UNIVERSITY OF ALBERTA

THEORY AND ALGORITHMS FOR PRECONFIGURATION OF SPARE CAPACITY IN MESH RESTORABLE NETWORKS

By

DEMETRIOS STAMATELAKIS ©

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

DEPARTMENT OF ELECTRICAL ENGINEERING

EDMONTON, ALBERTA

SPRING 1997

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

0-612-21210-6

Canada

University of Alberta

Library Release Form

Name of Author: Demetrios Stamatelakis

Title of Thesis: Theory and Algorithms for Preconfiguration of Spare Capacity in Mesh Restorable Networks

Degree: Master of Science

Year this Degree is Granted: Spring 1997

11215 93 Street
Fort St. John, B.C.
V1J 4W5, CANADA

Date: January 3, 1997

UNIVERSITY OF ALBERTA

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled

"Theory and Algorithms for Preconfiguration of Spare Capacity in Mesh Restorable Networks"

submitted by **Demetrios Stamatelakis** in partial fulfilment of the requirements for the degree of **Master of Science.**

Wayne D. Grover (Supervisor)

____

Pawel Gburzynski (External)

____

Jan Conradi (Internal)

Jan Conradi
____

Date: Dec 23/96

Dedicated to my wife Tanis
and to my parents Euthimios and Katherine

Abstract

Mesh-restorable transport networks offer a number of advantages, such as low capacity redundancy and flexible signal management. However, a disadvantage of mesh based restoration is that its restoration speed may not be fast enough to prevent voice and data connections from being dropped, when implemented on some types of transport switching nodes. Typically, an existing connection will not be dropped if a failure is restored within 2 seconds. Previous research in on-demand mesh restoration algorithms have established that it is technically possible to calculate a restoration pathset, in a distributed realtime response, within 2 seconds. However, due to the slow crossconnect time of some current digital crossconnect systems (DCS), it may not be possible to also implement the restoration in the 2 seconds available. One way to reduce this crossconnect formation time is to reduce the number of crossconnections, that have to be asserted in realtime, to form the restoration pathset by a strategy of preconnecting spare links, prior to failure, so as to maximize the statistical likelihood that, in the event of any failure, it will be found that required connections to form a restoration pathset are already in place within the network spare capacity.

This thesis experimentally evaluates the performance of a number of different elemental subgraphs as building blocks, with which to synthesize restorable networks with a high degree of usefully preconnected spare restoration paths. Preconnected paths supply immediately fully formed restoration paths to the restoration of span failures. Two classic graph patterns which are evaluated are the tree and the cycle. Subsequently, a pair of genetic algorithms are used to evolve pre-set patterns which are not constrained to belong to a certain pattern class. A theoretical upper limit is established to the number of pre-formed paths which the cycle and tree patterns can contribute to restoration. In addition, a theoretical upper limit is established to the number of pre-formed paths which any pattern, regardless of its configuration, can provide. The experimental and theoretical results suggest that the cycle may be the best candidate to pre-set, within a network's spare capacity, in anticipation of any span failure.

Acknowledgments

I would like to thank my wife Tanis, my parents Tim and Kathy, and parents in-law Mike and Betty for their love and support during the writing of my thesis.

I would like to thank my supervisor Dr. Wayne D. Grover for his guidance and encouragement over the course of the winding trail that eventually became my thesis.

I would also like to thank all of the people of TRLabs-Edmonton, students and staff alike, for providing such a friendly environment in which to carry out my research. I'd like to single out the following fellow members of the Networks and Systems group (a.k.a. the N&S Mafia) for providing a group which was both supportive and yet somewhat abusive :) : Danny Li, Mike MacGregor, Jim Slevinsky, Daniel Tse, Deepak Sarda, Yong Zheng, and, last but certainly not shortest, Jason Palm. I'd also like to mention the following people, who graduated and left TRLabs before myself: Ashish Duggal, Vipul Rawat, Rainer Irashko, Ping Wan, and Jeremy Sewall. TRLabs made it easy to make many friends.

Table of Contents

List of Tables

List of Figures

Abbreviations

ADM	Add-Drop Multiplexer
APS	Automatic Protection Switching
DCS	Digital Crossconnect System
DPCD	Distributed Preconfigured Cycle Design Protocol
DS-N	Digital Signal - Level N
FOTS	Fiber Optic Transmission System
GA	Genetic Algorithm
GA1	PC Design at the Span Level using a GA
GA2	PC Design at the Crossconnect Level using a GA
IP	Integer Program
IP1-cycle	IP design of PC cycles in an Existing Network Spare Plan
IP2-cycle	IP design of PC cycles and Network Sparing Plan
KSP	k-shortest paths
PC	Preconfigured / Preconfiguration
SHN	Selfhealing Network Protocol
SP	Shortest Path
SHR	Selfhealing Ring
STS-N	Synchronous Transport Signal - Level N
tree1	PC Tree Heuristic with Span Weights set Once
tree2	PC Tree Heuristic with Span Weights updated at Every Iteration

1. Introduction

Modern society has a greater dependence on information services than ever before. This information is carried on a world wide network of public telecommunication transport networks. The advent of high capacity digital technologies, in recent years, has permitted the increasing quantities of communication traffic to be carried in an economical and efficient manner. The large carrying capacity of these technologies, which is their greatest advantage, also increases the transport network's vulnerability to a single failure. A single multi-gigabit per second fibre optic cable can carry the capacity equivalent of tens of thousands of individual conversations and data connections. There would be significant financial, and social consequences if such a fibre failed and there was no means in place to rapidly reroute the traffic which flowed on it [1].

In recent years, alternatives have been studied to solve the problem of the restoration of a failed network span. These restoration methods basically fall into two categories: dedicated spare capacity routing and undedicated spare capacity routing. In dedicated spare capacity routing methods, such as automatic protection switching (APS) and self-healing rings (SHRs), a telecom network will have spare capacity added to the network which is dedicated to the restoration of specific working traffic flows. The spare capacity is preset so it contains restoration paths. When a network span fails, the switching equipment automatically reroutes the working traffic by switching the working flow from the failed span to the preset spare alternate route. In undedicated (also generically called "mesh") spare capacity routing methods, spare capacity is distributed throughout the network on each network span. This spare capacity is not dedicated to the restoration of any particular span failure. In the event of a failure, workings flows are rerouted over paths, which are dynamically calculated and formed on demand using the network's distributed spare capacity. In summary, in dedicated sparing restoration schemes the working traffic is rerouted through predefined restoration paths in the network spare capacity, while in undedicated sparing schemes the network configuration is modified, as needed, to form restoration paths on demand.

The primary advantage of dedicated capacity restoration methods is their speed. The spare capacity is effectively "hardwired" and, in the event of a span failure, the transport signals can be rapidly switched to the stand-by paths, which can result in restoration times of as little as 50 msec. However, there are disadvantages to this type of restoration. First, the total amount of spare capacity which is required to make such a design fully restorable, for all span failures, will be generally greater than the total amount of working capacity present in the network* [2]. The network will also be relatively inflexible in its configuration because all of its spare capacity is hardwired. The implication is that network traffic has to be accurately forecast, at the time of the network's construction, so that the network's fixed protection configuration will be able to support future traffic growth.

A mesh type restorable network, on the other hand, uses undedicated spare capacity and has the primary advantage of being fully restorable using an amount of spare capacity which can be 3 to 6 times less than required in a dedicated capacity design [2]. This reduction occurs because all the spare capacity in a mesh network is free to be re-used in the restoration of any span failure, whereas the spare capacity in a dedicated network can only be used in the restoration of a specific set of spans. An additional advantage, which mesh networks offer, is that both the working and spare capacity are fully reconfigurable to more easily accommodate future changes in offered traffic. The only difference between working capacity and spare capacity, in a mesh network, is that one is committed to service and the other is simply sitting idle. If traffic offered to the network should suddenly increase, in a certain area, it may be carried by putting some of the spare capacity into service. However, this may result in a network which is no longer fully restorable in the event of any span failure. More usually, the benefit is in placing new working capacity only where the growth is actually materializing which allows a mesh network to be less dependant on the actual forecast ordering. The primary disadvantage of undedicated spare capacity restoration is that its restoration speed will not be as fast as that of dedicated restoration. In the event of a span failure, alternate paths for the working traffic flowing

* -To appreciate this consider that fully diverse routes APS logically requires only 100% redundancy. However, by definition, the diverse protection routes cannot be on the same shortest route as the working paths and, therefore, are physically longer. It follows, therefore, that the redundancy of APS (and by related argument for rings) is always greater than 100%.

through the failed spans will have to be dynamically calculated through the network's unaffected spare capacity. Additional time may also be required to set the switches required to form dynamically determined restoration the paths.

1.1 Background on Mesh Transport Networks: Definitions

A mesh network contains nodes, where network traffic is sourced, terminated and routed, and spans, which connect adjacent network nodes and through which network traffic is carried.

A *link* is an individual bidirectional digital signal carrier joining a pair of adjacent transport network nodes. It could be a DS-3, STS-1 or STS-N *. In general, a link represents the multiplexed level at which digital signals are managed at the respective transport nodes. A link can either be a *working/worker* link or it can be a *spare* link. A working link is a link which is carrying live network traffic. A spare link is a link which is currently sitting idle within the network but is available to assist in the restoration of a network failure or to be used to absorb traffic growth.

A *span* is the set of all working and spare links which join a pair of adjacent network nodes. A span can contain many links but a link can only belong to one span. In other words, the set of all signals that would be intercepted by a slice through the direct route joining adjacent nodes is a span.

The nodes in a mesh network are a type of digital carrier switching machine called a digital crossconnect system (DCS.) A DCS permits the forming of a bidirectional switch or crossconnect between the ends of any pair of links which fall on the DCS. It is important to note, especially with regard to the work done in this thesis, that a DCS is a carrier signal management device and is completely distinct from the more familiar "telephone switch". Each link which appears at a DCS is terminated at a port card in the DCS. A crossconnection may be formed between a pair of links by connecting the port cards,

* The DS-3 rate is 44.736 Mb/s while the STS-1 rate is 51.880 Mb/s. Note that throughout this work it is these highly multiplexed transport signals that we are manipulating and not individual telephone calls. For further clarification see section 1.2 on the concept of a transport network.

which terminate the links, through the DCS's crossconnect matrix. The DCS are what give a mesh transport network its flexible configuration. Note that the re-configurations in the transport network are invisible to the conventional call switches of the network.

A *path* is a concatenation of network links which follow a route between a pair of network nodes. A *route* is any concatenation of network spans joining a pair of non-adjacent or adjacent nodes. The distinction between a path and a route is helpfully given in [2] which states, "that a link is to a path as a span is to a route, the former identifying individual capacity units of the network, the latter referring to the topology of the network only." A route can correspond to many paths but each path has only one route.

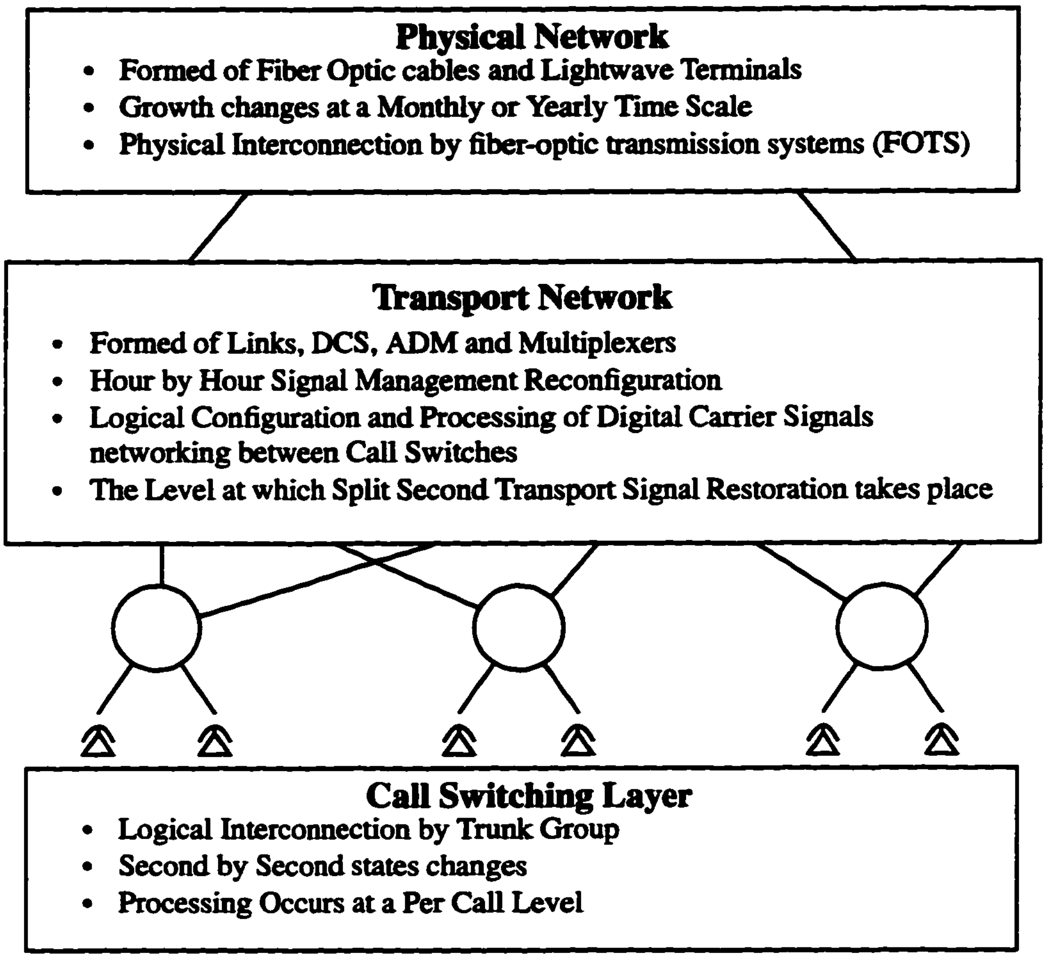
A *demand pair* is a pair of network nodes which require a certain quantity of network working demand units (e.g. DS-3s, STS-1s) to flow between them. Each unit of working demand is routed between the demand pair on a working path.

1.2 Background on Mesh Transport Networks: Logical Network Layers

The work in this thesis deals entirely with restoration of failures at the transport network layer. This is separate from the call switching layer where individual conversation and data connections are made. The transport layer deals only with the multiplexed digital carrier signals and not with individual conversations. Refer to Figure 1.1 for a diagram which shows the different network layers.

The physical network layer is formed of fiber-optic transmission systems (FOTS) and provides the physical transmission media through which signals are passed. Changes at the physical level occur at a monthly or yearly time scale, in response to traffic growth within the network.

The transport network contains DCS and add-drop multiplexers (ADM) to configure and process digital carrier signals. It also contains multiplexers which multiplex the traffic offered by the call switches onto the digital carrier signals. The configuration of the digital carrier signals form logical connections between the call switches. The transport network is also the level at which split second restoration of failures takes place and is the level at



Physical Network

- Formed of Fiber Optic cables and Lightwave Terminals
- Growth changes at a Monthly or Yearly Time Scale
- Physical Interconnection by fiber-optic transmission systems (FOTS)

Transport Network

- Formed of Links, DCS, ADM and Multiplexers
- Hour by Hour Signal Management Reconfiguration
- Logical Configuration and Processing of Digital Carrier Signals networking between Call Switches
- The Level at which Split Second Transport Signal Restoration takes place

Call Switching Layer

- Logical Interconnection by Trunk Group
- Second by Second states changes
- Processing Occurs at a Per Call Level

FIGURE 1.1 Logical Separation of the Different Network Layers

which all work in this thesis is done. Configuration changes at this level occur on an hour by hour time scale, as signal management takes place. All configuration changes at this level occur transparently to the call switching layer.

The call switching layer, switches individual conversations and data connections using the logical connections formed within the transport layer. Configuration changes occur on a second by second scale, as individual connections are formed and released.

1.3 Restoration of Mesh Network Span Failures

A restorable mesh network will contain both working links and spare links distributed among its spans. The working links will be configured into working paths which will carry the traffic between the network nodes. In all work to date, the spare links are unconnected and sit in a stand-by state. In the event of a span failure, the lost traffic, which flowed through the working links on the failed span, will be rerouted onto restoration paths formed from the network's spare links. The presence of the appropriately placed spare capacity distributed throughout the network permits the restoration of span failures. Refer to Figure 1.2 for an example of a restoration pathset formed in response to a span failure using the network's distributed spare capacity.

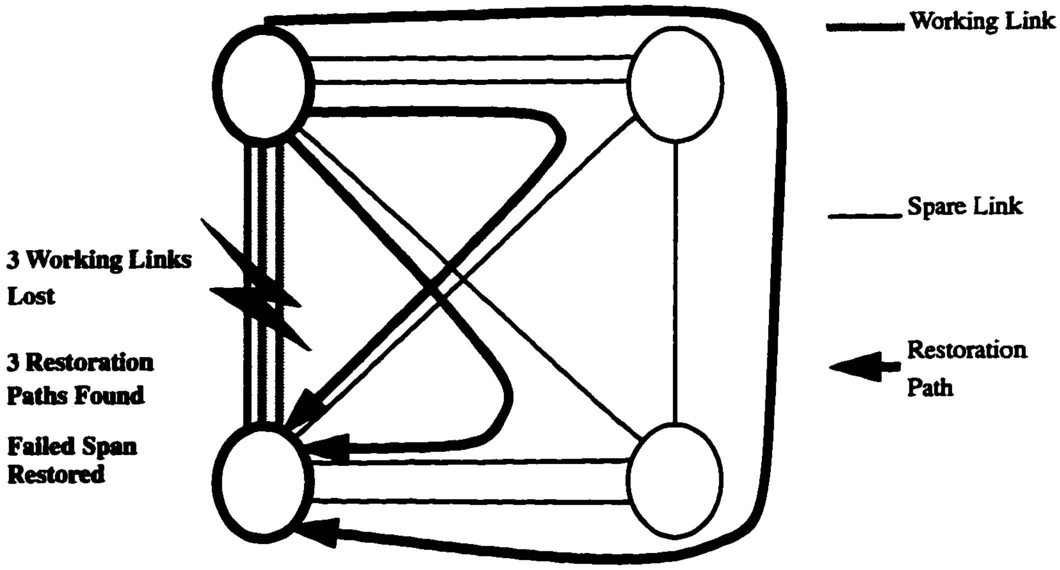


FIGURE 1.2 Restoration Pathset formed, from a Mesh Network's Distributed Spare Capacity, in Response to a Span Failure

A restorable mesh network will have an associated capacity redundancy which measures how much spare capacity it contains compared to working capacity. Network redundancy is given by:

$$Redundancy = \left(\sum_{i=1}^{S} s_i\right) \Big/ \left(\sum_{i=1}^{S} w_i\right) \qquad \text{(EQ 1.1)}$$

Where, S is equal to the number of network spans, s_i is equal to the number of spare links present on span i, and w_i is equal to the number of working links present on span i. Redundancy measures how efficiently a network is provisioned with its spare capacity.

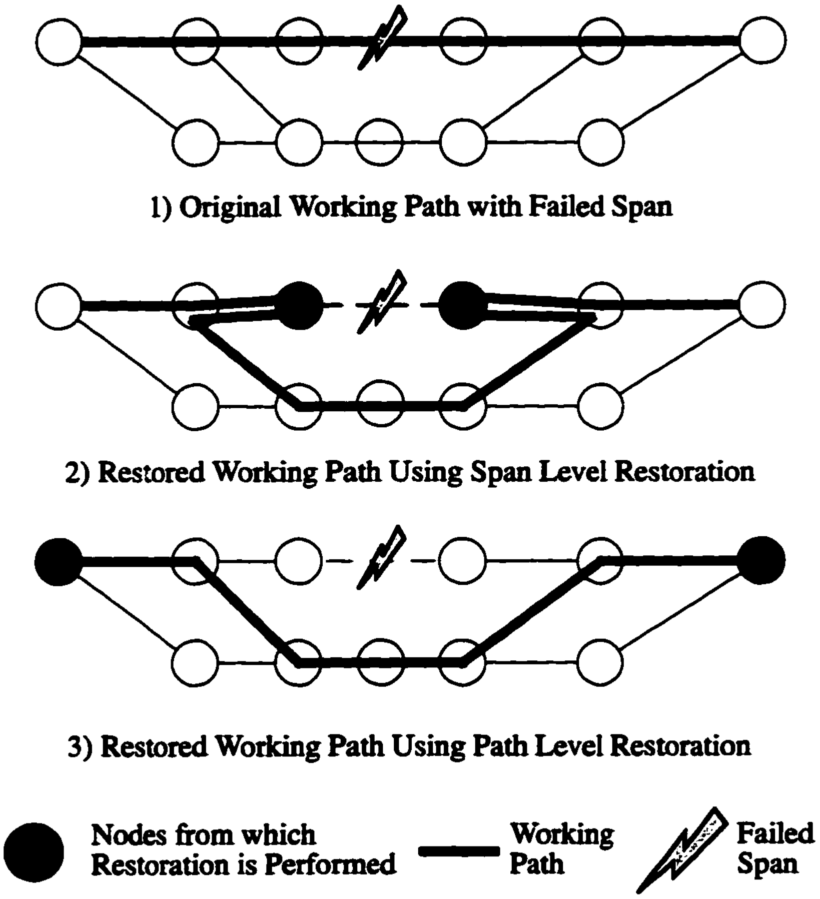
Also, a restorable mesh network will have an associated restorability which gives a measure of what fraction of working links lost, over all possible failed spans, can be recovered using a particular restoration method with a given set of spares. Network restorability is given by:

$$Restorability = \frac{\sum_{i=1}^{S} MIN(w_i, k_i)}{\sum_{i=1}^{S} w_i} \qquad \text{(EQ 1.2)}$$

Where, $MIN(A, B)$ is set equal to lower value of A or B, and k_i is equal to the number of restoration paths which are available for span i using a particular restoration method. Taking the minimum of the number of restoration paths for a failed span and the number of working links present on a failed span is required so that those spans which are super restorable (a span is super-restorable if has a larger number of restoration paths available to it than it has working links) are accounted for in the restorability calculation. A span is still only 100% restorable if it is super restorable.

Mesh restorable networks can be restored using span level restoration or path level restoration. Span level restoration restores a span failure by forming restoration paths, between the end nodes of the failed span, to reroute the lost traffic around the failure. Path level restoration restores a span failure by forming restoration paths between each demand pair which lost a working path because it was routed through the failed span. Path level restoration is more efficient in its use of spare capacity so, in general, a fully path level restorable network requires less spare capacity to achieve than a fully span level restorable network.

The reason that path level restoration is more efficient than span level restoration is that it directly reroutes traffic between the demand pairs affected by a span failure; span level restoration only reroutes traffic between the end nodes of the failed span with out consideration to the routing of the demands. This can result in restoration paths which utilize more spare links than is strictly necessary. Figure 1.3 shows an example of the restoration of a failed span using both span and path level restoration. However, path level restoration is more complicated to execute because restoration paths must be simultaneously formed between a number of different node pairs while in span restoration paths are only formed


1) Original Working Path with Failed Span

2) Restored Working Path Using Span Level Restoration

3) Restored Working Path Using Path Level Restoration

FIGURE 1.3 Span and Path Restoration

between the single pair of end nodes of the failed span. For this reason, and because path level restoration has only recently been evaluated for mesh network restoration [11,16], the work presented in this thesis will only be concerned with span level restoration.

A network's restorability (under span-restoration) will depend on the restoration algorithm used to form the restoration pathset. A network's restorability will be at a maximum, while using a minimum of spares, if the restoration pathset is equivalent to the failed span's max-flow number. A span's max-flow number is the maximum possible number of restoration paths which can be formed, through surviving network spares, to restore a failed span. Algorithms which can generate a max-flow pathset are generally fairly complex (at least $O\left(N^3\right)$ [2].) A compromise to an absolute maximum flow pathset is a pathset generated using a k-shortest paths (KSP) algorithm. This algorithm can be thought of as equivalent to iteratively taking out the shortest path possible from the network's spare capacity. KSP's benefit is that it can be calculated using an algorithm with a lower computational complexity, and, in practice, KSP pathsets can be formed in realtime, in a single iteration, by a self-organizing distributed autonomous restoration protocol known as the SHN [3,4]. Additionally, a study which compared KSP restoration to max-flow restoration [5] in a wide range of transport network models found that in 99.9%, of the network restoration cases studied, the two methods generated equivalent capacity restoration pathsets. Therefore, the network restorabilities calculated in this thesis were evaluated using a centralized KSP restoration algorithm as the reference solution sets.

In the event of a span failure, the signals which flowed through the working links on the failed span can be rerouted using calculated restoration paths which are formed from the network's spare links. The restoration pathsets are calculated using either a central mechanism or a distributed local mechanism. Using a central mechanism, the restoration pathset is calculated, using a central computer, and then downloaded to the network nodes which then form the restoration pathset. Use of a central mechanism introduces a delay in restoration due to the time required for the affected network nodes to alert the central mechanism of the span failure and due to the time required for the central mechanism to download the restoration pathset to the network nodes. Additionally, an accurate representation of the network's current configuration must be maintained so that the central mechanism generates a restoration pathset which is appropriate to the restoration of the failure. Using a distributed local mechanism, the restoration pathset is formed, in a distributed self-organizing pattern-forming process, using the computational power present in each of the network's DCS machines. The entire restoration pathset is evolved simultaneously as a composite pattern of mutually feasible restoration paths in the available spares. The use of a distributed restoration mechanism offers several advantages; it is not required that a database of the network configuration be maintained for the purpose of restoration, there is minimal delay in the initiation of the calculation of the restoration pathset, and the task of calculating the restoration pathset is distributed among the network's nodes which can result in reduced calculation time.

Both kinds of restoration mechanism, discussed in the previous paragraph, generate a restoration pathset in response to a span failure. There is another kind of restoration mechanism called preplanning which attempts to reduce signalling time and to eliminate calculation time by generating a restoration pathset for each possible span failure before any failure happens. The restoration pathsets are distributed to the network nodes in anticipation of a span failure. In the event of a span failure, each network node then proceeds to directly implement the crossconnections indicated in the node's locally stored restoration database for the span failure. The primary disadvantage of using preplanned restoration is that any changes to the network configuration, since the last update of the preplanning database, would not be included in the database and could compromise the restoration of a span failure. Preplanned restoration requires that the stored restoration pathsets be continuously updated to track changes in the network configuration.

A failure will have minimal impact on a network if it is restored within 2 seconds [6]. Mesh restorable networks have been widely evaluated since their first proposal in 1987 by Grover [3]. It has now generally been accepted that it is possible to *calculate* a restoration pathset within a time which is less than the 2 second limit using distributed methods essentially as proposed and patented in [3,4]. However, in practice, the total restoration time will include more than just the time required to calculate a restoration pathset. It may also include a significant time to actually effect the crossconnections to form the restoration pathset and switch the severed signals onto the restoration paths. Unfortunately, the current generation of DCS machines was not designed with specific consideration given to realtime network restoration. The current DCS implementations have crossconnect set times of up to 1 s, serially per crossconnection [7]. Although some part of these long crosspoint times may be artifacts of specifications developed for growth provisioning applications*, not restoration, and may not represent the machine's actual technical limits, it is of research interest to consider how to overcome (at a network level) the problem of slow crossconnections in realtime restoration. When considering the overall restoration time of a mesh restorable network, it has been found that a span failure may not be fully restored within the required 2 seconds [8] mainly due to the slow crossconnect time of some current DCS implementations.

1.4 Problem Introduction

The main disadvantage to using mesh based restorable networks is that the restoration time will be slower than when using dedicated spare capacity restoration methods such as APS or SHRs. It has been recognized, however, that as long as a span failure is restored within 2 seconds there will be minimal impact on services because connections in progress are not dropped. However, in the worst case a DCS machines may take up to 1 second to form each crossconnection. Such a slow DCS crossconnection time results, conceptually, in an interesting quandary; that it is possible to quickly calculate an efficient restoration pathset in response to a span failure but it may not be possible to actually *implement* the restoration pathset quickly enough to minimize the effects of the failure. This is ironic, as the challenge, in early mesh restorable network research, was to prove that the fast calculation of a restoration pathset was technically possible. Now that this problem has been solved, it is found that mesh restorable networks may still not be quickly restorable because of slow DCS crossconnect times.

Clearly in the regime where implementation, not computation, time dominates, the restoration time can be reduced if the crossconnection workload at each node can be mini-

* for instance, current specifications include 100 msec to test each crosspoint. During restoration, this may reasonably be waived as there is already an emergency on hand (pg. 406 in [2]).

mized. But how to do that in a mesh restorable structure is quite unclear, since it is only known at failure time which crossconnection pattern is needed. This thesis, therefore, attempts to evaluate a novel idea; that of partially or wholly preconnecting crossconnections in the network's spare capacity in statistical anticipation of a failure. An analysis of the network is performed and those crossconnections, which are deemed to be the most beneficial in contributing to the restoration of all possible span failures, are preconfigured. In the event of a span failure, the preconfigured * (PC) paths, which are contained within the structures formed by the preconfigured crossconnections within the spare capacity, can be applied to the restoration of the failure. If these structures are efficiently formed then it may be that a large proportion of restoration paths required by the span will already be formed within the network's spare capacity. The PC paths may be "pruned" out of the PC structures by selectively breaking preconfigured crossconnections to free the contained PC paths. If the PC paths contained within the network are not adequate to fully restoring the span failure, then a standard mesh network restoration method can be executed to scavenge additional restoration paths from the remaining spare capacity.

In other words, the idea and problem is as introduced in [9]:

> "Mesh restorable networks have $O(1/(\bar{d}-1))$ redundancy in the form of spare links to permit 100% span restorability via restoration re-routing; $\bar{d}$ is the network average node degree. In work to date, these spares are crossconnected only on demand, after a restoration pathset has been computed in response to an actual failure. This is because, in general, the restoration pathset for each possible network failure requires a significantly different pattern of restoration nodes and spare link crossconnections at each node. We hypothesize, however, that if a restoration re-routing mechanism is well characterised, it should be possible to analyse a mesh-restorable network to derive a description of the statistically most-likely crossconnection patterns that each node is called on to deploy, assuming all span failures are equally likely (or using relative failure probabilities, if known.) An example is the selfhealing network protocol, which results in k-shortest link-disjoint (KSP) pathsets for restoration. Predeployment of the statistically most advantageous pattern of crossconnection at each node should therefore accelerate restoration: On average, some fraction of the required crossconnections will already be in place at each node for a given failure. The DCS need then consume real time only for the crossconnections remaining in the KSP pathset developed by distributed restoration."

* the slightly more general term "preconfigured" can be thought of as synonymous with "pre-connected" throughout this work.

The design of an efficient PC plan must consider the placement of preconfigured crossconnection that will be effective in the contribution of PC restoration paths for all possible span failures. It must contain PC structures which can be converted into an effective restoration pathset for any failed span by selectively breaking preconfigured crossconnections within the structures.

1.5 Research Objectives

The objective of this thesis is to evaluate the effectiveness of strategies for establishing preconfigured spare capacity for the restoration of span failures in mesh restorable networks. It is hypothesized that the preconfiguration of network crossconnections can speed up the restoration of span failures by reducing the number of crossconnection events which are required to form a restoration pathset.

A number of different approaches will be considered for generating preconfigured restoration plans. Different kinds of elemental subgraphs or patterns will be evaluated for use in the formation of PC plans, including trees and cycles. Also, a pair of genetic algorithm PC design generation algorithms will be used to form PC pattern sets which can employ a heterogeneous mix of a variety of subgraph types.

The effectiveness of preconfigured restoration will be evaluated, and compared to KSP restoration, with regard to network restorability and with regard to the reduction or increase of the number of crossconnect events required to both form the restoration pathset in realtime and to prune-off unneeded portions of the subgraphs from which useful restoration paths are exploited.

1.6 Outline

Chapter 2 contains an introduction to the key concepts and terminology for preconfiguration and a review of previous work done in the design and evaluation of preconfiguration.

Chapter 3 presents a pair of heuristics which iteratively generate preconfigured trees within a network's spare capacity. The trees are generated using a maximum weight spanning tree algorithm from weights assigned to the network spans as determined by each span's contribution to the KSP restoration of all other network spans. By forming trees using these weights it is conjectured that a PC plan will result which contains efficient PC restoration paths.

Chapter 4 contains a pair of heuristics based on genetic algorithm (GA) techniques. These heuristics generate PC restoration plans using patterns evolved with a genetic algorithm. The patterns are unconstrained in their basic graph nature, i.e. they do not have to belong to a specific pattern type (such as cycle, segment, or tree.) The patterns are formed by the genetic algorithm on the basis of a fitness score which measures how effectively a pattern provides the network with already formed segments which are immediately useful for restoration. The performance of a PC plan, generated using a genetic algorithm, will indicate how effective a plan can be towards PC restoration, if it is unconstrained in the type of patterns it may can contain.

