarrow \boldsymbol{R}_0^+$, let $\boldsymbol{N}$ be the set of nodes, and $N$, $N'$ the number of nodes and nodes with odd degree, respectively. Then the procedure of MSCC ($G$, $w$; $C$) can be described as following:

(1) Group all nodes with odd degrees into a set $\boldsymbol{N'}$, i.e. $\boldsymbol{N'} = \{v \in \boldsymbol{N}: d(v) \text{ is odd}\}$, where $d(v)$ is the degree of the vertex $v$;

(2) Determine $w_{ij}$ for all $x, y \in \boldsymbol{N'}$, where $w_{ij}$ is the weight for the shortest path between $x$ and $y$ in $\boldsymbol{G}$;

(3) Generate the complete graph $H$ on $\boldsymbol{N'}$ with weight $w_{ij}$ to determine a perfect matching $K$ of minimal weight for ($H$, $w$).

(4) Determine the shortest path $W_{xy}$ from x to y in $G$ to form the set $W = \{W_{xy}, x, y \in \boldsymbol{N'}\}$.

(5) For each span $(x, y) \in K$ with the corresponding $W_{xy}$ found, add the parallel edges to $G$. Let $G'$ be the multigraph thus defined.

(6) Mark all of edges of on $W$, determine the second minimum weight perfect matching using the unmarked edges in $X$;

(7) Find the shortest path $W'_{xz}$ from $x$ to $z$ in $G$, and generate the set $W' = \{W'_{xz}, x, z \in X\}$;

(8) Merge the paths in $W$ and $W'$ to form a set of simple cycle $C_0$, to be removed. After removal, the resultant multigraph is now an Eulerian graph G'';

(9) Select an edge $E_1$ in $G''$ as the first edge, and find the shortest path between its two end nodes, Then merge the path and the edge to form a set of simple cycles $C_1$ for removal;

(10) Repeat step (9) to find the cycle $C_i$, $i = 2,3,4,...,k$, for a $k$ when the edge set of the network $G''$ becomes empty.

(11) Save all of the $C_i$, and denote $C = \{C_i\}$, $i = 1, 2, 3, ...k$.

This procedure is also captured in the pseudo code below:

```
MSCC (G,  w; C)
{
Let set N' = {n∈ N: d(n) is odd}, where d(n) is the degree of the node n;
for (int x=0,; x<N'; ++x)
  for (int y=0,;y<N'; ++y) find the distance of shortest path d(x, y) = Floyds(x,y; N,N');
Let H be the complete graph on X with weight function d(x, y).
K= MPM(H, d) (comments: Find minimal perfect matching K for H;)
for ((x, y∈ X; x,y<N'; ++x, ++y)
  find shortest path W_xy ;
  W = {W_xy , x, y∈ N' }
  Marked all the spans and nodes on the path of the minimal match, and denote as M';
  find the shortest path W'_xz from x to z in G\M', and denote W' = {W'_xz , x, z∈ N' };
  remove all the paths,  which form a cycle set C_0 , in W and W' from G, G" = G\ C_0 , C_0 ->C
  let S"= number of spans in G"
for (s =0; s< S"; ++s )
{
  if (G"!=NULL)
  {
    find the shortest path  p_s  between two end node of s using Dijkstra shortest path algorithm,
    set C_s+1 = p_s ∪ S ;
    C_s+1 ->C;
    G" = G"\ C_s+1 ;
  }
}
}       /*  end MSCC

Floyds(A, B; N,N')
{
for (int A=0;A<N';++A)
 for (int B=0;B<N';++B)
        if (Adj[A][B])
            for (int C=0;C<N;++C)
                if (Adj[B][C])
                        if (!Adj[A][C] || (Adj[A][B] + Adj[B][C] < Adj[A][C]))
                        Adj[A][C]=Adj[A][B]+Adj[A][C];
 }
```

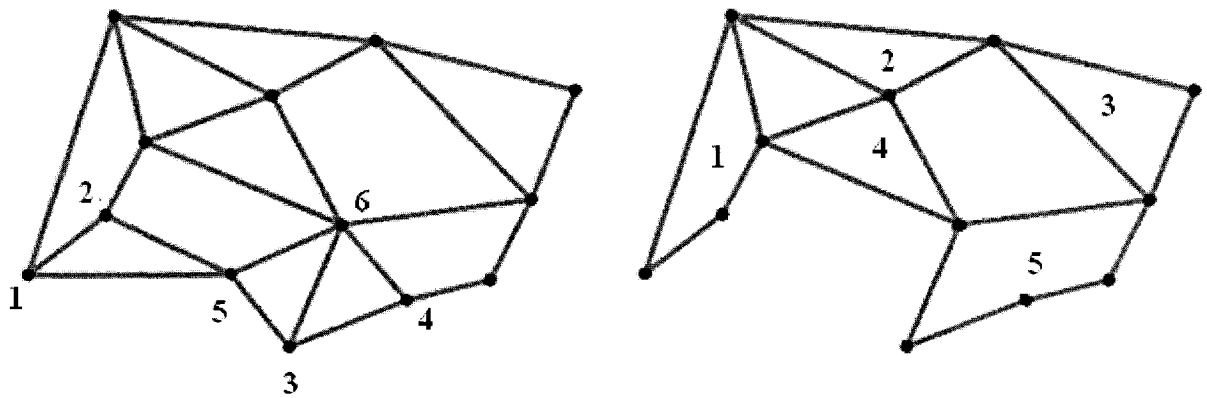
### 3.2.2 An Example

Take the topology of Canada (Appendix A3) as an example. We have previously determined the first minimum weight perfect matching in Section 3.1.2 and Figure 3.5. Now we augment the

links along the corresponding shortest paths 1-2 and 3-4; and obtain an Eulerian graph. Then we mark the edges along the found shortest paths, and find the 2nd minimum perfect match, which is {(1,3), (2,4)} shown in Figure 3.9.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | - | 1 | 2 | 3 |
| 2 | 1 | - | 2 | 3 |
| 3 | 2 | 2 | - | 1 |
| 4 | 3 | 3 | 1 | - |

**Figure 3.9: finding the 2nd minimum perfect matching in Canada Network**




(a) Find the 2nd minimum perfect matching (b) Find the ring cover for the Eulerian graph

**Figure 3.10: Implementing the MSCC to Canada network**

As shown in the Figure 3.10, we can find the corresponding shortest path 2-5-6-4 between the node pair of 2 and 4, and 1-5-3 for that of 1 and 3 (following steps (1) to (4)). Then we augment the path to make the graph Eulerian (step (5)) and find the second minimum matching (step (6)). Now by connecting the two groups of shortest paths, we can obtain one cycle, which is 1-2-5-6-4-3-5-1, which contains two simple cycles: 1-2-5-1 and 3-4-6-5-3 (step (7) and (8)).

After removing the close path 1-2-5-6-4-3-5-1, the graph becomes an Eulerian graph again. Then using Dijkstra's shortest path algorithm, we can find 5 cycles covering the remaining graph as shown in the Figure 3.10 (b)(step (10) and (11)).

## 3.3 Performance Analysis and Evaluation

According to the design, not only our MSCC can find the cycle cover, but it can also find the min-sum cost cover, which is very significant to network detection and protection because it can guarantee fully detection and 100% restorability for every link can be protected by at least one cycle.

We shall analyze the MSCC algorithm in three ways: computation complexity, comparison with other fault detection algorithms, and comparison of the redundancy with the *p*-cycle design. We use Microsoft Visual C++ 6.0 to implement of MSCC. We shall also make use of our heuristic algorithm *SP-Join & WCIDA* (to be detailed in the Chapters 4 and 5) to provide redundancy of *p*-cycles for comparison.

### 3.3.1 Complexity Analysis

We have discussed before that the complexity of finding cycle cover can be up to double exponential. But MSCC algorithm can guarantee the polynomial time complexity.

**Table 3.1 Number of node with odd degree in Networks**

| Item | Canada | NSFNET | ARPA2 | SmallNet | BellCore | Cost239 | ARB |
|---|---|---|---|---|---|---|---|
| # of Nodes | 13 | 14 | 21 | 10 | 15 | 11 | 20 |
| # of Links | 23 | 21 | 25 | 22 | 28 | 26 | 33 |
| # of Nodes with Odd Degree | 4 | 10 | 4 | 4 | 8 | 6 | 12 |

Our MSCC algorithm consists of determination of various shortest paths. To find a shortest path, we can implement either the Dijkstra's shortest path algorithm, which has the complexity of $O\ (S+N\ logN)$, or the Floyd's Algorithm, the complexity of which is $O\ (N^3)$ where $S$ is the number of spans and $N$ is the number of nodes. The complexity of finding the shortest paths is $O[N'^3\ (S+N\ log\ N)]$ (using Dijkstra's shortest path algorithm) which has been used twice in the algorithm, where $N'$ is the number of nodes with odd degrees.

Considering the above, the upper bound of complexity of MSCC is max $\{O\ [N'^3\ (S+N\ log\ N)],\ O\ (N'^3 N^3)\}$. Therefore the MSCC algorithm has a polynomial time complexity. The

Table 3.1 is the evaluation of number of nodes with odd degree for some typical networks. From the results of Table 3.1 discussed later, we observe that $N'$ is a fraction of $N$ or $O(N)$. Therefore the complexity in the worst case is $max\{O[N^3(S+N\log N)], O(N^6)\}$

### 3.3.2 Comparison between MSCC and Other Fault Detection Algorithms

We simulate our MSCC algorithm in four different networks, from which we can get better performance when compared to two other detection algorithms: Heuristic Depth First Searching (HDFS) and Shortest Path Eulerian Matching (SPEM) in [ZeHu04] (as reviewed in Ch. 1), which are two heuristic fault detection algorithms. Although HDFS and SPEM can obtain the good performance on number of monitors, we found MSCC can obtain even better results by making the performance comparison.

We tested and compared the proposed failure detection mechanism using MSCC on four example networks: NSFNET, ARPA2, SmallNet, and Bellcore, as shown in Appendix A.4. The basic properties have been shown in Tables 2.1 and 3.1, including the number of nodes, links and average nodal degree, and the number of the nodes with odd degree.

**Table 3.2: Comparison of cycle finding algorithms: HDFS, SPEM and MSCC**

| | NSFNET | | | ARPA2 | | | SmallNet | | | Bellcore | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Algorithm | HDFS | SPEM | MSCC | HDFS | SPEM | MSCC | HDFS | SPEM | MSCC | HDFS | SPEM | MSCC |
| # of cycle cover | 6 | 4 | 3 | 4 | 4 | 3 | 8 | 9 | 6 | 6 | 5 | 5 |
| $R_{link}$ | 28.6% | 19.0% | 14.3% | 16.0% | 16.0% | 12% | 36.4% | 40.9% | 27.3% | 21.4% | 17.9% | 17.9% |

The performances of the three cycle finding algorithms are compared in terms of the relative costs for fault detection. Let $R_{link} = T/L$, where $T$ is the number of required transceivers for monitoring, and $L$ is the number of links in the networks. The comparison results are listed in Table 3.2. which show that MSCC is most cost efficient because of using least cycle monitors for meshed network failure detection, compared to the other two heuristic algorithms.

### 3.3.3 MSCC Redundancy Performance on Various Protection Designs

Table 3.3 composes the design on Canada net, Cost239, ARB and NSFNET using MSCC algorithm. The denotation "XnYs" represents X nodes and Y spans. For example,13n23s represents there are 13 nodes and 23 spans in the network.

**Table 3.3: Performance for MSCC algorithm ("XnYs" represents X nodes and Y spans)**

| Networks | Canada | Cost239 | ARB | NSFNET |
|---|---|---|---|---|
| Number of Nodes and Spans | 13n23s | 11n26s | 20n33s | 14n21s |
| Number of node with odd degrees | 4 | 6 | 12 | 10 |
| Number of cycles in the cycle cover generated from MSCC | 7 | 7 | 7 | 3 |
| Redundancy for MSCC | 108.7% | 111.5% | 118.2% | 123.8% |
| Number of *p*-cycles required | 1 | 1 | 1 | 1 |
| Reduancancy for *p*-cycle | 56.5% | 42.3% | 60.6% | 66.7% |

The redundancy is obtained by equation (2.2) in Ch. 2. Here we assume the network is unit-working-capacity model, i.e. $w_i$=1 for every $i$. Therefore $\sum_{i=1}^{S} w_i$ becomes the number of links, $S$. Since every link failure needs to be protected by a cycle, we augment the shortest paths to the network to generate the cycle cover. Therefore $\sum_{i=1}^{S} s_i$ is the sum of $S$ and the number of augmenting edges. Take Canada network as an example, we have $S$= *23,* according to the analysis of Section 3.2, and the number of augmenting links is 2. Therefore *Redundancy*= (23+2)/23= 108.7%. From the results of redundancy for MSCC, we can see that the redundancies are all more than 100% for all cases, comparing to these of the *p*-cycle design.

But from the view of protection, the redundancy generated by MSCC is always more than 100% even for the unit-working capacity networks. The number of augmented links will be the dominant factor in the redundancy. We implement the algorithm for networks (see in Appendix A 4) in which we just consider the unit-working capacity case, and compare the results to the *p*-cycle design. If the network is not unit-working capacity, the redundancy will be much higher

than that for unit-working, because the spare capacity will be chosen to fit the biggest working copies on the cycle and this leads to higher redundancy.

Although $p$-cycle can achieve better redundancy efficiency by using $p$-cycle for protection (see Table 3.3) than the cycle covers obtained by MSCC, the $p$-cycle design cannot be used to detect failure because we cannot detect the link failure for the straddlers.

## 3.4 Application of MSCC: Failure Detection and Protection

In order to detect a network failure, a transceiver/monitor can be assigned to one node in each cycle along with a spare link that will act as a loopback supervisory channel. Usually all of the optical network performance indices, including optical power, optical spectrum, optical signal-to-noise ratio, and more importantly, bit error ratio, can also be measured along each monitoring cycle. Once a fault occurs in any link or node belongs to such a cycle, the monitor will trigger an alarm in this monitoring cycle and invoke a procedure to achieve fault detection and isolation. Consequently if the failure happens on a link, we will use the same channel as a spare link to protect the failure link.

Our graphical algorithm MSCC can be used for network failure detection and protection under the following proper assumptions: (1) network demands are routed a priori, (2) the cost of the network is a linear function of the demand on each span, (3) cycle capacity is not a constraint (i.e. idealized cycles that can handle as much demand as traverses a span), (4) the network topology is fixed, and (5) the cycle set covers every link in the network graph $G$.

Because the number of transceivers required for monitoring (also called monitors) is determined by the number of cycles in the cycle cover of each network example, we also use the ratio between the number of required transceivers and the number of links in a network in [ZeHu04], to measure the relative costs for fault detection.

The results in Table 3.1 show that MSCC is more cost efficient because of using less cycle monitors for meshed network failure detection, when compared to the other two heuristic algorithms.

## 3.5 Concluding Remark

Based on the above discussing, we know that the MSCC is designed to detect or monitor the failure and guarantee 100% restoration because every link can be located in a cycle, and after detecting the failure, and at same time minimize the total cost of the whole network. Compared

to other heuristic failure detection algorithms, MSCC on one hand can obtain the better performance on the view of number of monitors and relative cost of detection. Considering the protection on the other hand, MSCC is not better than *p*-cycle design because of the redundancy is more than 100%, but *p*-cycle can not be used to detect failure because the failure on straddlers can not detected. We can get the conclusion that MSCC is a better choice for failure detection rather than for the protection design.

Based on the above discussion, in the following chapters of the thesis, we will focus on *p*-cycle scheme solely on the view of restoration/protection.

# Chapter 4
# Algorithmic Approaches for Efficient Enumeration of Candidate *p*-Cycles

We will first study two basic operations on primary *p*-cycles, namely operations *Add* and *Join* proposed by Prof. Grover. As an extension, we have formulated the more generalized *SP-Add* (Add operation using Shortest Path), *SP-Join and SP-Merge* algorithms to operate of pre-selection of *p*-cycles.

## 4.1 Design Consideration

Dr. Grover proposes two merging operations [Grov03], *Add* and *Join*, as the basic idea for the *p*-cycle design after the work of SLA [ZhYa02]. The initial idea was to study operations that would transform the set of cycles from SLA into more efficient *p*-cycles and to develop a fully restorable *p*-cycle network design algorithm based on this approach. The main benefit of such an algorithm is that it avoids any form of explicit cycle-set enumeration. Some performance evaluation will be provided at the end. First of all, we need to formulate the problem as follows:

Input: The topology of the network, including number of nodes *N*, spans *S*, and end nodes of every span, cost for every span (to simplify the problem, we consider the cost as the hop number);

Output: *p*-Cycle candidate set;

Constraints: The cycles generated should be simple cycles, which cannot route the same node or span more than once.

Before we get to our heuristic algorithm, we shall summarize the Straddling Link Algorithm (SLA) to introduce the concepts and terminology used in this chapter.

## 4.2 Straddling Link Algorithm

The SLA [ZhYa02] attempts to generate p-cycles with respect to each network span. The main idea of SLA is to find a *p*-cycle that uses that span as a straddling span relationship. Specifically, we determine two node-disjoint shortest paths while restricting use of the span itself in either route. The two routes are then combined end-to-end to form what we have subsequently called a

*primary p-cycle*. The intent of the process is to generate a set of cycles with exactly one straddling span each, although from topological necessity some such cycles have more than one straddling span, and in other cases (typically involving degree-2 nodes) no primary cycle exists for a specific span. However, most primary cycles do contain no more than the single intended straddling span, and although their efficiencies will be better than simple cycles, they are relatively inefficient compared to other cycles that can be constructed in the network having many more straddles and hence potential efficiency in a capacitated design.

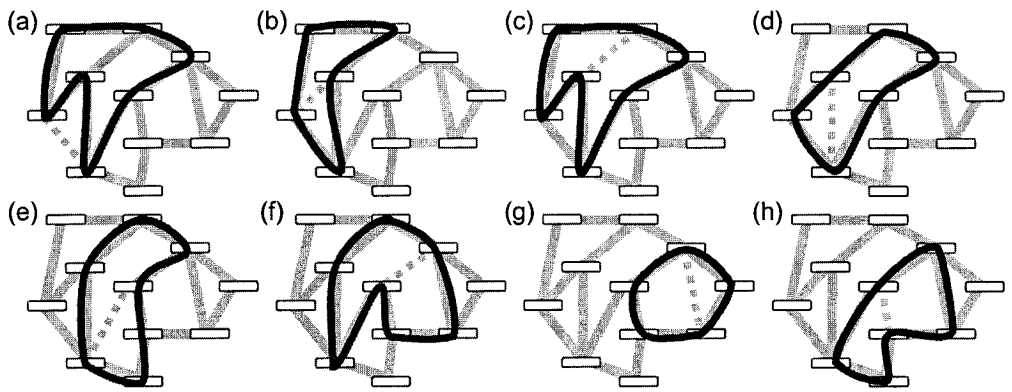



**Figure 4.1: One Set of Primary p-cycles for a Small 11-node 16-span Network Generated by the SLA**

Figure 4.1 shows a small network in which eight of spans have a primary cycle. Each graph in Figure 4.1 highlights one span (of the straddling relationship, the dashed line), and its primary cycle. By construction, all of the primary cycles will have at least one straddling span. However, it is possible that a primary cycle will have more than one straddler. In Figure 4.1, the primary cycles shown in panels (a) and (c) are actually the same cycle, each have two straddlers.

Since each primary cycle has at least one straddling span, then the p-cycle score of a primary cycle is

$$AE(p) \geq \frac{2+|S_{C,p}|}{|S_{C,p}|}, \qquad \text{(EQ4)}$$

where $S_{C,p}$ is the number of on-cycle spans. In most cases, a primary cycle has exactly one straddler, and so the inequality in (EQ4) becomes a strict equality.

## 4.3 Pre-selection Operations to *p*-Cycle Candidate

SLA is extremely fast and totally without exhaustive enumeration of the cycle-set based on the topology. On the other hand, if the set of such nominally "single straddler" cycles from SLA are used directly as the basis of a capacitated network design, the result is collectively quite inefficient because there are many small *p*-cycles of both low *AE* and actual efficiency. So we would like to improve the performance by introducing the *Add* and *Join* operators in the following.

*Add* and *Join* operations are intended to operate on suitable pairs of primary cycles (and subsequent cycles) to create new *p*-cycles with more straddling span relationships and hence higher efficiency.

### 4.3.1 The Add Operation

The *Add* operation operates on a pair of primary *p*-cycles when each has the other's straddler as one of their on-cycle spans, and (for reasons to be explained) the two straddler spans involved are not adjacent in the graph.

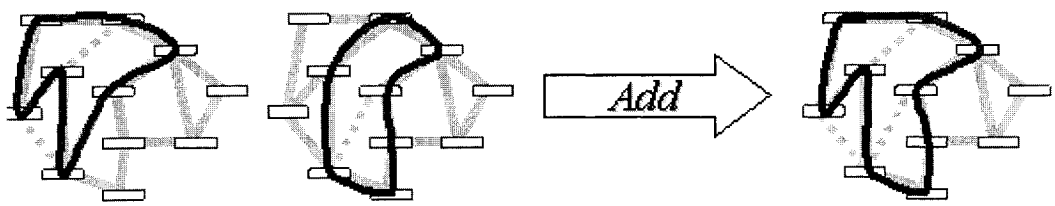



**Figure 4.2: An example of the *Add* conditions and Operation on Two Primary p-Cycles**

As an example, consider primary *p*-cycles *c* and *e* in Figure 4.1, and examine the qualifying condition and the resulting Figure 4.2. The key is that each cycle originally contains the other's straddler, and the resultant *p*-cycle contains both straddling spans as its new straddlers.

The exclusion condition of "contra-adjacency" means that the two straddlers cannot be both adjacent to each other and have their common node arrived at in opposite directions if a clockwise (or counterclockwise) "tour" of each of the two primary *p*-cycles is taken. The resulting *p-cycle* may have a straddling relationship to the two spans as a concatenated sub network.

As a forbidden case, examine *p-cycle*s e and f in Figure 4. The result in Fig 4.3 illustrates the set union is not strictly a cycle, nor does the identifiable outer cycle component by itself

protect either previous straddler span.

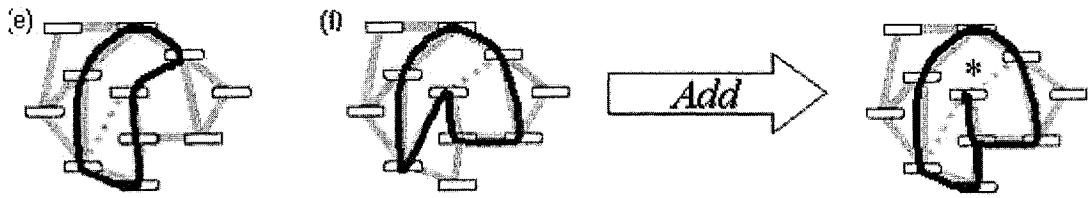



**Figure 4.3: Add Operation is not Allowed if the Straddling Spans are "Contra-adjacent"**

Mathematically, an *Add* operation is possible on any two primary *p*-cycles {A} and {B} with straddling spans $i_{s\{A\}}$ and $i_{s\{B\}}$ when $i_{s\{A\}} \in \{B\} \cap i_{s\{B\}} \in \{A\} \cap (i_{s\{A\}}$ not contra-adjacent to $i_{s\{B\}}$), in which case the new *p-cycle* becomes $\{C\} = \{A\} + \{B\} \equiv \{\{\{A\} - i_{s\{B\}}\} \cup \{\{B\} - i_{s\{A\}}\}\}$. The resultant *p-cycle* $\{C\}$ is then of total length (in hop counts): $|\{C\}| = |\{A\}| + |\{B\}| - 2$.

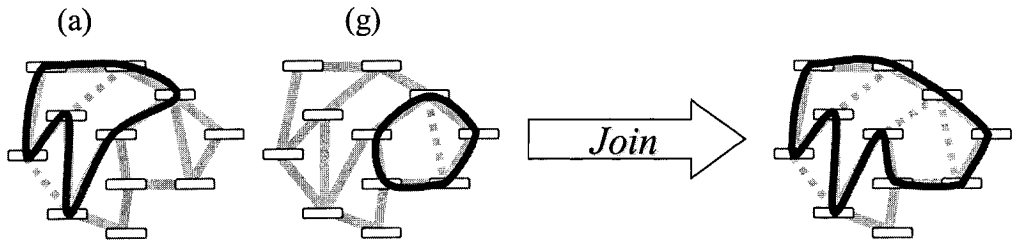



**Figure 4.4: Example of the *Join* condition and operation on primary *p*-cycles**

### 4.3.2 The Join Operation

The *Join* operation is possible when a pair of primary *p-cycles* have exactly one, non-straddler, span in common (i.e., on their cycles'). As an example, consider primary. *p-cycles* 2 and 6 in Fig. 4.1, the qualifying condition and the result are illustrated in Fig 4.4.

Mathematically, a *Join* operation is possible on any two primary *p*-cycles $\{A\}$ and $\{B\}$ with straddling spans $i_{s\{A\}}$ and $i_{s\{B\}}$ when $|\{B\} \cap \{A\}| = 1$. Under this condition the new *p*-cycle becomes $\{C\} = \{A\} + \{B\} - \{\{A\} \cap \{B\}\}$. The resultant *p-cycle* $\{C\}$ is then of total length (in hop counts) $|\{C\}| = |\{A\}| + |\{B\}| - 1$. Thus the *three* straddling spans $\{i_{s\{A\}}, i_{s\{B\}}, i_{s\{C\}}\}$ where $\{i_{s\{C\}}\} = \{A\} \cap \{B\}$ is a new straddler formed out of the single span where both previous primary *p*-cycles have now been joined.

*Join* is provably somewhat more advantageous than *Add* because it adds one straddler (+2

on score) and deletes one span (-1 on cost), whereas *Add* keeps the same number of straddlers (+0 on score) but deletes two spans from the total spans (-2 on cost).

### 4.3.3 Application of *Add* and *Join* to a Network

We now use the network of Figure 4.1 as a working example to show how an all-pairs application of *Add* and *Join* (where possible) on a network's primary *p*-cycles can produce a more efficient set of *p*-cycles. In the network example, there are seven unique primary *p*-cycles (remember that (a) and (c) are the same cycle), so we need to inspect only 21 combinations of cycle pairs for possible application of *Add* or *Join*.

**Table 4.1: Applying *Add* or *Join* to Primary *p*-cycles in Fig 4.1**

| A | B | Add/Join | AE(p) | | A | B | Add/Join | AE(p) |
|---|---|---|---|---|---|---|---|---|
| a | b | - | n/a | | c | e | Add- | 1.67 |
| a | c | - | n/a | | c | f | Add | 1.67 |
| a | d | - | n/a | | c | g | Join | 1.80 |
| a | e | Add | 1.44 | | c | h | - | n/a |
| a | f | Add | 1.44 | | d | e | Add | 1.75 |
| a | g | Join | 1.6 | | d | f | Add | 1.50 |
| a | h | - | n/a | | d | g | Join | 1.67 |
| b | c | - | n/a | | d | h | - | n/a |
| b | b | - | n/a | | e | f | - | n/a |
| b | e | - | n/a | | e | g | - | n/a |
| b | f | - | n/a | | e | h | - | n/a |
| b | g | - | n/a | | f | g | Add | 1.50 |
| b | h | - | n/a | | f | h | - | n/a |
| c | d | - | n/a | | g | h | Add | 1.57 |

Table 4.1 summarizes the combinations of primary *p*-cycles where either the *Add* or *Join* operation is feasible and gives the resultant *p*-cycle efficiencies; we ignore the cycle in Figure 4.1 (a) since it is the same as the one in Figure 4.1 (c). The resultant *p*-cycles have efficiencies ranging from 1.50 up to 1.80, with an average of 1.64, as compared to the original primary *p*-cycles whose efficiencies range from 1.29 up to 1.57 with an average of 1.40 (an improvement of 17%). In Table 4.1, a, b, c, d, e, f, g, h represent the graphs in Figure 4.1, respectively. *AE(p)* is the score of the *p*-cycle.

## 4.4 The SP Merge Algorithm

Although *Add* and *Join* operations result in more efficient *p*-cycles, programming the determination of pairs of cycles that satisfy the conditions required for the operations is complex.

However, *Add* and *Join* serve only as a conceptual stepping-stone to the following class of simpler operations, which we can perform on individual *p*-cycles, as opposed to *p*-cycle pairs. So we consider operations *SP-Add,* which merge the shortest path (SP) between the two end nodes of the spans on the primary cycles. They will eventually be utilized in our new algorithm to improve the primary *p*-cycle from the SLA.

### 4.4.1 The *SP-Add* Operation

The *SP-Add* operation uses the shotest path (Dijstra) algorithm [DuGr94] to merge the cycle-disjoint shortest path and the path between the end nodes of an on-cycle span in the primary cycle, with the purpose of forming a simple cycle from the merged paths. A *simple cycle* is defined here to be a loop without repeating edges or nodes.

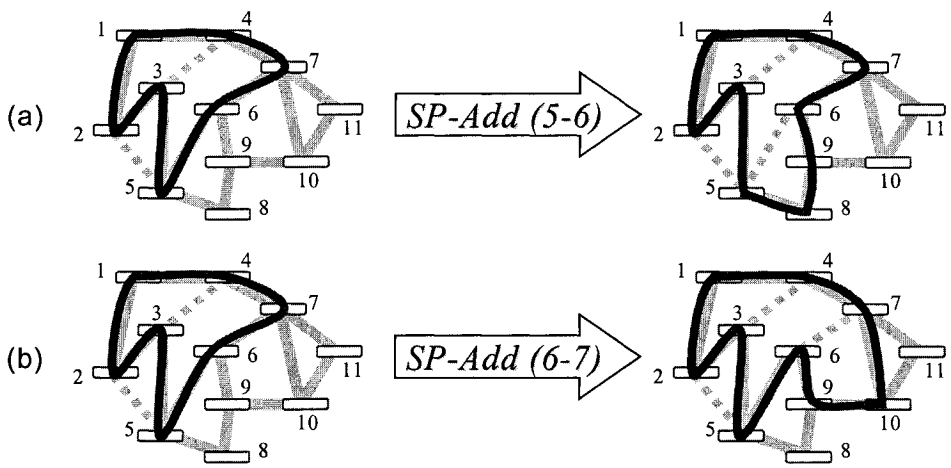



**Figure 4.5: The *SP-Add* Operation for Primary Cycle-2**

Figure 4.5 illustrates the concept of a simple cycle and the operation of *SP-Add* for primary cycle *a* of Figure 4.1. We have indexed the nodes in Figure 4.1 to facilitate description. We can ignore span (3,4) because it is the original straddling span of the cycle, i.e. it merges the 2nd shortest path (3-2-1-4) and the cycle-disjoint shortest path ( 3-5-6-7-4) between node 3 and node 4. For node 5 and node 6, we can find the cycle-disjoint SP 5-8-9-6 (disjoint with the cycle 5-6-7-4-1-2-3-5) that can be merged with path 5-3-2-1-4-7-6 to form a simple cycle (Figure 4.5(a)). The SP between node 6 and node 7, path 6-9-10-7, can also be merged with the path 5-3-2-1-4-7-6 to form a simple cycle (See Fig 4.5(b)).