Chapter 5 uses integer programming (IP) techniques to generate strictly optimal PC plans using preconfigured cycles as a homogeneous subgraph building block. The use of an integer program will generate a solution which is optimal with regard to the problem given to it; the IP generated preconfigured design will be optimal with regard to maximizing the number of useful PC restoration paths contained within the design. The generated PC designs give an upper limit to the effectiveness of using preconfigured cycles.

Chapter 6 presents a theoretical upper limit for PC effectiveness using the tree and cycle subgraph types, in terms of the number of immediately useful restoration paths that a single preconfigured pattern can provide. Additionally, the theoretical upper limit to the number of preconfigured paths, that can be contained within any preconfigured pattern regardless of its configuration, is evaluated. This analysis gives a clear theoretical insight into the intrinsic effectiveness of the various pattern types when applied towards preconfiguration, and goes a long way towards explaining the experimental results as they turn out.

Motivated by the findings of chapters 3 to 6, which strongly suggest the cycle as the best elemental subgraph for use in PC strategies, **chapter 7** presents a distributed preconfigured cycle generation protocol which can generate PC cycles within a network's spare capacity using only the computational power present within the network's DCS machines.

Chapter 8 summarizes the results and suggests future research for mesh network restoration which exploits the concept of spare capacity preconfiguration.

2. Preconfiguration Concepts and Prior Work

2.1 Introduction to Preconfiguration

As discussed, a mesh network has several advantages when it used as the solution to the network restoration problem. It is fully configurable, which allows for easy adaption to unforeseen growth, and it can provide a fully restorable solution with substantially less redundant (spare) capacity than a dedicated protection restoration method. The mesh network's primary disadvantage when used in restoration is its speed. It will not match the 50 msec restoration time that SHRs and APS can provide. However, such a fast restoration speed (50 msec) is not, strictly speaking, necessary in most instances as it is generally accepted that if a span failure in a network can be restored within 2 seconds then the failure's effects will still be negligible [6]. 50 msec of outage causes 1 or 2 error second. But restoration at say 1.5 seconds only causes only 2 or 3 error seconds. Few data sessions ever time out as quickly as 2 seconds and, for voice, 50 msec is a click as opposed to a pause of a second or two. On the other hand, above 2.5 seconds voice users are returned to dialtone! Therefore, 2 seconds is a very well reasoned and cost-effective target for restoration times. It has been shown that a highly efficient KSP-equivalent restoration pathset can be calculated in this 2 second time [3]. However, some existing DCS machines were not designed with restoration in mind and have crossconnect times which may be up to one second serially per crossconnection to be set. This suggests that, although it is technically possible to calculate a restoration pathset for a span failure within a reasonable amount of time, it may not be possible to form the crossconnections, required to form the restoration pathset, quickly enough to prevent large scale consequences within the network.

In these circumstances, the best way to reduce the total crossconnection time would be to reduce the number of crossconnections which must be made. It was first conjectured in [9] that an analysis of the working and spare capacity distribution within a network could generate a list of crossconnections, that if set in anticipation of a span failure, would contribute, in a statistical sense, a useful number of the crossconnections which would be required to form the restoration pathset of any possible span failure. If the preset crossconnections are to be useful towards the overall restoration of the network, they must be preset with consideration of the restoration of all possible span failures as it is not possible to predict, in advance, when a span will fail or which will be the next span to fail. In other words, crossconnections are preconnected or preconfigured (PC) in anticipation of the failure of a network span and, because it can not be known in advance which will be the next span to fail, crossconnections must be preconfigured with consideration given to all possible span failures. In the event of an actual failure, the preconfigured crossconnections can be used to form restoration paths for the failed span and, thus, will reduce the number of crossconnections which must be actively made (or closed) to restore the span failure. The preconfigured paths exist within the patterns, formed by preconfiguration throughout the network, and can be used by pruning extraneous crossconnections to free the paths.

Figure 2.1 shows an example of how the preconfiguration of a network's spare capacity can contribute restoration paths to multiple failed spans. In the example, a simple preconfigured pattern is formed using 4 spare links and 4 preconfigured crossconnections. This pattern contains within itself enough preconfigured paths to fully restore the working links which are lost on all possible span failures. The preconfigured paths are extracted from the PC plan by selectively breaking preconfigured crossconnections. Once the PC paths are freed, or isolated, from the PC patterns of which they were a part, the interrupted working signals can be rerouted over pruned-down elements of the PC paths. Figure 2.2 contains a similar example but shows the restorations paths available, for a failed span, within a complex PC pattern with a high nodal degree.

In contrast to standard mesh network restoration techniques which form restoration pathsets by forming crossconnections completely on demand, preconfigured restoration forms its restoration paths by unmaking some crossconnections, leaving a minimum of active make crossconnections to complete restoration, in general. An important consideration, from a real time viewpoint, is that pruning to isolate desired restoration paths may be done after the restored signals are already flowing. We return to issues related to crossconnection make and break speed and technology aspects later.

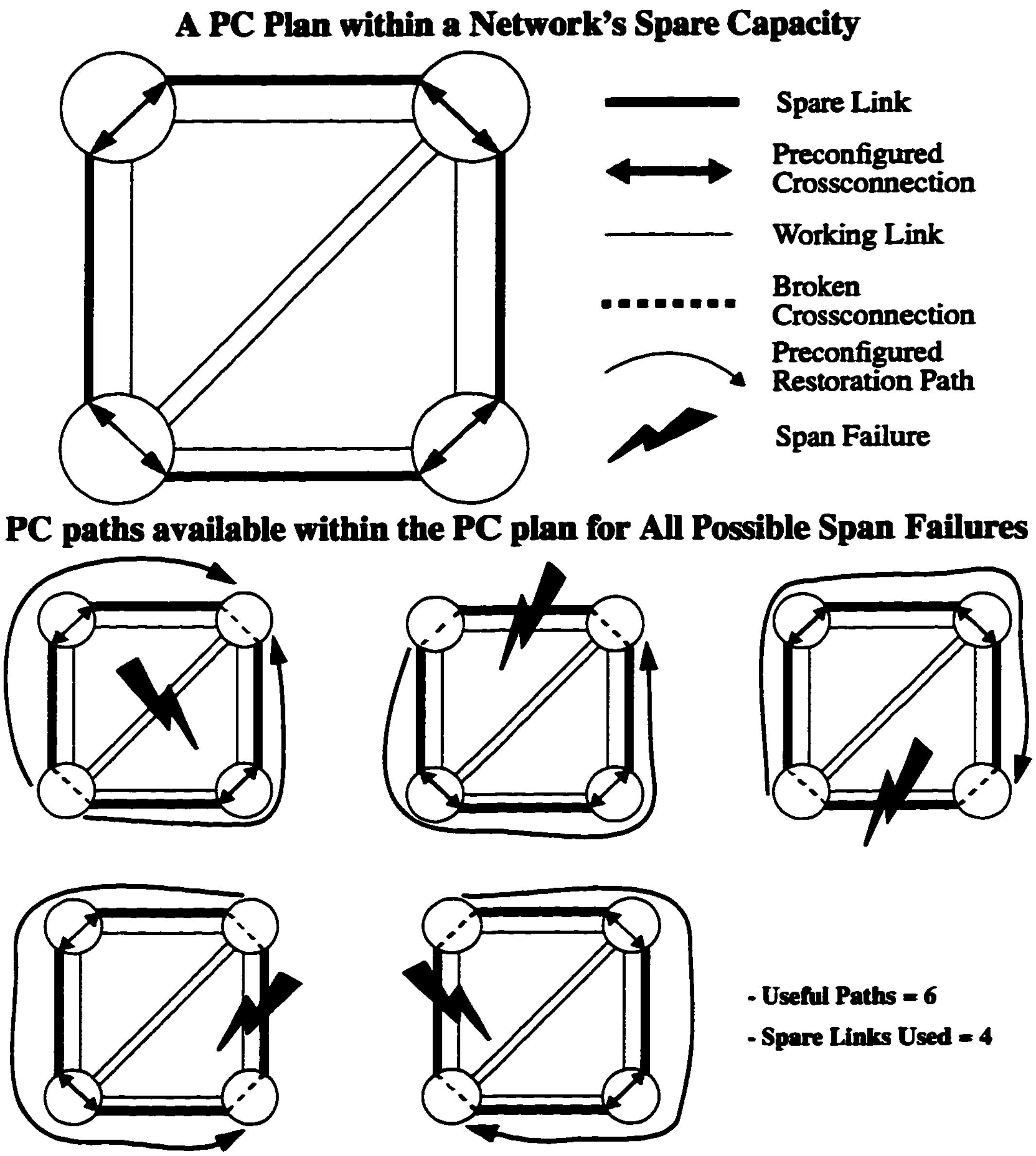

A PC Plan within a Network's Spare Capacity
Spare Link
Preconfigured Crossconnection
Working Link
Broken Crossconnection
Preconfigured Restoration Path
Span Failure
PC paths available within the PC plan for All Possible Span Failures
- Useful Paths = 6
- Spare Links Used = 4

FIGURE 2.1 A Preconfigured Pattern within a Network's Spare Capacity and the Preconfigured Restoration Paths available for All Possible Span Failures

2.2 Definition of Preconfiguration Terminology

Before further discussion of preconfiguration takes place, a number of definitions should be stated explicitly.

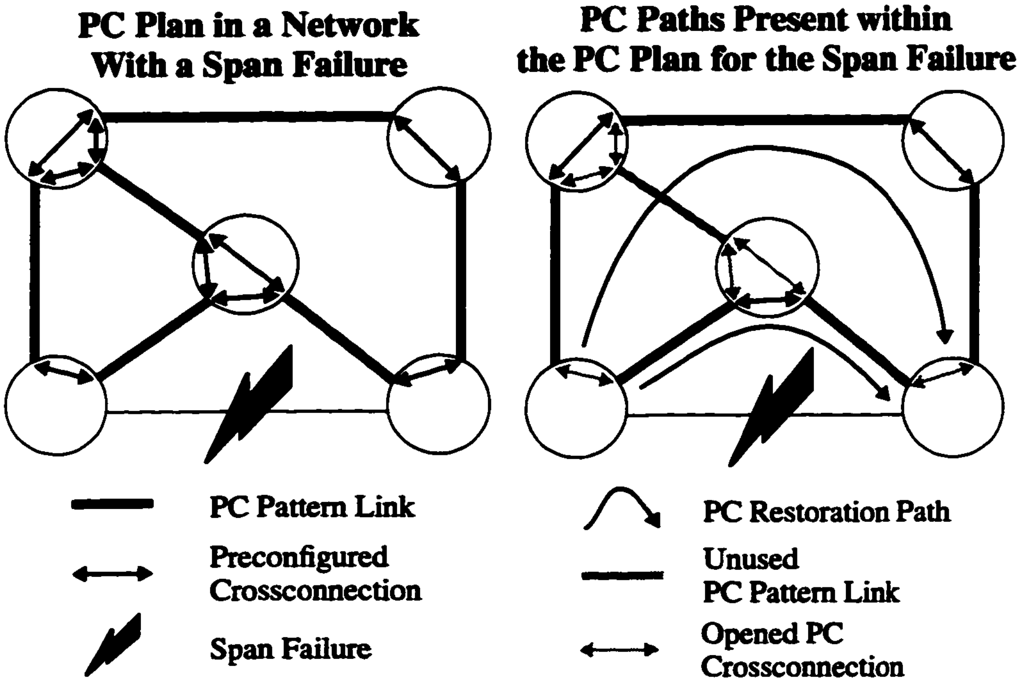


FIGURE 2.2 A PC Plan containing a PC Pattern which Requires Multiple Crossconnections on a Single Link and the PC Paths available For a Failed Span

A *preconfigured crossconnection* is a crossconnection at a node which connects a pair of spare links. The preconfigured crossconnection is set among the network's spare links in anticipation of a span failure so that it may attempt to contribute towards the restoration of the failure. A *preconfigura:ion plan* or *preconfigured restoration plan* refers to the network wide collection of all preconfigured crossconnections at all nodes and the spare links which they interconnect.

A *preconfigured pattern* is a pattern formed from a set of connected spare links and preconfigured crossconnections. A pattern results from a set of links which form a connected subgraph.

A *useful path*, or *useful PC path*, is a PC path, for a specific span failure, which is present within the network PC plan and is useful in contributing to the restoration of the failure. It is possible that a PC plan can contribute more PC paths, to a span failure, than there are working links on the failed span. If this is the case, then those PC paths, which are in excess of the number of working links, are not useful. If a PC plan can not contribute more PC paths, to a span failure, than there are working links on the span, then all supplied PC paths will be useful.

An *uncovered working link* is a working link which does not have a preconfigured restoration path available to it. This may not mean it is unrestorable. It may still be restorable using on-demand crossconnect operations. Here we mean only that a useful preconfigured replacement path is not provided for it in the PC plan. An *unallocated spare link* is a spare link which does not belong to any subgraph element of the preconfigured restoration plan for a network. A *pruned spare link* is a spare link which was a part of a PC pattern but was not used to form a PC path, for a span failure, and was "pruned" to isolate it from those allocated links which formed PC paths.

The *k-shortest path (KSP) restorability* of a network is the network's restorability when using a KSP algorithm to calculate a restoration pathset for each of the network's failed spans with the available span sparing values and with no PC plan (the spare capacity is not preconfigured.) The KSP restorability of a network gives an upper limit to the network's overall restorability with a restoration method which is near optimal in its efficiency. The *preconfigured (PC) restorability* of a network is the network's restorability when using only isolated sections of the PC paths, contained within its deployed PC plan, to restore each the network's potential failed spans. The PC restorability of a network indicates how effective the network's PC plan is at providing immediately useful intact paths for the restoration of the network's span failures. An efficient PC plan could generate a PC restorability which can approach the restorability which results when just using KSP restoration. The *"2-step" restorability* of a network is the network's restorability when, first, all possible useful PC paths are applied to the span failure and then, if needed, additional on-demand KSP paths are found within both the network's unallocated spare capacity and the network's pruned spare capacity. The 2-step restorability of a network indicates if the use of PC restoration has exacted a penalty on the overall restorability of a network.

2.3 Prior Work in Preconfiguration

The first work which proposed the possible utility of statistically-protective preconnection and established that preconfigured restoration had the potential to speed-up the restoration of mesh networks is in [9]. In this paper, an analysis is made which establishes the upper bound to the number of crossconnections which can be preconfigured for use in forming the KSP restoration paths for all possible span failures. That is, an upper limit is established to the fraction of the crossconnections, that are required to form the KSP restoration pathsets of all possible span failures, which can be preconfigured in anticipation of all possible span failures. This value establishes an upper bound because consideration is only given to the number of crossconnections between each span pair which are preconfigured at each node and not to which specific link level crossconnections are preconfigured. This is required so the crossconnections within the network nodes line up to form coherent restoration paths between the endpoints, where they are needed. This upper bound is useful in that it gives an indication of preconfiguration's rather surprising potential. In the test networks evaluated [9] it was found that on average the upper bound to the fraction of crossconnections, which are required to form the network's KSP restoration paths, that can be preconfigured was 79%.

A second technique recently developed by the same authors [10] addresses the problem of the coherent formation of preconfigured crossconnections to form useful preconfigured paths by using integer programming (IP) techniques to design a PC plan. This was done for a number of test networks but using only preconfigured linear segments as a homogeneous class of building blocks. A "segment" is a simple connected non-closed non-branching linear sequence. The segment's end nodes are of degree* one while the intermediate nodes are of degree two. The use of an IP to solve a problem will generate a solution which is optimal with regard to the definition of the problem given to the IP. Using this method, the IP generates a coherently preconfigured segment plan which is optimal in that the total number of unprotected working links in the network is minimized by the design,

* The degree of a node is the number of high level connections / spans connecting the node to other adjacent nodes in the network. At the level of the network, each node's degree will be determined by the number of spans falling on it. At the level of a pattern, a node's degree will be determined by the number of spans falling on it which are a part of the pattern.

given the initial sparing of an optimal spare capacity plan for KSP (on-demand) restoration. The technique generated PC restorabilities which varied from 30 to 40% and 2-step restorabilities which were either very close to or exactly 100%. These restorabilities show the promise of preconfigured restoration and were the genesis of the idea for this thesis. Since only segments, were used, why not consider elemental trees, cycles, cycle-trees, etc. and their arbitrary mixtures.

2.4 Technical Considerations

Now in this work several different types of preconfigured patterns will be considered; a number of these will have nodal degrees which are not constrained to be less than two. To form such a pattern would require that a single DCS port card which terminates a spare link be connected simultaneously to more than one other port card. That is, we will be accepting and allowing for a link at a node to be connected to more than one other link through the DCS's crossconnect matrix. This requires some consideration of the technical issues involved.

The transport signals on each of the network's links are continuous time, framed, isochronous, time-division multiplexed digital carrier signals. It is easy to visualize a single incoming side of a link being connected to the outgoing direction of multiple other links. Each outgoing signal would be a regenerated digital copy of the incoming signal. This is an example of one way broadcast and is, in practice, used today for distributing broadcast signals, such as video, from a single source to multiple destinations within the network.

However, the broadcast of multiple *incoming* signals into a single outgoing signal does not have any physical meaning with regard to digital signals although, in a graph theoretic sense, we are quite able to contemplate and deal with it. Electrically, however, if the outgoing side of a port card is to be connected to the incoming side of more than one other port card then some manner of selection is implied to determine which of the multiple incoming signals will actually be broadcast on the outgoing side. Isochronous time continuous signals cannot be simply added electrically in the same time-space and bandwidth without destroying the contents of the signals (unlike, for example, analog signals on different subcarriers. The one link entity is the carrier here.) This could take the form of hardware added to the DCS which would add 1:N selective switching to each port card, as illustrated in Figure 2.3. However, this might be a costly and significant modification to existing DCS machines.

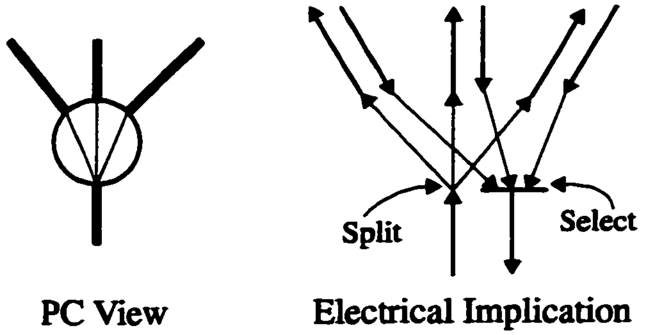


FIGURE 2.3 Electrical Implication of Permitting Multiple Crossconnections to a Single Port Card

In any event, for the purpose of evaluating pattern types which require multiple crossconnections joined to a single port card it is assumed that some method exists which permits the patterns to be formed. In other words, our present investigation is focused on the graph theoretic problem itself. If the benefits are strong, technical solutions will be further addressed with the motivation from this work. If such patterns show strong performance when applied to preconfigured restoration it could warrant modification to the design of future DCS machines.

A final technical consideration, which influence the way we tabulate results, is that the relative speed of forming (closing) and breaking (opening) a crossconnection may be different; that is, the required speed in closing a crossconnect may be different than in opening a crossconnect. In standard mesh network restoration crossconnections are only made (closed) when the restoration pathset is formed in realtime. However, when using PC restoration crossconnections are opened to free the desired PC paths from the PC pattern in which they were contained. In PC restoration crossconnections are *made* to connect the disrupted working flows to the PC paths and, also, if it is required that additional leftover restoration paths be formed within the network's remaining spare capacity (2-step restoration.) When comparing the PC and KSP restoration, therefore, the total number of each kind of crossconnection event (crossconnections closed and opened) will be counted and maintained separately. This is because there may be a significant difference in the time required to make a crossconnection and the time required to break one. By recording these separately we allow others to use our data more effectively in predicting overall realtime benefits from this work depending on their technical assumptions. The time required to form a crossconnection is made up, primarily, of a calculation time required to route the crossconnection through the DCS nodal switching matrix (assuming a multi-stage switching fabric), a switching time required to form the crossconnection within the matrix and a testing time to validate the crossconnection (which arguably may be waived in a restoration context.) However, the opening of a crossconnection does not require calculation as its internal path through the switch matrix is known in advance. There will likely not be a testing time associated with opening a crossconnection as the resources required to form the crossconnection are being freed and not allocated. This leaves only the time required to physically open the logical crossconnection which may actually be a multi-hop internal path in the DCS. These factors suggest that it will generally take more time to *make* a new crossconnection than to open an existing one.

2.5 Method used to Evaluate the Effectiveness of a PC Plan

In the chapters that follow, a PC plan is evaluated on the basis of its restorability (for both PC and 2-step restorabilities) and the number of crossconnection events required to extract the isolated PC paths from the PC plan and to form leftover (2nd-step) KSP paths (if required.) In this section, the basic method used to evaluate a PC plan's performance is discussed.

The are 3 steps in the evaluation of a PC plan for a specific span failure:

- Step 1: Connection state to PC subgraph abstraction. The raw PC plan, which is comprised only of a list of the link to link individual crossconnections which are preconfigured within the network, is analysed and a higher level abstraction of a list of the logical unit capacity patterns which are contained within the PC plan is obtained.

- Step 2: Exploit useful portions of subgraphs. All PC paths which are useful in restoring the specific span failure are extracted from each PC pattern and a record is maintained of the crossconnections which must be opened to free the PC paths from the pattern. The pruned spare links, which were not useful in forming PC paths, are available for use in 2nd-step restoration (if required) at the completion of this step.
- Step 3: "Top up" restoration on demand. If required, 2nd-step leftover KSP paths are extracted from the unallocated and now also released spare links remaining in the network. These spare links can include unallocated spare links and allocated spare links which were not used in forming PC paths (the pruned spare links generated in the previous step.) The leftover KSP paths try to make use of the remaining preconfigured crossconnections in the network to reduce the number of crossconnections events.

These steps are repeated for all possible span failures within the network. As each step is executed the number of PC paths and the number of leftover KSP paths is stored for each span failure. Additionally, a record is maintained of the number of crossconnections which had to be asserted to restore each span failure; a separate record is also maintained of the number of crossconnections which had to be opened. After the PC plan has been evaluated for all span failures, an overall representation of the total number of PC and leftover KSP paths can be used to calculate the pure PC and 2-step restorabilities. Additionally, the number of crossconnects closed, required in 2-step restoration, can be summed for all span failures to generate the total crossconnection closures required to form the 2-step restoration pathsets of all spans. A similar procedure can be used to generate the total crossconnection openings required to form the 2-step restoration pathsets of all spans. These totals give an overall network view of the crossconnection events required when using PC plus 2nd-step KSP restoration. The PC and 2-step restorabilities, as well as the crossconnection totals, can then be compared against equivalent metrics resulting from the use of on-demand KSP restoration alone. A more detailed explanation of each step follows in the next sub-sections.

2.5.1 Step 1: Abstraction of the PC Subgraphs contained within a PC Plan

As previously discussed, the first step in evaluating the performance of a PC plan is to extract, from the detailed crossconnection data which constitutes implementation of the PC plan, the logical subgraphs which are formed by the given set of preconfigured crossconnections. This is a straightforward step; the preconfigured crossconnections are traced through the network to form the PC patterns. When all PC crossconnections are accounted for, this stage of evaluation terminates, and a list of logical subgraphs is on hand.

2.5.2 Step 2: Exploitation of Useful PC Paths contained within each PC Pattern

For a particular span failure, each PC pattern will be evaluated to determine the PC paths which are contained within the pattern. This is done by iteratively taking the shortest path contained within the pattern which joins the end nodes of the failed span. Successive shortest paths are removed from the PC pattern until the subgraph is disconnected between the end nodes of interest. As each path is taken out, the PC crossconnections, which join the PC path's links to other links in the pattern, are pruned (a record is maintained of the number of crossconnections which are broken.) Figure 2.4 is an example of the extraction of PC paths contained within a PC pattern. In the example, two paths are removed successively from the one PC pattern by iteratively choosing the shortest path within the pattern. As each path is extracted, the crossconnections which connected it to the rest of the pattern are unmade. In the end, all that remains of the original pattern is a fragment which cannot contribute any PC paths but may be useful if released for use, when in the general case, some additional on-demand KSP paths are required (i.e., 2 steps to complete restoration.)

2.5.3 Step 3: Formation of Further on-demand KSP Paths (2 Step Restoration)

The final step, in evaluating the overall restorability of the network is to form KSP paths from the network's remaining spare capacity, if needed. At this stage unallocated spares and pruned spares are both available. A shortest path (SP) algorithm is used to iteratively generate successive single shortest paths to restore the span failure being tested. However, as each shortest path is generated, it is not simply formed from randomly chosen available spare links. Instead, a simple method is used which tries to make use of still present PC crossconnections to assist in forming the on-demand shortest paths. This further reduces then number of crossconnection closures required for the failure restoration.

Each iteratively generated on-demand SP path is formed in the following manner: at the starting node (one of the end nodes on the failed span is arbitrarily chosen to be the starting node) the first available spare link is chosen on the span which joins the starting node

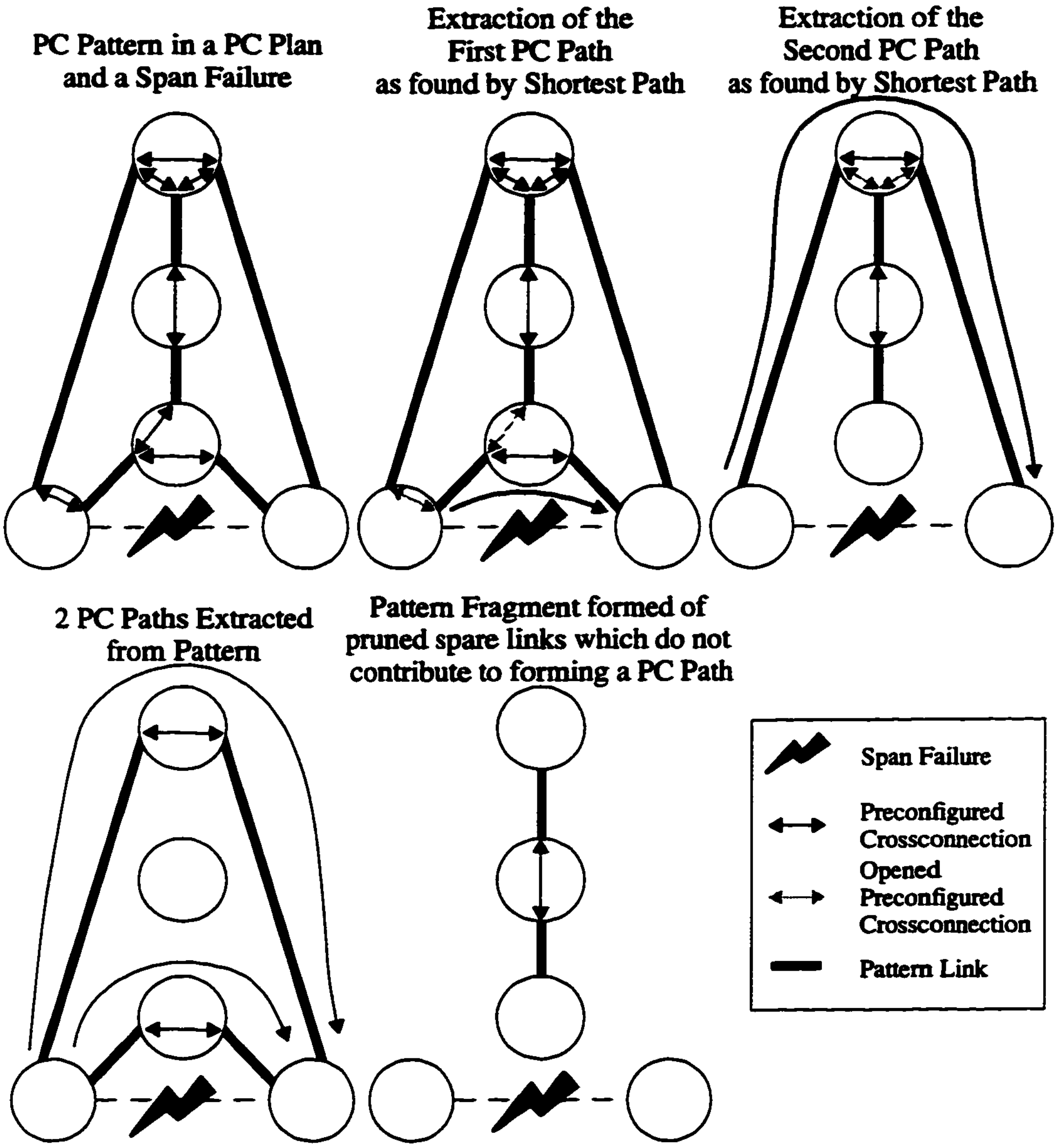

PC Pattern in a PC Plan and a Span Failure
Extraction of the First PC Path as found by Shortest Path
Extraction of the Second PC Path as found by Shortest Path
2 PC Paths Extracted from Pattern
Pattern Fragment formed of pruned spare links which do not contribute to forming a PC Path
Span Failure
Preconfigured Crossconnection
Opened Preconfigured Crossconnection
Pattern Link

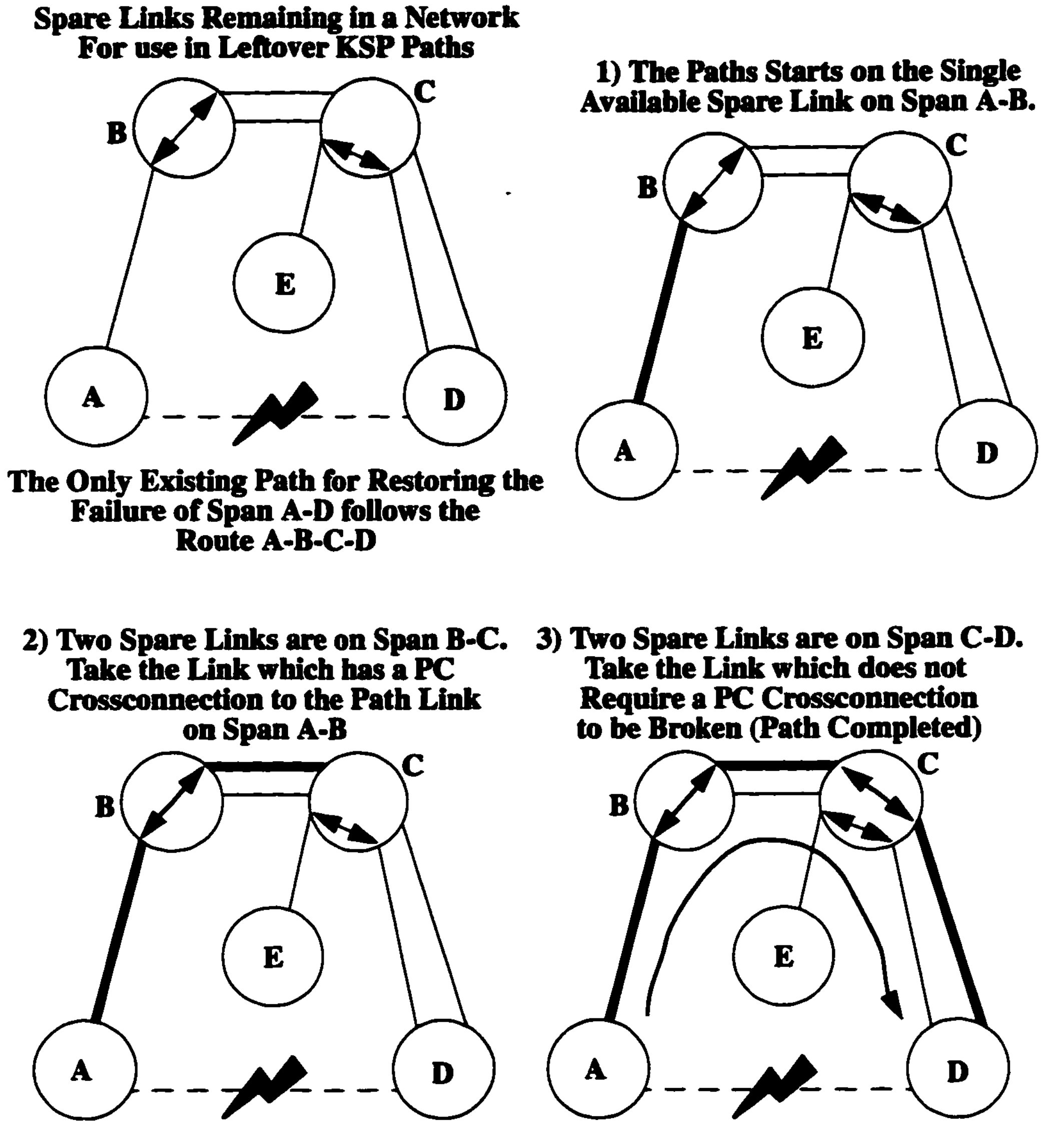
FIGURE 2.4 Extraction of the PC Paths contained within a PC Plan by successive removal of the Shortest Path within the Plan

to the next node along the route generated by the SP algorithm. At the next node, however, along the restoration route, the spare link is inspected to determine if a preconfigured crossconnection exists between it and a spare link on the span which connects the node to the next node along the route. If such a crossconnection exists then the construction of the SP restoration path continues along the spare link which is connected to this crossconnection; otherwise, an inspection of the unoccupied spare links which exist on the next span is made and the available spare link, which would require the least number of preconfigured crossconnections to be opened, is chosen. This continues until the SP path is fully formed. In summary, a SP route is converted into a path by choosing links, along the route, with preference given to those links which are in a preconnected fragmentary state which assists in forming the path, and to those links which minimize the number of preconfigured crossconnections which must be opened. This method is applied to the construction of each of the iteratively generated on-demand SP paths. Constructing the on-demand KSP paths in this way, helps to further reduce the number of crossconnection events by making use of preconfigured crossconnections to assist in the formation of the leftover 2nd-step KSP paths.