On the other hand, we find no cycle-disjoint SP between nodes 1 and 2 so that we can merge it with cycle *a* to form a simple cycle. The same discussion fits for node 1 and 4 (node 3

and 5, 4 and 7). If we implement the above algorithm, recounting the same cycles may happen because the design of the algorithm. For example, in Figure 4.1, two cycles (cycle (a) and cycle (c)) generated by two different spans are the same one.

```
Save_Cycle (NewCycle, CycleSet)
{
If CycleSet== NULL
  Add NewCycle to CycleSet;
  Find the CycleSet1 from CycleSet, in which the length of cycles is same with NewCycle;
  For every cycle p in CycleSet1
  {
        forward_route = {node on p enumerated in the formal order};
        backward_route = {node on p enumerated in the backward order};
        If the first node of NewCycle belongs to p;
        If not,
        If every other node on NewCycle is exactly the same as that in backward_route orderly
                        If not, add the cycle in the CycleSet;
                                Else continue;
                        End If
        End If
  }
}
```

**Figure 4.6: Pseudo-code of the *Save_Cycle* Algorithm**

We can now describe the *SP-Add* algorithm by the pseudo-code in Figure 4.7.

```
SP-Add(OriginalCycleSet)
{
/*  NewCycleSet is the name space for the new generated cycles. Its initial value is empty set. */
Initialize NewCycleSet
For each cycle p in OriginalCycleSet{
   Mark all spans and nodes on cycle p
   For each span i on cycle p {
       Set route r0 = {the on-cycle nodes on the directed route between two nodes of span i}
       Dijkstra (i, unmarked spans/nodes)->r'
       If r'! = NULL returns the route r'
       {
       Add route r' to route r0 to get route r
       If route r is simple cycle
       Save_Cycle( x, NewCycleSet);
        }
     }
   Unmark all spans and nodes
  }
  Return NewCycleSet
}

Dijkstra()
{
  Find the shortest route r between the end nodes of i using unmarked spans and nodes only
  Return route r;
}
```

**Figure 4.7: Pseudo-code of the *SP-Add* Algorithm**

The inputs of the function are the set of primary $p$-cycles, the adjacent list of the topology, the number of the nodes and spans, and the primary cycle set based on SLA, while the output is the merged cycles combining the spans on the primary cycle and the cycle-disjoint shortest path between the end nodes of an on-cycle span. To implement the algorithm, we start with an existing set of cycles, and for each cycle taken one cycle at a time, we visit each span in the cycle to see if a route that is node-disjoint from the cycle exists between the span's end nodes. If so, we create a new cycle as described above, and move on to the next span on the cycle, and so on for each cycle. The function returns the new set of cycles created, at most one new cycle for every pairing of original cycle and span on the cycle.

Note that the *Add* operation discussed before becomes a special case here. The simple cycle requirement can guarantee that no "stub" spans, which are like the star node in Figure 4.3, exist.

### 4.4.2 The *SP-Join* Operation

The *SP-Join* operation is similar to the *SP-Add* except that after we merged a cycle-disjoint route to the cycle, we do not stop and continue to start the first span of the new cycle use the *SP-Add* one more time and seeking another span that we can transform into a straddler in the same manner. We do this until we have visited every span in the original cycle. Each time we expand the cycle by adding a route to it, we ensure that it is not only cycle-disjoint from the original cycle, but also to every previous route we have added.

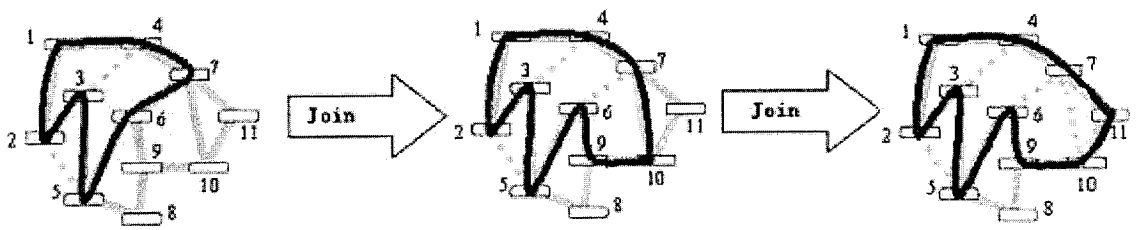



**Figure 4.8: The *SP-Join* Operation for Primary Cycles**

Figure 4.8 illustrates the concept of a simple cycle and the operation of *SP-Join* ( Join operation using Shortest Path) for primary cycle *a* of Fig. 4.1. For example, as we have discussed in the *SP-Add* operation, the SP 6-9-10-7 can be merged with cycle *c* to form a simple cycle, meanwhile the SP (10-11-7) between node 7 and node 10 can be merged again to the current cycle path 6-5-3-2-1-4-7 to form a simple cycle, we keep on the procedure until we can not

merge with more cycle-joint path to the cycle. Therefore we finally obtain the cycle 1-2-3-5-6-9-10-11-7-4-1.

Beside the cycles generated by iteratively implementing *SP-Add*, we also add the primary cycles to the cycle candidate set, and finally we add the set of faces to the accumulated set of candidate cycles for a subsequent capacity planning design. In the SP-Join algorithm, we don't explicitly produce the faces themselves, but rather we find for each span the shortest cycle for which it is an on-cycle span. While this will not necessarily produce the set of faces in all cases, it is equivalent, or nearly so, in most real transport networks. The purpose is only enrichment of the candidate cycle set. Note that the France network is actually nonplanar, so strictly speaking, a complete set of faces does not exist.

```
SP-Join (OriginalCycleSetA)
{
Initialize SPAddCycleSetB
  SPAddCycleSetB = SP-Add (OriginalCycleSetA)
  Initialize NewCycleSet (Comments: the NewCycleSet is the set containing new generating cycles)
  For each cycle p in SPAddCycleSetB {
    Set route r0 = {the on-cycle nodes on the directed route between two nodes of span i}
    For each span i on route r0 {
      Mark all spans and nodes on r0
      Add_Cycle (route r0)
      Unmark all spans and nodes
    }
  }
  Add OriginalCycleSetA to NewCycleSet
  Add SPAddCycleSetB to NewCycleSet
}
Add_Cycle (route r0)
{
Mark all spans and nodes on route r0
For each span i on route r0 {
      Dijkstra (i, unmarked spans/nodes)->r'
    If r'! = NULL returns the route r' {
      Add route r' to route r0 to get route r
      If route r is simple cycle
      Save_Cycle (r, NewCycleSet)
    Unmark all spans and nodes
    }
    Add_Cycle (route r)
  }
}
```

**Figure 4.9: Pseudo-code of the *SP-Join* Algorithm**

We use a recursive function *Add-Cycle* to implement the iterative operation. SP-Join can be described by the Pseudo-code in Fig 4.9.

### 4.4.3 The *SP-Merge* Operation

If we can merge where possible all shortest paths between two end nodes of an on-cyle span, we can avoid even more unnecessary duplication of coverage by a set of primary *p-cycles*.

Take Figure. 4.10 as an example, and consider the primary cycle A-C-B-E-D-A that was formed by combining path A-C-B and path A-D-E-B, the two edge-disjoint shortest paths of the straddler AB.

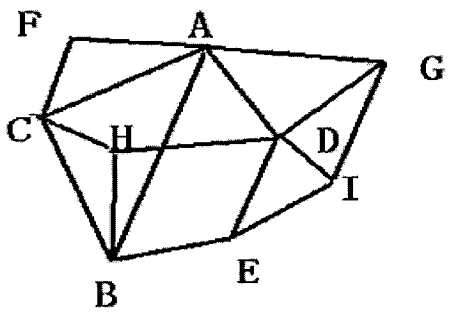



**Figure 4.10: Merge the shortest paths on cycle ACBEDA**

```
SP-Merge(OriginalCycleSet)
{
Initialize NewCycleSet
For each cycle p in OriginalCycleSet {
  Set route r0 = {the on-cycle nodes on the directed route between two nodes of span i}
  Mark all spans and nodes on route r0
  For each span i on route r0{
    Dijkstra (i, unmarked spans/nodes) -> r'
    If r'! = NULL returns the route r' {
      Add route r' to route r0 get the route r
      Mark all spans and nodes on route r
      If route r is simple cycle
      Save_Cycle( r, NewCycleSet)
      }
    }
    Unmark all spans and nodes
  }
  Return NewCycleSet
}
```

**Figure 4.11: Pseudo-code of the Algorithm *SP-Merge***

For each on-cycle span, we can search out the shortest paths between its two end-points. Then we merge the path that can form a simple cycle with the on-cycle span. For example, we

can merge A-F-C, the shortest path of span AC with the primary cycle A-C-B-E-D-A. Likewise, we merge C-H-B, the shortest path for span CB; we merge E-I-D for span ED; and finally D-G-A for DA. Consequently, we obtain the cycle A-F-C-H-B-E-I-D-G-A that can cover all nodes in the graph, and the *p*-cycle score is improved from (5+2)/5 =1.4 to (9+14)/9 = 2.6. Motivated by the above observation, we propose the *SP-Merge* algorithm presented in Figure 4.11. In the algorithm we also save the intermediate generated cycles to extend the candidate cycles.

## 4. 5 Performance Analysis

We test the 3 algorithms, *SP-Add*, *SP-Join* and *SP-Merge,* based on the topologies of USA Long Haul network and the France Telecom network. We tabulated the results to compare the algorithm and analysis of the size of the candidate cycle sets and time complexity.

The USA Long Haul network is a 28-node, 45-span network [MuFi97], and the France Telecom network is a 43-node, 71-span European long haul network inspired by France Telecom's network [FrTe03]. Both networks have sets of working span capacities that arise from shortest-path mapping of one lightpath requirement between each node pair. This produces working channel counts on spans ranging from one to 97, and averaging 28.3 in the USA network and ranging from one to 224, and averaging 54.5 in the France network.

**Table 4.2: Performance Comparison of 4 Different Operations**

| | **USA** | | | | France | | | |
|---|---|---|---|---|---|---|---|---|
| | **SLA** | **SP-Add** | **SP-Merge** | **SP-Join** | **SLA** | **SP-Add** | **SP-Merge** | **SP-Join** |
| Average *p*-cycle length | 6.25 | 7.63 | 9.07 | 14.71 | 6.26 | 7.61 | 9.13 | 18.25 |
| AE (*p*) | 1.37 | 1.6 | 1.53 | 1.80 | 1.39 | 1.41 | 1.54 | 1.96 |
| Number of *p*-cycles | 24 | 57 | 102 | 374 | 34 | 89 | 154 | 879 |

### 4.5.1 AE (*p*) and the Average Cycle Length

In Table 4.2, we compare the performance of *SP-Add*, *SP-Join* and *SP-Merge* on the sets of primary cycles in the two test case networks, in terms of average *p*-cycle length (in hops) and average unit cost *p*-cycle efficiencies (AE). From the above results, we can see that *SP-Join* can obtain the best *AE(p)*, which improves the SLA by 25% for USA and 41% for France.

### 4. 5.2 Size of the Candidate Cycle Set

Based on the SLA design, we can claim that the upper bound of the number of cycles obtained is *S*. With no more than *S* primary cycles, *SP-Add* and *SP-Merge* can generate *SN* (number of cycle multiples the length of possible longest cycle) cycles at most, and in the worst case, *SP-Join* will generate no more than *SN(N-5)* cycles, because the SLA forms a cycle from at least 4 nodes, and the *SP-Add* from at least 5 nodes. Actually the number of *p*-cycles produced by *SP-Join* will be much less because:

1. If one of the lengths of shortest paths is more than 1, we will use less steps than *N-5* steps;
2. The function ***Save_Cycle()*** can be used to avoid re-enumerating the same cycles produced by different straddlers;
3. **There are many on-cycle spans for which node-disjoint shortest path between its end nodes does not exist.**

The above pre-selection operations significantly reduce the number of *p*-cycles as candidates. For example, 7321 cycles can be found in USA network using ILP, about 15 (~7321/495) times of that of using *SP-Join*.

Since SLA uses Dijkstra's shortest path Algorithm of which the time complexity is $O(S+N \log N)$, SLA has a complexity of $O(S^2 + SN \log N)$.

From the design discussed above, we know every cycle is generated using Dijkstra SP algorithm exactly once, therefore the number of cycles the algorithm generated determines how many times the Dijkstra's SP algorithm is executed. Consequently, the time complexity of the algorithm is the product of the number of cycles and complexity of Dijkstra's SP algorithm. Finally, the complexity of *SP-Add* and the *SP-Merge* is $(NS^3 + S^2 N^2 \log N))$, because as discussed *SP-Add* and the *SP-Merge* can generate at most *SN* cycles. The complexity of *SP-Join* is $O[(NS^3 + S^2 N^2 \log N)*(N-5)]$, because the number of *p*-cycle is *SN (N-5)* at most.

In summary, all three operations have a polynomial time complexity.

### 4. 5.3 The Pre-Select ILP Model: An Optimality Example

To test if our heuristic algorithm works well, we incorporate the SCP (*p*-Cycle Spare Capacity Placement) ILP model [GrSt00] with the operations we developed above to build up our PSILP (P re-Select ILP) model. We then run it in a capacitated network, and use the output in an

optimal model to plan the spare capacity according to the real working link distribution. Finally, we can compare the results with those obtained from the pure optimal cycle selection algorithm. Here we shall take the *SP-Join* as an illustration because the procedure can produce more candidate cycles. We can obtain the optimal *p*-cycle set when we use these candidate cycles as input to the ILP (Integer Linear Programming) as show in Equation. (4.1) and presented as SCP in [GrSt00].

Let $P$ be the set of p-cycles, $E$ the set of all network spans, and $S = |E|$ the number of network spans. Let $s_j$ and $w_j$ be respectively the number of spare and working links on span *j*. Let $n_i$ be the number of unit capacity copies of cycle *i* in the *p*-cycle design. The parameter $p_{i,j}$ is the number of spare links required on span *j* to build a copy of *p*-cycle *i*. The value of $p_{i,j}$ is 1 if cycle *i* passes over span *j*; otherwise it is 0. The parameter $c_j$ is the cost of span *j*. The parameter $x_{i,j}$ is the number of paths that a single copy of *p*-cycle *i* provides for restoration of span *j*. The parameters $x_{i,j}$ and $p_{i,j}$ are evaluated in advance for each cycle in ***P***. Note that $x_{i,j}$ can be either 0,1 or 2 for every *i, j*. It is zero if either or both of the end nodes of prospective failure span *j* are not on the cycle *i*, it is 1 if both of the span end nodes are on the cycle and are adjacent along the cycle. The value of $x_{i,j}$ is 2 if both of the failure span end-nodes are on the cycle but they are not adjacent to one adjacent to one another on the cycle, i.e. span *j* has a straddling relationship to cycle *i*.

$$\text{Minimize} \sum_{j=1}^{S} c_j s_j$$

s.t.

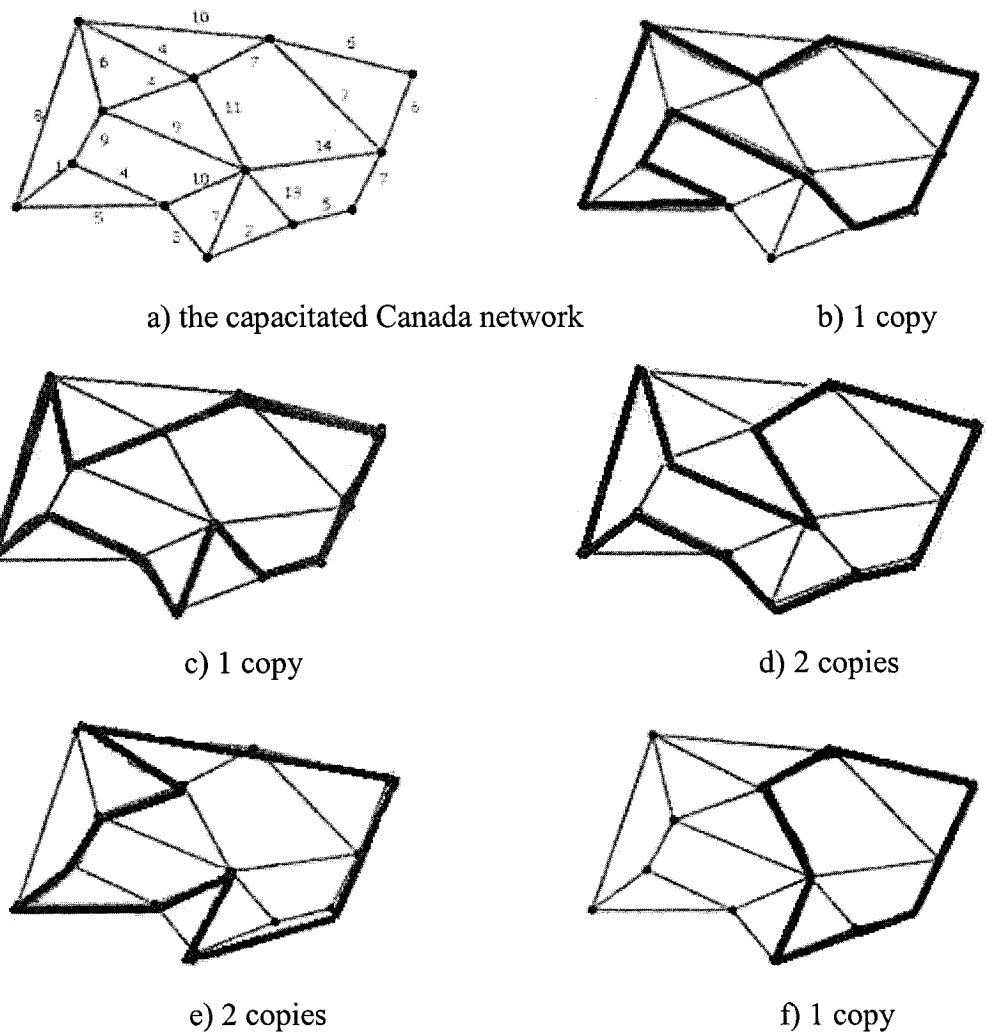
$$s_j = \sum_{i=1}^{|P|} p_{i,j} \cdot n_i \qquad \forall (j \in E)$$

$$w_j \le \sum_{i=1}^{|P|} x_{i,j} \cdot n_i \qquad \forall (j \in E)$$

$$n_i \ge 0 \qquad \forall i = 1,2,\Lambda\ |P| \qquad (4.1)$$

The traditional pure ILP is to find all of the possible cycles using FDS (First Depth Search) algorithm, and plan the capacity according to SCP. We take Canada network as an example (see Figure 2.4).

If we use the Pure ILP by using DFS algorithms, we can find 410 cycles. Based on the cycle set, we can get 5 different optimal cycles if we implement the SCP model. The results of the computation simulation can be seen in Figure 4.12 where one obtains a logical redundancy of 53.8% by implementing pure ILP. The solution composes seven unit-capacity *p*-cycles (we call a unit capacity as a copy) using five distinct cycles. Three of them (c, d and e) are Hamiltonian cycles, the other two (b and f) are not. The number of copies is the number of unity-capacity *p*-cycles instantiated on each of the cycles shown.



a) the capacitated Canada network b) 1 copy

c) 1 copy d) 2 copies

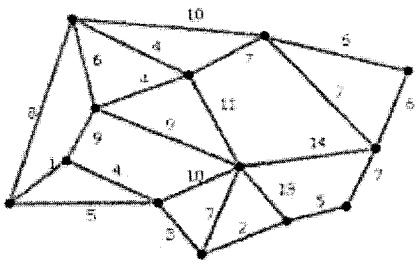
e) 2 copies f) 1 copy

**Figure 4.12: Pure ILP Optimal Capacitated p-cycle Design**

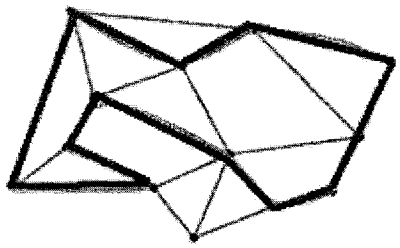
We can obtain the optimal redundancy by using the *SP-Join* algorithm, but without enumerating all of the possible cycles. Using the *SP-Join* for Canada networks, we can obtain 127 cycles, from which we can implement four distinct cycles by using seven unit-capacity *p*-cycles as shown in Fig 4.13. The logical redundancy is also 53.8%. Since the size of the

candidate set has decreased from 410 to 127 cycles (decreased to 31%), the complexity is consequently decreased. This example shows how efficient our heuristic algorithm can be.

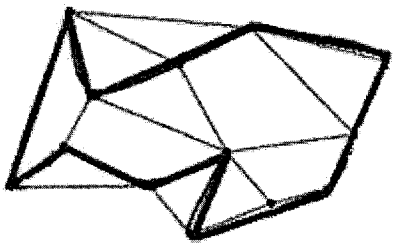
Below we list the results for other networks in Chapter 5.



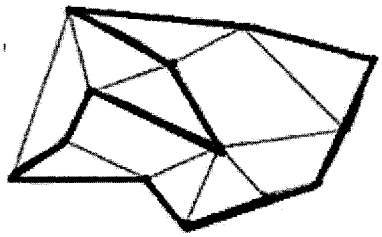
a) the Capacitated Canada Network



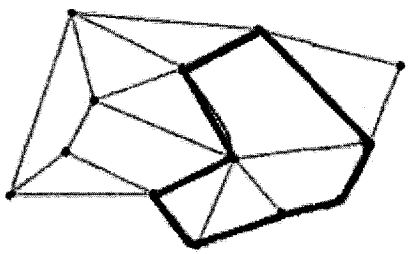
b) 2 copies



c) 3 copies



d) 1 copy



e) 1 copy

**Figure 4.13: *SP-Join* ILP Optimal capacitated p-cycle design**

Unlike using SP-Join, we cannot obtain optimal results by implementing SP-Add and SP-Merge. We got only 9 cycles from SP-Add and a redundancy of 74.05% under the same assumption of working capacity. From SP-Merge, we only obtained 8 cycles and redundancy is 64.56%. The results for other networks are listed in Chapter 5

## 4.6 Concluding Remark

To make the heuristic pre-selection more efficient for the capacitated network, we have introduced in this chapter three operations based on the SLA algorithm: *SP-Add*, *SP-Join* and *SP-Merge*. By producing non-primary *p*-cycles with more straddling span relationships, and hence network capacity efficiency, the study allows us to appreciate the unnecessary duplication of coverage that a set of primary *p*-cycles may involve and to seek out further methods to improve the *p*-cycle score.

We tested the operations to get some performances such as the size of the candidate set, the average length, and the scores. Through our SPILP exercise, we have also shown that *SP-Join* operation is very efficient because the redundancy can be up to the same with using the pure ILP. Comparing to SP-Join, redundancy by using SP-Add and SP-Merge are 74.05% and 64.56% respectively. Together with the comparison results from Chapter 5, we shall demonstrate that the SP-Join provides the best performance.

# Chapter 5
# *P*-Cycle Algorithm for Capacitated Networks

Since many Integer Linear Programming (ILP) methods for *p*-cycle network design can only handle small or medium-sized networks, we develop in this chapter a developed, heuristic, capacitated, iterative, design algorithm for use in large networks. And we make comparison between two different algorithms.

## 5.1 Design Consideration

In an actual design where working capacities differ on each span, one can expect to see *p*-cycles of different sizes, not only the cycles of high a priori efficiency because true efficiency is essentially dependent on protection of working capacity not protected by other *p*-cycles in the design. We focus on the heuristic solution of spare capacity planning problem described in the following:

Input: *G(N, S)*, topology of the network with ***N*** the set of nodes and ***S*** the set of spans; $s$ is the number of spans and $n$ is number of nodes; and the *p*-cycle candidate set is $P$; the working capacity distribution for every spans is $W= \{ w_j,\ 0<=j<=s,$ where $w_j$ is the working capacity of span $j\}$.

Output: The spare capacity distribution $\{ s_j,\ 0<=j<=s,$ where $s_j$ is the spare capacity of span $j\}$

Constraints: The spare capacity cannot be greater than the working capacity for every span in the network.

Take the Canada network in Figure 4.12 as an example, we can see only three of the five cycles are the Hamiltonian (which has the best *AE (p)* = 2.54), where the other two are not. However, they are actually the key part of an economic solution because of differential working capacity considerations. Smaller but more precisely placed *p*-cycles are required to match the actual capacity distribution to be protected, thus complementing the Hamiltonians.

In view of this, we shall first introduce two more meaningful efficiencies: the *weighted capacity efficiency* $E_w$, and the *unity capacity efficiency* $E_u$. These new measures give us not only the evaluation of a *p*-cycle's ability to protect hypothetical working capacity, but also give us an indication of a *p*-cycle's suitability to a specific working capacity state. There could be

other efficiencies, but here we just tried two known measures. The difference of them is $E_u$ is not working capacity weighted measure; while in $E_w$ we consider the working capacity as a weight. For example, if two $p$-cycles have common score, if we use $E_u$ as efficiency, we can randomly choose one of them, but according to the definition of $E_w$, we will choose the $p$-cycle which can protect more working capacity. In the specific case, the unit working capacity model, it is obviously that the two efficiencies are the same.

The formulas are the following, respectively. Let $S$ be the set of spans in the network, $c_i$ be the cost of a unit of capacity on span $i$, and $X_{p,i}$ be the number of protection relationships available to span $i$ from $p$-cycle $p$ ($X_{p,i} = 1$ if $i$ is an on-cycle span, and $X_{p,i} = 2$ if $i$ is straddling span). Then the weighted capacity efficiency is

$$E_w(p) = \frac{\sum_{\forall i \in S} w_i X_{p,j}}{\sum_{\forall i \in S | X_{p,j}=1} c_i} \quad . \qquad (5.1)$$

In (5.1), $w_i$ is the amount of unprotected working capacity on span $i$ at the present time. Also we have the unity capacity efficiency of the p-cycle,

$$E_u(p) = \frac{\sum_{\forall i \in S} \alpha_i X_{p,j}}{\sum_{\forall i \in S | X_{p,j}=1} c_i} \quad . \qquad (5.2)$$

In (5.2), $\alpha_i = 1$ if the current working capacity on the span $i$ is not zero, and $\alpha_i = 0$ otherwise.

Note that $E_w$ considers capacity as the weight in the efficiency formula (5.1); it is possible not to only guess a $p$-cycle's ability to protect the hypothetical working capacity, but also to determine the actual suitability in a specific working capacity. That is, cycles that can provide more working capacity should have a higher priority to be chosen. In view of this efficiency $E_w$ is more effective than $E_u$. Equation (2.1) is the special case for $E_w$ if we assume $w_i (i=1, 2, \ldots s)$ always is 1, and also special case for $E_u$ if we assume the working capacity on each span is not zero.

Taking advantage of the two efficiencies, our heuristic algorithms will be designed to protect one unit working capacity every time using the cycle. After that, remove the working capacity that has been protected and consider the unprotected working capacity in the same way until all of the working capacity will be protected.

## 5.2 Efficiency-Based Algorithms

The two different efficiencies can be used to implement our WCIDA which stands for the Working-Capacity-based Iterative *p*-cycle network Design Algorithm (*WCIDA*) according to the two different efficiencies. The basic steps are as follows:

(1) Enumerate all of the candidate cycles according to the algorithm described in Chapter 4 (e.g. *SP-Join* or *SP-Merge,* etc) and determined a unity or non-unity working capacity on each span;
(2) For each candidate cycle, calculate the actual efficiency (could be either of $E_w$ or $E_u$) of each *p*-cycle;
(3) Select the *p*-cycle with the maximal actual efficiency (the best cycle), if there is more than one cycle with the same maximal efficiency, then randomly choose one;
(4) Subtract one unit from each on-cycle span in the cycle we just placed, and two from each straddler, which updates the working capacity values of the network. These new working capacity values represent the amount of as yet unprotected working capacity on each span;
(5) Repeat step (2) through (4) until the working capacity in each span is reduced to 0;
(6) Obtain the redundancy from the total number of copies of the spare links and copies of the working links.

```
WCIDA(X)
{
Initialize CycleSet, work[], and CycleUse[]
CycleSet = EnumerateCycles()
While work[i] > 0 for all spans i
{
  BestCycle = 0
  For each cycle p in CycleSet
  {
    Calculate X(p)
    If X(p) > X(BestCycle)
    {
      BestCycle = p
    }
  }
  CycleUse[BestCycle] = CycleUse[BestCycle] + 1
  For each on-cycle span i in BestCycle
  {
    work[i] = work[i] - 1
  }
  For each straddling span i in BestCycle
  {
    work[i] = work[i] - 2
  }
}
Return CycleSet and CycleUse
}
```

**Figure 5.1: Pseudo-Code of *WCIDA(X)***

The pseudo-code of the algorithm using the efficiency *X(p)* (either $E_w(p)$ or $E_u(p)$) is in Figure 5.1. In the pseudo-code, the ***EnumerateCycles***() function is merely a space-holder for whatever

method we wish to use to enumerate a set of cycles. Since we use the output of *SP-Merge* or *SP-Join* as the input for *WCIDA(X),* we have the following variations: *Join-WCIDA* and *Merge-WCIDA* output of *SP-Merge* or *SP-Join* as inputs.

For example, *Join-WCIDA(X)* denotes the approach where we use *WCIDA(X)* to plan the spare capacity for the network from the candidates generated by *SP-Join.*

## 5.3 Performance Evaluation

We implement the two capacity planning algorithms *WCIDA* ($E_w$) and *WCIDA* ($E_u$) in two difference models: one is identical working capacity network, and the other is non-identical working capacity network. Four different networks (Canada, USA, France and Europe) have been tested based on the candidates generated from *SP-Merge* and *SP- Join*, respectively. After that, the heuristic and ILP algorithms are compared in terms of complexity and redundancy.

We implement the ILP model using AMPL and CPLEX 7.0. Here we test 4 networks (Canada, USA, France, and Europe), and we run the simulations of our algorithms on 1.2 MHz Intel processor using MS Windows 2000 with 2GB of Ram. We run the simulations of our algorithms on 1.2 MHz Intel processor using MS Windows 2000 with 2GB of RAM.

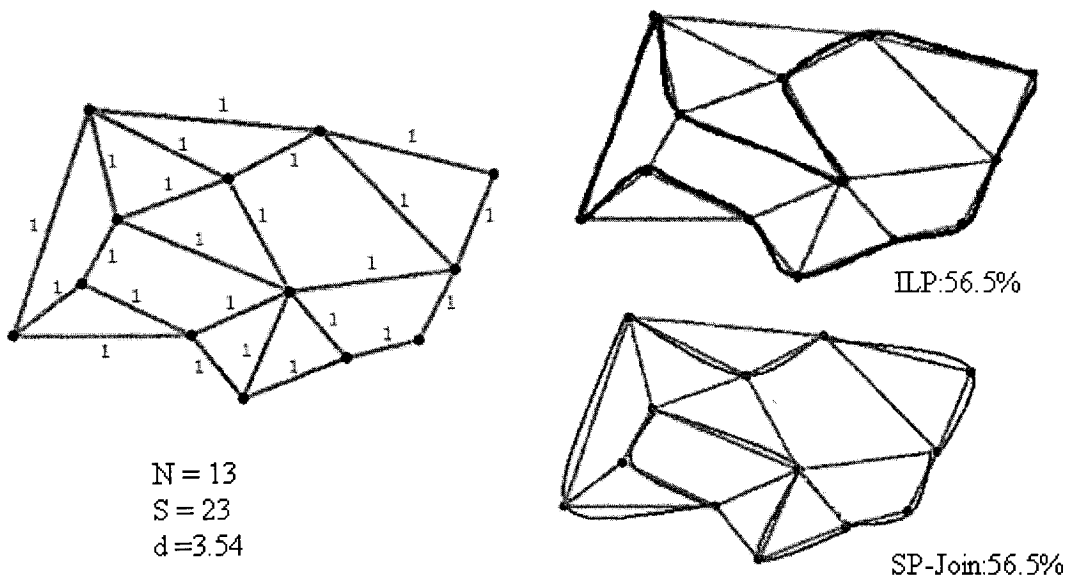



**Figure 5.2: *p*-Cycle Solutions for an Identical-Working Hamiltonian Network**

### 5.3. 1 Identical Working Capacity Network

For the networks with unit working capacity, it is obvious that we can obtain the same redundancy for $X= E_w$ and $X= E_u$ because the working capacity is 1 for every span. According to the definition of $E_w$ and $E_u$, we can easily find that $E_w = E_u$.