Figure 2.5 is an example of how a leftover KSP path is formed. In the example the only possible leftover KSP path follows the route A-B-C-D. At node A, construction of the path is started on the only available spare link on span A-B. At node B, there are two spare links on span B-C which can continue the construction of the path. However, one of the links has a preconfigured crossconnection which connects it to the path link on span A-B. So, this link is chosen to continue the construction of the path so as to reduce the number of 2nd-step crossconnections which must be asserted. At node C, there are two possible spare links, on span C-D, with which the construction of the path can be completed. However, one of the spare links has a preconfigured crossconnection connecting it to a spare link on span C-E while the other has no crossconnection. Therefore, in order to minimize the number of crossconnections which must be broken, the path construction is completed on the spare link which has no preconfigured crossconnections.

2.6 Validation of PC Plan Generation and Evaluation

For all the PC design methods, discussed in the following chapters, PC plans were evaluated using an independent PC evaluation program which implemented the evaluation method discussed in section 2.5. Each design method generated a detailed list containing all of the preconfigured link to link crossconnections contained within the method's generated PC plan. In other words, the program which implements each design method and the program which evaluates the PC plan are separate and independent of each other. The



Spare Links Remaining in a Network For use in Leftover KSP Paths

The Only Existing Path for Restoring the Failure of Span A-D follows the Route A-B-C-D

1) The Paths Starts on the Single Available Spare Link on Span A-B.

2) Two Spare Links are on Span B-C. Take the Link which has a PC Crossconnection to the Path Link on Span A-B

3) Two Spare Links are on Span C-D. Take the Link which does not Require a PC Crossconnection to be Broken (Path Completed)

FIGURE 2.5 Routing of On-demand 2nd-Step KSP Paths which Attempts to Make Use of Preconfigured Crossconnections

operation of the PC evaluation program was hand verified for the different design methods using a small test network (Net1.)

Each program, which implemented a PC design methods, generated a raw output file which contained the detailed link to link crossconnections contained in the generated PC plan. The evaluation of this PC plan was performed by the program described in the previous paragraph. The PC patterns generated within an output PC plan were inspected for each test network, to verify that the category of generated pattern was correct. Finally, the execution of each PC design program was hand verified using a small test network (Net1.) Full source code listings for all PC design and evaluation programs are available in [17].

2.7 Properties of the Test Networks

There were 5 test networks used with the PC design methods presented in the following chapters. The networks are labelled Net1, Net2, Net3, Net4, and Net5. Figures 2.6 to 2.10 show each network's topology. Appendix A contains each network's standard network interface file (SNIF) which gives detailed information about the network, specifically about the number of working and spare links which are present on each network span. The working and sparing placement plan, for each network, was designed using the integer program (IP) method presented in [11]. The resulting design is optimal, in the sense that the total physical fibre distance of the working and spare capacity is minimized while insuring that the network is fully restorable when using on-demand max-flow span restoration. The network designs, which result, when using this method are very capacity efficient and will give a true indication of how effective PC restoration can be when it is deployed within an efficiently spared network plan.

We now go on to consider the performance of tree subgraphs as the elemental building block for preconfiguration.

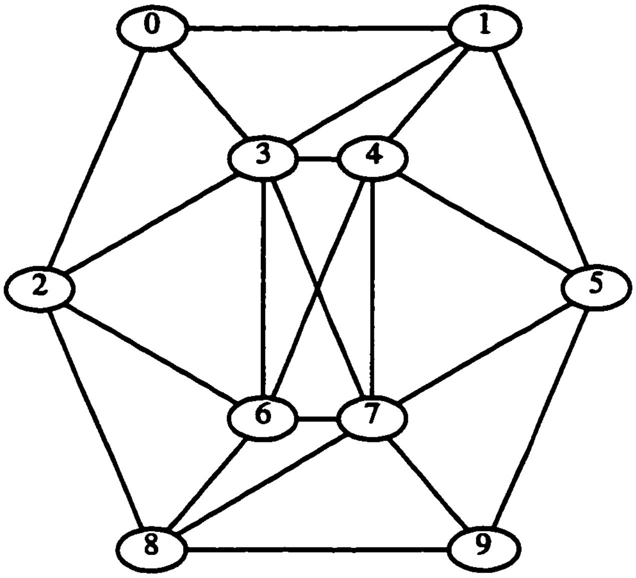


FIGURE 2.6 Test Network Net1

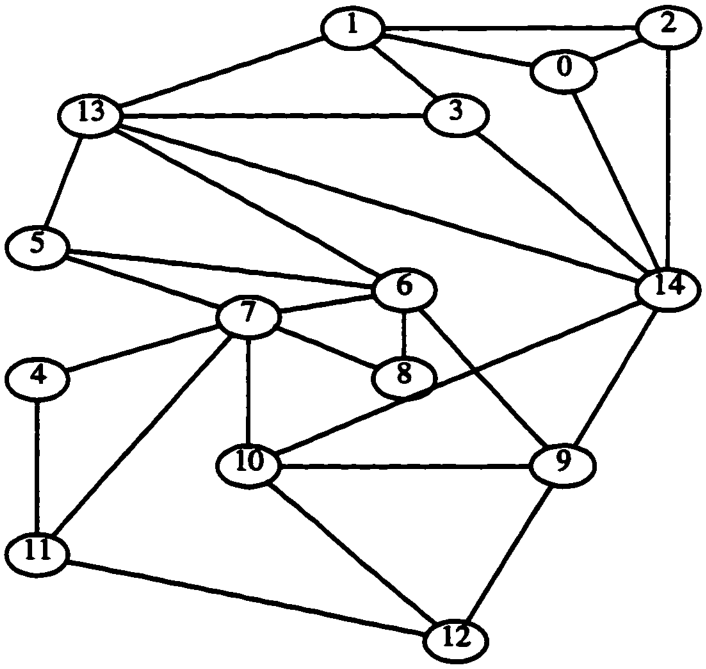


FIGURE 2.7 Test Network Net2

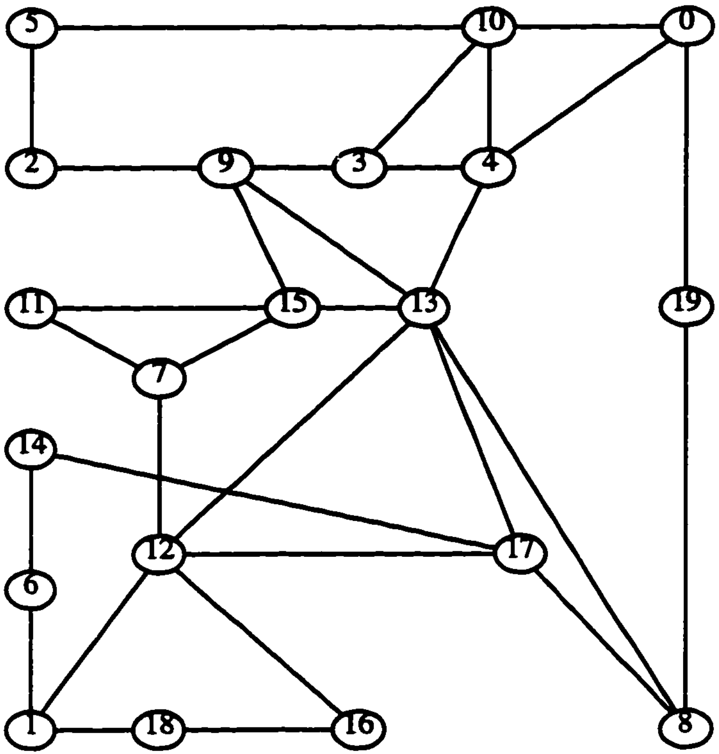


FIGURE 2.8 Test Network Net3

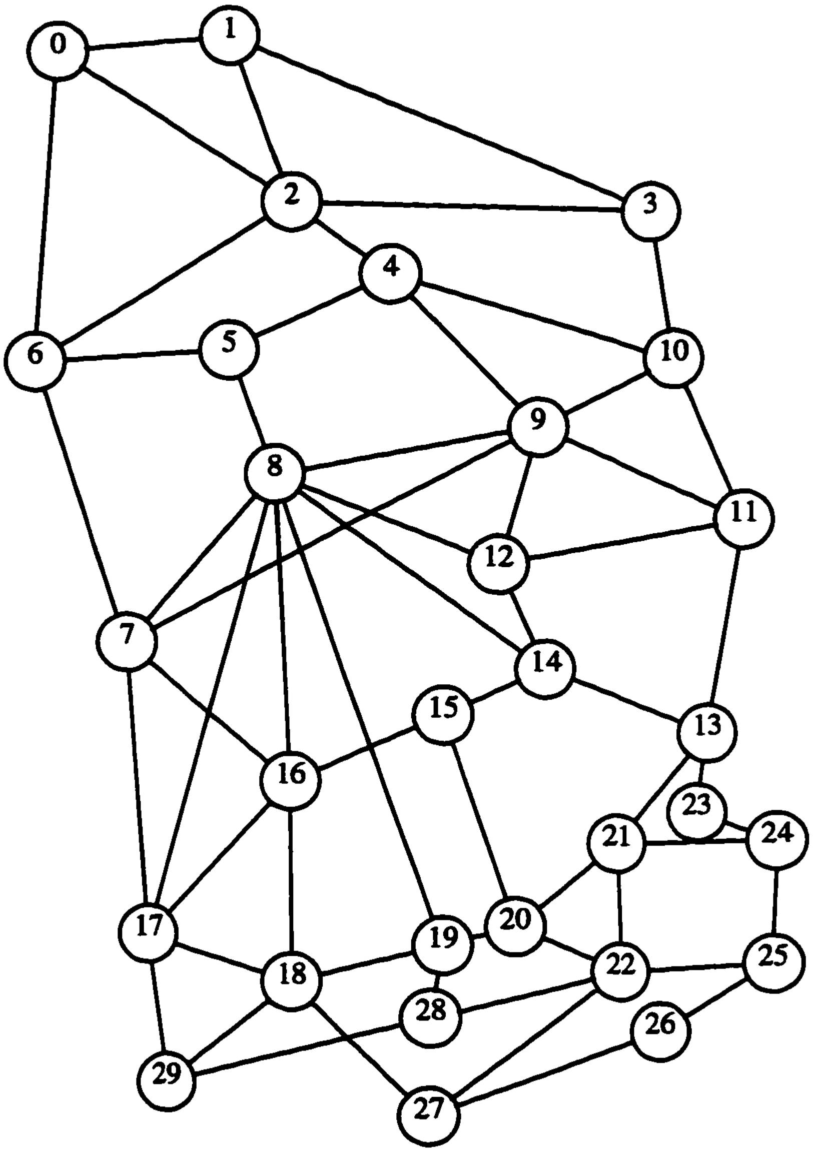


FIGURE 2.9 Test Network Net4

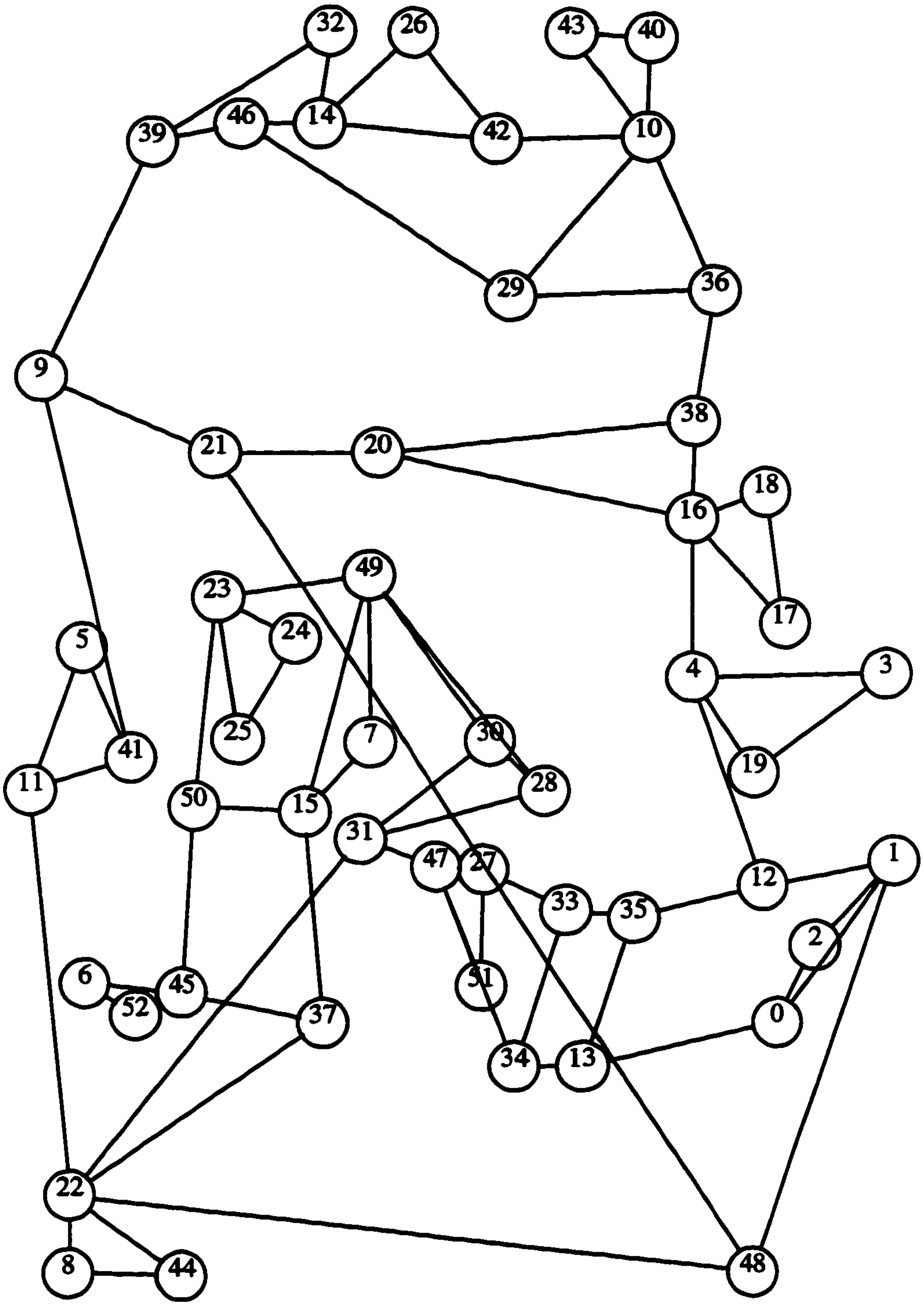


FIGURE 2.10 Test Network Net5

3. Studies on Preconfigured Spanning Trees

The tree is a class of pattern which has no limitation on its nodal degree. The current generation of DCS are not designed to directly form a pattern in which any node has a nodal degree greater than 2 (although some support broadcast unidirectionally.) Nonetheless, in this chapter a pair of heuristics, which generate preconfigured spanning trees in a network's existing spare capacity plan, are tested to see if the tree pattern can be of theoretical or conceptual benefit to preconfigured restoration.

3.1 Concepts

A tree is defined as a collection of fully connected nodes in which there is only a single path between any pair of nodes in the tree. A tree may not contain cycles and, as a result, the failure of any single span will disconnect the tree (a path will no longer exist between every pair of tree nodes.) A spanning tree is a tree which, when superimposed over a network, covers all connected network nodes.

In a network which has a weight associated with each network span, a maximum weight spanning tree is a tree, from the set of all possible spanning trees, for which the sum of the weights of the tree spans is at a maximum. That is, the sum of the weights belonging to such a tree can not be exceeded (although it may be equalled) by the sum of the span weights of any other spanning tree.

The heuristics, which are described in this chapter, generate preconfigured trees by iteratively building maximum weight spanning trees in a network's spare capacity. The span weights are determined by running a k-shortest path (KSP) algorithm for the failure of all network's spans and counting the number of times that each span contributes to the restoration of another span. An example of how the span weights are determined is given in Figure 3.1. Note, in the example, that although the network can offer a number of restoration paths, which exceeds the number required for full restoration by two of the spans, only the number of paths which are directly used in restoring the spans is counted in determining the span weights. Thus, the span weights are determined so they are representative of each span's actual contribution to the KSP restoration of other spans.

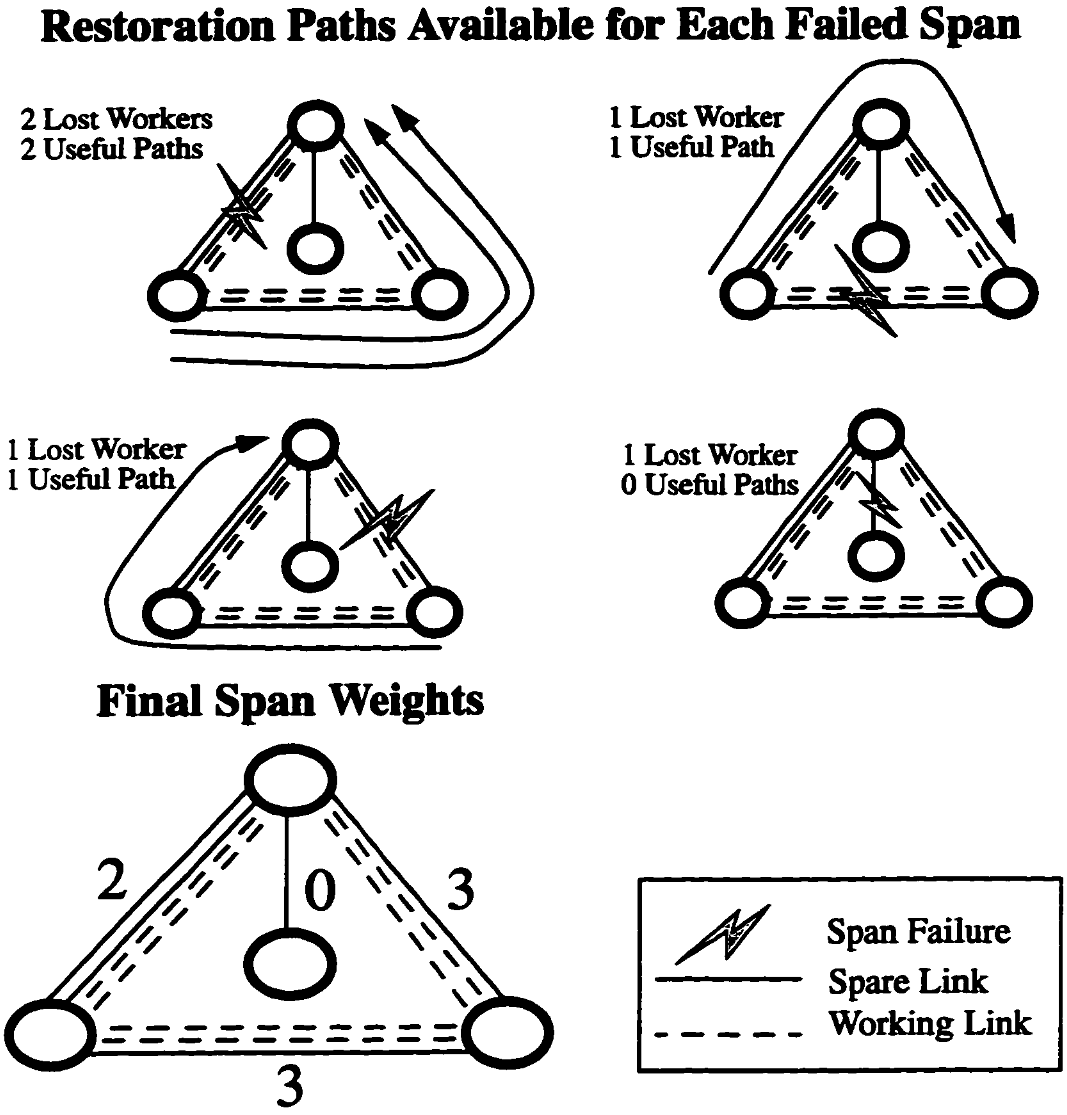


FIGURE 3.1 Determination of Span Weights for Use in the Tree Heuristics by Counting the Contribution of Each Span to the Working Link Restoration of the Failure of Other Spans

After the span weights are determined, preconfigured trees can then be designed in the network's spare capacity by the repeated application of Prim's spanning tree algorithm [12] to form a series of maximum weight spanning trees based on the span weights. At each application of the spanning tree algorithm, one or more copies of the determined maximum weight spanning tree will be constructed at a link level within the network's spare capacity. The algorithm terminates when it is no longer possible to find another maximum weight spanning tree in the network's spare capacity (a network's spare capacity is finite and, so, can only hold a finite number of trees.)

By building PC spanning trees using spans with a high contribution to the KSP restoration of other spans, the idea is that the likelihood of the tree design containing efficient preconfigured restoration paths should be increased.

Two versions of this basic method will be presented. The first version will form all spanning trees on the basis of a set of span weights which are determined only once at the start of execution. The second version will form spanning trees using a set of span weights which are determined at the start of each iteration. This should help track changes caused in the network's configuration by the addition of PC trees in previous iterations. The two versions of the maximum weight spanning tree heuristic are presented in the following sections.

3.2 Tree Design with Span Weights set Once

The first preconfigured tree heuristic constructs a tree PC design by calculating the span weights in the network once, at the start of execution, and then building successive preconfigured trees using only the original span weights. The span weights do not change as the network configuration is modified by the addition of PC trees.

3.2.1 Method

First, the heuristic runs a KSP algorithm to determine the restoration pathset which recovers, to as large a degree as possible, the working links lost for each potential failed span. While the KSP algorithm is run, the heuristic tracks the number of times each span contributes a spare link to the restoration of another span's failed working link. When the KSP restoration pathset has been evaluated, the total number of times that each span has contributed to the restoration of another span is set equal to the span's weight. This gives each span a weight which is proportional to its contribution to span restoration.

After the span weights have been determined, a maximum weight spanning tree is determined using Prim's algorithm. Next, the number of copies the spanning tree, to be

```
START buildPCtrees_with_SpanWeights_set_at_start_of_execution {
    findKSPrestorationPathsetsForAllSpans
    calculateSpanWeightsFromKSPpathsets
    for (i = 1 to NumberOfSpans) {
        unallocated_spares[i] = spans[i].spares
    }
    done = FALSE
    while (not done) {
        tree = prim (weights, unallocated_spares)
        if (tree exists) {
            treeSpans = tree.treeSpans
            numTree = INFINITY
            for (i = 1 to size of treeSpans) {
                if (unallocated_spares[treeSpans[i]] < numTree)
                    numTree = unallocated_spare[treeSpans[i]]
            }
            adjustUnallocatedSpares(tree, numTree, unallocated_spares)
            preconfigureTree (tree, numTree)
        } else {
            done = TRUE
        }
    }
}
```

FIGURE 3.2 Pseudo-code representation of Preconfigured Tree Heuristic tree1.

preconfigured in the network's spare capacity, is determined. The number taken is set equal to the number of available spare links on the tree span which has the least amount of available spare capacity. The reason that multiple copies of the tree are taken is that, in general, rerunning Prim's algorithm, after having taken only a single copy, would return exactly the same spanning tree as in the previous run because the span weights do not change between runs. With unchanging span weights, a different tree will only be generated if the span set, from which a tree can be formed, is changed. If the number of copies taken is equal to the number of spare links on the tree span with the least amount of available spare capacity then at least one network span will be depleted of all available spare capacity. This will reduce the set of valid spans from which a tree may be formed and force the next run of Prim's algorithm to generate a different tree. After the number of link-level instances of the tree to take is determined, the number of available spare links on each span, which is covered by the tree, is adjusted downward by the number of trees taken (each copy of the tree requires one spare link on each tree span.) Subsequent trees are built using repeated applications of the procedure described in this paragraph. The algorithm terminates when it is no longer possible to build a PC tree in the network's available spare capacity. Refer to Figure 3.2 for a pseudo-code representation of this heuristic which will be referred to as algorithm tree1.

3.3 Tree Design with Span Weights updated Iteratively

A second preconfigured tree heuristic, is similar to the tree heuristic discussed in the previous section, as it builds trees iteratively in a network's spare capacity. However, it differs in that it updates the restorability contribution weights of the spans at the start of each iteration and takes only one copy of the maximum weight spanning tree that is formed per iteration. Evaluating the span weights at the start of each iteration reflects changes in the network restoration plan which are caused by the addition of preconfigured trees in previous iterations. The restoration pathset is modified because the PC trees contain paths which can contribute towards the restoration of a failed span's working links. These PC paths modify the restoration pathset by reducing the number of working links which are directly vulnerable to a span failure which results in a reduction of the number of restoration paths which must be calculated using KSP restoration. Also, the PC paths occupy spare links which are no longer available for use in KSP restoration (tree links which are not used to form a PC path can be recycled for use in 2nd-step on-demand KSP restoration, if required) and this can result in the KSP restoration paths being distributed among different spans than in previous iterations. Since the span weights are calculated on the basis of the KSP restoration pathset, which is evaluated after taking all useful PC paths, the span weights will change as PC trees are added to the network. Updating the span weights, at the start of each iteration, should result in a more efficient preconfigured tree design because a maximum weight spanning tree will be formed on the basis of the current network configuration.

3.3.1 Method

At the start of each iteration, a 2nd-step on-demand KSP restoration pathset is evaluated for each potential network span failure after first taking advantage of all useful PC paths from the PC trees present from previous iterations. The network span weights are determined from this KSP pathset by counting the number of times each span contributes to the restoration of another span. The KSP pathset, for each potential failed span, is determined with consideration to the reduction in the amount of spare capacity which can be used to form 2nd-step KSP paths and to the reduction in the number of uncovered working network links (as more PC trees are added to the network, additional useful PC paths should be available for use by previously uncovered working links. These paths will reduce both the number of uncovered working links and the number of available spare links for use in 2nd-step KSP paths.) The 2nd-step KSP pathset will be modified by the reduction in the number of uncovered working links because fewer KSP restoration paths will need to be calculated. It will also be modified by the use of spare links in forming PC paths because this will reduce the number of spare links which each span can contribute to the KSP algorithm. This could modify the configuration of the 2nd-step KSP pathset, if it is forced to include on-demand restoration paths with different routes, than in previous iterations, due to the changes in available network sparing. The spare capacity which is available for use in the calculation of the KSP pathset is made up of the unallocated spare capacity and the pruned spare capacity present in the network. In summary, at the start of an iteration, the 2nd-step on-demand KSP restoration pathsets are evaluated for each potential span failure, using a PC plan formed of the generated PC trees from previous iterations, and it is from these pathsets that the span weights are calculated for use in the current iteration.

At the start of execution, the algorithm calculates the span weights as described in the previous paragraph. Once the span weights are calculated, Prim's algorithm is executed to find a maximum weight spanning tree and a single copy of this tree is preconfigured into the network's spare capacity. The network's unallocated spare capacity is decremented to account for the links used in forming the tree and the network's uncovered working links are also decremented to account for the preconfigured paths that exist within the tree. The adjusted network configuration is used to recalculate the network span weights and the next tree is formed on the basis of these updated weights. The algorithm continues, in this manner, until it is no longer possible to build further preconfigured trees in the unallocated network spare capacity, at which point it terminates. Refer to Figure 3.3 for a pseudocode representation of this PC tree design algorithm which will be referred to as tree2.

```
START buildPCtrees_with_SpanWeights_evaluated_at_start_of_each_iteration {
    for (i = 1 to NumberOfSpans) {
        unallocated_spares[i] = spans[i].spares
        uncovered_working[i] = spans[i].working
    }
    done = FALSE
    trees = ∅
    while (not done) {
        findLeftoverKSPrestorationPathsetsForAllSpans
            (unallocated_spares, uncovered_working, trees)
        calculateSpanWeightsFromKSPpathsets
        tree = prim (weights, unallocated_spares)
        if (tree exists) {
            treeSpans = tree.treeSpans
            numTree = 1
            trees = trees U tree
            adjustUnallocatedSpares(tree, numTree, unallocated_spares)
            adjustUncoveredWorking (tree, numTree, uncovered_working)
            preconfigureTree (tree, numTree)
        } else {
            done = TRUE
        }
    }
}
```

FIGURE 3.3 Pseudo-code representation of Preconfigured Tree Heuristic Tree2.

3.4 Results

3.4.1 Results for Tree1

Figure 3.4 shows an example of the type of pattern that preconfigured tree heuristic tree1 generates in test network Net1. Table 3.1 contains the network restorability results over all possible span failures for restoration using only KSP, restoration using only pure PC, and 2-step restoration. Table 3.2 contains the total count of crossconnection events, over all possible failed spans, for KSP restoration alone and for 2-step restoration. Figures 3.6 and 3.7 give counts of the number of crossconnection events required at each node to form a restoration pathset, for each span failure, in test network Net2 when using only on-demand KSP restoration and when using 2-step restoration. The crossconnect totals for each node appear sequentially within the space, on the x-axis, corresponding to each span failure.

3.4.2 Results for Tree2

Figure 3.5 shows an example of the type of pattern that algorithm tree2 generates in test network Net1. Table 3.3 contains the network restorability results over all possible span failures for restoration using only KSP, restoration using only pure PC, and 2-step restoration. Table 3.4 contains the total count of crossconnection events, over all possible failed spans, for KSP restoration alone and for 2-step restoration. Figures 3.8 and 3.9 give counts of the number of crossconnection events required at each node to form a restoration pathset, for each span failure, in test network Net2, when using only on-demand KSP restoration and when using 2-step restoration. The crossconnect totals for each node appear sequentially within the space, on the x-axis, corresponding to each span failure.