We take the Canada network in Fig. 5.2 as an example. We first run the ILP to obtain the optimal solution, which happened to be a Hamiltonian cycle as well. We obtained a redundancy 56.5%. We then tried *Join-WCIDA(X)* and *Merge-WCIDA(X)* for $X= E_w$ and $X= E_u$ respectively), and found the *Join-WCIDA* ($E_w$) algorithm can provide the optimum performance. Although the Hamiltonian cycles are different obtained by using SP-Join and ILP, they provide the same redundancy because of the property of Hamiltonian cycles.

**Table 5.1: Heuristic *p*-Cycle Solutions in Identical-Working Test Networks Using Merge/Join-*WCIDA* ($E_w$)**

| Network | Canada | | USA | | France | | Europe | |
|---|---|---|---|---|---|---|---|---|
| | *SP-Merge* | *SP-Join* | *SP-Merge* | *SP-Join* | *SP-Merge* | *SP-Join* | *SP-Merge* | *SP-Join* |
| Spare/working | 13/23 | 13/23 | 36/45 | 37/45 | 67/71 | 55/71 | 11/26 | 11/26 |
| Hamiltonian | Yes | Yes | No | No | No | No | Yes | Yes |
| Redundancy | 56.5% | 56.5% | 80% | 82.22% | 94.37% | 77.5% | 42.31% | 42.31% |

We have tabulated the results for 4 different networks in Table 5.1. Take Canada network as an example in Table 5.1. Since we assume it to be a unit working capacity model, we found the total working capacity is 23 (the number of span). Also whether we use *SP-Merge* (or *SP-Join*) or not, the result is still a Hamiltonian cycle, and the spare capacity is the number of nodes. The redundancy is therefore the same at 56.5%.

Case of Hamiltonian Cycle

From the above result, we appear to have found a Hamiltonian cycle a topology is the best solution with 100% restoration and observe that the least Hamiltonian and redundancy is *s/n*. The conclusion can be obtained theoretically below: i.e. the redundancy of Hamiltonian cycle is the lowest bound.

Consider now a Hamiltonian network of average nodal degree $d$, with $n$ nodes and $s$ spans. Every span has two end nodes, giving the average nodal degree $d = 2s/n$ or $s = n*d/2$ and $AE\ (p) = [n + 2*(s-n)]/n = (2s-n)/n$. According to the definition of a Hamiltonian cycle, the length of the cycle is the number of nodes; therefore the number of spare link on the cycle is equal to the number of nodes $n$. According to the working capacity distribution of the network, we know that every span reserves a channel for working, so the total working capacity is the number of the spans $s$.

Based on the above facts, a simple calculation of the logical (i.e., hop-count based) redundancy in this homogenous case would be:

$$\frac{\sum_{i=1}^{S} s_i}{\sum_{i=1}^{S} w_i} = \frac{n}{s} = \frac{n}{n*d/2} = \frac{2}{d}.$$

Thus, where applicable, a single Hamiltonian $p$-cycle is considered to be extremely efficient for networks with unit-working capacity. The result $(2/d)$ would be the lower bound on redundancy for a strictly homogenous network. However, in a general case without being strictly homogenous, a straddling span can actually carries twice as much working capacity as the capacity of the $p$-cycle that protects it. Therefore if we take this into account in the lower bound, the sum of working spans will be:

$$\sum_{i=1}^{S} w_i = n*1 + (s-n)*2 = n + 2*(n*d/2-n) = n(d-1).$$

So the redundancy is $\frac{\sum_{i=1}^{S} s_i}{\sum_{i=1}^{S} w_i} = \frac{n}{n(d-1)} = \frac{1}{d-1}$.

It is an NP-complete problem to check whether a topology is Hamiltonian. So far there are no efficient algorithms to find the Hamiltonian cycle in the network. But using our heuristic algorithm, The Hamiltonian cycle can sometimes be obtained (such as, by using *SP-Join* algorithm), but there is no guarantee, because there are no efficient algorithm can guarantee it even for the Hamiltonian graph.

### 5.3.2 Non-identical Working Capacity Network

For a non-identical case, we assume all demands take the shortest-path to produce the working

capacities on each span (also see Chapter 2). The traffic matrix of Canada network (in Table 5.2) is an example of the end-to-end demand distribution. In this matrix, the number 1 in $i^{th}$ row and $j^{th}$ column, stands for the unit traffic in between the node pair $i$ and $j$.

**Table 5.2: Traffic Demand Matrix for the Canada Network**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **2** | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **3** | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **4** | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **5** | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **6** | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **7** | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| **8** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| **9** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| **10** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| **11** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| **12** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| **13** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

**Table 5.3: Distribution Matrix of Working Capacity for the Canada Network**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 4 | 6 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 4 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 0 | 0 | 0 | 7 |
| **3** | 6 | 0 | 0 | 0 | 9 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 8 | 0 | 0 | 0 | 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 9 | 1 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 | 5 | 4 | 0 | 10 | 0 | 3 | 0 | 0 | 0 | 0 |
| **7** | 0 | 11 | 9 | 0 | 0 | 10 | 0 | 0 | 7 | 0 | 14 | 0 | 0 |
| **8** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 5 | 0 | 0 | 0 |
| **9** | 0 | 0 | 0 | 0 | 0 | 3 | 7 | 2 | 0 | 0 | 0 | 0 | 0 |
| **10** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 7 | 0 | 0 |
| **11** | 0 | 0 | 0 | 0 | 0 | 0 | 14 | 0 | 0 | 7 | 0 | 6 | 7 |
| **12** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 6 |
| **13** | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 6 | 0 |

Table 5.3 is the working capacity distribution matrix according to the traffic matrix using a shortest path routing. In the matrix of Table 5.3, the number in $i^{th}$ row and $j^{th}$ column is the working capacity on the span between node $i$ and $j$. Appendix B also provides the working capacity distribution of other networks.

**Table 5.4: Redundancy Comparisons between *WCIDA* ($E_w$) and *WCIDA* ($E_u$)**

| | Europ | Canada | USA | France |
|---|---|---|---|---|
| ADD-- *WCIDA*($E_w$) | 74.42% | 74.05% | 116.26% | 117.24% |
| ADD- *WCIDA*($E_u$) | 74.42% | 81.65% | 121.21% | 122.01% |
| MERGE- *WCIDA*($E_w$) | 53.49% | 64.56% | 102.51% | 110.6% |
| MERGE- *WCIDA*($E_u$) | 53.49% | 67.09% | 112.33% | 119.8% |
| JOIN- *WCIDA*($E_w$) | 47.67% | 56.96% | 94.89% | 98.89% |
| JOIN- *WCIDA*($E_u$) | 55.81% | 64.56% | 97.32% | 113.10% |

We test the algorithms *WCIDA* ($E_w$) and *WCIDA* ($E_u$) respectively using Canada network. We use 127 cycles produced by *SP-Join* algorithm in Canada network as the input *p*-cycle set. If we use $E_u$ as the efficiency, we can get 8 distinct *p*-cycles. The sum of spare capacity is 102, giving a redundancy of 64.56%. The maximum length of candidates is 13, and the minimum length is 4. If we use $E_w$ as the efficiency, we can get 7 distinct *p*-cycles, the sum of spare capacity is 90, and redundancy is 56.96%. The maximum length of candidates is 13, and the minimum length is 4.

We have also tried *Add-WCIDA*($E_w$), *Add-WCIDA*($E_u$), *Merge-WCIDA*($E_w$), *Merge-WCIDA*($E_u$), *Join-WCIDA*($E_w$) and *Join-WCIDA*($E_u$) in four networks: Canada, USA, Europe (cost 293), and France network. The simulation results can be found in Table 5.4.

We also built up the comparison charts in Figure 5.4 and 5.5. From the results, we can sum up the following conclusions based on the outputs:

1) Smaller but more precisely placed *p*-cycles could be the parts of candidates because of the differential working capacity considerations;
2) $E_w$ is more effective as the actual efficiency than $E_u$.

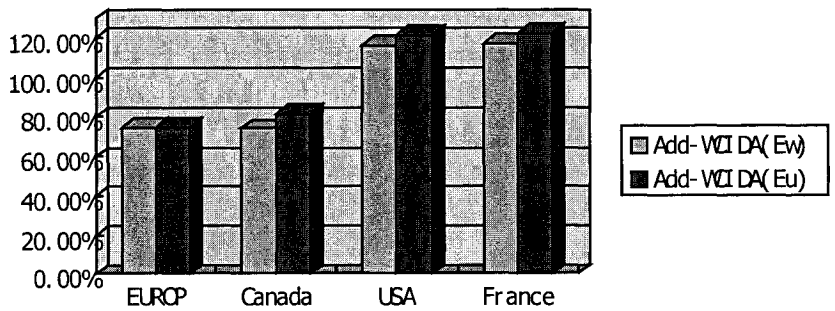



**Figure 5.3: Redundancy for Mon-unit Working Network using Add-WCIDA($E_w$) and Add-WCIDA($E_u$)**

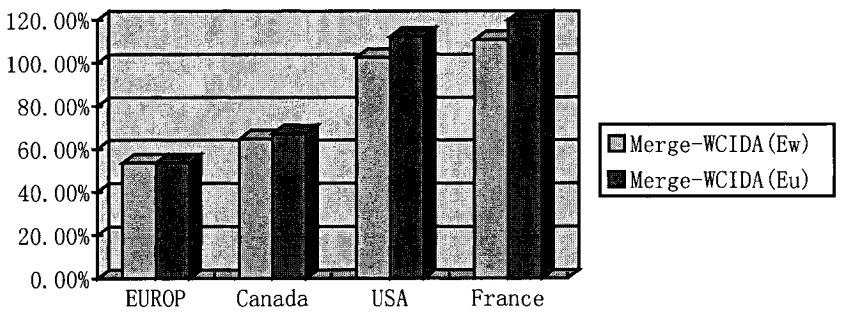



**Figure 5.4: Redundancy for Non-unit Working Network using *Merge-WCIDA($E_w$)* and *Merge-WCIDA($E_u$)***

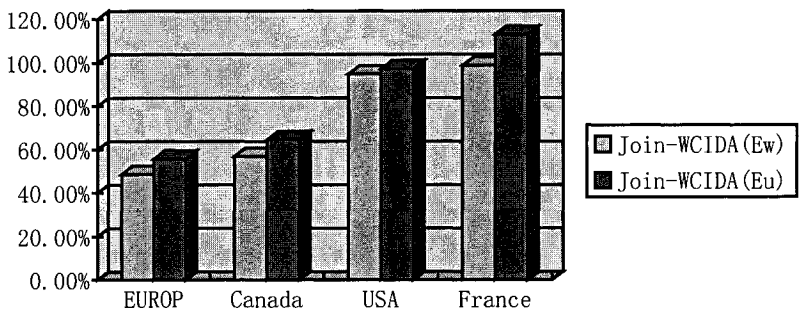



**Figure 5.5: Redundancy for Non-unit Working Network using *Join-WCIDA($E_w$)* and *Join-WCIDA($E_u$)***

Join-WCIDA ($E_w$) appears to have the lowest redundancy requirement among all networks. The plausible reason can that $E_w$ is more capacitated than $E_u$ because if there are two cycles with same $E_u$, we will randomly choose any of the cycles by using Join-WCIDA($E_u$), but

the cycle with more working capacity will have a higher priority by using Join-WCIDA($E_w$). Obviously the latter choice is more efficient. Therefore Join-WCIDA ($E_w$) can obtain better performance on the point of view of redundancy.

Another important finding is that the Hamiltonian cycle does not perform well for the networks with non-unit working capacities. Since the distribution of the working capacities is not even for this kind of networks, one Hamiltonian cycle can not guarantee 100% restoration. Although more copies of Hamiltonian cycles can guarantee 100% restoration, the trade-off is the increase in redundancy. Therefore considering different cycle varieties, especially smaller cycles, can usually obtain even more efficient p-cycles set. If we review the example in Section 4.5.3, we find there are four different cycles provide the optimum redundancy by using our PSILP algorithm, but only two of them are Hamiltonian cycles; the other two are not.

**Table 5.5: Comparison between Heuristic and Optimal Algorithm**

| Network | Canada | | USA | | France | | Europe | |
|---|---|---|---|---|---|---|---|---|
| | *SP-Merge* | *SP-Join* | *SP-Merge* | *SP-Join* | *SP-Merge* | *SP-Join* | *SP-Merge* | *SP-Join* |
| #of p-cycles | 70 | 127 | 102 | 374 | 154 | 879 | 119 | 383 |
| Min hops | 3 | 3 | 3 | 3 | 3 | 3 | 4 | 3 |
| Average hops | 7.4 | 8.6 | 9.1 | 14.7 | 9.5 | 18.3 | 6.3 | 8.0 |
| Max hops | 13 | 13 | 20 | 28 | 25 | 38 | 11 | 11 |
| Total working | 158 | | 1273 | | 4043 | | 86 | |
| Redundancy (ILP) | 53.8% | | 91.44% | | 96.19% | | 44.19% | |
| Redundancy (WCIDA) | 64.56% | 56.96% | 102.51% | 95.52% | 110.6% | 98.89% | 53.49% | 48.84% |
| Difference | 10.76% | 3.16% | 12.15% | 4.08% | 14.41% | 2.7% | 9.3% | 4.65% |

## 5.4 Comparison between Optimal and Heuristic Algorithms

Based on the conclusion of the last session, we consider using the *WCIDA ($E_w$)* algorithm for capacity planning. To compare the optimal and heuristic algorithms, we assume their input or the cycle candidates are of the same candidate group (say using *SP-Join*). See Table 5.5.

From the results, we find that the difference between the optimal and heuristic algorithms is no more than 5%. But the complexity of the heuristic algorithm is quite a bit lower than the optimal one. Considering the CPU time, the difference is not too much, for example, for Canada network, the CPU time of both of ILP and heuristic (*WCIDA* ($E_w$)) algorithm are 130 ms, and the reason is the size of input is quite small.

As discussed before, the heuristic algorithm is faster than the optimal algorithm, because of the complexity. The time complexity of *WCIDA* algorithm is of the same order as the number of cycles, i.e. *O(C)*, where *C* is the number of *p*-cycles. Therefore if we take the *SP-Join* as the *p*-cycle producing algorithm, the complexity of *WCIDA* will be $O(SN^2)$. However the ILP is a NP-hard problem.

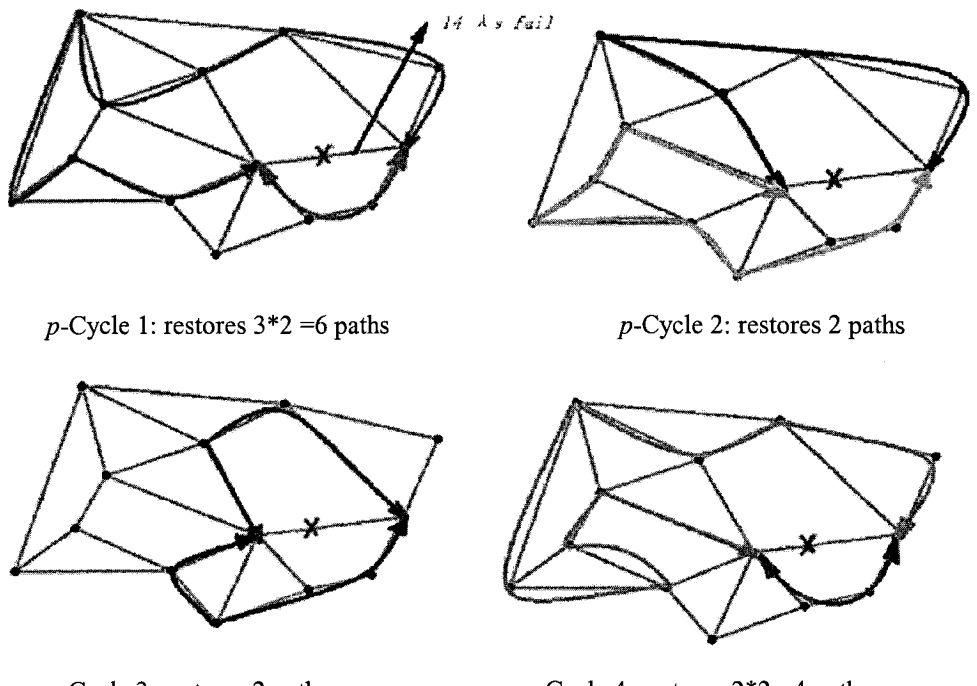
```
Specific Cycle()
{
For every span S_i
        {
        Initial StraddlerSet_i (); OncycleSet_i ()
        While (working[i]>0)
        {
        For every p-cycle P_j
        {
                If S_i is a straddler of P_j
                working = working[i] –2;
                Save P_j in StraddlerSet_i ();
                If working[i] <=0
                Return;
        }
        if working[i] >0
        For every p-cycle P_j
        {
                if S_i is a on-cycle span of P_j
                working = working[i]-1;
                Save P_j in OncycleSet_i ();
                If working[i] <=0
                Return;
        }
}
}
```

**Figure 5.6: Pseudo code of enumeration of protection for specific span**

## 5.5 Real time Operation

What we discussed now is to provide (pre-plan) specific $p$-cycles with a single span failure. It is called real-time because we want to recover in real-time whenever and wherever this span failure occurs.