Table 3.1: Network Restorability Results using Standard KSP and PC Restoration with Tree Heuristic Tree1

Network	KSP Restorability (%)	PC Restorability (%)	2-step Restorability (%)
Net1	100	37.32	97.89
Net2	96.87	35.04	98.01
Net3	100	35.32	100
Net4	100	37.7	100
Net5	100	31.22	98.45

Table 3.2: Total Network Crossconnect Events Results using Standard KSP and PC Restoration with Tree Heuristic Tree1

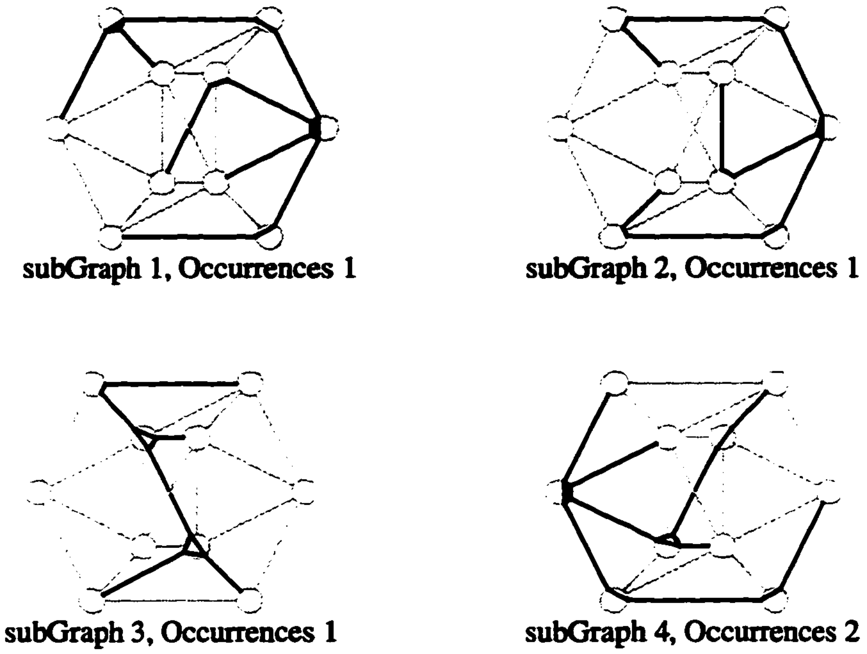
Network	Xpts Closed KSP	Xpts Closed 2-step	Xpts Opened 2-step	Total Xpt Events 2-step
Net1	310	169	583	752
Net2	3984	2644	6184	8828
Net3	13787	7879	22464	30343
Net4	84389	51084	175349	226433
Net5	8529	5726	16851	22577

Table 3.3: Network Restorability Results using Standard KSP and PC Restoration with the Tree Heuristic Tree2

Network	KSP Restorability (%)	PC Restorability (%)	2-step Restorability (%)
Net1	100	38.74	99.30
Net2	96.87	36.04	97.15
Net3	100	36.19	100
Net4	100	39.96	100
Net5	100	33.68	100

Table 3.4: Total Network Crossconnect Events Results using Standard KSP and PC Restoration with Tree Heuristic Tree2

Network	Xpts Closed KSP	Xpts Closed 2-step	Xpts Opened 2-step	Total Xpt Events 2-step
Net1	310	171	501	672
Net2	3984	2737	6205	8942
Net3	13787	9135	24353	33488
Net4	84389	55661	157214	212875
Net5	8529	6235	18462	24697



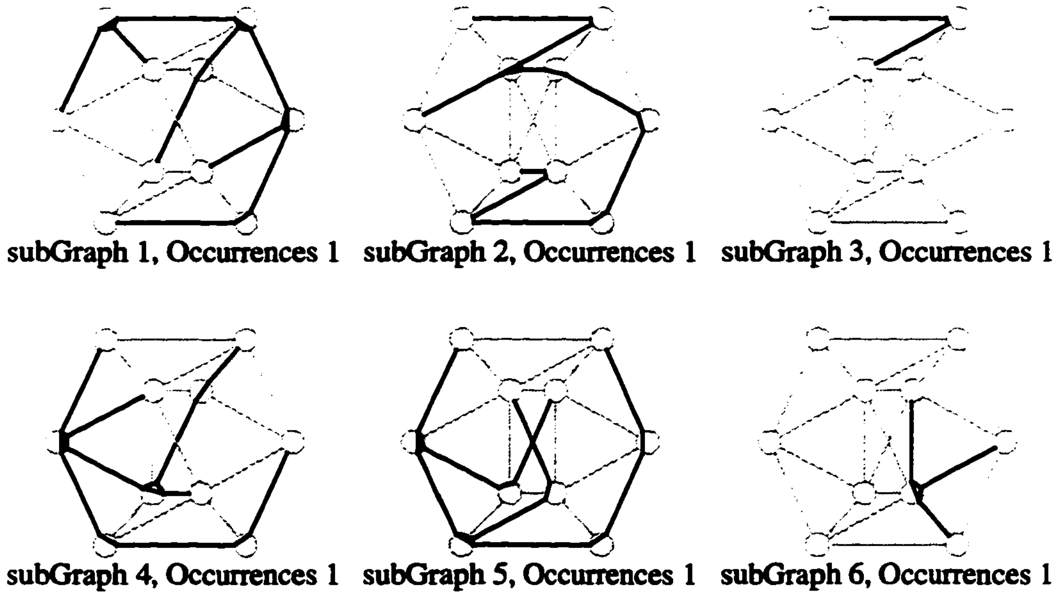
subGraph 1, Occurrences 1

subGraph 2, Occurrences 1

subGraph 3, Occurrences 1

subGraph 4, Occurrences 2

FIGURE 3.4 Example of the Patterns Generated using Tree HeuristicTree1 in Net1



subGraph 1, Occurrences 1

subGraph 2, Occurrences 1

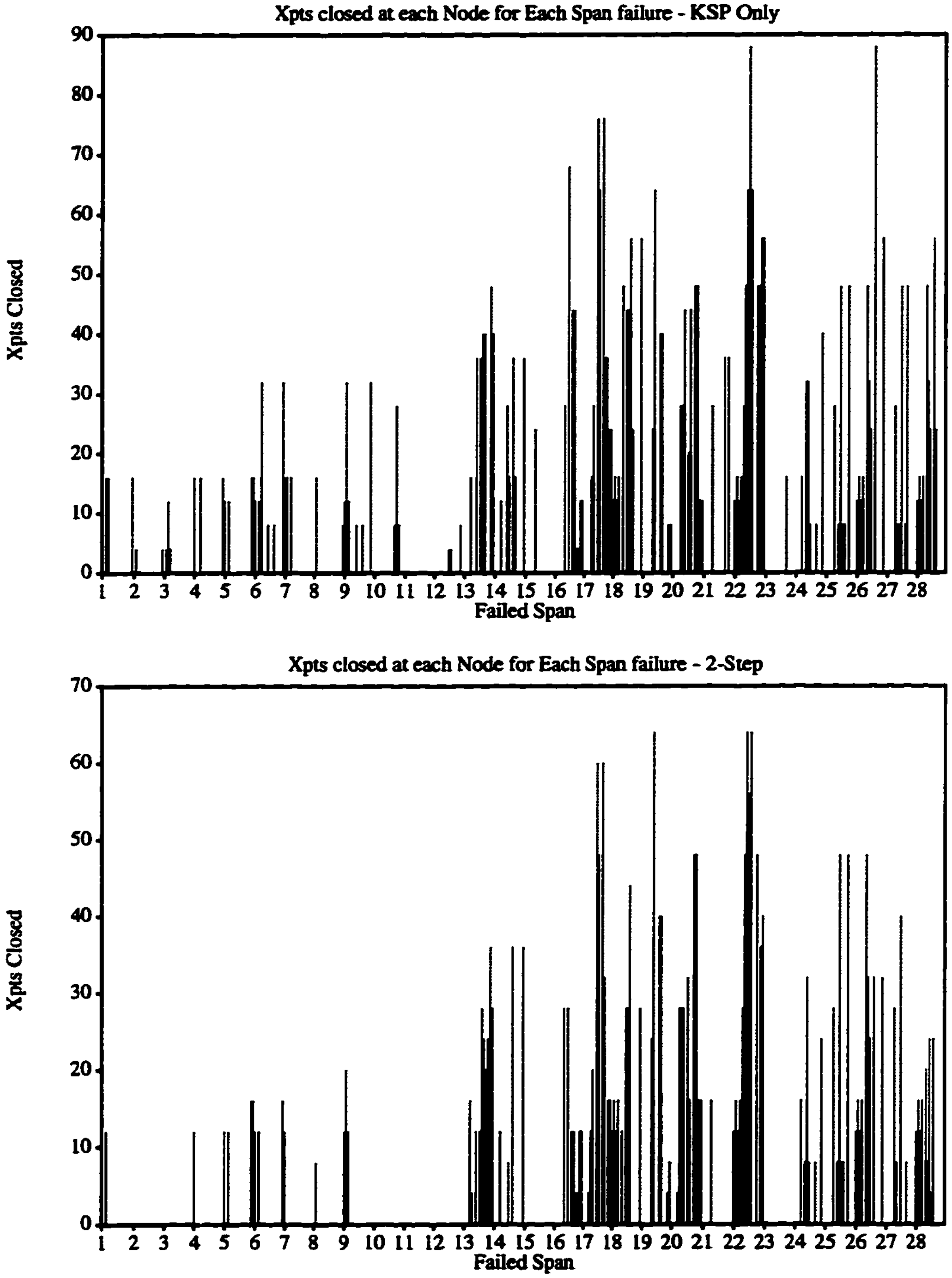
subGraph 3, Occurrences 1

subGraph 4, Occurrences 1

subGraph 5, Occurrences 1

subGraph 6, Occurrences 1

FIGURE 3.5 Example of the Patterns Generated using Tree Heuristic Tree2 in Net1


Xpts closed at each Node for Each Span failure - KSP Only

Xpts closed at each Node for Each Span failure - 2-Step

FIGURE 3.6 Crossconnections Closed at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method Tree1. For Restoration using only Conventional KSP Restoration and for 2-Step Restoration.

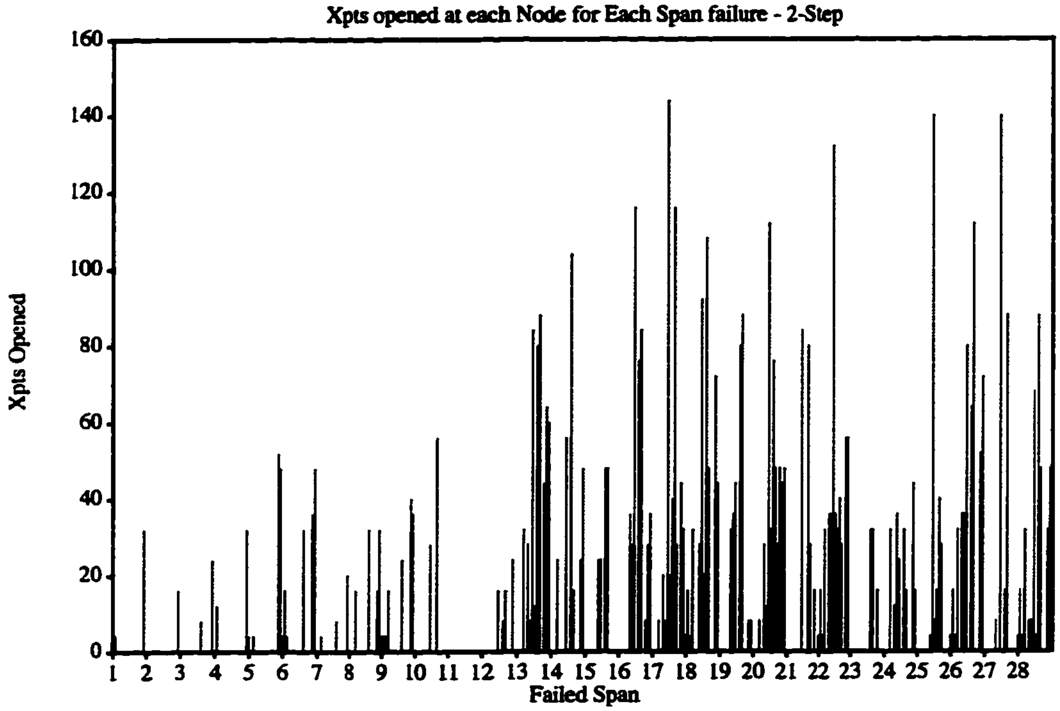
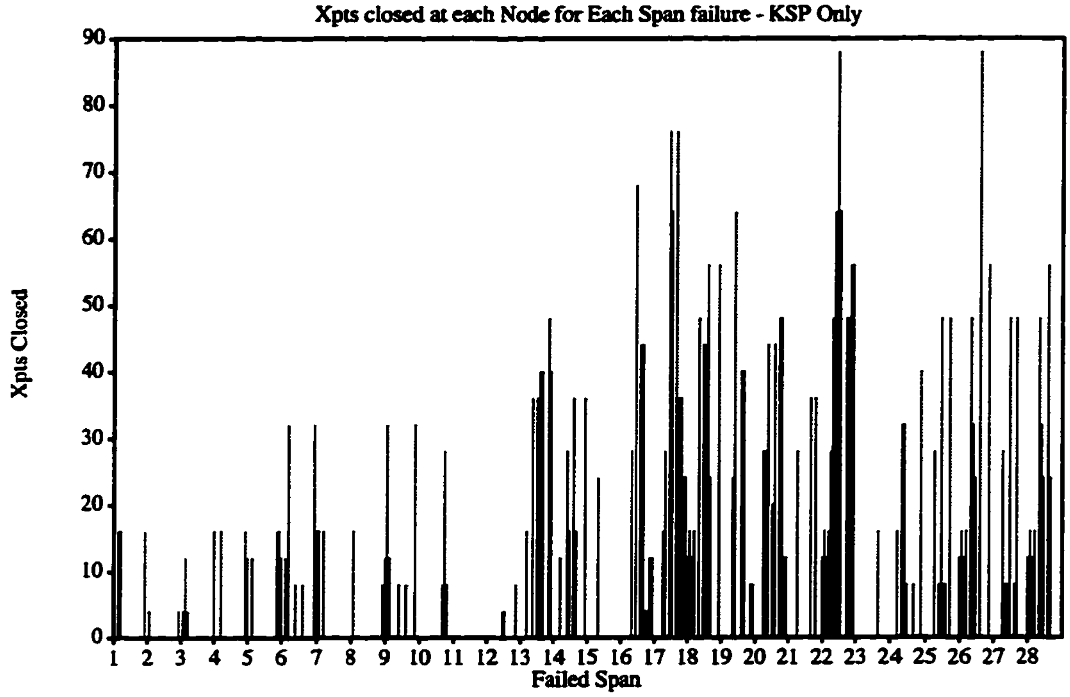


FIGURE 3.7 Crossconnections Opened at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method Tree1. For Restoration using 2-Step Restoration.

3.5 Conclusions

The tree pattern type appears to have a relatively poor performance when applied to preconfigured restoration. For both tree heuristics the restorability when using PC restoration without leftover KSP restoration was typically in the range of 30% to 40%. The PC plus leftover KSP restoration was quite good, however, typically being at or near fully 100% restorability which would indicate that the addition of preconfigured trees to a network's spare capacity will not hurt overall restorability to any large degree. The PC restorability was marginally better in the PC plan generated by tree2 compared to the PC plan generated by tree1. The difference was typically only 1%-2%, however.

The use of the PC tree also dramatically increased the total number of crossconnection events required to restore a span failure compared to just using KSP restoration. However, the number of crossconnections which were closed when using PC restoration was always



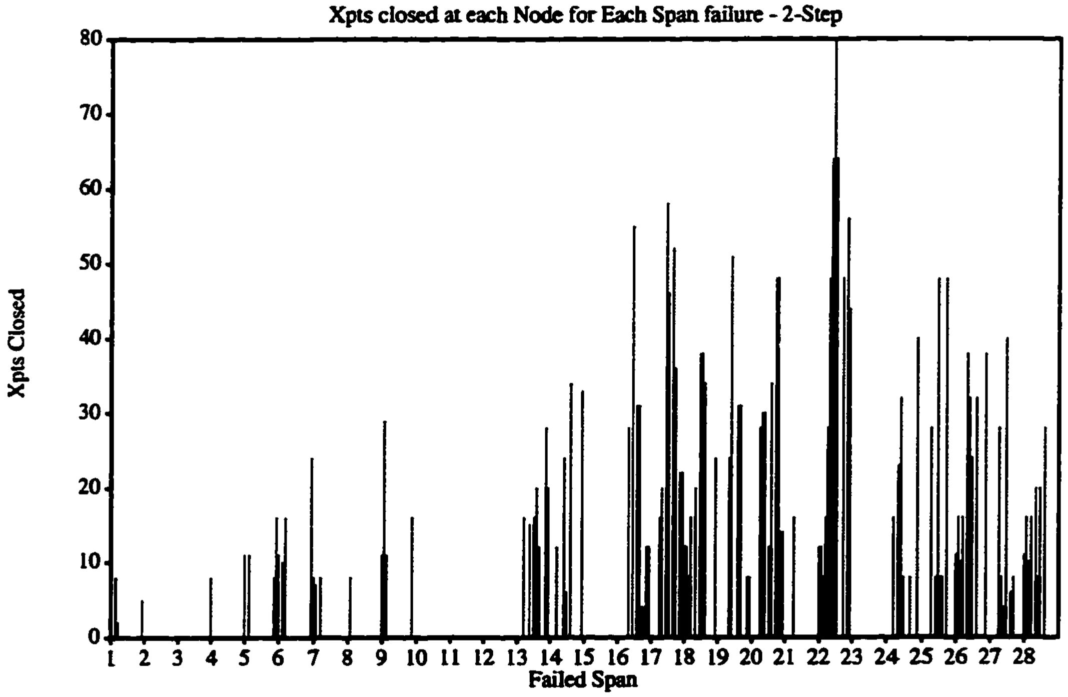


FIGURE 3.8 Crossconnections Closed at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method Tree2. For Restoration using only Conventional KSP Restoration and for 2-Step Restoration.

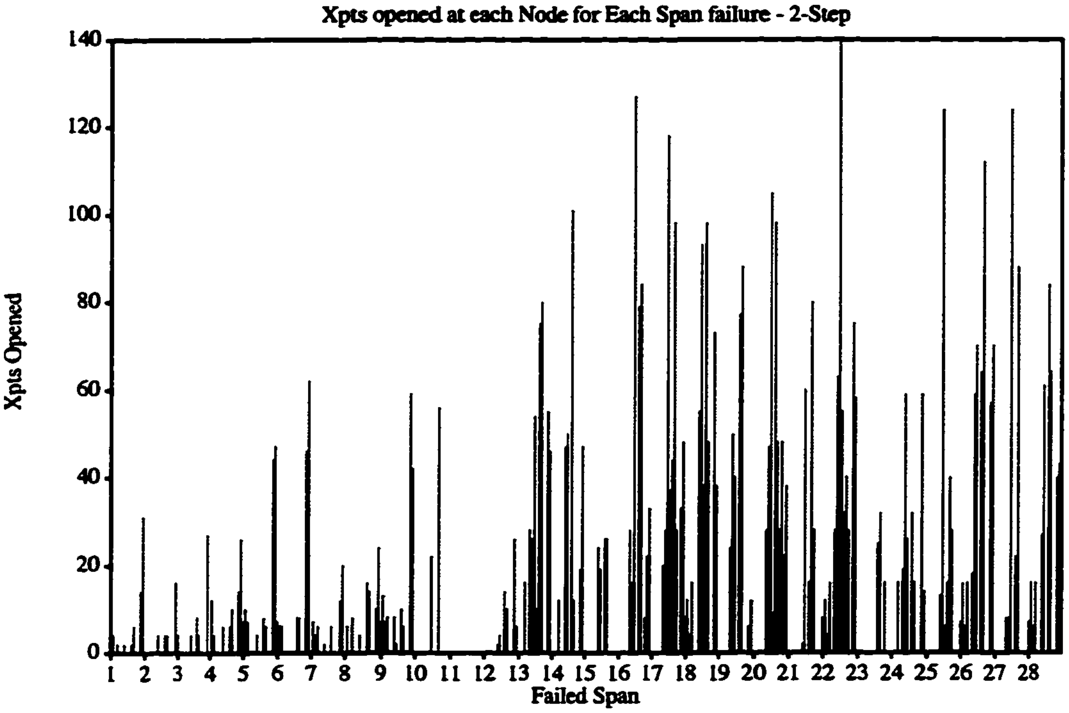


FIGURE 3.9 Crossconnections Opened at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method Tree2. For Restoration using 2-Step Restoration.

lower than the number of crossconnections which had to be closed when using KSP restoration. But PC restoration will generally require that crossconnections also be opened so that the PC branches are pruned off of the desired PC paths for restoration. The number of crossconnections which would have to be opened when using preconfigured trees is quite large compared to the number which have to be made and, when considering the total number of crossconnect events between PC and KSP restoration, PC restoration will require a considerable number of crossconnect events to implement if using directly preconfigured trees. This is because it is assumed that a preconfigured crossconnection exists between each pair of links, which meet at a node and belong to the same PC tree. The number of such preconfigured crossconnections which exist at a node for a particular PC link-level tree will be equal to $\binom{N}{2}$ or $\frac{N(N-1)}{2}$, where N is equal to the number of tree links which meet at the node. Therefore, to form a PC tree, the number of preconfigured crossconnects at a single node will increase with the square of the number of tree links which meet at the node. As PC trees have no constraint on their nodal degrees, the total number of preset crossconnections can be quite large.

However, as previously discussed, whether this is harmful depends on the method used to implement the ability to have multiple crossconnections meeting at a single port card in a DCS. If a method is used which uses a fast selection switch then having a large number of crosspoints preconfigured within the network may not be harmful to the overall restoration time. Also, there may exist an asymmetry between the time required to close a crossconnection and the time required to open one. If the time required to open a crossconnection is significantly less than the time required to close a crossconnection, then the overall restoration time may still be better when using this PC method than when using KSP restoration alone. This is because the number of crossconnections which must be made when using PC tree restoration is always less than the number of crossconnections which must be made when using KSP restoration alone.

4. Genetic Algorithms for Preconfiguration

In the previous chapter, the tree was considered as a preconfigured element. It was unconstrained in nodal degree but still belonged to a category of pattern with certain defining properties. In this chapter, a genetic algorithm (GA) is used to generate preconfigured patterns without constraint on the heterogeneous mixture of differing pattern classes. That is, the patterns contained within the design do not have to belong to a common pattern class (i.e. cycle or tree.) The patterns are generated on the basis of a fitness scoring formula which favours those patterns which are the most efficient in providing useful preconfigured restoration paths.

4.1 Concepts

Genetic algorithms are an optimization/search method modelled after the processes found in natural evolution [13,14]. For a genetic algorithm (GA) to attempt to optimize a problem it requires an encoding scheme which can represent a potential solution as a string formed from some symbol set and it requires a fitness function by which it can assign a score to a string (actually, to the solution the string represents) that evaluates how well it solves the problem. The string representing a particular solution can be viewed as the "genome" of that solution while its score is equal to its fitness.

The GA generates an initial "population" of solutions from randomly generated strings and evaluates the fitness of each "individual" in the population. This initial population is used to generate a new population, which is to form the next "generation", by combining pairs of strings from the initial population to generate new strings. Pairs of strings are chosen probabilisticly for this operation, with a probability directly proportionate to each string's fitness relative to the average fitness of the population, until the population of the next generation is filled (equal in size to the initial population.) This process of evaluation and reproduction is repeated to produce successive generations of populations. The GA can terminate either when an individual is found that meets a certain fitness level, after a certain number of generations have passed, and/or when the population converges (a population has converged when all the members of the population contain the same solution or have the same score and generating further generations would be of limited benefit.)

An explanation of why a GA is effective in finding a good solution to a problem is given by the schema theory [14]. According to this theory, a GA operates by evaluating the effectiveness of the schema which are present in the individuals of the population. A schemata is a particular pattern of genes which can appear in an individual's defining genome. For example, the bit sequences 100100 and 101101 both contain the schemata defined by 10x10x, where x is a wild card bit location. Each schemata represents a hyperplane of the solution search space. A schemata can be either beneficial or harmful to a pattern's overall fitness depending on what its effect is on the solution. The overall fitness of an individual is determined by the interaction of all of the schema present in its genome. If an individual contains a beneficial schemata, it will, on average, have a relatively high contribution to the formation of the next generation. On the other hand, if an individual contains a harmful schemata, it will, on average, have a relatively low contribution to the next generation. The net effect is that the representation the beneficial schema have in the population will increase as the number of generations increases while the representation of the harmful schema will decrease. This will result in an overall increase of both the average and highest individual score present in the population.

Crossover is the most common method used to combine two individuals to produce two new individuals from the genomes of the original pair. Crossover operates by cutting two strings in half at the same location and connecting the first half of the first cut string with the second half of the second cut string and vice versa to generate two new strings from the original two. Figure 4.1 shows an example of the crossover operator

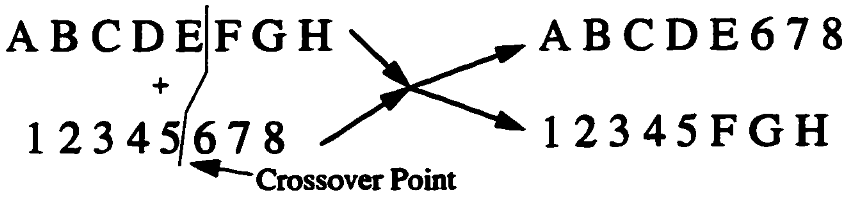


FIGURE 4.1 Example of the Genetic Algorithm Operation of Crossover

In Figure 4.1, the two strings on the left are mixed together using a crossover point between the fifth and sixth string location to produce two new strings (shown on the right.)

The example above is of a one point crossover operator. It is possible to have 2 or more crossover points. The location of the crossover point is, generally, chosen at random.

When the crossover operator acts on two fit individuals with beneficial schemata it can produce individuals with a higher concentration of good schemata. However, the crossover operator can also disrupt a schema if the crossover point should occur between one of the bits spanning the edges of the schema's defining bits. This would result in the different parts of the schema being separated along the crossover point. The crossover operator can also create new schema, that were not present in either of the original individuals, by combining fragments of bit sequences from the original sequences to form new schema. Thus, the crossover operator can recombine existing schema, disrupt existing schema and create schema which previously did not exist. Whether the results of the crossover operation are beneficial or not is determined by the GA's fitness function.

The mutation operator is used to perturb the search into parts of the search space which may not be represented by the individuals in the population. Mutation operates by flipping a gene in an individual's defining genome. The probability that a mutation will occur at a particular gene is set to some small value (typically 0.1% or less, per iteration or generation.) An individual modified by a mutation operation would thrive if it contained a new schema which provided it with benefit; otherwise, the selection process would be biased against it.

4.2 PC Design Using Genetic Algorithms at the Span level

The first GA representation for PC strategies tries to design good preconfigured patterns using individual spans as building blocks. A pattern is formed from the union of a number of spans. Each span contributes a single spare link to forming the pattern and all the links in a pattern which fall on a node are fully connected; that is, there is a crossconnection formed between each pair of the pattern links which meet at a node. The scoring function, which is used by the GA to assign a score to the pattern represented by each individual, is the ratio of the number of useful preconfigured paths contained in the individual's pattern to the number of spare links which would be required to construct the individual's pattern. This scoring function gives an advantage to individuals which represent a pattern that can provide a large number of usefully preconfigured paths relative to the number of spare links required to construct the pattern. This encourages the generation of preconfigured patterns that make efficient use of the network's finite spare capacity. This GA based design algorithm will be referred to as GA1.

The GA continues execution until the population converges (converges in the sense that all members of the population have the same score), at which point a preconfigured pattern with a good score should result. By scoring on the basis of preconfigured paths per pattern link, a pressure for efficient preconfiguration of network spare capacity is created.

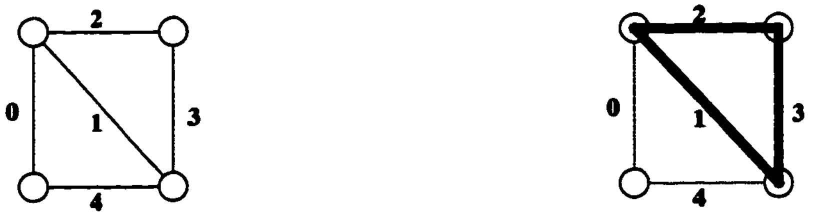
4.2.1 Representation of a Pattern at the Span Level

A preconfigured pattern can be represented with a bit string of length N_{BS} in a network with S spans. N_{BS} is given by the following expression:

$$N_{BS} = \sum_{i=1}^{S} \delta_i \qquad \text{(EQ 4.1)}$$

Where, δ_i is one if span i has at least one spare link which is not being used (to form a preconfigured pattern, or restore a span failure) or zero if span i has no free spare links. N_{BS} is equal to the number of spans which are able to contribute at least one free spare link to form a preconfigured pattern.

A specific pattern can be represented with a particular sequence of ones and zeros in a bit string of length N_{BS} If a position in the bit string is set equal to one then the pattern contains a link on the span which corresponds to that bit position. If another position is set equal to zero then the pattern does not contain a link on the span corresponding to that bit position. An example, of the span level representation of a pattern using a bit string, follows in Figure 4.2.



A Network with 5 spans.

Pattern from Binary Digit:
01110
01234

FIGURE 4.2 Example of How a Preconfigured Pattern can be Represented at the Span Level, for use in GA1

In the example in Figure 4.2, there is a simple network containing 4 nodes and 5 spans. A triangular pattern is represented in the network by the binary string 01110, which has a zero for the two network spans which are not in the pattern, and a one for the three spans which are. Note that in this example all of the network spans are assumed to have at least one available spare link to contribute to the construction of a preconfigured pattern. If, for example, span 4 were completely depleted of available spare links it would not be represented with a bit in the pattern's binary string. In this case, the same pattern could be represented with the binary string 0111, which is only 4 bits long as opposed to 5 bits in the original example.

It should be noted that there is nothing within this representation that does not allow a single binary string to represent a pattern which contains several disconnected elements. The binary string only determines what spans contribute a link to the pattern and it could be that the pattern links join together to form two or more disconnected subpatterns. However, there is no advantage in attempting to encourage the creation of bit strings which represent fully connected pattern; if a disconnected pattern has a high fitness score, it can still effectively provide PC paths even if its subpatterns are disconnected.

4.2.2 Fitness Score

A GA requires a fitness function to assign a score to the solutions contained in the population pool. A good preconfigured pattern will tend to contain a large number of useful preconfigured paths compared to the number of spare links required to form the pattern. The score used to evaluate the fitness of an individual i is given by:

$$F_{S_i} = \frac{P_{PC_i}}{SL_i} \qquad \text{(EQ 4.2)}$$

Where F_{S_i} is the fitness of individual i, P_{PC_i} is the number of useful preconfigured paths contained in the pattern represented by individual i, and SL_i is the number of spare links used in the construction of preconfigured individual i. This fitness score will favour those individuals that represent patterns which make efficient use of network spare capacity to deliver preconfigured paths, for the given working demand flows and network topology.

4.3 Design Using Genetic Algorithms at the Crossconnection level

The previous method (GA1) used a representation of a pattern at the span level to design preconfigured patterns in the network's spare capacity. The span level method has the advantage of needing only a relatively small binary string size to represent a pattern (the string can, at most, be of a length equal to the number of spans in the network.) Having a short defining string translates into a relatively small solution search space in which good patterns must be searched for. However, the previous representation assumes that all pattern links, where they meet at a node, are fully connected; i.e.) that is there is a crossconnection between each pair of pattern links meeting at a node. As the number of pattern crossconnections at a node varies with the square of the number of pattern links meeting at the node, the number of crossconnections can increase rapidly as the degrees increases at the vertices of the patterns.

Representing a pattern at the crossconnect level could reduce the total number of crossconnections preset in a preconfiguration plan while maintaining a high level of preconfigured restorability. Reducing the total number of preset crossconnects can translate into faster restoration times by reducing the number of preset crossconnections which have to be broken to utilize preconfigured paths in span restoration or, at least, permit the PC plan to be formed using fewer and smaller 1:N selection switches within the DCS. This GA based design algorithm will be referred to as GA2.

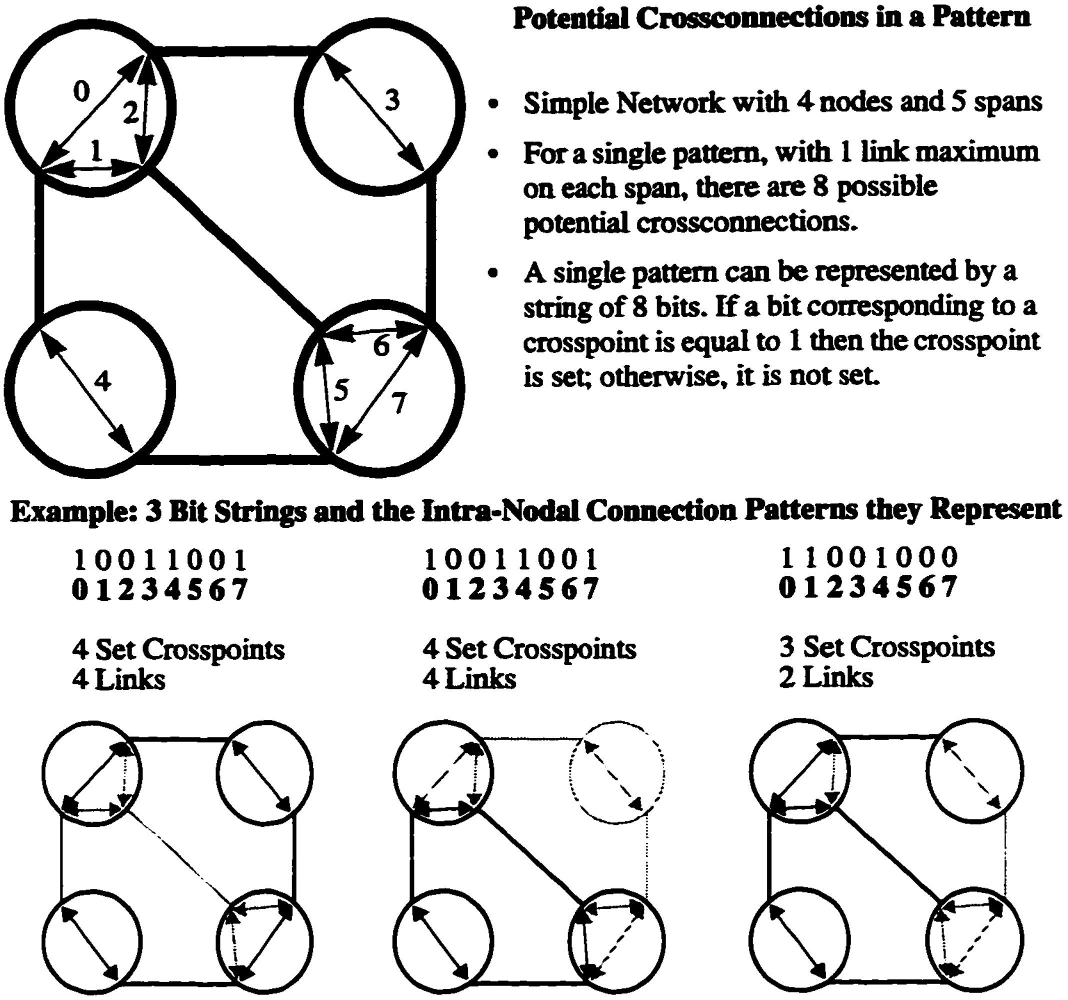
4.3.1 Representation of a Pattern at the Crossconnection Level

A pattern can be represented at the crossconnect level using a binary string with a length equal to the number of possible crossconnects which could exist in a pattern. In the pattern, the crossconnection would be set if its corresponding bit position in the defining binary string was equal to one; otherwise, it would not be set.