For all candidate cycles, we test every span $s_i$ to see if it is one of straddlers in the candidate cycle, if so we save the cycle as one of the protection cycles of $s_i$, and reduce 2 copies links of $s_i$. When we try all of the candidate cycles, and the span is no longer a straddler for all candidate cycle and not all of the working capacity in the span have been protected, we will try all the cycle to see if the span is on-cycle for every cycle, if so we save the cycle as protection cycle of $s_i$, and reduce one copy for the total working capacity, until all the span can be saved. This process happens after the *WCIDA* algorithm to use the obtained cycles. The pseudo code is the following Fig 5.6: In this algorithm, we will give the higher priority to $p$-cycle on which the specific span is of the straddler relationship because the span can protect two copies using one copy of spare capacity.



$p$-Cycle 1: restores 3*2 =6 paths

$p$-Cycle 2: restores 2 paths

$p$-Cycle 3: restores 2 paths

$p$-Cycle 4: restores 2*2 =4 paths

**Figure 5.7: Illustrating of the real time phase**

Figure 5.7 is an example to illustrate how to choose efficient *p*-cycles to protect a failed span. In this figure, the failed span has 14 copies working capacities to be recovered. Using the above algorithm, we find 5 *p*-cycles from which the span can be protected 100%, where *p*-cycle #1 provides 6 protection paths for the failure spans, *p*-cycle #2 provides 2 paths, *p*-cycle #3 provides 2, and *p*-cycle #4 provides 4 as well, so the total protection paths is 6+2+2+4= 14.

## 5.6 Concluding Remark

In this chapter, we propose the heuristic algorithm *WCIDE* to plan the spare capacity of the network. Two efficiency algorithms have been tested based on the real networks. We find that the $E_w$ is better than $E_u$ as a parameter of the heuristic algorithm through comparing the redundancy under the same condition for different network.

Another conclusion we can draw is that the *SP-Join* is generally better than *SP-Merge*, because the former can get more efficient *p*-cycles as candidates, so it is closer to the optimal results. Because the difference of redundancy between ILP and *SP-Join-WCIDA* ($E_w$) is very small, less than 5%, and the heuristic algorithm is of lower time complexity, we can claim the *SP-Join-WCIDA* ($E_w$) is feasible and efficient.

# Chapter 6
# Design Guideline

Our research has provided a relatively complete design of optical failure detection and protection scheme. We consider a general DWDM network with a general mesh topology in the thesis. The cycle protection is always a kind logical scheme that can be implemented at the fiber layer, the DWDM or OC-n channel layer, etc. The following are three design guidelines for using algorithms for different goals in network planning and evaluations.

## 6.1 For Failure Detection with Limited Capacity

If detection is the only objective, the MSCC algorithm is a good choice, especially for the small unit-working capacity networks. We allocate a spare capacity for failure monitors. Once the single failure being detected by the cycle monitors no matter where the failure is, on the spans or on the nodes of a network, we can pre-configure the detection cycles to detect it, and then the spare channel can be shared to protect the failure. The main point of the MSCC algorithm is to find a min-cost cycle cover of the network so that every span relies on one of the cycles. The design can be applied to the non-unit working capacity network, if the failure happens to one link of the span.

The disadvantage of MSCC is the cost or the redundancy for the protection; in general, the redundancy of the networks is more than 100%. So if we just consider the purpose of protection, the ring speed and mesh efficiency design, *p*-cycle, will be a better solution because of its efficiency.

Requirements of Current Networks:

Size of the network: small networks (# of nodes <=20, # of links <= 30),

Working capacity: not special requirement;

Spare capacity: at least one;

Connectivity: bi-connected;

Design Procedure:

1. Check if the network is the Eulerian, if so decompose the network to small cycles by removing the span and the shortest path between the two end node in the remaining graph;
2. If the network is non-Eulerian, implement the MSCC and find the cycle cover for the network;
3. Reserve a channel along every cycle from the cover for monitor;
4. Once a failure happens, trigger the monitoring procedure; let the monitor route along the cycle to locate the failure.

## 6.2 For Failure Detection/Protection of Networks with Limited Capacity

If both detection and protection are the goals of research, MSCC algorithm is still a good choice, especially for the small unit-working capacity networks. The procedure is therefore very similar to the above except with some difference as noted below.

Requirements of Current Networks:

Size of the network: small networks (# of nodes <=20, # of links <= 30);

Working capacity: unit working capacity;

Spare capacity: at least one;

Connectivity: bi-connected;

Design Procedure:

1. Check if the network is the Eulerian, if so decompose the network to small cycles by removing the span and the shortest path between the two end node in the remaining graph;
2. If the network is non-Eulerian, implement the MSCC and find the cycle cover for the network;
3. Reserve a channel along every cycle from the cover for both detecting and protecting;
4. Once a failure happens, trigger the monitoring procedure by letting the monitor route along the cycle to locate the failure.
5. Protect the failure by switching the working link where the failure happens to the spare link on the protection path of the cycle.

## 6.3 For Failure Protection of Networks with Unlimited Capacity

This is the scenario for networks where working links can be 100% restorable i.e. the spare capacity can be unlimited, we would recommend implementing the joint design using heuristic algorithms.

To find the pre-selected p-cycle candidates, we use the cycles generated by SLA as seeds, and repeat the Dijkstra's shortest path algorithm to extend the cycles to bigger ones in order to get the improving efficiency, the *SP-Merge* and *SP-Join* work very well both for the unit and non-unit working capacity small network. But for more complicate network (such as USA and France), the *SP-Join* seems more efficient than others.

For the capacity design step, we also use the heuristic method *WCIDA*, and so far we find the $E_w$ is the best as efficiency to plan the spare capacity through comparing to the other one. Especially the redundancy output of the heuristic algorithm is very close to the optimal one.

Requirements of Current Networks:

Size of the network: no specific requirements (suitable for the big network like France, USA, etc.);

Working capacity: no specific requirements;

Spare capacity: unlimited;

Connectivity: bi-connected;

Design Procedure:

1. Find the efficient $p$-cycle candidates using SP-Merge or SP-Join;
2. Use the WCIDA algorithm to find spare capacity on every span and the redundancy of the network;
3. If the failure happens on the span level, find the all the possible $p$-cycle to protect it.

# Chapter 7
# Conclusion

We have proposed a heuristic algorithm using the Eulerian graph theory to solve the failure-detecting/protection problem. To reduce the span time of fault detection and survivability, the detection channel can be shared as a spare one to restore from the failure. A comparison with other detecting algorithms has shown that the MSCC algorithm can provide better performance at the expense of >100% redundancy. This algorithm can achieve two objectives despite its scarcity of spare capacity.

We have also developed and tested heuristic algorithms for providing $p$-cycle survivable transport network designs. The basic approach is to first enumerate a set of $p$-cycle candidates based on a primary set (generated from SLA) through different operations, such as *SP-Add*, *SP-Join,* and *SP-Merge*. Through testing, we find that the set of $p$-cycle has high individual (for a single $p$-cycle) and collective (for the whole network) efficiency. We have found a simple algorithm for capacitated $p$-cycle design without enumerating all of the possible cycles. We also found the *SP-Join* operation is generally better than *SP-Merge*, because the former can get more efficient $p$-cycles as candidates; so it is closer to the optimal results.

Any of above operations can be used to get the input cycles for planning capacity stage where we also take the heuristic (iterative) algorithms based on different actual efficiencies ($E_w$ and $E_u$). After comparing the two efficiencies, we find that $E_w$ has better performance than $E_u$. The heuristic capacity-planning algorithm is of less complexity than the optimal algorithm, and very close to the optimal results.

## 7.1 Future Work

There is still improvement desired of our achievement in this thesis. Further development of algorithms based on *Add* and *Join* search strategy can be studied. Consider how to decrease the size of the candidate set to make it more efficient because we found the final solution of the subset is quite small in size. Take the Canada network for example; although the *SP-Join* has decreased the number of candidates from 410 to 126, the final solution of $p$-cycles is just 4

distinct cycles, which is 3.17% of the total $p$-cycles. The large amount of $p$-cycles not only require too much memory, but also make the process of planning capacity more complicated, so reducing the size of the candidate to a smaller set is very necessary. A possible solution is to delete the non-essential cycles during the process of generating the $p$-cycle candidate, which could be the next project.

Another possible solution is to design a protection algorithm to solve the multi-failure problems. A third way is to get the simulation results within the limitation of the hops or length of cycles according to the actual requirements.

# References


[Ampl00] AMPL Software, Version 10.6.16 - Win32, AMPL Software Pty Ltd

[ArGr00] P. Arijs, M. Gryseels, and P. Demeester, "Planning of WDM Ring Networks", *Journal of Photonic Network Communications*, Vol. 2, No. 1, 2000, pp.33-51

[AsSh01] C. Assi, A. Shami, M. A. Ali, R. Kurtz, D. Guo, "Optical networking and real-time provisioning: an integrated vision for the next-generation Internet," *IEEE Network Magazine*, July/August 2001, pp.36-45.

[CoLe90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge MA, 1990.

[Dijk59] Dijkstra, E. W., "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, 1(1959), 296-71.

[DoHe03] J. Doucette, Donna He, W. D. Grover, Oliver Yang, "Algorithmic Approaches for Efficient Enumeration of Candidate p-Cycles and Capacitated p-Cycle Network Design," *Proc. Workshop on Design of Reliable Communication Networks (DRCN 2003)*, Banff, AB, Canada, pp. 212-220, 19-22 October 2003.

[DuGr94] D.A. Dunn, W.D. Grover, M.H. MacGregor, "A comparison of *k*-shortest paths and maximum flow methods for network facility restoration," *IEEE Journal on Selected Areas in Communications*, Jan. 1994, vol. 12, no. 1, pp. 88-99.

[EdJo73] Jack Edmonds and Ellis L. Johnson, Matching, Euler Tours, and the Chinese Postman, *Mathematical Programming,* 5:88-124, 1973.

[ElCh97] G. Ellinas, G. K. Chang, M.Z. Iqbal, John Gamelin, and Mamun Ur Rashid Khandker, "Wavelength-selective Cross-connect Architecture Interconnecting Multi-wavelength Self-healing Rings", *OFC '97*, Dallas, USA, February 1997, pp. 315-317

[Floy62] Floyd, R.W., "Algorithm 97: Shortest Path," *Comm. ACM*, 5, no. 6(1962), 345.

[FrTe03] France Telecom (2003) [Online], "France Telecom," Available: http://www.francetelecom.com/, 14 August 2003 [date accessed].

[GaHe94] L. M. Gardner, M. Heydari, J. Shah, I.H. Sudborough, I.G. Tollis and C. Xia, "Techniques for Finding Ring Covers in Survivable Networks", *Proc. of IEEE GLOBECOM* 1994, November 1994, pp. 1862-1866.

[GrDo02] Grover, W: D. Doucette, J. E., "Advances in Optical Network Design with p-Cycles: Joint Optimization and Pre- selection of Candidate p-Cycles," *Proc. of the IEEE- LEOS Summer, Topical Meeting on All Optical Networking*, Mont Tremblant, Quebec, July 15- 17, 2002, pp. 49-50 (paper WA2).

[Grov03] Wayne D. Grover, "Mesh-based Survivable Networks", Prentice Hall, 2003.

[Grov97] Wayne D. Grover, "Network Survivability: A Crucial Issue for the information Society", *IEEE Canadian Review*, summer, 1997, pp16-21

[GrSc02] C.G. Gruber and D.A Schupke, "Capacity-efficient Planning of Resilient Networks with p-cycles," in *Proc. of Networks*, 2002.

[GrSh03] Wayne D. Grover and Gangxiang Shen, "Extending the p-cycle concept to path-segment protection," *Proc. of ICC2003*, ON3-Protection/restoration I, Anchorage, Alaska, USA on May 11-15, 2003, pp. 1314-1319.

[GrSt98] W.D. Grover, D. Stamatelakis, "Cycle-Oriented Distributed Preconfiguration: Ring-like Speed with Mesh-like Capacity for Self-planning Network Restoration", *Proc. of IEEE ICC '98*, Atlanta June 7-11, 1998, pp. 537-543.

[GrSt00] Grover, W. D.; Stamatelakis, D., "Bridging the ring- mesh dichotomy with p-cycles," *Second International Workshop on the Design of Reliable Communication Networks (DRCN)*, Munich, Germany, April 9 - 12, 2000.

[Hama97] S. Hamada, "WDM Four-fiber Ring with Add/drop Acoustic-optic Tunable Filter and 4x4 Optical Switch", *OFC '97*, Dallas, USA, February 1997, pp. 313-314

[HsHo72] H. T. Hsu and P. A. Honkanen, "A Fast Minimal Storage Algorithm for Determining All the Elementary Cycles of A Graph," Computer Science Dept., Pennsylvania State Univ., University Park, 1972

[Ilog00] ILOG CPLEX 7.0, Copyright © 2000 ILOG, http://www.ise.ufl.edu/ilog/ cplex70/.

[JaDh01] Raj Jain and Sudheer Dharanikota, "Internet Protocol over DWDM - Recent Developments, Trends and Issues," in *Global Optical Communications - Business Briefing* published by World Market Research Centre Ltd (www.wmrc.com), London, UK, 10 pp., July 2001. http://www.cse.ohio-state.edu/~jain/papers/ftp/ipowdm.pdf.

[Jain99] R. Jain, "IP Over DWDM"(CIS788.08Q Class Lecture), 1999. http://www.cis.ohio-state.edu/~jain/cis788-99/h_aipwd.htm

[John75] Donald B. Johnson, "Finding all the Elementary Circuits of a Directed Graph", *SIAM J. Computing*, Vol. 4, No. 1, March 1975, pp. 77-84.

[KaRe03] J. Kang, M. J. Reed, "Bandwidth Protection in MPLS Networks Using *p*-cycle Structure," *Proc. Workshop on Design of Reliable Communication Networks (DRCN 2003)*, Banff, AB, Canada, pp.-, 19-22 October 2003.

[Kuhn55] H. W. Kuhn, "The Hungarian method for the assignment problem", Naval Res. Logist. Quart. 2: 83-97, 1955.

[Kwan62] Kwan, M. K. "Graphic Programming Using Odd or Even Points," Chinese Math. 1, 273-277, 1962.

[LaBo02] Jean-Francois Labourdette, Eric Bouillet, Ramu Ramamurthy, Georgios Ellinas, Sid Chaudhuri and Krishna Bala, "Routing Strategies for Capacity-Efficient and Fast-Restorable Mesh Optical Networks", *Journal of Photonic Network Communications*, July-Dec. 2002, pp215-217.

[Liuh04] Haomei Liu, "Tree Topology For Link Protection/Restoration In Photonic Networks", *Master Thesis*, University of Ottawa, August, 2004.

[LiYa03] Haomei Liu, Yang, O., Shah-Heydari, S, "A Tree-based Link Protection Algorithm", *Electrical and Computer Engineering 2003 Canadian Conference, (IEEE CCECE 2003)*. Volume: 2, May 4-7, 2003, pp.939 – 942.

[MaDe76] P. Mateti and Narsingh Deo, "Algorithms for Enumerating All Circuits of a Graph," *SIAM J. Computing*, Vol. 5, No.1, March 1976, pp 90-99

[Male02] M. Ajmone Marsan, E. Leonardi, M. Mellia, A. Nucci and A. Grosso, "Design of Logical Topologies in Wavelength-Routed IP Networks," *Journal of Photonic Network Communications*, July-Dec 2002, pp.423-442

[MaPa02] G. Maier, A. Pattavina, S. Patre, M. Martinelli, "Optical Network Survivability: Protection Techniques in the WDM Layer," *Photonic Network Communications*, 4:3/4, 251-269, 2002.

[MoGr97] G. D. Morley and W. D. Grover, "A Comparative Survey of Methods for Automated Design of Ring-based Transport Networks", *TRLabs Technical Report*. http://www.ee.ualberta.ca/~grover/RingBuilder/TRLabs_TR_97_04.PDF

[MoGr98] D. Morley, W. Grover, "A comparative survey of methods for automated design of ring-based transport networks," *TRLabs TR-97-04*, Jan. 1998 (78 pages).

[MoGr99a] D. Morley and W.D. Grover, "Current Approaches in the Design of Ring-based Optical Networks," *Proc. IEEE CCECE 1999*, Edmonton, Canada, May 1999, pp. 220-225

[MoGr99b] G. D. Morley and W. D. Grover, "Comparison of Mathematical Programming Approaches to Optical Ring Networks," *Proc. CCBR'99*, Ottawa, Canada, November 1999, pp. 173-184

[MoSh91] B. Moret and H. Shapiro. *Algorithm from P to NP: Design and Efficiency.* Benjamin/Cummings, Redwood City, CA, 1991.

[MuKi97] Kazutaka Murakami and Hyong S. Kim, "Comparative Study on Restoration Schemes of Survivable ATM Networks," *Proc. IEEE INFOCOM'97*, Kobe, Japan, 1997, pp.345-352.

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

[PuKu02] Nicolas Puech, Josue Kuri and Maurice Gagnaire, "Topological Design and Lightpath Routing in WDM Mesh Networks: A Combined Approach", *Journal of Photonic Network Communications*, July-Dec 2002, pp.443-456.

[RaLa99] F. Rabhi and G. Lapalme, "Algorithms: A Functional Programming Approach," Addison-Wesley, 1999.

[RaLu00] B.Rajagopalan, J. Luciani, D. Awduche and B. Cain "IP over optical networks: Architecture aspects," *IEEE Communications*, pp.94-102, Sep.2000

[RaMu99a] Ramamurthy, S., and Mukherjee, B, "In Survivable WDM Mesh Networks, Part I –Protection," *Proc., IEEE Infocom '99, New York, pp. 744-751, March 1999.*

[RaMu99b] Ramamurthy, S., and Mukherjee, B, "In Survivable WDM Mesh Networks, Part II –Restoration," *Proc. of IEEE ICC'99,* Vancouver, Canada, pp. 2023-2030, June 1999.

[RaSi98] R. Ramaswami and K. N. Sivarajan, "Optical Networks: A Practical Perspective," Morgan Kaufmann Publishers, 1998.

[ScGr02] D.A. Schupke, C.G. Gruber, A. Autenrieth, "Optimal Configuration of *p*-Cycles in WDM Networks," *IEEE International Conference on Communications (ICC),* Anchorage, AK, USA, May 11-15, 2003.

[Schu03] D.A. Schupke, "The tradeoff between the number of deployed p-cycles and the survivability to dual fiber duct failures," *IEEE International Conference on Communications (ICC),* Anchorage, AK, USA, May 11-15, 2003.

[ShBo00] Gangxiang Shen, Sanjay K. Bose, Tee Hiang Cheng and Chao Lu "Designing WDM Optical Network for Reliability: Algorithm with Efficient Light-Path Routing for Path Protection, " *Proc. of OFC'2000*, Maryland, U.S.A, March 2000.

[ShYa04] Shahram Shah-Heydari and Oliver Yang, "Hierarchical Protection Tree Scheme for Failure Recovery in Mesh Networks," *Photonic Network Communications*, vol.7, no.2, March 2004, pp. 145-159.

[ShYa01] Shahram Shah-Heydari and Oliver Yang, "A tree-based algorithm for protection/restoration in optical mesh networks," *Electrical and Computer Engineering, 2001 Canadian Conference*, Volume: 2, 13-16 May 2001 Pages: 1169 - 1174 vol.2

[Skie90] Skiena, S. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Reading, MA: Addison-Wesley, 1990.

[StGr00a] D. Stamatelakis, W.D. Grover, "Theoretical Underpinnings for the Efficiency of Restorable Networks Using Pre-configured Cycles ("*p*-cycles")," *IEEE Transactions on Communications*, vol.48, no.8, August 2000, pp. 1262-1265.

[StGr00b] D. Stamatelakis, W. D. Grover, "IP Layer Restoration and Network Planning Based on Virtual Protection Cycles," *IEEE JSAC Special Issue on Protocols and Architectures for Next Generation Optical WDM Networks*, vol.18, no.10, October, 2000, pp. 1938 - 1949.

[StGr00c] D. Stamatelakis, W.D. Grover, "Network Restorability Design Using Pre-configured Trees, Cycles, and Mixtures of Pattern Types," *TRLabs Technical Report TR-1999-05*, Issue 1.0, October 2000.

[StGr99] Demetrios Stamatelakis and Wayne D. Grover, "Rapid Span or Node Restoration in IP Networks Using Virtual Protection Cycles", *Proc. of CCBR'99*, Ottawa, Nov. 1999.

[StSu02] S. Stanic, S. Subramaniam, H. Choi, G. Sahin, and H. A. Choi, "On monitoring transparent optical networks," *Int'l Conf. on Parallel Proc. Workshops*, Aug. 2002, pp.217-223

[Temp81] Temperlev, H. N. V.,Graph Theory and Applications, Halstead Press, New York (1981).

[ThSo01] S. Thiagarajan and A. K. Somani, "Traffic grooming for survivable WDM mesh networks", *Proc. OPTICOMM 2001*, Aug. 2001.

[T1A193] T1A1.2 Working Group on network Survivability Performance, Report No. 24. *A Technical Report on Network Survivability Performance*, November 1993.

[ToNe94] M. To, P. Neusy, "Unavailability analysis of long-haul networks," *IEEE J. on Sel. Areas in Comm*, vol.12, no.1, January 1994, pp100-109.

[VePo02] Presentation by A.J. Vernon, J. D. Portier,"Protection of Optical Channels in All-Optical Networks," *18th Annual National Fiber Optic Engineers Conference (NFOEC 2002)*, pp. 1695-1706, Dallas, TX, Sept.2002.

[Wang99] G. Q. Wang, "Optical Label Switching and Routing Architecture", 1999, internal document of Nortel Networks

[Wase91] Ondria J. Wasem, "An Algorithm for Designing Rings for Survivable Fiber Networks", *IEEE Transactions on Reliability*, Vol. 40, No. 4, Oct 1991, pp. 428-432

[Wils72] Wilson, R. J.,Introduction to Graph Theory, Academic Press, New York (1972).

[WuLa90] Tsong-Ho Wu & Richard C. Lau, "A Class of Self-Healing Ring Architecture for SONET Network Applications", *Proc. of GLOBECOM 90*, pp444-449, 1990

[WuTs95] Tsong-Ho Wu, "Emerging Technologies for Fiber network Survivability", *IEEE Communications Magazine*, Feb. 1995, pp.60-74

[YeDi00] Yinghua Ye, Sudhir Dixit and Mohamed Ali, "On Joint Protection/Restoration in IP-Centric DWDM-Based Optical transport Networks", *IEEE Communications Magazine*, June 2000, pp174-182.

[ZeHu04] Hongqing Zeng, Changcheng Huang, Alex Vukovic, "Monitoring Cycles for Fault Detection in Meshed All-Optical Networks," *2004 International Conference on Parallel Processing Workshops (ICPPW'04)*, August 15 - 18, 2004, Montreal, pp. 434-439

[Zhan02a] Hanxi Zhang, "DWDM Network Protection/Restoration with Ring/Cycle Topology," Master Thesis, University of Ottawa, March, 2002

[Zhan02b] Yuna Zhang, "Distributed tree Protection Scheme for DWDM Network Protection/Restoration," Master Thesis, University of Ottawa, August, 2002

[ZhYa02a] Hanxi Zhang and Yang, O., "Finding protection cycles in DWDM networks," *Communications, 2002. ICC 2002. IEEE International Conference,* Volume: 5, 2002 Page(s): 2756 -2760 vol.5.

[ZhYa02b] Hanxi Zhang, Oliver Yang, "DWDM Network Protection/Restoration with Ring/Cycle Topology", Master Thesis, University of Ottawa, March, 2002

[ZhYa02c] Yuna Zhang and O. Yang, "A distribute tree algorithm for WDM network protection/restoration," *High Speed Networks and Multimedia Communications 5th IEEE International Conference*, 3-5 July 2002 Pages: 289 – 294

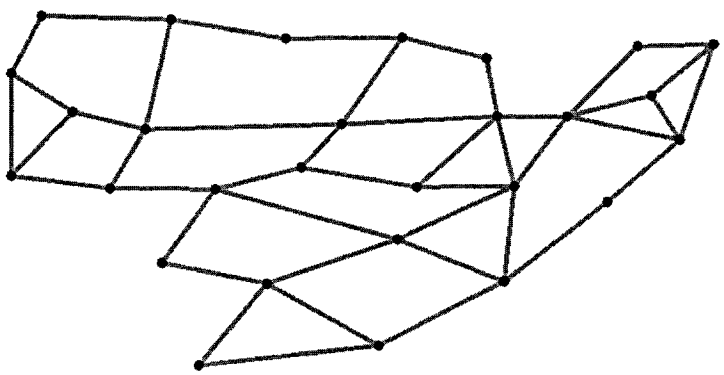
[ZhYa02d] Yuna Zhang and Oliver Yang, "Different implementations of token tree algorithm for DWDM network protection/restoration," *Computer Communications and Networks, 2002. Proceedings*, pp.290-295

[ZhZh04] Zhenrong Zhang, Wen-De Zhong, Biswanath Mukherjee, "A Heuristic Method for Design of Survivable WDM Networks with p-Cycles," *IEEE Commu Letter*, July 2004.

# Appendix A
# Topologies of the Sample Networks

This appendix stipulates the physical topologies and snif files of the few networks used in the evaluation of our algorithms in Chapters 4 and 5. The ".sniff" file describes the node to link adjacency relationships between the nodes and spans (links) in a network graph. For example, from the snif file of the USA long haul network, we can see that there are 28 nodes in the network; span 1 connects nodes 1 and 2, span 2 connects nodes 1 and 5. The 'Cost' column is 1 and represents the number of hops.

## A.1 Topology of USA Long Haul Network

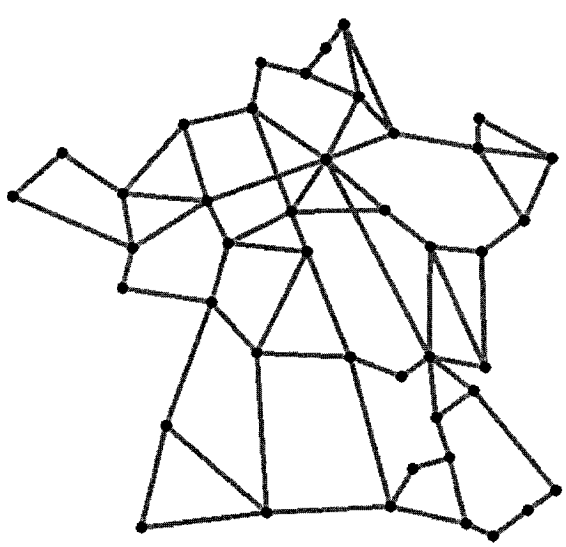


The topology part of ".snif" file:

| Span | NodeA | NodeB | Distance |
|---|---|---|---|
| 1 | 1 | 2 | 1 |
| 2 | 1 | 5 | 1 |
| 3 | 2 | 3 | 1 |
| 4 | 2 | 7 | 1 |
| 5 | 3 | 4 | 1 |
| 6 | 4 | 8 | 1 |
| 7 | 4 | 13 | 1 |
| 8 | 5 | 6 | 1 |
| 9 | 5 | 9 | 1 |
| 10 | 6 | 7 | 1 |
| 11 | 6 | 9 | 1 |
| 12 | 7 | 8 | 1 |
| 13 | 7 | 10 | 1 |
| 14 | 8 | 12 | 1 |
| 15 | 8 | 14 | 1 |
| 16 | 9 | 10 | 1 |
| 17 | 10 | 11 | 1 |

| | | | |
|---|---|---|---|
| 18 | 11 | 12 | 1 |
| 19 | 11 | 22 | 1 |
| 20 | 11 | 25 | 1 |
| 21 | 12 | 15 | 1 |
| 22 | 13 | 14 | 1 |
| 23 | 14 | 15 | 1 |
| 24 | 14 | 16 | 1 |
| 25 | 14 | 21 | 1 |
| 26 | 15 | 21 | 1 |
| 27 | 16 | 17 | 1 |
| 28 | 16 | 19 | 1 |
| 29 | 16 | 20 | 1 |
| 30 | 16 | 21 | 1 |
| 31 | 17 | 18 | 1 |
| 32 | 18 | 19 | 1 |
| 33 | 18 | 20 | 1 |
| 34 | 19 | 20 | 1 |
| 35 | 20 | 24 | 1 |
| 36 | 21 | 22 | 1 |
| 37 | 21 | 23 | 1 |
| 38 | 22 | 23 | 1 |
| 39 | 22 | 26 | 1 |
| 40 | 23 | 24 | 1 |
| 41 | 23 | 27 | 1 |
| 42 | 25 | 26 | 1 |
| 43 | 26 | 27 | 1 |
| 44 | 27 | 28 | 1 |
| 45 | 26 | 28 | 1 |

## A.2 Topology of France Telecom Network



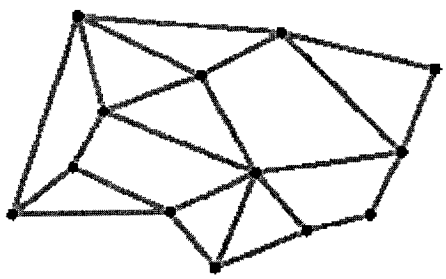
The topology part of ".snif" file:

| Span | NodeA | NodeB | Distance |
|---|---|---|---|
| 1 | 1 | 2 | 1 |
| 2 | 1 | 3 | 1 |
| 3 | 2 | 3 | 1 |

| | | | |
|---|---|---|---|
| 4 | 2 | 18 | 1 |
| 5 | 3 | 19 | 1 |
| 6 | 4 | 18 | 1 |
| 7 | 2 | 5 | 1 |
| 8 | 6 | 8 | 1 |
| 9 | 8 | 9 | 1 |
| 10 | 9 | 10 | 1 |
| 11 | 10 | 11 | 1 |
| 12 | 11 | 12 | 1 |
| 13 | 7 | 8 | 1 |
| 14 | 7 | 15 | 1 |
| 15 | 4 | 5 | 1 |
| 16 | 4 | 17 | 1 |
| 17 | 12 | 13 | 1 |
| 18 | 7 | 13 | 1 |
| 19 | 13 | 15 | 1 |
| 20 | 14 | 15 | 1 |
| 21 | 15 | 16 | 1 |
| 22 | 16 | 4 | 1 |
| 23 | 17 | 18 | 1 |
| 24 | 18 | 19 | 1 |
| 25 | 19 | 20 | 1 |
| 26 | 19 | 22 | 1 |
| 27 | 17 | 22 | 1 |
| 28 | 17 | 36 | 1 |
| 29 | 15 | 35 | 1 |
| 30 | 36 | 37 | 1 |
| 31 | 15 | 38 | 1 |
| 32 | 14 | 38 | 1 |
| 33 | 14 | 39 | 1 |
| 34 | 39 | 40 | 1 |
| 35 | 40 | 41 | 1 |
| 36 | 40 | 42 | 1 |
| 37 | 41 | 42 | 1 |
| 38 | 41 | 43 | 1 |
| 39 | 42 | 43 | 1 |
| 40 | 42 | 34 | 1 |
| 41 | 34 | 35 | 1 |
| 42 | 37 | 35 | 1 |
| 43 | 35 | 36 | 1 |
| 44 | 36 | 28 | 1 |
| 45 | 35 | 23 | 1 |
| 46 | 36 | 28 | 1 |
| 47 | 22 | 23 | 1 |
| 48 | 23 | 21 | 1 |
| 49 | 20 | 21 | 1 |
| 50 | 21 | 26 | 1 |
| 51 | 25 | 26 | 1 |
| 52 | 24 | 25 | 1 |
| 53 | 24 | 21 | 1 |
| 54 | 23 | 24 | 1 |
| 55 | 23 | 27 | 1 |
| 56 | 24 | 27 | 1 |
| 57 | 27 | 28 | 1 |
| 58 | 35 | 28 | 1 |
| 59 | 28 | 29 | 1 |
| 60 | 29 | 30 | 1 |