The length of the binary string would be equal to N_{BS} which is given by:

$$N_{BS} = \sum_{i=1}^{N} \binom{dp_i}{2} = \sum_{i=1}^{N} \frac{dp_i(dp_i - 1)}{2} \qquad \text{(EQ 4.3)}$$

Where, N is the number of network nodes, and dp_i is the maximum degree that a pattern can have on node i (dp_i is equal to the number of spans at node i which have at least one spare link available for contribution to forming a pattern). It is assumed in equation 4.3 that each pattern can use, at most, a single spare link on each span. Refer to Figure 4.3 for an example of how a pattern can be represented at the crossconnect level. In the example there is a simple network with 4 nodes and 5 spans. A pattern is permitted to utilize, at most, one spare link on each span which is able to provide a spare link. In this example there are 8 possible crossconnections which could be set in a pattern. A pattern can be represented using a string with 8 bits where each bit represents whether a particular crossconnection is set within the pattern or not (our convention is, if a bit is set to 1 then the crossconnect represented by that bit is set; otherwise, it is not set.) This representation indirectly determines whether a pattern has a spare link on a particular span because a spare link on a span is only present in the pattern if there is a crossconnection connecting it to another spare link.



Potential Crossconnections in a Pattern

- Simple Network with 4 nodes and 5 spans
- For a single pattern, with 1 link maximum on each span, there are 8 possible potential crossconnections.
- A single pattern can be represented by a string of 8 bits. If a bit corresponding to a crosspoint is equal to 1 then the crosspoint is set; otherwise, it is not set.

Example: 3 Bit Strings and the Intra-Nodal Connection Patterns they Represent

10011001 **01234567**	10011001 **01234567**	11001000 **01234567**
4 Set Crosspoints 4 Links	4 Set Crosspoints 4 Links	3 Set Crosspoints 2 Links

FIGURE 4.3 Example of How a Preconfigured Pattern can be Represented at the Crossconnection Level, for use in GA2

4.3.2 Fitness Score

As in the previous GA based preconfigured design method, a fitness score is required as a basis for evaluating solutions. However, if the patterns were evaluated as in the previous section there would not be a strong drive to reduce the number of crossconnections set in the preconfigured patterns because a pattern's fitness was set equal to the ratio of the number of useful PC paths it could provide to the number of spare links required in its construction. The function is only evaluated at a link level, not at a crossconnect level, so if a pattern contained one or more PC crossconnections which do not contribute to the formation of a useful PC path, the pattern's fitness score would not reflect this. For this method, a good preconfigured pattern would tend to contain a large number of preconfigured paths compared to the number of crossconnections which would be required to form the pattern. The score used to evaluate the fitness of an individual i is given by:

$$F_{X_i} = \frac{P_{PC_i}}{XPT_i} \qquad \text{(EQ 4.4)}$$

Where F_{S_i} is the fitness of individual i, P_{PC_i} is the number of useful preconfigured paths contained in the pattern represented by individual i, and XPT_i is the number of crossconnections used in the construction of the pattern represented by individual i. Such a fitness score favours those individuals which deliver a large number of preconfigured paths per crossconnection required to form the pattern.

4.4 Method

The basic method is the same for both GA1 and GA2 with only the fitness function and representation method differing. These algorithms builds up patterns in a network's spare capacity iteratively. The GA is run repeatedly to generate a preconfigured pattern with the genome length and fitness function adjusted according to the network configuration at the start of the iteration. A number of the patterns are then taken, and the network's configuration is adjusted to include the patterns.

The final pattern which is generated by the GA will depend on two factors: unallocated spare links available within the network and uncovered working links present in the network. The network's unallocated spare links will represent a hard limit to the form the pattern generated by the GA may take, as the pattern may only use a link on spans which have at least a single available unallocated spare link. The network's uncovered working links will represent a soft limit to the form the pattern may take as the pattern will only be rewarded, in its fitness score, for the useful PC paths that it contains. If a pattern overprovides a network span, whose working links are covered with PC paths supplied by previously generated PC patterns, with PC paths it will not be punished but it will not be rewarded either. Consideration, of the limits imposed on the pattern generation by the current network configuration of unallocated spare links and uncovered working links, suggest that it would be reasonable to place multiple copies of the pattern generated by the GA, at each iteration. This is because, depending on the state of the network's PC plan at the current iteration, taking only a single copy of the pattern will not change the distribution of the spans requiring PC paths or of the spans able to contribute unallocated spare links to the construction of the pattern. Re-running the GA, after only taking a single copy, could result in an individual which is similar to the previously generated individual because fitness would be evaluated based on a similar distribution of uncovered workers and the configuration of the pattern would be limited by a similar distribution of unallocated spares. Therefore, the number of the pattern which are taken is set equal to the lower of the number required to change the distribution of the uncovered workers and the number required to change the distribution of the unallocated spares.

The limiting pattern number due to the network's unallocated spares is equal to the number of copies of the pattern which result in the depletion of all unallocated spares on one or more network spans. The limiting pattern number due to the network's unallocated workers is equal to one less than the number of copies of the pattern which cause one or more network spans, with a non-zero number of uncovered working links, to be overprovided with PC paths. Spans with no uncovered working links are not considered in evaluating this limit because these spans no longer influence the fitness score used by the GA to evaluate its individuals (only useful PC paths are considered when evaluating an individual's fitness score.) The number of copies of an individual is equal to the lower of these two limits. However, at least a single copy of each individual will be taken at each iteration even if the uncovered working link limit should be equal to zero (this may happen if taking even a single copy of a pattern will cause a network span to be overprovided with PC paths.) Taking multiple copies of each pattern, if possible, reduces the number of times that the GA must be run and speeds up execution time.

After the network configuration is modified to accommodate the pattern, the GA is rerun and a new pattern is generated. The algorithm continues in this manner until it is no longer possible to preconfigure a pattern in the network spare capacity or the pattern produced is not useful to preconfigured restoration. Refer to Figure 4.4 for a pseudocode representation of this algorithm.

```
START building GA PC patterns at the span level{
    for (i = 1 to NumberOfSpans) {
        unallocated_spares[i] = spans[i].spares
        uncovered_working[i] = spans[i].working
    }
    done = FALSE
    while (not done) {
        goodPattern = runGA_untilConvergence
            (unallocated_spares, uncovered_working)
        numPattern = INFINITY
        if (goodPattern exists) {
            numPattern =
                numberPatternToTake
                    (goodPattern, unallocated_spares, uncovered_working)
            adjustAvailableSpares
                (goodPattern, numPattern, unallocated_spares)
            adjustUncoveredWorking
                (goodPattern, numPattern, uncovered_working)
            preconfigurePattern (goodPattern, numPattern)
        } else {
            done = TRUE
        }
    }
}
```

FIGURE 4.4 Pseudo-code representation of the Use of a Genetic Algorithm to iteratively builds Preconfigured Patterns.

4.5 Results

The GA PC design heuristics were programmed and executed using a publicly available library of GA procedures, written in C code, called LibGA* [15].

Each PC iteratively generated PC pattern was generated using a generational GA. The population pool size, of each execution of the GA, was set to the 4 times the length of the solution's representative bit string as determined by Equation 4.1 or 4.3. The initial population, of the GA, was formed of randomly generated bit strings. The mutation rate was set equal to 0.1% and a mutation of a bit would simply invert it. Pairs of individuals were selected for the crossover operator using a roulette wheel method of selection. Roulette wheel selection operates by giving each individual in a population a slot which is proportional in size to the individual's fitness relative to the average fitness of the population. The wheel is then "spun" and pairs of individuals are selected for crossover until the next generation's population pool is filled. Crossover was performed using a simple one-point swap of a pair of individual's defining strings. In addition, for each crossover operation, there was a 10% chance that the individuals would not be crossed over but would be directly copied into the next generation's pool without modification. Finally, each GA was operated using elitism; that is, when the next generation is formed, a pair of copies of the best individual found, so far, is automatically placed into the next generation.

4.5.1 Genetic Algorithm Design of PC Patterns represented at the Span Level

Figure 4.5 gives an example of the type of patterns that GA1 generates in test network Net1. Table 4.1 contains the network restorability results over all possible span failures for restoration using only KSP, restoration using pure PC, and 2-step restoration. Table 4.2 contains the total crossconnection events, over all possible failed spans, for KSP restoration alone and for 2-step restoration. Figures 4.7 and 4.8 give counts of the number of crossconnection events required at each node to form a restoration pathset, for each span failure, in test network Net2, when using only on-demand KSP restoration and when using

*- available by anonymous ftp, over the internet, at: ftp.aic.nrl.navy.mil/pub/galist/src/ga/libga100.tar.Z

2-step restoration. The crossconnect totals for each node appear sequentially within the space, on the x-axis, corresponding to each span failure.

4.5.2 Genetic Algorithm Design of PC Patterns represented at the Crossconnect Level

Figure 4.6 gives an example of the type of pattern that GA2 generates in test network Net1. Table 4.3 contains the network restorability results over all possible span failures for restoration using only KSP, restoration using only pure PC, and 2-step restoration. Table 4.4 contains the total crossconnection events, over all possible failed spans, for KSP restoration alone and for 2-step restoration. Figures 4.9 and 4.10 give counts of the number of crossconnection events required at each node to form a restoration pathset, for each span failure, in test network Net2, when using only on-demand KSP restoration and when using 2-step restoration. The crossconnect totals for each node appear sequentially within the space, on the x-axis, corresponding to each span failure.

Table 4.1: Network Restorability Results using Standard KSP Restoration and PC Restoration with GA1

Network	KSP Restorability (%)	PC Restorability (%)	2-step Restorability (%)
Netl	100	74.64	93.66
Net2	96.87	82.48	95.73
Net3	100	81.71	98.37
Net4	100	85.88	99.58
Net5	100	88.32	100

Table 4.2: Total Network Crossconnect Events Results using Standard KSP Restoration and PC Restoration with GA1

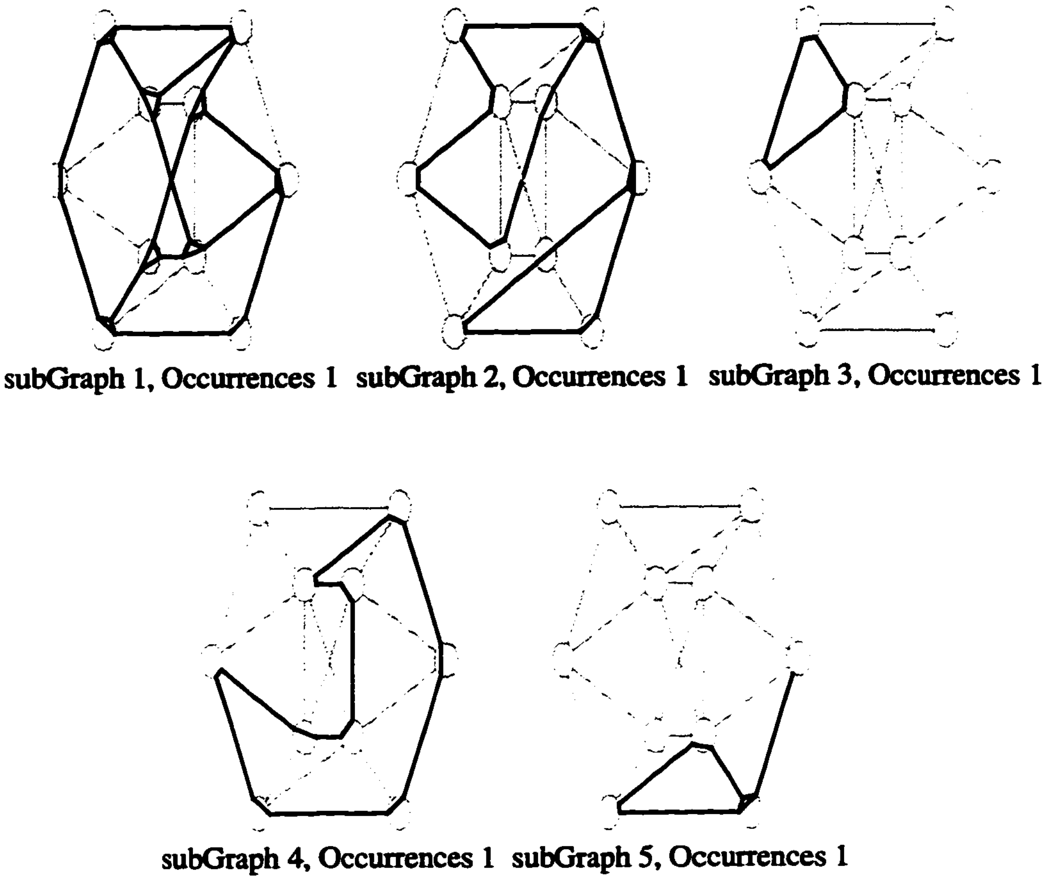
Network	Xpts Closed KSP	Xpts Closed PC	Xpts Opened PC	Total Xpt Events PC
Netl	310	75	585	660
Net2	3984	700	5722	6422
Net3	13787	2916	20044	22960
Net4	84389	12862	196553	209415
Net5	8529	1427	18157	19584

Table 4.3: Network Restorability Results using Standard KSP Restoration and PC Restoration with GA2

Network	KSP Restorability (%)	PC Restorability (%)	2-step Restorability (%)
Net1	100	73.94	90.85
Net2	96.87	76.64	99.00
Net3	100	78.19	99.43
Net4	100	82.78	97.37
Net5	100	89.87	99.82

Table 4.4: Total Network Crossconnect Events using Standard KSP Restoration and PC Restoration with GA2

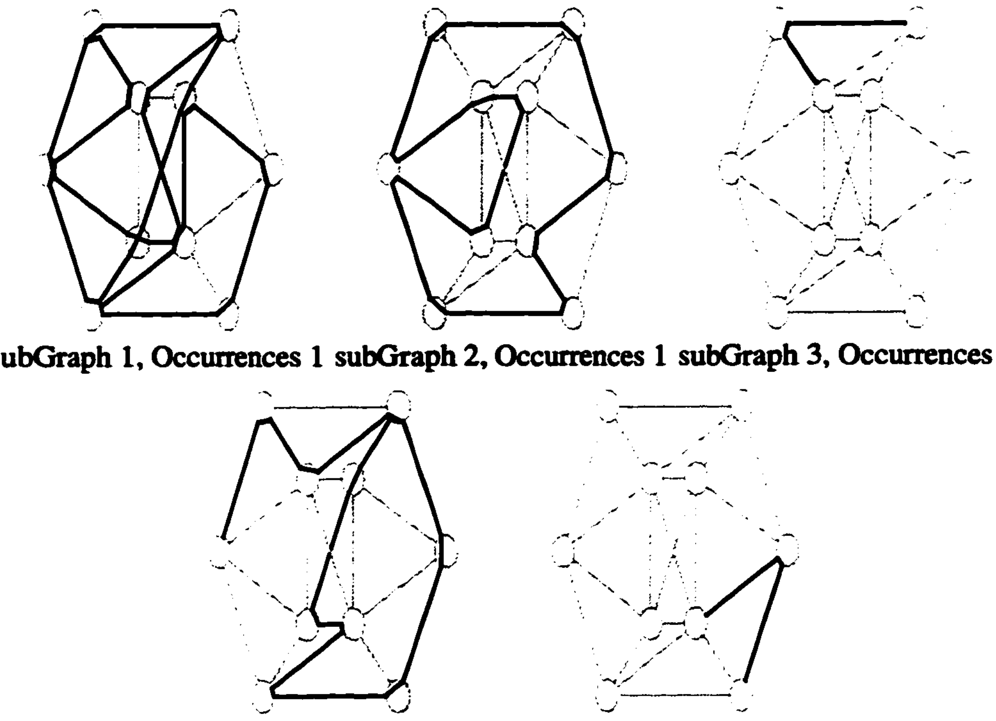
Network	Xpts Closed KSP	Xpts Closed PC	Xpts Opened PC	Total Xpt Events PC
Net1	310	46	253	299
Net2	3984	911	3645	4556
Net3	13787	3133	11886	15019
Net4	84389	14049	103963	118012
Net5	8529	947	8967	9614



subGraph 1, Occurrences 1 subGraph 2, Occurrences 1 subGraph 3, Occurrences 1

subGraph 4, Occurrences 1 subGraph 5, Occurrences 1

FIGURE 4.5 Example of the Patterns Generated using GA1 with Net1



subGraph 1, Occurrences 1 subGraph 2, Occurrences 1 subGraph 3, Occurrences 1

subGraph 4, Occurrences 1 subGraph 5, Occurrences 1

FIGURE 4.6 Example of the Patterns Generated using GA2 with Net1

4.6 Conclusions

The preconfigured restoration offered by the PC plans designed by the genetic algorithms tested in this chapter was quite effective compared to the performance of the preconfigured tree algorithms. The designs generated by both genetic algorithms were restorable, using only preconfigured paths, in the 70% to 90% range. Overall restorability using PC restoration plus leftover KSP did seem to be hurt somewhat, by the use of genetic algorithm designed PC plans, with overall restorability being mainly in the range of 95% to 100% but in one case dropping to 90%. The reason that the overall restorability of a network may be hurt when using PC restoration is that the useful PC paths which are applied to the restoration of a failure are constrained by the configuration of the PC pattern from which they are pruned. This can results in PC paths utilizing more spare capacity,

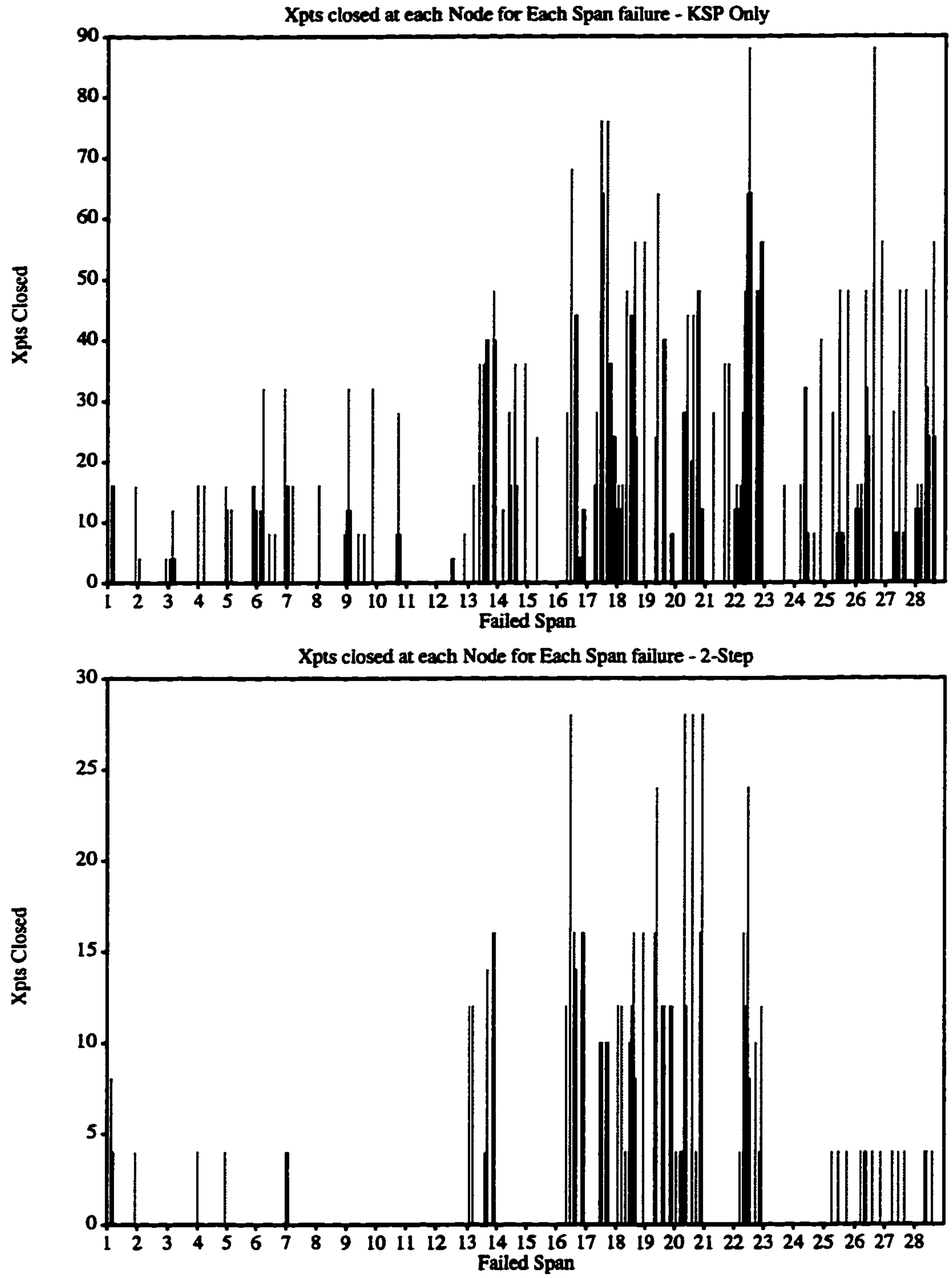


FIGURE 4.7 Crossconnections Closed at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method GA1. For Restoration using only Conventional KSP Restoration and for 2-Step Restoration.

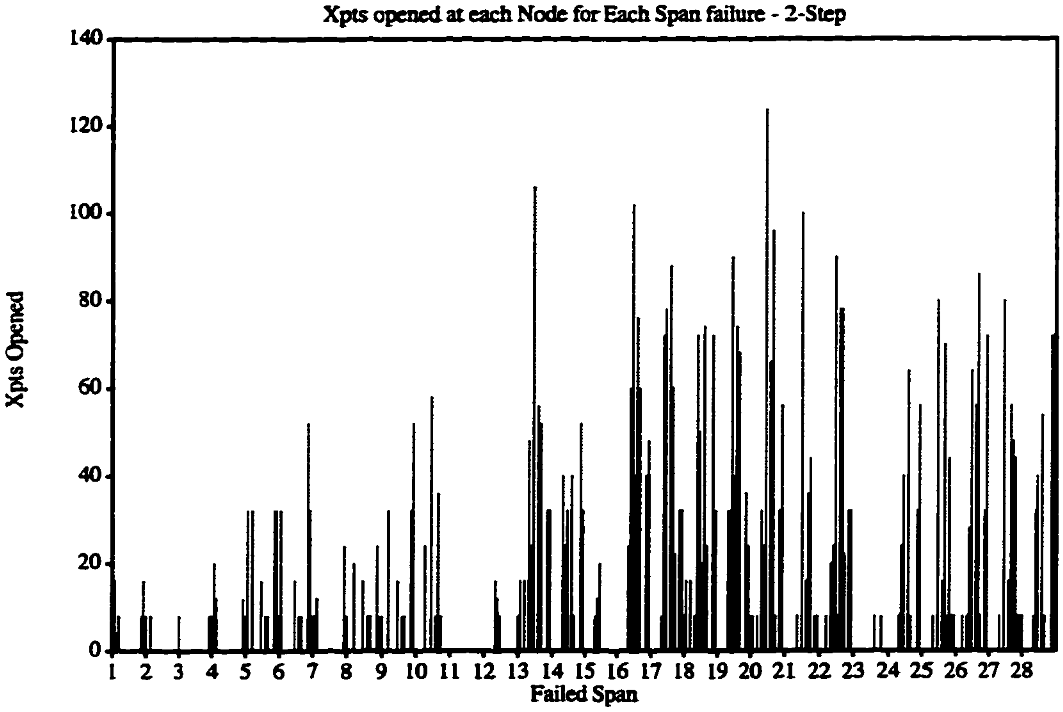
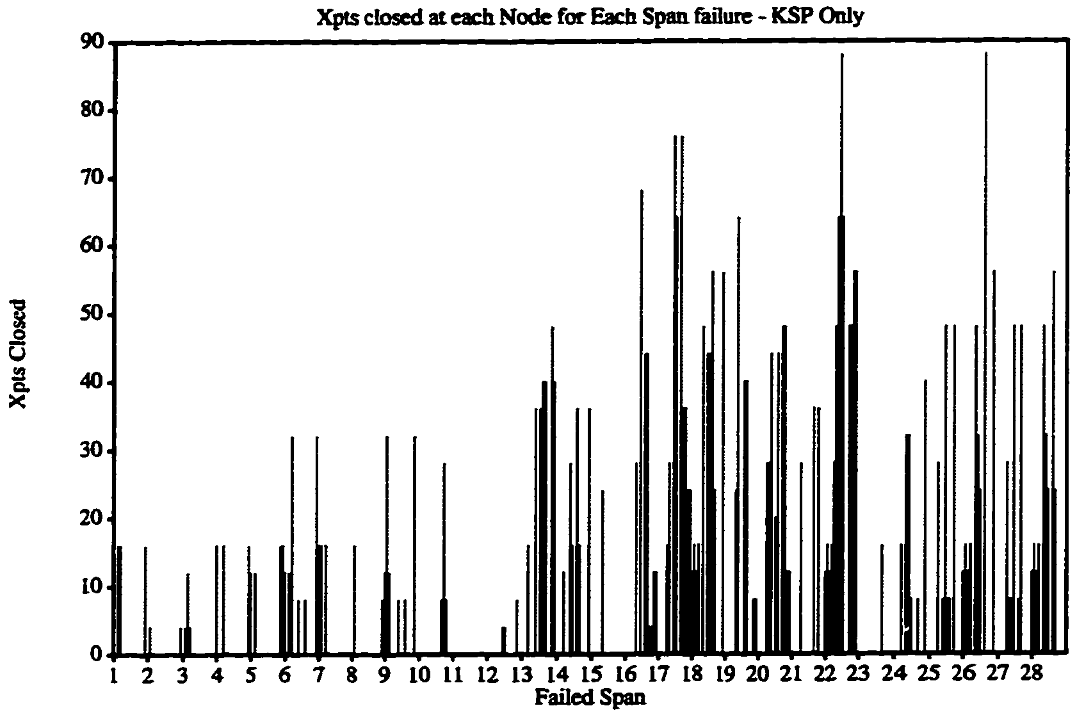


FIGURE 4.8 Crossconnections Opened at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method GA1. For Restoration using 2-Step Restoration.

than the equivalent restoration when using only on-demand KSP paths, and can hurt the network's overall restorability.

The genetic algorithm which represented patterns at the span level of representation (GA1) generated high levels of crossconnection events when restoring span failures. The number of crossconnections which are set using PC restoration are quite low compared to the number set when using only KSP restoration. However, the number of crossconnections are opened is very high which contributes to a high overall number of crossconnect events. The patterns which are generated are not limited in their nodal degrees and, so, a large number of preconfigured crossconnections can be required to form the patterns as with the PC tree pattern type. The span level representation used a scoring function which rewarded patterns which offered a large number of preconfigured path in relation to the spare capacity required to form the pattern. This score does encourage the efficient use of a network's spare capacity but does not place any limit on the number of preconfigured crossconnections. However, as with the PC tree, the implementation method which per-



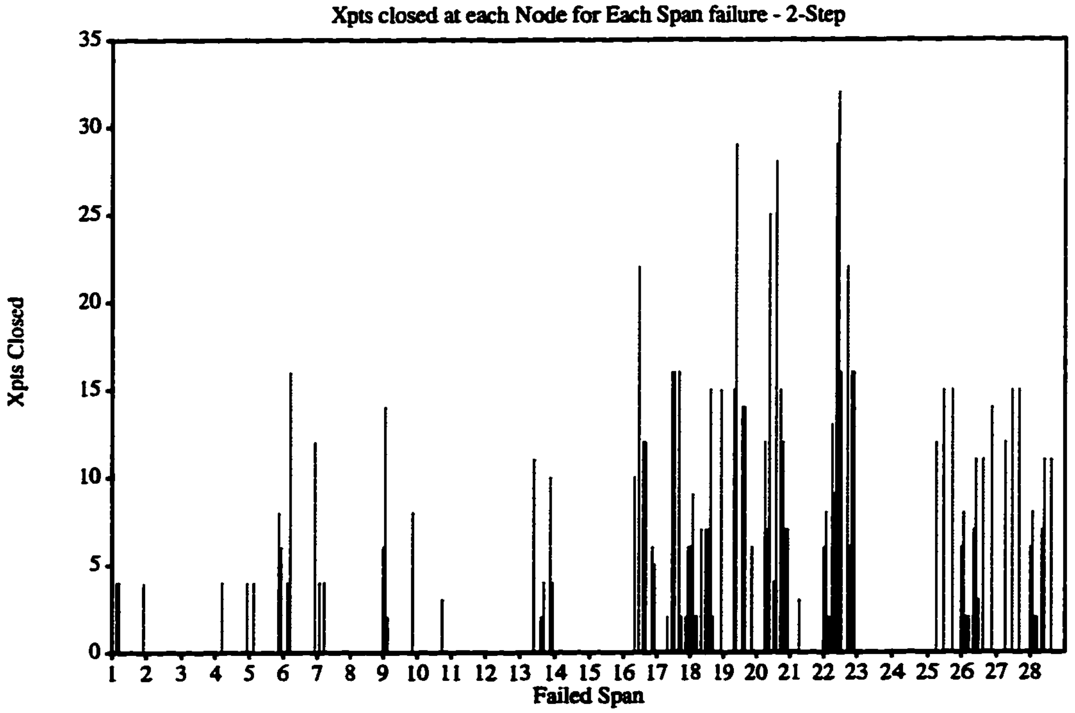


FIGURE 4.9 Crossconnections Closed at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method GA2. For Restoration using only Conventional KSP Restoration and for 2-Step Restoration.

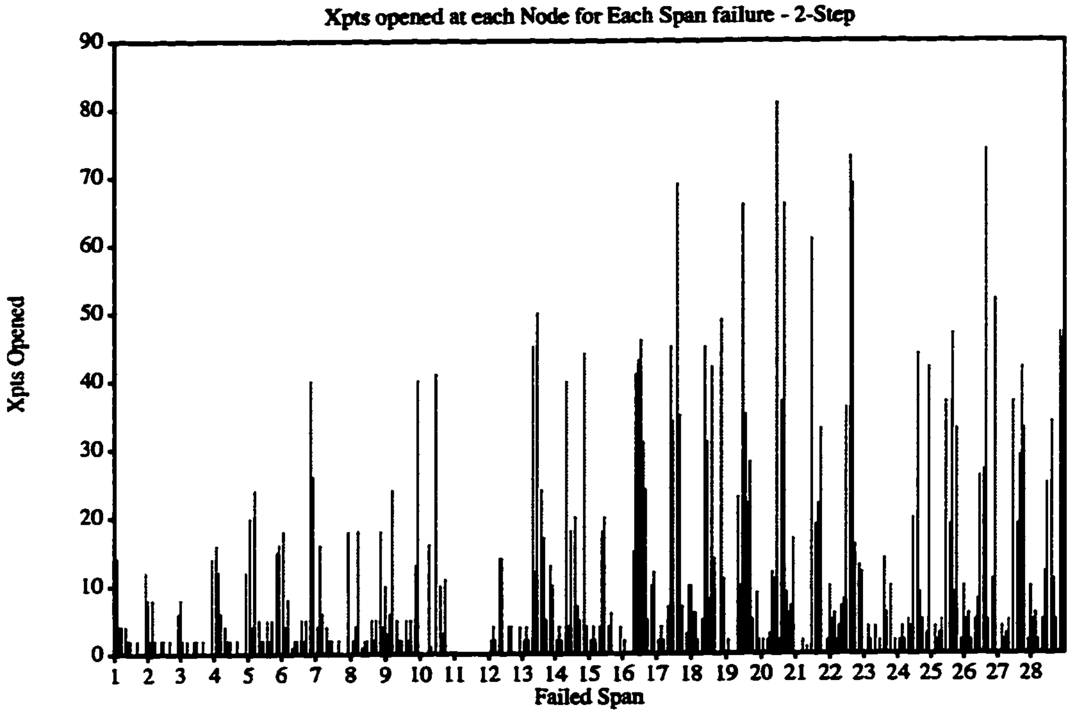


FIGURE 4.10 Crossconnections Opened at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method GA2. For Restoration using 2-Step Restoration.

mits multiple input port cards would influence whether the large number of crossconnection to be opened would be harmful or not. The relative speeds of crossconnection open and close operations would also be a factor, as the number of crossconnections closures were dramatically reduced when using the PC restoration plan developed by GA1.

The genetic algorithm which represented patterns at the crossconnect level (GA2) performed reasonably well with regard to total number of crossconnection events. As with the PC plan generated by GA1, the PC plan produced by GA2 reduced the number of crossconnections which had to be made compared to KSP restoration. Additionally, it greatly reduced the number of crossconnections which had to be opened to expose the desired PC paths when compared to GA1. The representation used for this method scored patterns according to the number of PC paths which a pattern could provide compared to the number of crossconnects which are required to form the pattern. This score encourages the production of patterns which contain minimal crosspoints while providing a good number of PC paths.

5. Cycle-Oriented Preconfiguration using Integer Programming Techniques

In the previous chapters, designs for preconfigured restoration were generated using complex types of patterns, such as the trees generated by the spanning tree heuristics, and the unconstrained patterns generated by the genetic algorithm techniques. These patterns are complex in the sense that there is no constraint on the nodal degrees that a pattern can have (other than the nodal degrees of the underlying network.) As previously mentioned, there is currently no provision in modern DCS to directly connect together more that two port cards, although, a multiple unidirectional connection may be allowed by the DCS implementation.

A cycle is an example of a pattern which can, however, be directly implemented in any current DCS. In this chapter integer programming (IP) techniques are used to generate preconfiguration restoration plans using cycles as the building element. An IP, if defined and run successfully, will result in a solution which is optimal with regard to the problem given it. A PC cycle design produced by an IP will be optimal (as long as the problem is not constrained to reduce it to a manageable size) so a PC design generated by an IP will indicated the best performance that can be achieved using only cycles as building blocks.

5.1 General Setup of an Integer Program to Generate a Preconfigured Restoration Design from a Pre-defined Set of Patterns in a Network's Pre-existing Spare Capacity Plan

This method is a modified version of a method used to generate preconfigured restoration plans using preconfigured segments as design elements [10]. It is modified so that instead of considering a particular pattern type, such as segments, it has a finite set of pre-defined and arbitrary patterns from which it can draw on to generate a preconfigured design. This method uses IP techniques to form a design from the pre-defined patterns in the set. The result is optimal only with regard to the patterns contained within the set; a constrained or minimal pattern set will result in a constrained or sub-optimal solution.

Assume there exists a pattern set P of size N_P equal to the number of patterns contained in the set. Each of the patterns in the set has a unique configuration and properties. The contents of the set are preset and there are no constraints as to the type of pattern contained within the set.

For a pre-existing sparing plan, a particular PC design formed from pattern set P has the following constraint on its use of the spare capacity present on each network span j:

$$s_j \geq \; SPC_j = \sum_{i=1}^{N_P} (spc_{i,j}) n_i \qquad \text{(EQ 5.1)}$$

Where, s_j is the number of spare links available on span j, SPC_j is the number of spare links utilized by the PC design on span j, n_i is the number of copies of pattern i present in the PC design and $spc_{i,j}$ is the number of spare links required on span j to form a single copy of pattern i. The constraint given in eq. 5.1 exists for all network spans, and it requires that the spare capacity used on each span to form a PC design does not exceed the amount of available spare capacity.

The number of preconfigured paths, that a PC design can provide to span j (for its restoration) is given by the relationship:

$$PC_j = \sum_{i=1}^{N_P} (pc_{i,j}) n_i \qquad \text{(EQ 5.2)}$$

Where, PC_j is the total number of PC paths available for the restoration of span j, and $pc_{i,j}$ is the number of preconfigured paths that a single copy of pattern i of set P can provide for failed span j.

An additional constraint, on a PC design, is that for each span j it is required that the following relationship be met:

$$u_j + PC_j = w_j + r_j \qquad \text{(EQ 5.3)}$$

Where, w_j is the number of working links which are on span j, u_j is the number of working links on span j which do not have a PC path available to them (i.e. the amount by which PC_j is less than w_j,) and r_j is the number of PC paths which are in excess of the number needed on span j (i.e. the amount by which PC_j is greater than w_j.) Note that either u_j or r_j must be equal to zero. This constraint defines the relationship, on each span, between the number of working links which are not protected by PC paths, the number of working links, the total number of available PC paths, and the number of PC paths which exist but, for the given span, are not needed for further PC restoration.

Equations 5.1, 5.2 and 5.3 can be combined to form a set of constraints that can be passed to an IP engine to form a PC design using pattern set P. Constraints that the variables be positive and integer are also required to limit the IP to meaningful physical solutions. The IP constraint set, for a network with S spans, is given by:

$$\begin{aligned}
&s_j \geq \sum_{i=1}^{N_P} (spc_{i,j})\, n_i \qquad \forall j = 1, 2, \ldots, S \\
&u_j + \sum_{i=1}^{N_P} (pc_{i,j})\, n_i = w_j + r_j \qquad \forall j = 1, 2, \ldots, S \\
&0 \leq u_j \leq w_j \qquad \forall j = 1, 2, \ldots, S \\
&r_j \geq 0 \qquad \forall j = 1, 2, \ldots, S \\
&n_i \geq 0 \qquad \forall i = 1, 2, \ldots, N_P
\end{aligned} \qquad \text{(EQ 5.4)}$$

Where, $spc_{i,j}$ and $pc_{i,j}$ are pre-calculated before the IP engine is executed and take the form of integer coefficients in the relationships. Both $spc_{i,j}$ and $pc_{i,j}$ must be greater than or equal to zero. s_j and w_j are set by the network's configuration and also take the form of integer coefficients. They must also be non-negative.

In addition to a constraint set, and IP engine also requires an objective function to minimize or maximize. This function (which is minimized) is given by:

$$\sum_{i=1}^{S} u_i \qquad \text{(EQ 5.5)}$$

Minimizing the objective function given by eq. 5.5 results in a PC design, formed from the elements of P, which is optimal in the sense that total number of unprotected working links is at a minimum.

Using the constraints given by eq 5.4 and the objective function given by eq 5.5 with an IP engine will generate a PC design, in a pre-existing network working and sparing plan, which maximizes the number of useful PC paths available to in a network to assist in the restoration of a span failure.

This formulation, called IP1-general, will be used shortly in test cases where the pattern set P contains all distinct cycles of the network graph and the network sparing is given.

5.2 General Setup of an Integer Program to Generate a Fully Restorable Preconfigured Restoration Design and a Network Sparing Plan from a Pre-defined Set of Patterns in a Networks Pre-defined Working Routing Plan

In the previous section, a PC design was formed for a network whose working capacity and spare capacity had been pre-defined. In this section, a PC plan is designed for a network in which the working capacity plan has been placed but the spare capacity has not. It is required that the PC plan be designed so that the network is fully restorable using only preconfigured restoration and that the PC plan be designed so that the spare capacity, required to contain it, be minimized. As in the previous section, a design is formed from a pre-defined set of PC patterns.

First a finite set P of patterns, from which the PC design will be formed, is defined. The set contains N_P patterns and there is no constraint, in the general formulation, on the type of pattern which the set can contain although our interest will be in later constraining the design pattern set to consider cycles only.

For a particular PC design, the number of spare links which will be required on span j is given by:

$$s_j = \sum_{i=1}^{N_P} (spc_{i,j}) n_i \tag{EQ 5.6}$$

Where, s_j is the number of spare links which are required on span j, $spc_{i,j}$ is the number of spare links required on span j to form a single copy of pattern i, and n_i is the number of copies of pattern i which are present in the design.

The PC design is required to be fully restorable using only PC restoration as set by the following constraint:

$$w_j + r_j = PC_j = \sum_{i=1}^{N_P} (pc_{i,j}) n_i \tag{EQ 5.7}$$

Where, w_j is the number of working links which exist on span j, r_j is the number of PC paths which exceed the amount required to fully restore span j, PC_j is the total number of PC paths available for span j, and $pc_{i,j}$ is the number of PC restoration paths that a single copy of pattern i can provide for span j. Eq. 5.7 is a slightly modified version of eq. 5.3 with the number of unprotected working links, u_j, set to zero due to the requirement that the network be fully restorable using only the PC design.

Combining eqs. 5.6 and 5.7 results in the following constraint set:

$$s_j = \sum_{i=1}^{N_P} (spc_{i,j}) n_i \qquad \forall j = 1, 2, \ldots, S$$

$$w_j + r_j = \sum_{i=1}^{N_P} (pc_{i,j}) n_i \qquad \forall j = 1, 2, \ldots, S$$

$$s_j \geq 0 \qquad \forall j = 1, 2, \ldots, S$$

$$r_j \geq 0 \qquad \forall j = 1, 2, \ldots, S$$

$$n_j \geq 0 \qquad \forall i = 1, 2, \ldots, N_P$$

(EQ 5.8)

Where, $spc_{i,j}$, $pc_{i,j}$ and w_j take the form of pre-defined non-negative integer coefficients. All variables are required to be integer and positive. In this IP method, unlike the method in the previous section, s_j is a variable as a PC plan must be formed while also forming a network sparing plan.

An objective function is required which must be minimized or maximized by the IP engine. Because a network sparing plan is being generated to contain the PC plan, a function which minimizes the total required network sparing would produce an efficient design. The objective function, which is to be minimized, is given by:

$$\sum_{j=0}^{S} d_j s_j \qquad \text{(EQ 5.9)}$$

Where, d_i is the actual physical length of span j. The summation given by eq. 5.9 is equal to the total physical distance of the transmission medium (be it optical fibre or coaxial cable) that is required to provision the network's spare capacity. Minimizing this sum results in an economical savings.

Using eq 5.8 and eq 5.9 with an IP engine will generate a PC design and a network sparing plan from a network's pre-existing working routing plan and a pattern set. The PC design generated with this IP solution will be able to fully restore any failed network span using only PC paths.

This formulation, called IP2-general, will be used shortly in test cases where the pattern set P contains all distinct cycles of the network graph.

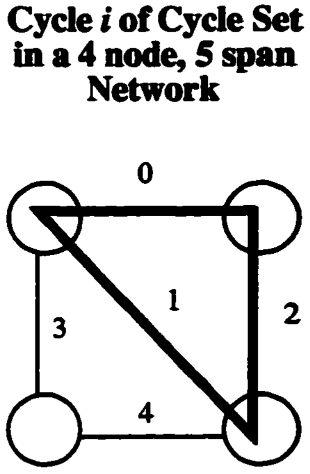
5.3 Setup of an Integer Program to Generate a Preconfigured Restoration Design using Cycles in a Network's Pre-existing Spare Capacity Plan

The general method given in section 5.1 formed a PC design using a combination of patterns drawn from a set containing a number of pre-defined patterns (IP1-general). In this section, the method will be used with a set containing only distinct cycles of the network graph. The method, constrained to use only cycles, will be referred to as IP1-cycle.

The constraints and objective function required for use in an IP are given by equations 5.4 and 5.5. However, it is required that the coefficients $pc_{i,j}$ and $spc_{i,j}$ be evaluated for the cycles contained in the pattern set. The coefficients w_i and s_i are set by the network's working and spare capacity placement plan.

The coefficient $spc_{i,j}$ is equal to the number of spare links required on span j to build a single copy of pattern i. For a cycle i, in a predefined set of cycles, $spc_{i,j}$ would be set equal to 1, if cycle i passes over span j; otherwise, $spc_{i,j}$ would be equal to 0. Refer to Figure 5.1 for an example of how $spc_{i,j}$ is determined.

The coefficient $pc_{i,j}$ is equal to the number of preconfigured paths that cycle i can provide to assist in the restoration of failed span j. For a cycle, $pc_{i,j}$ can be either 0, 1 or 2. It is zero, for a particular cycle and failed span, if either of the span's end nodes are not on the cycle. It is one if both of the span's end nodes are on the cycle and the end nodes are adjacent to one another along the cycle. It is two if both of the span's end nodes fall beneath the cycle and the end nodes are not adjacent to one another on the cycle. Figure 5.2 shows these different cases as an example of how $pc_{i,j}$ is calculated.



Pattern Span Utilization Coefficients $spc_{i,j}$

$spc_{i,0} = 1$

$spc_{i,1} = 1$

$spc_{i,2} = 1$

$spc_{i,3} = 0$

$spc_{i,4} = 0$

FIGURE 5.1 Determination of the Pattern Span Utilization Coefficients of a Cycle for use with PC design using IP techniques

5.4 Setup of an Integer Program to Generate a Fully Restorable Preconfigured Restoration Design and a Network Sparing Plan using Cycles in a Networks Pre-defined Working Routing Plan

The general method given in section 5.2 formed a PC design and a network sparing plan using a combination of patterns drawn from a set containing a number of pre-defined patterns (IP2-general.) In this section, the method will be used with a set containing only cycles and will be called IP2-cycle.

The constraints and objective function required for use in an IP are given by equations 5.8 and 5.9. The coefficient values given by $pc_{i,j}$ and $spc_{i,j}$ are evaluated for the cycles contained in the pattern set as in section 5.3. The w_i coefficients are set by the network's working routing plan.

5.5 Results

For both integer program design methods, Table 5.1 summarizes the maximum cycle length which was used to form the cycle set used with the previously described integer programming techniques. The maximum cycle length is set in maximum logical hops.

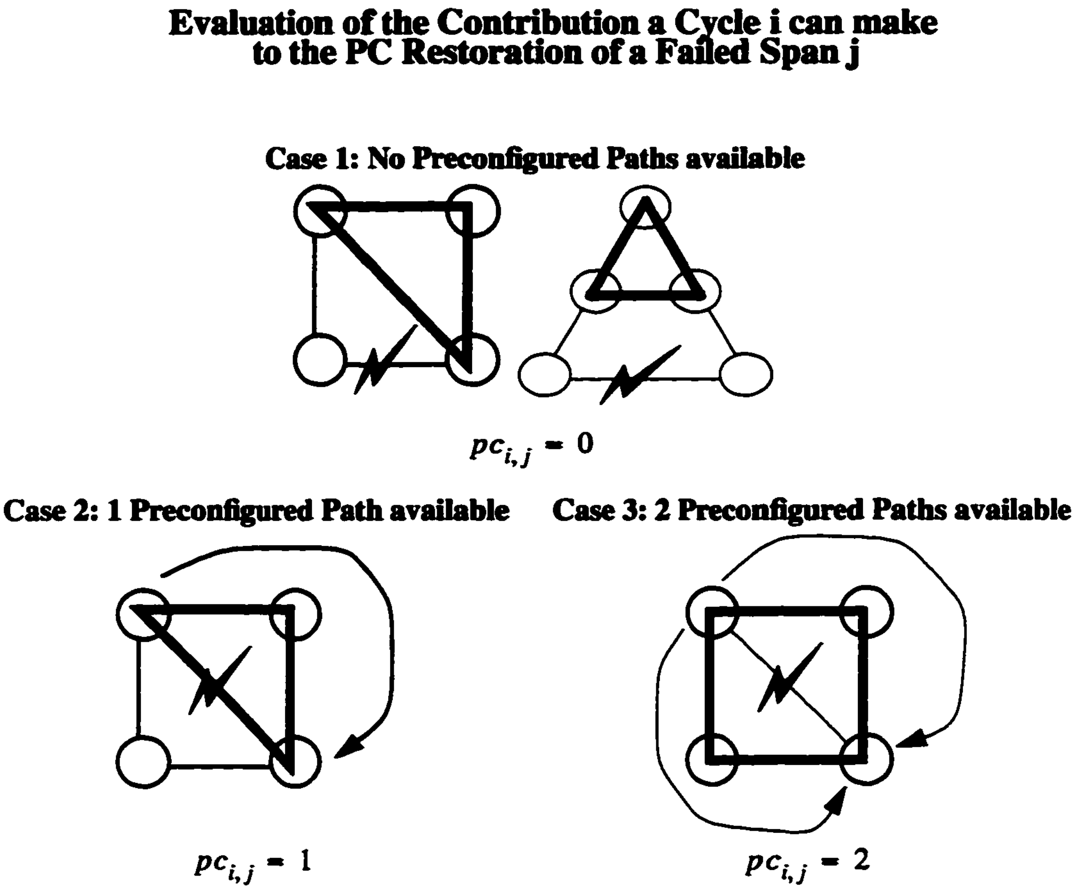

Evaluation of the Contribution a Cycle i can make to the PC Restoration of a Failed Span j
Case 1: No Preconfigured Paths available
$pc_{i,j} = 0$
Case 2: 1 Preconfigured Path available
Case 3: 2 Preconfigured Paths available
$pc_{i,j} = 1$
$pc_{i,j} = 2$

FIGURE 5.2 Evaluation of the number of preconfigured paths that a single preconfigured cycle can provide towards the restoration of a failed span

5.5.1 Preconfigured Cycle Design using Integer Program Techniques in a Network's Existing Spare Capacity Plan (IP1-Cycle)

Refer to Figure 5.3 for an example of the type of PC pattern that is generated in Net1 by IP1-Cycle. Table 5.2 contains the network restorability results over all possible span failures for restoration using only KSP, restoration using pure PC, and 2-step restoration. Table 5.3 contains the total crossconnection events, over all possible failed spans, for KSP restoration alone and for 2-step restoration. Figures 5.5 and 5.6 give counts of the number of crossconnection events required at each node to form a restoration pathset, for each span failure, in test network Net2, when using only on-demand KSP restoration and when using 2-step restoration. The crossconnect totals for each node appear sequentially within the space, on the x-axis, corresponding to each span failure.

5.5.2 Fully Restorable Preconfigured Cycle Design and Spare Capacity Placement using Integer Program Techniques (IP2-Cycle)

Refer to Figure 5.4 for an example of the type of PC pattern that is generated in Net1 by IP2-Cycle. Table 5.4 contains the network restorability results over all possible span failures for restoration using only KSP, restoration using pure PC, and 2-step. Table 5.5 contains the total crossconnection events, over all possible failed spans, for KSP restoration alone and for 2-step restoration. Table 5.6 contains the total sparing which is generated by this method, for each of the 5 test networks, and compares it the original sparing Figures 5.7 and 5.8 give counts of the number of crossconnection events required at each node to form a restoration pathset, for each span failure, in test network Net2, when using only on-demand KSP restoration and when using 2-step restoration. The crossconnect totals for each node appear sequentially within the space, on the x-axis, corresponding to each span failure.

Table 5.1: Maximum Cycle Length permitted within the Cycle Sets used with the Integer Programs

Network	Maximum Cycle Length (Logical Hops)
Net1	∞
Net2	∞
Net3	∞
Net4	12
Net5	25

Table 5.2: Network Restorability using Standard KSP and PC Restoration using IP1-Cycle

Network	KSP Restorability (%)	PC Restorability (%)	2-Step Restorability (%)
Net1	100	93.66	93.66
Net2	96.87	96.58	100
Net3	100	96.86	99.95
Net4 (*)	100	75.88	98.91
Net5 (*)	100	94.93	99.95

Table 5.3: Total Network Crossconnect Events Results for Standard KSP and for PC Restoration using IP1-Cycle

Network	Xpts Closed KSP	Xpts Closed PC	Xpts Opened PC	Total Xpt Events PC
Net1	310	0	174	174
Net2	3984	92	2260	2352
Net3	13787	366	7086	7452
Net4 (*)	84389	20032	55419	75451
Net5 (*)	8529	312	4067	4379

*- *The Integer Program did not return a solution which is guaranteed to be optimal. Due to the size of the design problem, for Net4 and Net5, the IP engine terminated before finding a strictly optimal solution because it ran out of memory. The solution returned is the best found prior to the point of memory exhaustion failure*

Table 5.4: Network Restorability Results for Standard KSP and for PC Restoration using IP2-Cycle

Network	KSP Restorability (%)	PC Restorability (%)
Net1	100	100
Net2	100	100
Net3	100	100
Net4 (*)	100	100
Net5 (*)	100	100

Table 5.5: Total Network Crossconnect Events for Standard KSP and for PC Restoration using IP2-Cycle

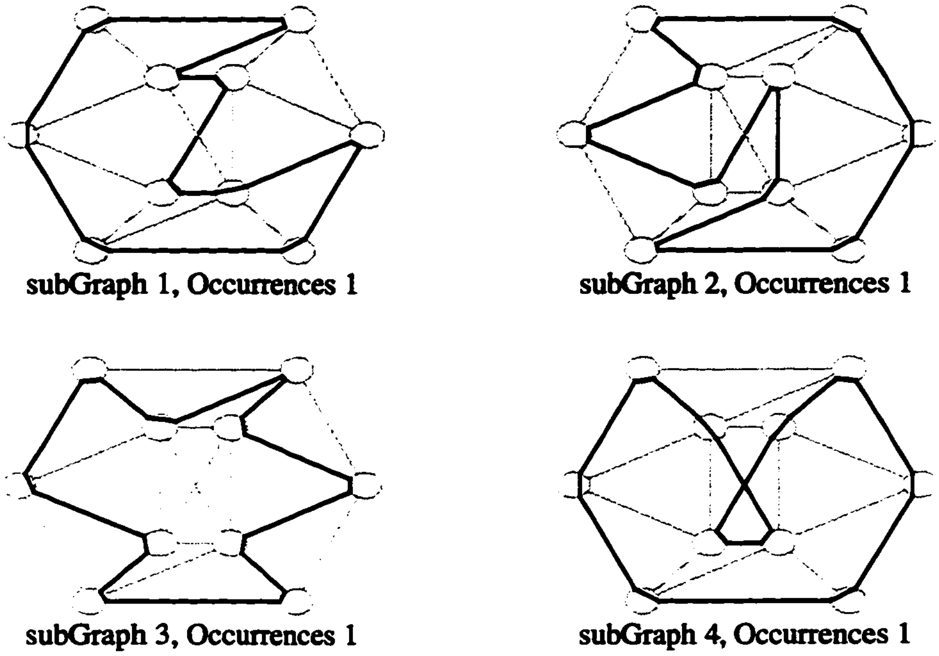
Network	Xpts Closed KSP	Xpts Closed PC	Xpts Opened PC	Total Xpt Events PC
Net1	291	0	190	190
Net2	4176	0	2108	2108
Net3	20744	0	6860	6860
Net4 (*)	86048	0	45216	45216
Net5 (*)	13727	0	3850	3850

**- The Integer Program did not return a solution which is guaranteed to be optimal. Due to the size of the design problem, for Net4 and Net5, the IP engine terminated before finding a strictly optimal solution because it ran out of memory. The solution returned is the best found prior to the point of memory exhaustion failure*

Table 5.6: Comparison between the Sparing Plans of IP2-Cycle and the Original Network Sparing Plan

Network	Total Real Fiber Distance Original Spare Plan	Total Real Fiber Distance PC Cycle Spare Plan	% Excess between Original and PC Cycle
Net1	44	48	9.09
Net2	6388	6584	3.07
Net3	177804	175476	-1.31
Net4 (*)	874189	1036240	18.54
Net5 (*)	672876	647241	-3.81

**- The Integer Program did not return a solution which is guaranteed to be optimal. Due to the size of the design problem, for Net4 and Net5, the IP engine terminated before finding a strictly optimal solution because it ran out of memory. The solution returned is the best found prior to the point of failure.*



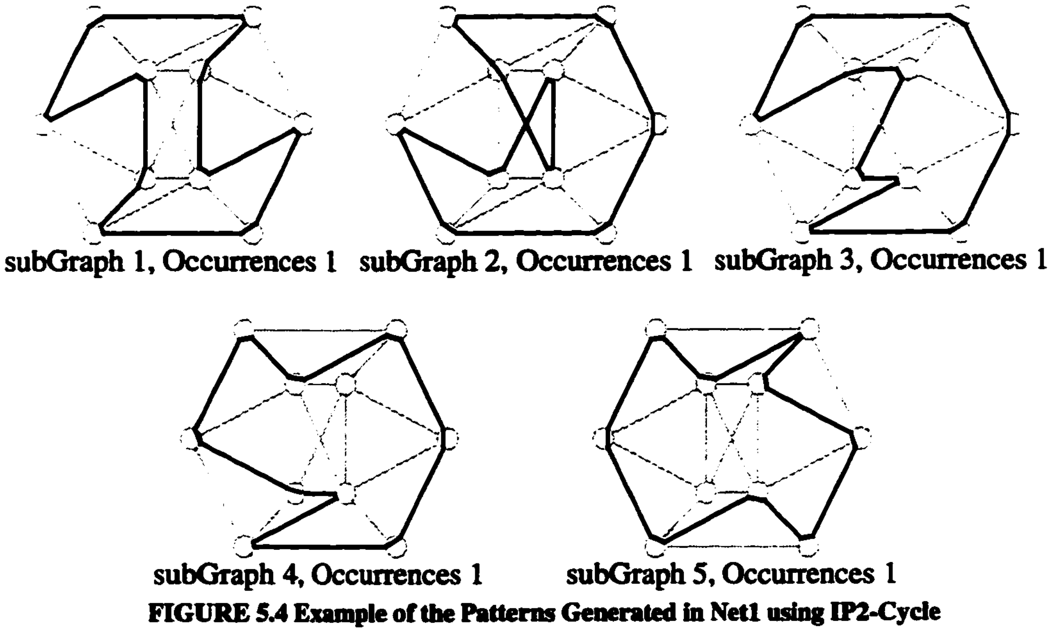
subGraph 1, Occurrences 1

subGraph 2, Occurrences 1

subGraph 3, Occurrences 1

subGraph 4, Occurrences 1

FIGURE 5.3 Example of the Patterns Generated in Net1 using IP1-Cycle



subGraph 1, Occurrences 1

subGraph 2, Occurrences 1

subGraph 3, Occurrences 1

subGraph 4, Occurrences 1

subGraph 5, Occurrences 1

FIGURE 5.4 Example of the Patterns Generated in Net1 using IP2-Cycle

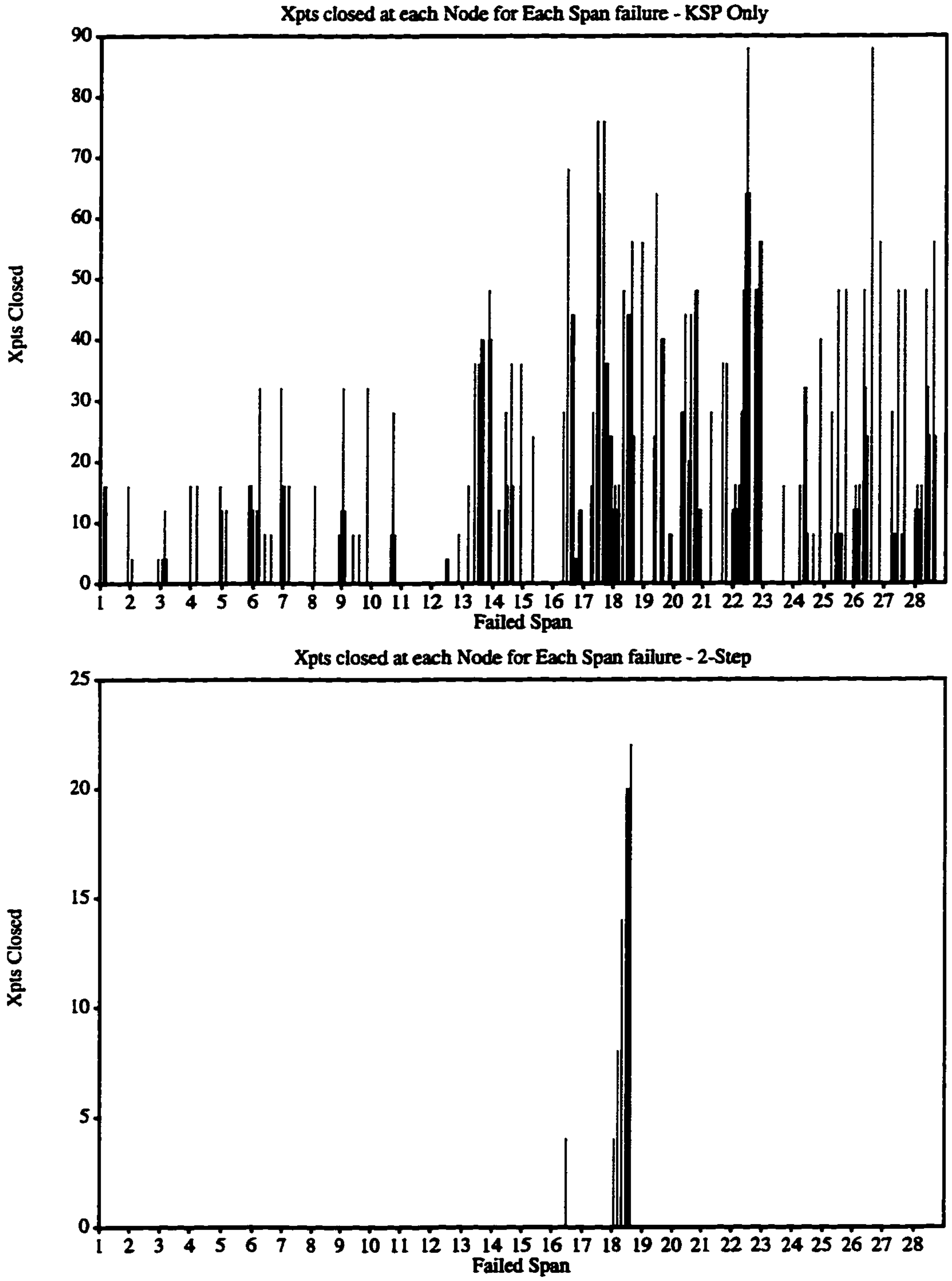


FIGURE 5.5 Crossconnections Closed at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method IP1-Cycle. For Restoration using only Conventional KSP Restoration and for 2-Step Restoration.

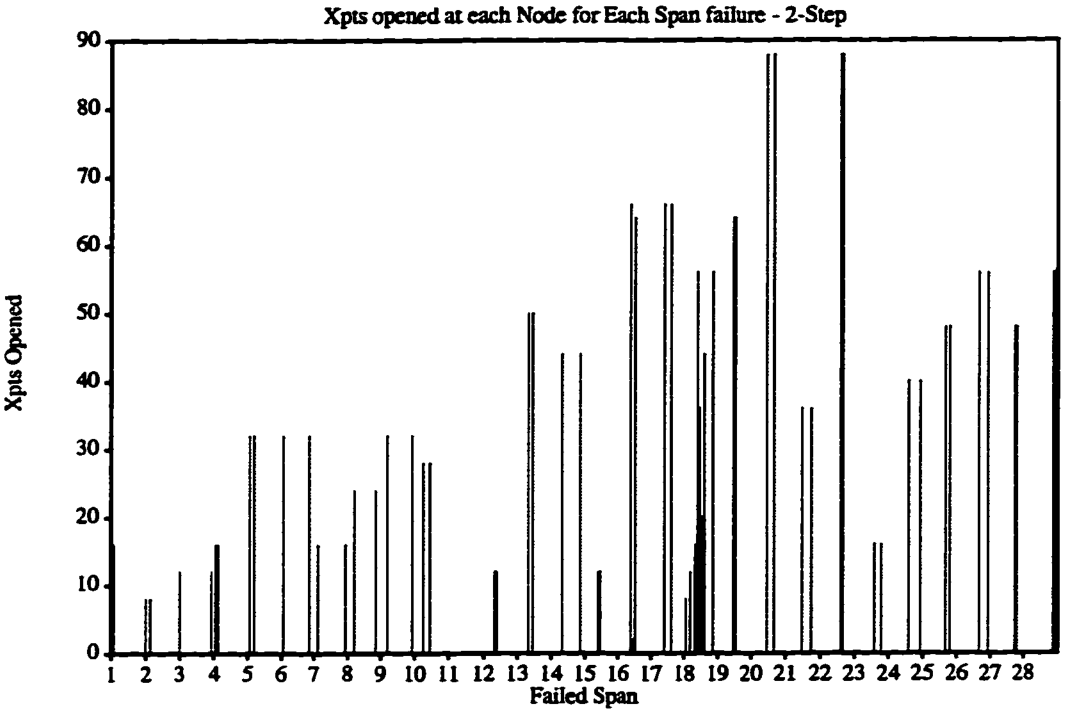


FIGURE 5.6 Crossconnections Opened at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method IP1-Cycle. For Restoration using 2-Step Restoration.

5.6 Conclusions

The designs generated using IP techniques, with cycles as the preconfigured element, offered very good preconfigured restorability and preconfigured restoration crossconnection event counts.

For the design of PC cycle plans in a network's existing sparing plan, the PC restorability which resulted ranged from 75% to 97% in the test networks. The low score of 75% which resulted in network Net4 is most likely due to the severely restricted cycle set which was used by the IP to form the PC plan. Because of Net4's complexity, a hop limit of 12 was used in its IP design. The remaining networks had PC restorabilities in the range of 94% to 97%. The overall restorability, using PC restoration and leftover KSP restoration, was generally quite good with values ranging from 94% to 99%.