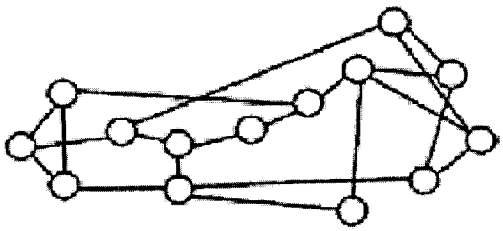
| | | | |
|---|---|---|---|
| 61 | 35 | 33 | 1 |
| 62 | 33 | 34 | 1 |
| 63 | 32 | 34 | 1 |
| 64 | 33 | 32 | 1 |
| 65 | 30 | 33 | 1 |
| 66 | 30 | 31 | 1 |
| 67 | 31 | 32 | 1 |
| 68 | 37 | 38 | 1 |
| 69 | 38 | 39 | 1 |
| 70 | 6 | 5 | 1 |
| 71 | 5 | 9 | 1 |

## A.3 Topology of the Canada Network

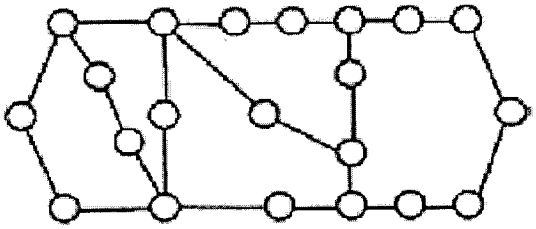


| Span | NodeA | NodeB | Distance |
|---|---|---|---|
| 1 | 1 | 2 | 1 |
| 2 | 1 | 3 | 1 |
| 3 | 1 | 4 | 1 |
| 4 | 1 | 13 | 1 |
| 5 | 2 | 3 | 1 |
| 6 | 2 | 7 | 1 |
| 7 | 2 | 13 | 1 |
| 8 | 3 | 5 | 1 |
| 9 | 3 | 7 | 1 |
| 10 | 4 | 5 | 1 |
| 11 | 4 | 6 | 1 |
| 12 | 5 | 6 | 1 |
| 13 | 6 | 7 | 1 |
| 14 | 6 | 9 | 1 |
| 15 | 7 | 8 | 1 |
| 16 | 7 | 9 | 1 |
| 17 | 7 | 11 | 1 |
| 18 | 8 | 9 | 1 |
| 19 | 8 | 10 | 1 |
| 20 | 10 | 11 | 1 |
| 21 | 11 | 12 | 1 |
| 22 | 11 | 13 | 1 |
| 23 | 12 | 13 | 1 |

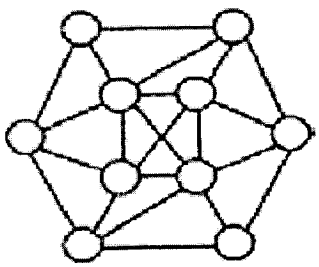
## A.4 Topologies of the Other Networks for MSCC



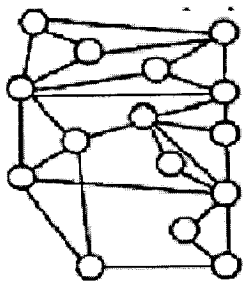
NSFNET: 14 nodes, 21 links



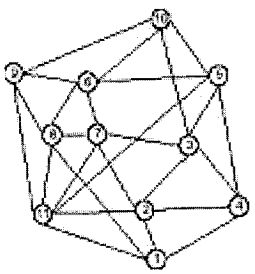
ARPA2: 21 nodes, 25 links



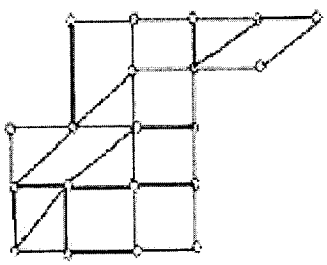
SmallNet: 10 nodes, 22 links



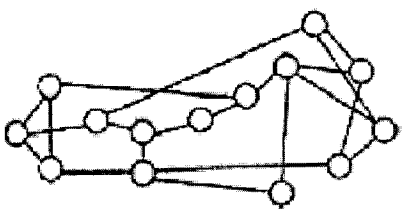
Bellcore: 15 nodes, 28 links



COST239: 11nodes, 26spans



ARB : 20nodes, 33spans



NSFNET: 14nodes, 21spans

# Appendix B
# The Working Capacity Distribution of the Sample Networks under the Unit-end-to-end Demand

This appendix stipulates the working capacity distribution for deferent networks under unit end-to-end traffic model that we discussed in Chapter 5.

## B.1 The France Network

| Span | NodeA | NodeB | Working Capacity |
|---|---|---|---|
| 1 | 1 | 2 | 23 |
| 2 | 1 | 3 | 19 |
| 3 | 2 | 3 | 42 |
| 4 | 2 | 18 | 32 |
| 5 | 3 | 19 | 69 |
| 6 | 4 | 18 | 21 |
| 7 | 2 | 5 | 71 |
| 8 | 6 | 8 | 39 |
| 9 | 8 | 9 | 40 |
| 10 | 9 | 10 | 57 |
| 11 | 10 | 11 | 21 |
| 12 | 11 | 12 | 35 |
| 13 | 7 | 8 | 93 |
| 14 | 7 | 15 | 107 |
| 15 | 4 | 5 | 77 |
| 16 | 4 | 17 | 53 |
| 17 | 12 | 13 | 63 |
| 18 | 7 | 13 | 12 |
| 19 | 13 | 15 | 87 |
| 20 | 14 | 15 | 89 |
| 21 | 15 | 16 | 89 |
| 22 | 16 | 4 | 75 |
| 23 | 17 | 18 | 42 |
| 24 | 18 | 19 | 19 |
| 25 | 19 | 20 | 46 |
| 26 | 19 | 22 | 52 |
| 27 | 17 | 22 | 41 |
| 28 | 17 | 36 | 58 |
| 29 | 15 | 35 | 226 |
| 30 | 36 | 37 | 25 |
| 31 | 15 | 38 | 30 |
| 32 | 14 | 38 | 2 |
| 33 | 14 | 39 | 59 |
| 34 | 39 | 40 | 49 |
| 35 | 40 | 41 | 21 |
| 36 | 40 | 42 | 22 |
| 37 | 41 | 42 | 24 |
| 38 | 41 | 43 | 5 |
| 39 | 42 | 43 | 37 |
| 40 | 42 | 34 | 113 |
| 41 | 34 | 35 | 118 |

| | | | |
|---|---|---|---|
| 42 | 37 | 35 | 27 |
| 43 | 35 | 36 | 35 |
| 44 | 36 | 28 | 27 |
| 45 | 35 | 23 | 179 |
| 46 | 36 | 28 | 29 |
| 47 | 22 | 23 | 88 |
| 48 | 23 | 21 | 78 |
| 49 | 20 | 21 | 56 |
| 50 | 21 | 26 | 41 |
| 51 | 25 | 26 | 1 |
| 52 | 24 | 25 | 41 |
| 53 | 24 | 21 | 15 |
| 54 | 23 | 24 | 56 |
| 55 | 23 | 27 | 43 |
| 56 | 24 | 27 | 10 |
| 57 | 27 | 28 | 21 |
| 58 | 35 | 28 | 37 |
| 59 | 28 | 29 | 49 |
| 60 | 29 | 30 | 25 |
| 61 | 35 | 33 | 114 |
| 62 | 33 | 34 | 15 |
| 63 | 32 | 34 | 12 |
| 64 | 33 | 32 | 33 |
| 65 | 30 | 33 | 62 |
| 66 | 30 | 31 | 37 |
| 67 | 31 | 32 | 9 |
| 68 | 37 | 38 | 20 |
| 69 | 38 | 39 | 18 |
| 70 | 6 | 5 | 39 |
| 71 | 5 | 9 | 53 |

## B.2 the USA Network

| Span | NodeA | NodeB | Working Capacity |
|---|---|---|---|
| 1 | 1 | 2 | 45 |
| 2 | 1 | 5 | 13 |
| 3 | 2 | 3 | 27 |
| 4 | 2 | 7 | 73 |
| 5 | 3 | 4 | 31 |
| 6 | 4 | 8 | 64 |
| 7 | 4 | 13 | 19 |
| 8 | 5 | 6 | 19 |
| 9 | 5 | 9 | 19 |
| 10 | 6 | 7 | 67 |
| 11 | 6 | 9 | 16 |
| 12 | 7 | 8 | 126 |
| 13 | 7 | 10 | 58 |
| 14 | 8 | 12 | 55 |
| 15 | 8 | 14 | 145 |
| 16 | 9 | 10 | 31 |
| 17 | 10 | 11 | 82 |
| 18 | 11 | 12 | 37 |
| 19 | 11 | 22 | 97 |
| 20 | 11 | 25 | 36 |
| 21 | 12 | 15 | 25 |
| 22 | 13 | 14 | 21 |
| 23 | 14 | 15 | 10 |

| | | | |
|---|---|---|---|
| 24 | 14 | 16 | 79 |
| 25 | 14 | 21 | 67 |
| 26 | 15 | 21 | 16 |
| 27 | 16 | 17 | 69 |
| 28 | 16 | 19 | 36 |
| 29 | 16 | 20 | 48 |
| 30 | 16 | 21 | 51 |
| 31 | 17 | 18 | 31 |
| 32 | 18 | 19 | 1 |
| 33 | 18 | 20 | 7 |
| 34 | 19 | 20 | 3 |
| 35 | 20 | 24 | 30 |
| 36 | 21 | 22 | 58 |
| 37 | 21 | 23 | 49 |
| 38 | 22 | 23 | 28 |
| 39 | 22 | 26 | 66 |
| 40 | 23 | 24 | 28 |
| 41 | 23 | 27 | 42 |
| 42 | 25 | 26 | 4 |
| 43 | 26 | 27 | 3 |
| 44 | 27 | 28 | 7 |
| 45 | 26 | 28 | 33 |

# Appendix C
# Protection *p*-Cycle Obtained for the USA Network

In this appendix, we use the USA network as an example to present the results obtained with our algorithm. These results include the total number of spans of a network, the total number of nodes, the candidate *p*-cycles, the number of copies of *p*-cycles, and the total working and spare capacity and redundancy of the network. Details can be found in our internal CCNR report.

# Appendix D.
# Graph Theory Algorithms

## D.1 Binary Heap Implementation of Dijkstra's Algorithm

We would like to implement Dijkstra's algorithm using a binary heap. In this algorithm heap H would be the collection of nodes with finite temporary distances while the key of a node would be its temporary distance. The following subroutines are also required to carry out this algorithm and can be found in [Zhan02a]:

1) create-heap(H): Create an empty binary heap
2) find-min(i, H): Find and return a node i of minimum key.
3) insert(i, H): Insert a new node i with a predefined key.
4) decrease-key(value, i, H): Deduce the key of node i by 'value'.
5) delete-min(i, H): Delete a node i of minimum key.

```
   heap-Dijkstra algorithm;
begin
        create-heap(H);
        d(j) := ∞ for all j ∈ N;
        d(s) := 0 and pred(s) := 0;
        insert(s, H);
        while H ≠ ∅ do
        begin
                find-min(i H);
                delete-min(i H);
                for each (i, j) ∈ A(i) do
                begin
                        value := d(i) + Cij;
                        if d(j) > value then
                                if d(j) = ∞
                                then d(j) := value,
                                pred(j) := i, and  insert(j, H);
                        else set d(j) := value, pred(j) := i,
                                and decrease-key(value, i, H)
                end;
        end;
end;
```

## D. 2 MPM (Minimum-weighted Perfect Matching) Algorithm

This algorithm supplements the explanation in Section

Let $A=(a_{ij})_{n*n}$, where $a_{ij}$=1 if nodes i and j are adjacent, else $a_{ij}$ = 0 be the incidence matrix of G, the non-negative weight function $w$, $x=(x_e)_{e\in E}$ and $x_e \in Z^+ \cup \{0\}$. Mathematically, the model of MPM can be described as follows:

Min $wx^T$

subject to $Ax^T = I^T$ and $x>=0$,

Let $w_{ij}$ be the weight of the edge *(i,j)*, and $x_{ij}$ is a variable for $i$ =*1, 2, ..., n* and $j$ =*1,2,...,n,* where n be is the number of nodes. The value of $x_{ij}$=1 means that the edge *(i,j)* is included in the matching, whereas $x_{ij}$ =0 means that it is not, We can formulate the general matching problem for non-bipartite weighted graph (called the Primal Problem) as follows:

$$\text{Min} \sum_{i,j} w_{ij}x_{ij}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i= 1,2,...,n \qquad \text{(D1)}$$

$$x_{ij} \geq 0 \qquad 1 \leq i \leq j \leq n, \qquad \text{(D2)}$$

$$\sum_{i,j\in S_k} x_{ij} + y_k = s_k, \qquad y_k \geq 0 \qquad for\ k = 1,2,...,N, \qquad \text{(D3)}$$

where $y_k$ is the slack variable to satisfy the constraint D3.

Instead of providing a procedure for the Primal solution, we provided in the following the procedure of the Dual problem to be solved in Section 3.1.2:

Input: The topology of the graph *H*, and an n*n symmetric matrix $\{w_{i,j}\}$ of nonnegative integers; n is even.

Output: The perfect matching *M* which has the smallest total cost under $w_{i,j}$.

*MPM*(H, w )

{

for all $v_i \in V$ do $x_i = \frac{1}{2}\min_j\{w_{ij}\}$;

for all k do $y_k = 0$;

M = $\phi$;(comment: initialize the maximum matching of G)

$J_b = \phi$;(comment: $\bar{J}_b$ contains all odd sets $S_k$ with $y_k < 0$ )

while $|M| < n/2$ do

begin

construct the admissible graph $G_J$ by including all edges $[v_i, v_j]$ with $x_i + x_j + \sum_{i,j \in S_k} y_k = w_{ij}$,

and shrinking all sets $S_k$ in $\bar{J}_b$ ;

find the maximum matching in $G_J$ starting from the current matching M;

let $G_C$ be the current graph at the conclusion of the (unweighted) maximum matching algorithm for $G_J$;

let O be the set of outer vertices in $G_C$, I the set of inner ones, $\Psi_O$ the set of outer pseudonodes, and

$\Psi_I$ the set of inner pseudonodes;

calculate $\theta_1 = \min(\delta_1, \delta_2, \delta_3)$ (comment: Equations 3.3)

for all $v_j \in O$ do $x_j = x_j + \theta_1$;

for all $v_j \in I$ do $x_j = x_j - \theta_1$;

for all $S_k \in \Psi_O$ do $y_k = y_k - 2\theta_1$;

for all $S_k \in \Psi_I$ do $y_k = y_k + 2\theta_1$;

recover the maximum proper matching M of (V, $J_e$) from the maximum matching of $G_J$;

let $\bar{J}_b = \left\{ S_k \in \bar{J}_b \cup \Psi_O : y_k < 0 \right\}$

end

}