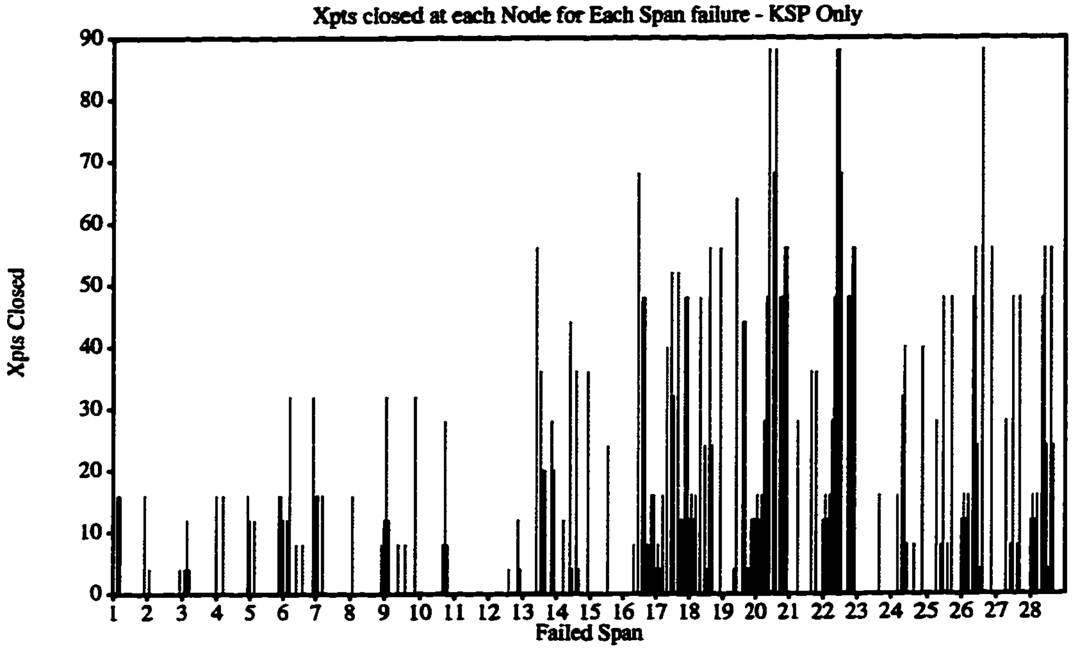


FIGURE 5.7 Crossconnections Closed at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed withinNet2 using method IP2-Cycle. For Restoration using only Conventional KSP Restoration.

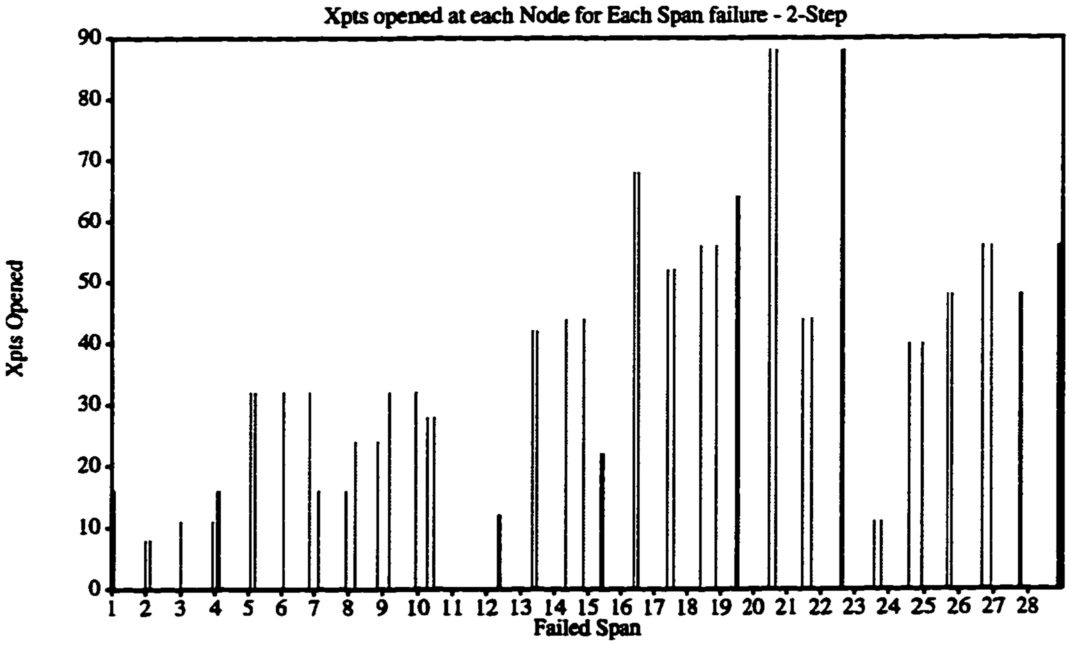


FIGURE 5.8 Crossconnections Opened at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using method IP2-Cycle. For Restoration using 2-Step Restoration.

The combined design of fully restorable PC cycle plans and spare capacity plans which the second IP technique generated were also quite efficient. The network's are, of course, fully restorable using only the preconfigured paths available within the network's PC plan. The price which was paid, in terms of excess spare capacity compared to the original sparing plans, so that the network would be fully restorable using only PC paths ranged from 18.54% to -3.81%. The large excess sparing requirement of 18.54% occurred for Net4 and the likely reason for this excessive sparing is, again, that the cycle set used to form the design was severely restricted due to the size and complexity of Net4. The two network test cases where the overall network sparing was actually reduced, for the fully restorable PC design, are likely due to the fact that the original sparing plans were generated using an IP engine with a constrained problem definition. The IP problem was formulated so that a solution would be generated in which all of the network's spans would be fully restorable using restoration paths, in the network's spare capacity, which routed around each failed span. However, much as with IP cycle design, the IP may only form restoration/sparing plans from the paths contained in a predefined pathset. In a large network the total number of possible paths, which join the end nodes of every possible failed span, will quickly become very large and, as a result, the problem will require that a constraint be placed on the maximum path length which can be considered.

The performance, in regards to the total crossconnection events required to achieve restoration, was also good using preconfigured cycles. For the case where a PC plan is designed in an existing sparing plan (IC1-Cycle), the total crossconnect closures in the test cases were all substantially lower than the total crossconnect closures required for KSP restoration. The actual reduction ranged from approximately 76% to 100%. The reduction in total crossconnect closures was even more pronounced for the fully restorable PC plan generated by the IP2-Cycle where all test networks had 0 closures. In addition, both IP1-Cycle and IP2-Cycle had a low total number of opened crossconnection which is a direct result of the cycle's simple structure and low nodal degree. Because a cycle can, by definition, have only degree 2 nodes contained within itself, the number of crossconnections required to form a cycle is relatively small.

An additional point, in favour of PC cycles, is, if a cycle is useful towards contributing PC paths towards the restoration of a span failure, then only the nodes on the ends of the span failure need prune crossconnections in the cycle to extract the usefully preconfigured paths. The implication of this is that no signalling is required to other network nodes to inform them of a failure as the only nodes which need do any pruning are the failed span's end nodes. Therefore, the PC paths present within a plan composed of PC cycles could be extracted and utilized more quickly than the plans designed using the tree and genetic algorithm methods. Figure 5.8, which shows the node level crossconnect opens in Net2 for IP2-Cycle, shows how only the end nodes on a failure need prune any crossconnects to extract the plan's PC paths; each span failure has only a pair of spikes which correspond to the failed span's end nodes while the other nodes have a crossconnect workload of zero.

6. Theoretical Considerations and Interpretation

Intuitively, it would seem that the performance of the more complex, high nodal degree patterns would be superior to that of a simple type of pattern, such as the cycle, due to the large number of preconfigured paths which a high complexity pattern could contain. However, in the previous chapter the performance of preconfigured cycles was investigated and was found to be quite good even compared to that of the unconstrained patterns generated by the genetic algorithms in chapter 4. How can this be explained or understood?

In this chapter, the theoretical upper limit to the number of preconfigured paths, which various classes of pattern can contain, is evaluated to provide insight on this issue. The upper path limit is evaluated for the preconfigured tree/segment class and for the preconfigured cycle class. Additionally, the theoretical upper path limit that any pattern, covering a certain number of nodes, can achieve is evaluated and compared to that of the tree and cycle class.

6.1 Maximum Useful PC Paths for Preconfigured Trees / Segments

The upper limit to the maximum number of preconfigured paths per preconfigured pattern link can be found for the tree type (of which segments are a subset.) A tree is any fully connected set of nodes for which there exists one, and only one, path between each pair of nodes, along the tree's branches.

Assume a full mesh network with a very large number of nodes, and a very large amount of spare and working capacity on the spans connecting the nodes. This assumption allows any path (save those preconfigured paths which traverse a failed span) which exists in a preconfigured pattern between any pair of network nodes to be a *useful path*. Now assume that there is a tree of size N nodes which is preconfigured in the network's spare capacity.

6.1.1 Spare Links Utilization in a Preconfigured Tree of size N

The number of spare links s used to form the tree pattern is given by:

$$s = N - 1 \qquad \text{(EQ 6.1)}$$

6.1.2 Derivation of Maximum for a Preconfigured Tree spanning N nodes

By definition, a tree can provide only a single path between each pair of its nodes. However, a preconfigured tree can not provide a restoration path to any span which lies directly beneath the tree because the failure of such a span splits the tree into two disconnected patterns where each of the disconnected patterns contains one of the failed span's end nodes. The maximum number of spans for which the tree can provide a restoration path is given by the difference between the maximum number of spans bounded by the tree and the number of spans which form the tree. Therefore, the maximum number of paths P that a preconfigured tree can provide is:

$$P = \binom{N}{2} - (N-1) = \left(\frac{N}{2} - 1\right)(N-1) \qquad \text{(EQ 6.2)}$$

6.1.3 Maximum Number of Preconfigured Paths per Preconfigured Tree Spare Link

Next, the efficiency measure of interest, the maximum number of PC paths per pattern link, E, is given by:

$$E = \frac{P}{s} = \frac{\left(\frac{N}{2} - 1\right)(N-1)}{N-1} = \frac{N}{2} - 1 \qquad N \geq 2 \qquad \text{(EQ 6.3)}$$

Therefore, the maximum number of preconfigured paths that a single preconfigured tree can provide per preconfigured pattern link is roughly half the number of nodes in the tree.

6.2 Maximum for a Preconfigured Cycle covering N nodes

A cycle is defined as a set of fully connected nodes where each node in the pattern has degree two (i.e. each node is connected to two other nodes.)

Assume there exists a full mesh network with a large number nodes and a large amount of spare and working capacity on its spans, as in the previous section. Now assume that a cycle with N nodes is preconfigured in the network's spare capacity.

The number of spare links s which are required to form a cycle spanning N nodes is:

$$s = N \qquad \text{(EQ 6.4)}$$

A cycle can provide preconfigured restoration paths for two kinds of potential failed span. The first kind of span is one which is not actually a part of the cycle but its end nodes fall beneath the cycle. For this kind of span, the pattern can provide 2 preconfigured restoration paths. The second kind of span is one which lies beneath the cycle. For this kind of span the pattern can provide 1 preconfigured restoration path. The maximum number of spans, which a cycle with N nodes can bound, is equal to $\binom{N}{2}$. N of these spans are of the second type of span which lie directly beneath the cycle and the remaining $\binom{N}{2} - N$ spans are of the first type of span, which do not lie directly beneath the cycle, but whose end nodes do fall beneath it. Therefore, the maximum number of preconfigured paths P that a cycle with N nodes can provide is:

$$P = 2\left[\binom{N}{2} - N\right] + 1N = N(N-2) \qquad \text{(EQ 6.5)}$$

The maximum number of paths per pattern link, E, is given by:

$$E = \frac{P}{s} = N-2 \qquad N \geq 3 \qquad \text{(EQ 6.6)}$$

Therefore, the maximum number of paths that a cycle can provide for each of the spare links used in its formation, varies directly with the number of nodes used in the cycle. This compares quite well with the maximum associated with preconfigured trees found in the previous section.

6.3 Upper Limit to the Number of Preconfigured Paths for any Preconfigured Pattern with N nodes

An upper limit to the maximum number of preconfigured paths can be derived for any preconfigured pattern covering N nodes. Once again, assume there is a full mesh network with a very large number of nodes, and a very large amount of spare and working capacity on the spans connecting the nodes. Also, assume there exists, in the network's spare capacity, a preconfigured pattern of an arbitrary type which spans N nodes.

6.3.1 Spare Link Utilization

The number of spare links s used in the formation of the pattern can be found from the general equation:

$$s = \frac{1}{2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{d_i} \alpha_{i,j} \qquad \text{(EQ 6.7)}$$

Where N is the number of nodes in the pattern, d_i is the number of pattern spans which fall on node i in the pattern, and $\alpha_{i,j}$ is the number of spare links that the pattern has on span j at node i.

6.3.2 Upper Limit to the Number of Preconfigured Paths

The maximum number of preconfigured restoration paths, for all potential failed spans, that a pattern node i can source is given by:

$$P_i = P_{PS_i} + P_{NPS_i} \qquad \text{(EQ 6.8)}$$

Where P_{PS_i} is the number of preconfigured paths that can be sourced for spans which are in the pattern, P_{NPS_i} is the total number of preconfigured paths that can be sourced for all spans which are not in the pattern (but whose end nodes fall beneath the pattern,) and P_i is the total number of preconfigured paths that node i can source for all potential span failures.

For a span not in the pattern, node i can source a single outgoing restoration path for each link which is a part of the pattern and falls on node i. Therefore, P_{NPS_i} is equal to:

$$P_{NPS_i} = (N-1-d_i)\sum_{j=1}^{d_i}\alpha_{i,j} \qquad \text{(EQ 6.9)}$$

Where $(N-1-d_i)$ is the number of spans which fall on node i but are not a part of the pattern, and $\sum_{j=1}^{d_i}\alpha_{i,j}$ is the number of pattern links which fall on node i.

For a span which is in the pattern, node i can have an outgoing restoration path for each pattern link which is a part of the pattern but not on the failed span. Therefore, P_{PS_i} is equal to:

$$\begin{aligned} P_{PS_i} &= \sum_{j=1}^{d_i}\left[\left(\sum_{k=1}^{d_i}\alpha_{i,k}\right)-\alpha_{i,j}\right] \\ &= \sum_{j=1}^{d_i}\left(\sum_{k=1}^{d_i}\alpha_{i,k}\right)-\sum_{j=1}^{d_i}\alpha_{i,j} \\ &= (d_i-1)\sum_{j=1}^{d_i}\alpha_{i,j} \end{aligned} \qquad \text{(EQ 6.10)}$$

Where $\left(\sum_{k=1}^{d_i}\alpha_{i,k}\right)-\alpha_{i,j}$ is the number of pattern links which fall on node i excluding those pattern links which are on span j.

Summing the contributions from equations 6.9 and 6.10 gives:

$$\begin{aligned} P_i &= P_{PS_i} + P_{NPS_i} \\ &= (N-1-d_i)\sum_{j=1}^{d_i}\alpha_{i,j} + (d_i-1)\sum_{j=1}^{d_i}\alpha_{i,j} \\ &= (N-2)\sum_{j=1}^{d_i}\alpha_{i,j} \end{aligned} \qquad \text{(EQ 6.11)}$$

Where P_i is the maximum number of preconfigured restoration paths that node i can source for the failure of all possible spans.

The limit for the maximum number of preconfigured paths that can exist within a pattern for a potential span failure j, is determined by the maximum number of paths that the span's end nodes can source in the pattern, and by the maximum number of paths that other nodes in the pattern can carry. In general, the limiting factor to the number of paths the pattern can provide will be the number of paths that the span's end nodes can source. Therefore:

$$PS_j \le MIN(\beta_{j,a_j}, \beta_{j,b_j}) \qquad \text{(EQ 6.12)}$$

Where PS_j is the maximum number of paths available for span j, a_j and b_j are span j's end nodes, and $\beta_{j,k}$ is the number of preconfigured paths that node k can source for span j. What equation 6.12 states is that the maximum number of preconfigured paths that can be provided for a failed span j will be less than or equal to the smaller of the number of paths that each of the span's end nodes can source.

A less rigorous upper limit for PS_j can be found by considering the average of β_{j,a_j} and β_{j,b_j}. This average will be greater than or equal to the upper limit found in equation 6.12 because equation 6.12 is found by taking the smaller of β_{j,a_j} and β_{j,b_j}. The average of two numbers will always be greater than the smaller of the two numbers unless the two numbers are equal in which case the average will be equal to the two numbers. Therefore, another upper limit for PS_j is:

$$PS_j \le min\,(\beta_{j,a_j}, \beta_{j,b_j}) \le \frac{1}{2}(\beta_{j,a_j} + \beta_{j,b_j}) \qquad \text{(EQ 6.13)}$$

The largest number of paths P that any preconfigured pattern, covering N nodes, can provide is given by:

$$P = \sum_{j=1}^{N_S} PS_j \le \sum_{j=1}^{N_S} \frac{1}{2}(\beta_{j,a_j} + \beta_{j,b_j}) \le \frac{1}{2}\sum_{i=1}^{N} P_i = \frac{1}{2}\sum_{i=1}^{N} (N-2) \sum_{j=1}^{d_i} \alpha_{i,j} \qquad \text{(EQ 6.14)}$$
$$\le \frac{1}{2}(N-2) \sum_{i=1}^{N} \left(\sum_{j=1}^{d_i} \alpha_{i,j} \right)$$

Where N_S is the number of spans in the network. The summations $\sum_{j=1}^{N_S} \frac{1}{2}(\beta_{j,a_j} + \beta_{j,b_j})$

and $\frac{1}{2}\sum_{i=1}^{N} P_i$ are equivalent. The first summation sums the preconfigured paths that can be sourced by each span's end points, over all spans, while the second summation sums the total number of preconfigured paths, for all spans, that each node can source, over all nodes. The final result must be the same for both summations.

Therefore, the upper limit to the number of preconfigured paths that any preconfigured pattern can provide is given by:

$$P \le \frac{1}{2}(N-2) \sum_{i=1}^{N} \left(\sum_{j=1}^{d_i} \alpha_{i,j} \right) \qquad \text{(EQ 6.15)}$$

The pattern which, for which equation 6.15 was derived, is assumed to be fully connected; that is, at each pattern node, there exists a crossconnection between each pair of pattern links on the node. The equation still holds for simpler patterns whose link ends are not fully connected because these patterns can be realized, from fully connected patterns, by breaking crossconnections; the breaking of a crossconnect can only reduce the number of PC paths which the pattern contains and, so, the upper limit given in equation 6.15 will still hold.

6.3.3 Maximum Number of Preconfigured Paths per Preconfigured Link

The maximum number of preconfigured paths per preconfigured spare links, E, for any preconfigured pattern covering N nodes can be found by expressions contained in equation 6.7 and equation 6.15. Therefore, E is given by:

$$E = \frac{P}{s} \le \frac{\frac{1}{2}(N-2)\sum_{i=1}^{N}\sum_{j=1}^{d_i}\alpha_{i,j}}{\frac{1}{2}\cdot\sum_{i=1}^{N}\sum_{j=1}^{d_i}\alpha_{i,j}} \le N-2 \qquad \text{(EQ 6.16)}$$

6.3.4 Discussion and Conclusions

In this section, the upper limit to the number of preconfigured paths that two simple patterns, tree/segment and cycle, could provide were determined. The maximum number of paths that any possible pattern could provide was also determined. The patterns were placed on a very large full mesh network with very large working capacity. This condition makes any preconfigured path between any node pair useful in the event of the failure of the span connecting that node pair.

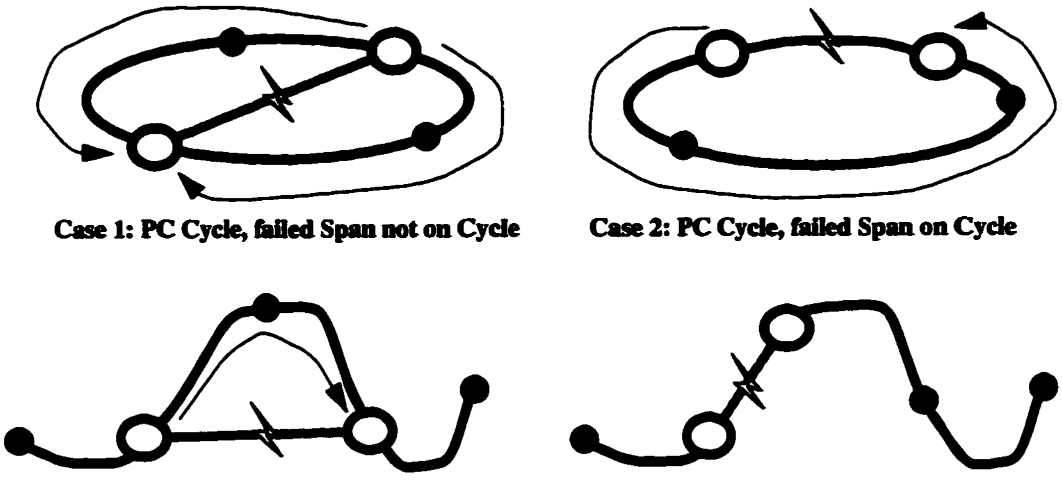
Table 6.1: Comparison of Maximum Preconfigured Paths Available for Different Pattern Types

Pattern Type	Maximum number of Preconfigured Paths per Preconfigured Pattern Link, for a pattern covering N nodes
Segment/Tree	$\frac{N}{2}-1 \qquad N \ge 2$
Cycle	$N-2 \qquad N \ge 3$
Any Possible Pattern	$N-2$

A comparison of the maximum number of preconfigured paths per preconfigured pattern link which can be provided by a single preconfigured cycle of size N against a single preconfigured tree of size N, gives the preconfigured cycle roughly a 2:1 advantage in the maximum number of preconfigured paths per spare link which it can provide. The cycle has this advantage because less of its preconfigured links are "wasted" when providing a restoration path to a failed span. For example, in Figure 6.1, there exists a preconfigured cycle and a preconfigured tree. Cases 1 and 3 are for a span failure where the span's end nodes fall beneath the preconfigured pattern but the failed span does not, and the cycle is able to provide two preconfigured paths while the tree is only able to provide one. The cycle links are fully utilized providing preconfigured paths but the tree links which do not fall on the tree's single preconfigured path cannot be used to form additional preconfigured paths. Cases 2 and 4 are for span failures which fall fully beneath the pattern. The cycle is able to provide a single preconfigured path while the tree is unable to provide any paths. The cycle is able to provide more paths because there exists 2 paths between each pair of cycle nodes, while a tree/segment has, by definition, only 1 path between each pair of tree nodes.

Another result of interest is that for a given pattern covering N nodes a cycle can provide a maximum of $N-2$ preconfigured restoration paths per spare link in the cycle. This is equal to the maximum that any configuration of preconfigured pattern covering N nodes can achieve. This result suggests that more complex patterns, with high pattern nodal degrees and more than one pattern link on a single span, will not necessarily be better performers than simpler patterns (such as cycles.)

Simpler patterns have the advantage of having low pattern nodal degree which results in fewer crossconnections which must be set (the number of crossconnections in a pattern tends to go with the square of the nodal degree in the pattern.) Also, simple patterns which only have, at most, a pattern nodal degree of 2, can be preconfigured in today's existing DCS, while it is unclear how a pattern nodal degree greater than 2 can be preconfigured (such a configuration would require a single port to be simultaneously connected to more than one other DCS port) without modification to the DCS's software and/or hardware.



Case 1: PC Cycle, failed Span not on Cycle

Case 2: PC Cycle, failed Span on Cycle

Case 3: PC Tree, failed Span not on Tree

Case 4: PC Tree, failed Span not on Tree

FIGURE 6.1 Comparison of Preconfigured Paths in PC Cycles and PC Trees/Segments

7. A Cycle-Oriented Distributed Preconfiguration Algorithm

The results of chapters 5 and 6 indicate that using cycles as the building blocks of a preconfigured restoration plan can be quite effective and efficient. The method given in chapter 5 for generating a PC plan uses an integer program engine which is quite computationally intensive and does not lend itself to real time generation of a PC plan. However, the result is capacity optimal with regard to the cycle set used to generate the PC plan.

Ideally, a PC design algorithm would be able to be run using only the computational power available within the network. Using such an algorithm would allow a network to directly respond to any changes made to its configuration with no need for outside intervention. The main advantages of this are that configuration changes in the network can be responded to quickly by the network, and the need for data gathering to determine the current configuration of the network can be eliminated (at least for the purpose of generating a PC design using a central designer.)

In this chapter a distributed preconfigured cycle design (DPCD) algorithm, which can be run in a network's nodes, is presented. The algorithm is distributed in the sense that its execution is spread amongst the significant processing power present in the DCS machines which form a mesh network's nodes. A network's nodes can execute the algorithm using only the information which is available locally at each node. This algorithm is a heuristic so the results are not guaranteed to be optimal but, for the advantage of being to design a PC plan using only the processing power present within the network, sub-optimal designs may be acceptable.

7.1 Method

This DPCD algorithm is based on the selfhealing network (SHN) protocol [3,4]. The SHN protocol implements a distributed algorithm which generates efficient path sets for span restoration and it does this using statelet based processing. A statelet is embedded on each network link and contains the current state of the link as described by a number of fields. A link is a bidirectional signal carrier and, so, each link, viewed locally by a node, has both an incoming statelet and outgoing statelet. An incoming statelet arrives at a node on a link, and originates from the adjacent node connected to the receiving node through the link. An outgoing statelet is broadcast from a node through a link, and arrives at the adjacent node which is connected to the broadcasting node through the link.

Each outgoing statelet has, in general, an incoming statelet which forms its precursor. An incoming statelet forms a precursor to an outgoing statelet if the incoming statelet was broadcast to form the outgoing statelet. In other words, an incoming statelet which forms the precursor of an outgoing statelet is the source for the outgoing statelet. An incoming statelet can be the precursor for many outgoing statelets but an outgoing statelet can have only one precursor incoming statelet.

As a statelet is broadcast throughout a network, it forms a statelet broadcast tree which, at each node in the tree, is rooted at the precursor incoming statelet from which the outgoing statelets are propagated. The particular route which a statelet travelled from the sender node to its present location is called the statelet broadcast route.

The distributed algorithm operates by the application of a set of rules at each node which preferentially broadcast certain incoming statelets over others. Each node acts only on the basis of information available to it only but the interaction of the nodes' actions results in the generation of a solution which is efficient at a global level.

7.1.1 Statelet Format

The statelet format used in the DPCD algorithm has 5 main fields:

- *index*: Each statelet belongs to an index family. At a node, when an outgoing statelet is broadcast from an incoming statelet, the outgoing statelet's index field is set equal to the index field of the incoming statelet from which it was broadcast.
- *hopcount*: As a statelet is relayed from node to node, a count of the number of hops it has taken is maintained. The hopcount in an incoming statelet is equal to the number of times that the statelet has been relayed by previous nodes to reach the node at which it has arrived.

- *sendNode*: All statelet broadcasts originate at a single node which acts as the sender node for the network. The sendNode field is set equal to the name of the sender node, and is the same for all statelets in the network.
- *numPaths*: As a statelet is relayed through the network, the number of useful PC paths which the statelet could provide if it could successfully loop back to the sender node to form a cycle, is evaluated. This field, in an incoming statelet, is set equal to the value which was evaluated at the node from which the incoming statelet originated.
- *route*: This field contains the route, from the sender node, which a statelet broadcast followed to become an incoming statelet at a particular node. This field is a vector with a size equal to the number of nodes in the network. Each node in the network is represented by a cell in the route vector. Each cell corresponding to a node contains the name of the node which was the predecessor of the node along the route. That is, the cell corresponding to a node points back to the node which fell before the cell node on the route. If a node does not fall on the route then its cell is set equal to NIL. As a node broadcasts a statelet to an adjacent node, the route field is updated so that the receiving node's cell points to the broadcasting node. Using the route vector allows the route, which was taken by a statelet broadcast to reach a particular node, to be traced back to the sender node.

These 5 fields form the basis of the statelet used in this distributed PC cycle design algorithm.

7.1.2 Network Node Roles

In the SHN protocol there are three node roles: that of the sender, the chooser and the tandem nodes. The sender node is the source of the broadcast pattern and the chooser node is the recipient of the broadcast pattern. The tandem nodes in the network relay received statelets according to the rules of the SHN protocol. The SHN protocol is concerned with the restoration of a span failure at the span level. The end nodes on a failed span become the sender and chooser nodes (which end node takes on which role is chosen in an arbitrary manner.) All other nodes become tandem nodes as broadcast statelets reach them. So in summary, the sender node initiates a statelet broadcast pattern, the tandem nodes relay the broadcast pattern according to the SHN protocol, and the chooser node receives the broadcast pattern. When the chooser receives an incoming statelet, then there exists a restoration path for the failed span. The path is found by tracing the exact link-level path that the statelet followed from the sender to reach the chooser in a process called reverse linking.

However, in the DPCD algorithm, the objective is different from that of the SHN protocol. In the SHN, it is required that paths joining the end nodes of a failed span be found, in order that the span be restored. In the DPCD algorithm, it is required that cycles which are maximally effective at providing PC paths, over all possible span failures, are to be found. The node roles in the DPCD algorithm are modified to reflect that cycles are to be generated rather than point to point paths. In the DPCD, there are two possible node types: that of a combined sender/chooser node which sources and receives the statelet broadcast pattern, and that of the tandem nodes which relay the broadcast statelet pattern.

The combined sender/chooser or "cycler" node initiates the statelet broadcast, the tandem nodes relay it according to the DPCD algorithm's statelet processing rules, and, eventually, the statelet broadcast pattern loops back to touch the cycler node. At this point, there exists a cycle which the cycler node can choose to take. The cycle is defined by following the route taken by the statelet broadcast pattern to return back to the cycler node. However, the cycler does not take immediately take the cycle indicated by the first incoming statelet it receives. Instead, it samples the incoming statelets and updates a record of the best received cycle each time it receives a new incoming statelet. When there is no further improvement in the best received cycle, the cycler node terminates the statelet broadcast and initiates the construction of a single link-level copy of this cycle. Each initiation of a statelet broadcast by a cycler node will eventually result in the construction of a cycle (if a cycle exists.)

7.1.2.1 . The Tandem Node Statelet Broadcast Rules

The bulk of the processing takes place in the tandem nodes of a network in both the SHN protocol and the DPCD algorithm. The tandem nodes determine what the actual shape of the statelet broadcast pattern will be through the repeated application of the statelet broadcast rules locally at each node.

The basic statelet broadcast rule is quite simple; for each incoming statelet, try to broadcast a single outgoing copy of it on a spare link, whose outgoing side is unoccupied, on each span except for the span on which the incoming statelet arrived. This is a simple broadcast of an incoming statelet and attempts to broadcast the statelet to the largest extent possible. Simple broadcasts of incoming statelets typically only happen at the start of the cycler node's initiation of the statelet broadcast because a single incoming statelet can be broadcast to form the precursor of multiple outgoing statelets and, as the tandem nodes relay incoming statelets, the spare link's outgoing sides, at each node, are quickly occupied by outgoing statelets. Refer to Figure 7.1 for an example of a simple broadcast of an incoming statelet. In the example, a single incoming statelet arrives on the incoming side of a node's spare port. There are no other incoming statelets present at the node so the incoming statelet is broadcast to the largest extent possible, forming the precursor of a single outgoing statelet on each network span (except the span on which the incoming statelet arrived.)

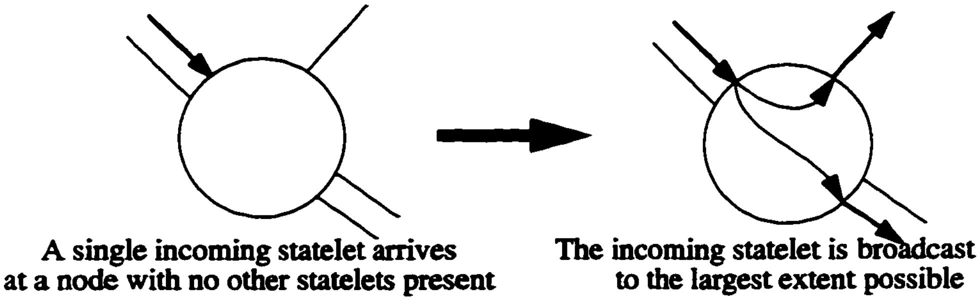
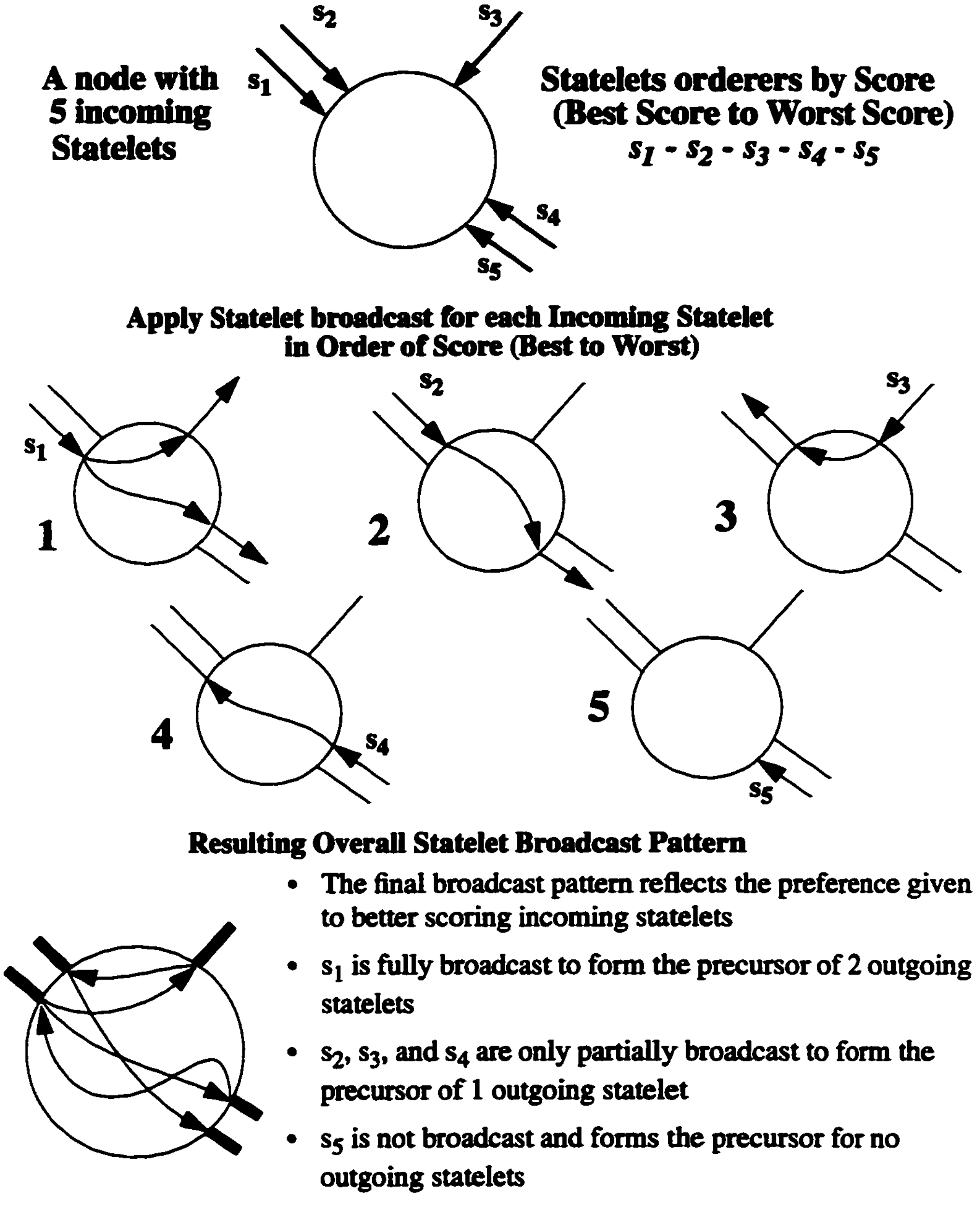

A single incoming statelet arrives
at a node with no other statelets present
The incoming statelet is broadcast
to the largest extent possible

FIGURE 7.1 Example of a Simple broadcast of an Incoming Statelet at a Tandem Node

A modification to the simple statelet broadcast, is the score based statelet broadcast. With the addition of this rule, an incoming statelet is broadcast to the largest extent possible prioritized by a score which is assigned to it (the rules which evaluate the score are discussed in the next section.) If multiple incoming statelets are present at a node then the statelets compete, on the basis of their scores, for a maximum statelet broadcast. A consequence of a node's incoming statelets being broadcast on the basis of their scores is that if a superior statelet arrives at a node and there are no free spare links available, on one or more spans, then the incoming statelet will, on each span with no unoccupied spare link, displace the single outgoing statelet whose precursor has the worst score. An incoming statelet can be broadcast on a span in one of two ways. The first way is by simple broadcast, which happens if there is at least one spare link on the span whose outgoing side is unoccupied by a statelet. The second way is by displacing an outgoing statelet whose precursor's score is both worse than that the score of the incoming statelet being considered for broadcast, and, also, the worst of the scores of all the incoming statelets which form a precursor to an outgoing statelet on the span. If there is no spare link, on a span, whose outgoing side is unoccupied and the incoming statelet's score is not good enough to displace an existing outgoing statelet, then the incoming statelet will not be broadcast.

This competition among the incoming statelets distributes the limited number of outgoing statelet positions among the incoming statelets with preference given to those statelets with good scores. The statelet broadcast pattern, at a node, is updated continuously as new incoming statelets appear. However, the overall pattern will have the same appearance as if the following procedure was followed. First a list of incoming statelets is formed and evaluated to assign each statelet a score. Next the list is sorted so the statelets are ordered from best score, at the start of the list, to worst score, at the end. Next, the incoming statelets are broadcast, without statelet displacement, sequentially as each appears in the list, to the fullest extent permitted by the unoccupied outgoing side of the spare links in the network. The result of applying this procedure is that high scoring incoming statelets will be fully broadcast, medium scoring incoming statelets will be partially broadcast, and low scoring incoming statelets may not be broadcast at all. This is analogous to the "compete" procedure in the SHN. Refer to Figure 7.2 for an example of statelet broadcast by statelet score.

As previously mentioned, each statelet belongs to an index family. When an incoming statelet is broadcast to form an outgoing statelet, the index to which the incoming statelet belongs is also assigned to the outgoing statelet. All statelets of a common index in a network can be traced back to a single outgoing statelet, at the cycler node, which forms the basis of that index family. As incoming statelets are broadcast to form the precursor of multiple outgoing statelets, it can occur that multiple incoming statelets can appear at a node that all belong to the same index family. An additional rule, which is added to the broadcast rules of a tandem node, is that there can only be one outgoing statelet, which belongs to a particular index family, on any given span. The broadcast rules are processed, as previously described, with preference given to those incoming statelets with the best


A node with 5 incoming Statelets
s_2
s_3
s_1
Statelets orderers by Score (Best Score to Worst Score)
$s_1 - s_2 - s_3 - s_4 - s_5$
s_4
s_5
s_1
1
s_2
2
s_3
3
4
s_4
5
s_5

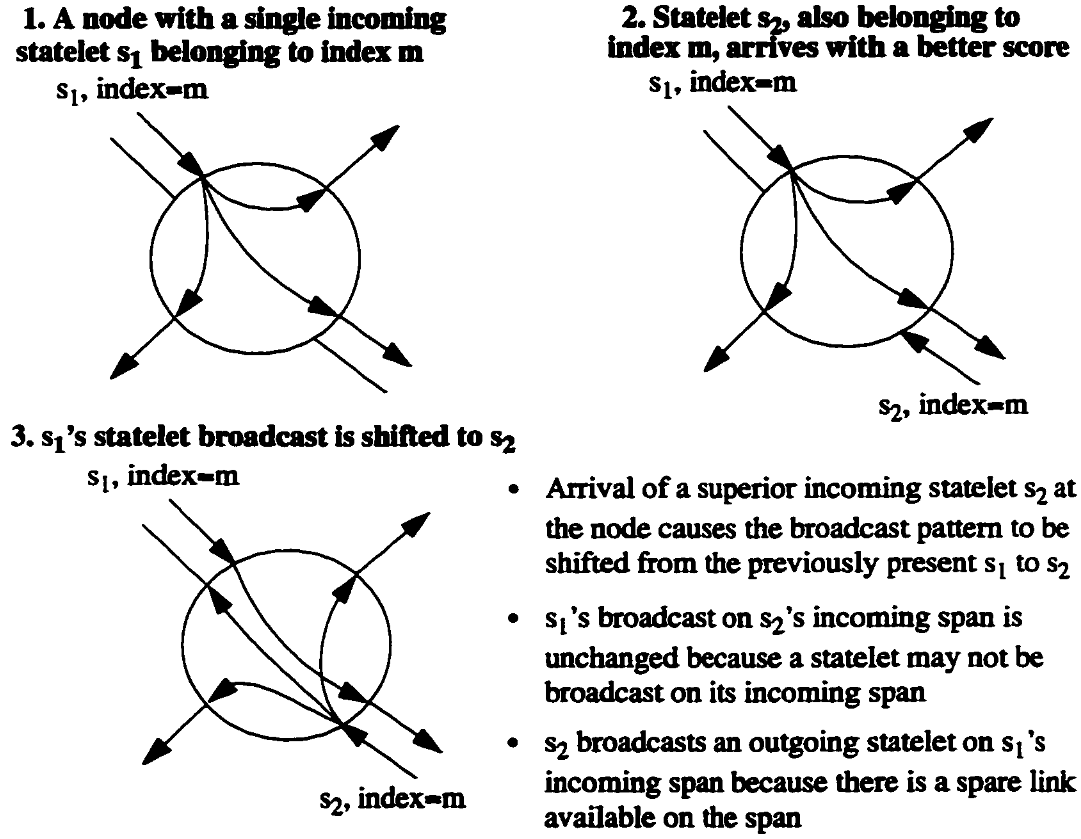
Apply Statelet broadcast for each Incoming Statelet in Order of Score (Best to Worst)

Resulting Overall Statelet Broadcast Pattern

- The final broadcast pattern reflects the preference given to better scoring incoming statelets
- s_1 is fully broadcast to form the precursor of 2 outgoing statelets
- s_2, s_3, and s_4 are only partially broadcast to form the precursor of 1 outgoing statelet
- s_5 is not broadcast and forms the precursor for no outgoing statelets

FIGURE 7.2 Example of the broadcast of Incoming Statelets with Preference given to Incoming Statelets with Good Scores

scores but there is an additional requirement that only a single outgoing statelet of any given index family can exist on any span. The incoming statelet, which forms the precursor of this outgoing statelet, has the best score of all incoming statelets of that index family which could potentially broadcast an outgoing statelet on that span. This rule requires that a new incoming statelet will displace a pre-existing outgoing statelet of the same index family, if the new incoming statelet would form a superior precursor. This has the effect of shifting the broadcast pattern for a particular index family from the original precursor statelet to the superior incoming statelet. Refer to Figure 7.3 for an example of the arrival of a superior incoming statelet where an incoming statelet, of the same index family, exists.



1. A node with a single incoming statelet s_1 belonging to index m

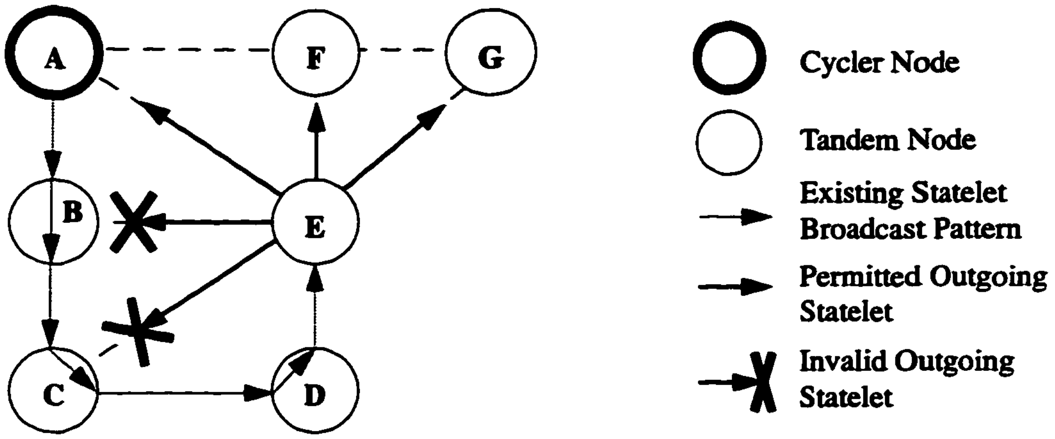
2. Statelet s_2, also belonging to index m, arrives with a better score

3. s_1's statelet broadcast is shifted to s_2

- Arrival of a superior incoming statelet s_2 at the node causes the broadcast pattern to be shifted from the previously present s_1 to s_2
- s_1's broadcast on s_2's incoming span is unchanged because a statelet may not be broadcast on its incoming span
- s_2 broadcasts an outgoing statelet on s_1's incoming span because there is a spare link available on the span

FIGURE 7.3 Example of the Shifting of a Statelet Broadcast from an Original Incoming Statelet belonging to an Index Family to a New Incoming Statelet, of the Same Index Family, with a Better Score

The final broadcast rule, and the only departure from the tandem node broadcast rules defined in the SHN protocol, is that an incoming statelet may not be the precursor of an outgoing statelet on a span which connects the node, where the incoming statelet arrived, to a node which touches the incoming statelet's broadcast route. This route is stored in the route field of the incoming statelet. The goal of the DPCD protocol is to generate cycles for preconfiguration by evaluating the statelet broadcast patterns which originate and end at the cycler node. In order that only simple cycles be generated (that is cycles which traverse each cycle node only once), the statelet broadcast at the tandem nodes is limited to nodes not previously touched by the incoming statelets broadcast route. The only case where an outgoing statelet may be broadcast to a node falling on the broadcast route is when a tandem node broadcasts to the cycler node. This is required so that the statelet broadcasts may form cycles by terminating on the cycler node. Refer to Figure 7.4 for an example of the valid spans a tandem node may broadcast an incoming statelet on.


A
F
G
B
E
C
D
Cycler Node
Tandem Node
Existing Statelet Broadcast Pattern
Permitted Outgoing Statelet
Invalid Outgoing Statelet

- An existing Statelet Broadcast travels from the Cycler Node A to Tandem Node E along route A-B-C-D-E
- The Incoming Statelet arriving at Tandem Node E can be Broadcast to Nodes A, F, or G. Nodes F and G are valid broadcasts as they are Tandem Nodes which have not previously been touched by the Statelet Broadcast Pattern. Node A is a valid broadcast because it is the Cycler Node.
- Nodes B and C are not valid as they have previously relayed the Statelet Broadcast.
- If an outgoing statelet was sent to the Cycler Node, the Statelet Broadcast Pattern would form a cycle following A-B-C-D-E-A.

FIGURE 7.4 An Example of the Tandem Node Statelet Broadcast Rule that Limits the Cycles Generated to Simple Cycles which visit a Cycle Node only once

In summary, a tandem node's statelet broadcast rules will broadcast each incoming statelet to the largest extent warranted by the statelet's evaluated score. If all spare links on a span are occupied by outgoing broadcast statelets and an incoming statelet, with a good score, is able to broadcast on the span but currently does not, then the incoming statelet can displace the outgoing statelet whose precursor statelet has the worst score of all precursors for the span. An incoming statelet may only form the precursor for an outgoing statelet whose destination is a node which has not been previously touched by the incoming statelet's statelet broadcast route. The single exclusion to this rule is that a outgoing statelet may be broadcast to the cycler node. An additional rule is that only a single outgoing statelet of a given index family may appear on a span. If multiple incoming statelets exist, which are of a common index family and are able to be broadcast on a span, then the statelet with the best score is broadcast. These rules result in the cycler node receiving incoming statelets whose statelet broadcast route traces out cycles which begin and terminate at the cycler node.

7.1.2.2 . Tandem Node Evaluation of a Statelet's Score

In the previous section a description of how a tandem node broadcasts incoming statelets on the basis of each statelet's evaluated score was given. In this section it will be described how an incoming statelet is assigned a score.

The DPCD protocol attempts to generate effective PC designs using cycles as PC elements. To generate good cycles a score is required which reflects the quality that makes a particular cycle effective at providing span failures with PC paths. The score is chosen to reflect the potential that an incoming statelet's broadcast route can form a PC cycle with a high ratio of number of effective PC restoration paths to number of spare links required to form the cycle. This score reflects how efficiently the cycle would use the spare links required to construct it to provide PC paths for span restoration.

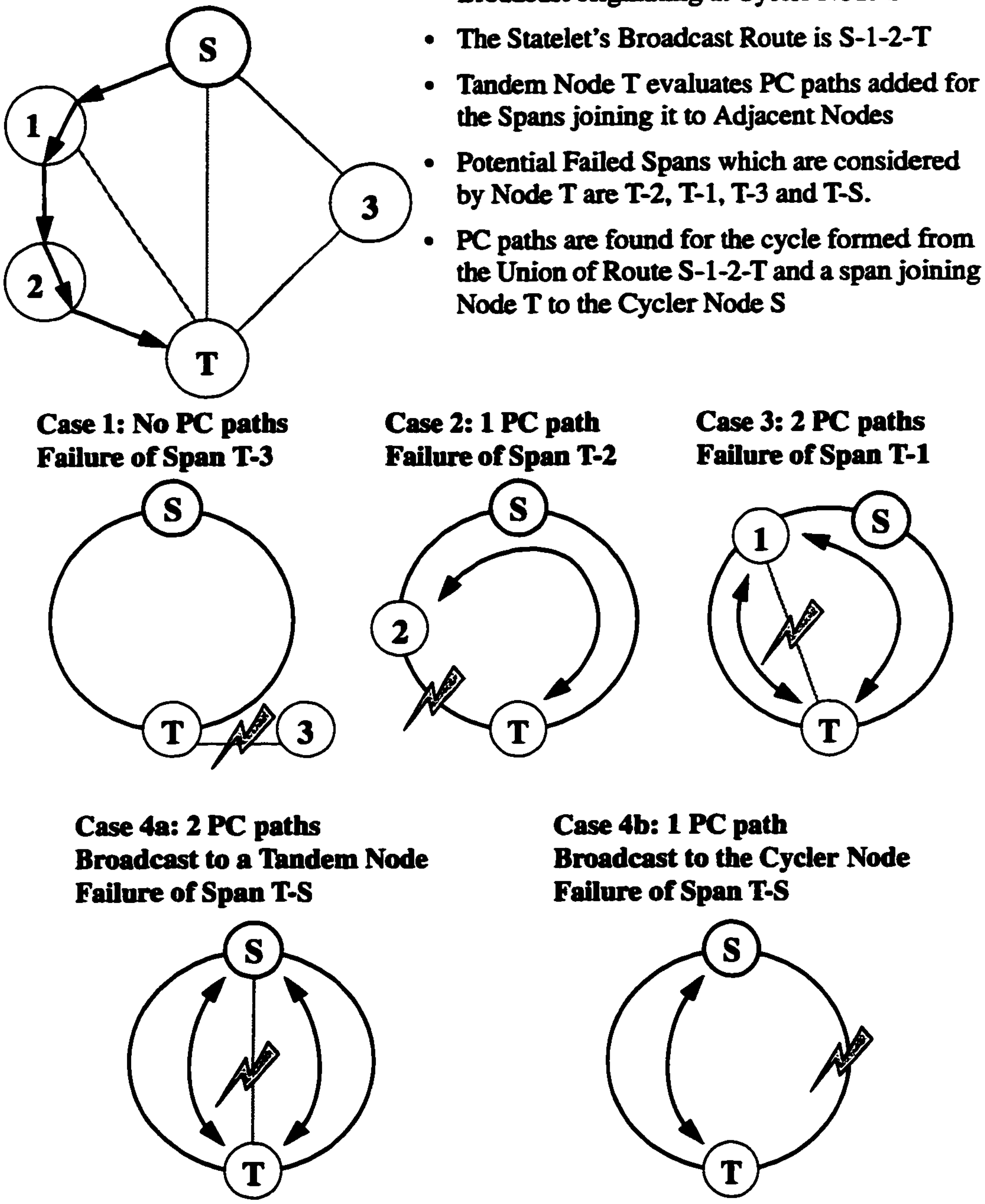
When an incoming statelet first arrives at a tandem node, the tandem node must first evaluate the statelet's score before it can attempt to broadcast the statelet. The statelet's score is equal to the ratio of the number of useful PC paths that the cycle, formed from a union of the incoming statelet's broadcast route and an imaginary span joining the tandem node and the cycler node, can provide and the number of spare links which would be required to form the statelet's broadcast route. The number of useful PC paths contained in the cycle are only counted for real network spans and are not considered for the imaginary span which closes the cycle (unless the imaginary span should coincide with a real span.) The number of spare links required to form the broadcast route can also be viewed as the number of spare links required to form the cycle when not considering the imaginary span. This score measures the potential that an incoming statelet's broadcast route, as evaluated locally at the tandem node, would form an effective PC cycle if it should loop back and touch on the cycler node.

The count of the number of PC paths which a statelet broadcast route could provide, if it was shorted out to form a cycle, is updated incrementally as the statelet broadcast route is relayed from tandem node to tandem node. When a tandem node receives an incoming statelet it evaluates the number of PC paths which were added in the step between the incoming statelet's originating node and itself. This number is added to the total PC paths, as evaluated at the originating node, which is contained in the incoming statelets numPaths field, to form a local numPaths value for the incoming statelet. The local numPaths number is divided by the value of the incoming statelet's hopcount field to generate the incoming statelet's score. If the incoming statelet should form the precursor to an outgoing statelet, the outgoing statelet's numPaths field will be set equal to the incoming statelet's evaluated local numPaths.

The PC path increment required to form the local path count for an incoming statelet is evaluated by examination of the statelet's route field. If the route touched on any of the nodes which are adjacent to the tandem node then the incoming statelet's route could provide PC paths for the spans which connect the tandem node to adjacent nodes. The rule used to evaluate the number of PC paths which are provided to a span connecting the tandem node to an adjacent span is quite simple. If a span connects the tandem node to a node, which is present in the route field, then the statelet broadcast route could provide one or two preconfigured paths; if the node is not present, the route can provide no paths. If the node is present, the route can provide only a single PC path if the adjacent span is that on which the incoming statelet, belonging to the broadcast route, arrives. The route can provide two PC paths if the span connects the tandem node to an adjacent node which is on the route but is not directly behind the tandem node on the route. The case where no

PC paths are provided, occurs because there is no path possible in a cycle for a pair of nodes which are not both a part of the cycle. The case, where a single PC path is available, occurs when a span, which connects a pair of cycle nodes directly along the cycle, fails because the only path possible is that which connects the pair of nodes around the failed span. The case, where two PC paths are available, occurs when a span, which connects a pair of cycle nodes but is not directly on the cycle, fails because there are two paths possible between any pair of nodes on a cycle, and, in this case, the cycle is not broken by the failure of a span. Therefore, unlike in the previous case, both paths can be exploited to aid in restoring the span failure. It should be noted that the total number of PC paths provided by a statelet broadcast route is not what is calculated but the total number of useful PC paths. If an incoming statelet's broadcast route is evaluated and it is found to overprovide a span with PC paths, either because the span has very few working links or the span's working links have been provided PC paths by previously formed cycles, then only the number of paths which are useful to restoring a span are counted. It can be determined if a PC path would be useful in restoration because each node maintains a list of the number of uncovered working links on the spans which join it to other nodes and this list is updated as PC cycles are formed in the network.

The method used in the previous paragraph to evaluate the total PC path count for an incoming statelet is valid only when considering the incoming statelet for broadcast to another tandem node. If a statelet is being considered for broadcast on a span which connects the broadcasting tandem node to the cycler node then the total PC count must be evaluated in a slightly different manner. The rules given in the previous paragraph are valid except that the number of PC paths which are available for a span, which has the cycler as an end node, can be at most one. This is because when an outgoing statelet is broadcast to the cycler node, it is closing the statelet broadcast route to form a cycle. The span connecting the broadcasting tandem node to the cycler node will fall directly beneath the cycle formed by the broadcast route and a cycle can only provide only a single PC path to the failure of such a span. This difference requires, in effect, that a tandem node evaluate two different scores for each incoming statelet: a score to be used for broadcast to other tandem node, and a score to be used for broadcast to the cycler node. Both scores are still



- An Incoming Statelet arrives at Node T with Broadcast originating at Cycler Node S
- The Statelet's Broadcast Route is S-1-2-T
- Tandem Node T evaluates PC paths added for the Spans joining it to Adjacent Nodes
- Potential Failed Spans which are considered by Node T are T-2, T-1, T-3 and T-S.
- PC paths are found for the cycle formed from the Union of Route S-1-2-T and a span joining Node T to the Cycler Node S

Case 1: No PC paths
Failure of Span T-3

Case 2: 1 PC path
Failure of Span T-2

Case 3: 2 PC paths
Failure of Span T-1

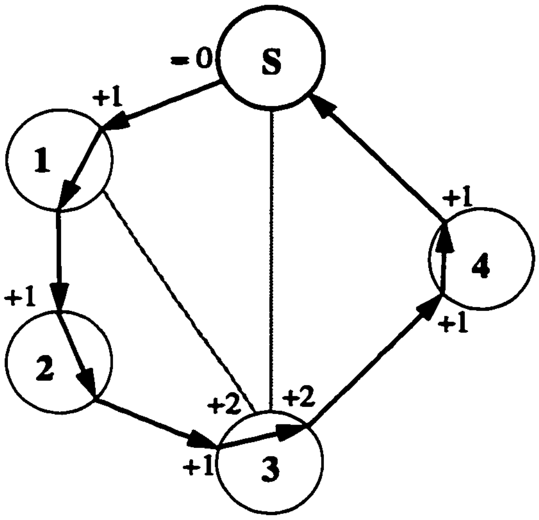
Case 4a: 2 PC paths
Broadcast to a Tandem Node
Failure of Span T-S

Case 4b: 1 PC path
Broadcast to the Cycler Node
Failure of Span T-S

Added PC paths for Broadcast to a Tandem Node = 0 + 1 +2 + 2 = 5
Added PC paths for Broadcast to the Cycler Node = 0 + 1 +2 + 1 = 5

FIGURE 7.5 Tandem Node evaluation of the Incremental Preconfigured Path Count of an Incoming Statelet

calculated by taking the ratio between the total PC paths, as calculated locally, to the value of the incoming statelet's hopcount field. Refer to Figures 7.5 and 7.6 and for examples of how the total PC path count is evaluated and updated at successive tandem nodes along a statelet broadcast route.



- A Statelet Broadcast which reach the Cycler with Route S-1-2-3-4-S
- As each Tandem Node receives the Statelet Broadcast it increments the Total Preconfigured Path Count
- The Total Number of Preconfigured Paths is equal to 9
- 1 PC path is available for the failure of a cycle span
- 2 PC paths is available for a non-cycle span whose end nodes are cycle nodes

Total PC paths for Cycle S-1-2-3-4-S = 0 + 1 + 1 + (1+2+2) + 1 + 1 = 9

FIGURE 7.6 Evaluation of the Total PC path Count by the Tandem Nodes along a Statelet Broadcast

7.1.2.3 . Cycler Node Initiation of Statelet Broadcast

All statelet broadcasts originate at the cycler node. To initiate statelet broadcast, the cycler node places a single outgoing statelet on an available spare link on each span falling on the cycler node. Each outgoing statelet is assigned a unique index. If a span does not have an available spare link, either because the span has no spare links or because the span's spare links have been previously used to form PC cycles, then no outgoing statelet is placed on it. Each outgoing statelet has its sendNode field set to the name of the cycler node, its hopcount field set to a value of 1 hop, and its numPaths field set to zero. Also, each outgoing statelet has all the cells in its route field set to NIL except for the contents of the cell, corresponding to the node which receives the outgoing statelet, which is set to the name of the cycler node.

After initiation of statelet broadcast the cycler node initiates the sampling of incoming statelets.

7.1.2.4 . Cycler Node Sampling of Incoming Statelets

After the statelet broadcast is initiated, the cycler node waits and samples the incoming statelets arriving on its spare links. Each successful arrival of an incoming statelet represents a cycle with a score attached to it. The score for each incoming statelet is found by taking the ratio of the incoming statelet's numPaths field to that of its hopcount field. The value is equal to the number of useful PC paths which the cycle can provide per spare link which would be required for its construction. At the start of statelet sampling the cycle resets a timer and then waits. As new statelet events arise on the incoming sides of the cycler node's spare links, it examines the new incoming statelet and determines its score. If the cycle represented by the new statelet has a score which better the scores of all previously received statelets, the cycler node records the cycle and its score and resets the timer. When the timer times out, the cycler node will initiate the building of the best non-zero scoring cycle which it has received. If the cycler has, either, received incoming statelets which have a score of zero or has received no incoming statelets at all, the cycler will not initiate the building of a cycle. Instead, it will terminate its role as the cycler node and signal another predetermined node to assume the role of the cycler node.

This sampling of incoming statelets is markedly different from the way received incoming statelets are treated by the chooser node in the SHN protocol. Immediately upon reception of an incoming statelet, the chooser node will react by initiating reverse linking, back along the incoming statelet's broadcast path, to form a restoration path between the sender and chooser nodes. The reason for this is that a statelet broadcast in the DPCD protocol will tend to improve in its score as it progresses unlike in the SHN where a statelet broadcast will tend to worsen in score. This difference is a consequence of the different goals of the two protocols. The SHN uses statelet broadcasts to represent node to node paths, between the sender and chooser nodes, and, in general, a shorter path makes more efficient use of network spare capacity than a longer path and is more reflective of the real time pressure in restoration. Therefore, a SHN statelet broadcast's score will worsen as it progresses and becomes longer. As shown in the previous section, a cycle tends to contain more PC paths as it grows in size and, therefore, the score of a DPCD statelet broadcast, which represents a cycle, will improve as it increases in size. This means that reacting immediately to an incoming statelet in the DPCD protocol will, in general, give a cycle which does not have very good score. Sampling the incoming statelets forces the cycler node to give the generation of incoming statelets, with better scoring broadcast routes, a chance to occur. The DPCD has the luxury of permitting the sampling of signatures because it is being run in anticipation of a span failure, and not in response to one, and so does not require the fast real time response present in the SHN.

7.1.2.5 . Cycler Node Initiation of the Building of a Cycle

If the cycler node has found a non-zero scoring cycle in the statelet sampling stage, it will initiate the building of a single copy of the cycle. As mentioned in the previous section, a statelet broadcast in the DPCD protocol will improve in score as it progresses. A consequence of this is that there is no guarantee that the actual link level path, that a incoming statelet's broadcast path traversed to reach the cycler node, will still be occupied by the statelet's broadcast. Because a statelet broadcast's score improves as it progresses there is nothing to protect its root from being displaced by other better scoring statelet broadcasts in the tandem node broadcasts rules. At each tandem node, a statelet broadcast builds upon the statelet broadcasts preceding it at other tandem nodes. The relatively low scoring portion of the broadcast will occur earlier in the broadcast route and can be displaced by the arrival of new higher scoring incoming statelets at the preceding tandem nodes. A consequence of this is that the network wide statelet broadcast pattern will not eventually grow static, unlike in the SHN protocol, but will continue to fluctuate as the roots of broadcast patterns are continuously displaced by the arrival of better scoring statelets. However, in the previous sampling stage, the cycler samples the incoming statelets and maintains a record of the best scoring cycle. The record contains only the route the cycle's statelet broadcast travelled to reach the cycler node, not the actual path, but this is all that is required to build a single copy of the cycle. It is guaranteed that the spare capacity required to construct a copy of the cycle will be available because the cycle's statelet broadcast route was only broadcast on unoccupied spare links.

The first step in initiating the construction of a cycle, is that the cycler node cancels all outgoing statelets. This causes a termination of the statelet broadcasts as the cycler node is the source of all statelet broadcasts within the network. In effect, although the statelet broadcasts would continue to fluctuate if left to proceed, the cycler node brings the statelet broadcasts to a conclusion. Next, the cycler examines the route field which was stored for the cycle. The cell corresponding to the cycler node will contain the name of the tandem node which preceded the cycler node on the broadcast route. The cycler node will then scan for the first unoccupied spare link on the span joining the cycler node to the node indicated by the route field. The cycler node will then place an outgoing statelet, on this spare link, with a copy of the cycle's route field and a special reserved index family which indicates the outgoing statelet is not a normal statelet broadcast but a cycle construction request.

The tandem node, upon reception of the cycle construction statelet from the cycler node, will cancel all outgoing statelet broadcasts. Then it will examine the route field on the construction request statelet for the name of the tandem node which preceded it on the cycle's original broadcast route. It will then locate the first unoccupied spare link on the span joining it to the node which preceded it on the broadcast route. Next, an outgoing construction request statelet will be placed on this link, instructing the preceding route node to continue construction of the cycle. The tandem node's local list of uncovered span working links will be updated, to reflect the PC paths which are available in the newly formed cycle, so that future iterations of statelet broadcasts can be scored accurately. Also, the tandem node will form a crossconnection between the link on which the incoming construction request statelet arrived and the link on which it sent the outgoing construction request statelet. The preceding tandem nodes on the broadcast route will react in a similar manner, continuing the construction of the cycle.

Eventually, the construction of the preconfigured cycle will loop back to the cycler node and an incoming cycle construction statelet will arrive on a spare link. The cycler node then forms a crossconnection between this spare link and the link on which it originally placed the outgoing statelet which initiated the cycle construction thus completing the cycle. The cycler node also adjust its local list of uncovered working links in a similar manner to that of the tandem nodes. Finally, the cycler node cancels all outgoing statelets, and initiates a new statelet broadcast, as described in section 7.1.2.3.

7.1.3 Global Execution of the DPCD Protocol

The previous section gave a network node level description of the DPCD protocol. In this section a network level representation of the protocol's execution will be given. The DPCD protocol is an example of a greedy algorithm. A greedy algorithm is an algorithm which iteratively constructs a solution to a problem on the basis of the state of the problem at the start of the iteration. The state of the problem is modified, after each iteration, by the application of the iteratively generated solution. The DPCD protocol can be seen to be greedy because it iteratively generates a single cycle per iteration and the cycle is formed on the basis of the current network configuration as represented by the number of unoccupied spare links and the number of uncovered working links present in the network spans. As the cycle is generated these network values are modified by the presence of PC paths in the newly generated cycle and the utilization of spare links needed to construct the cycle.

Globally, the DPCD protocol executes in the following manner. Each node in the network takes a turn at being the cycler node in an order which is predetermined and is stored locally within the nodes. As each node assumes the role of the cycler node, it iteratively generates cycles, using the rules outlined in the previous section, until it can either no longer generate a cycle or the cycles that it can generate all receive a zero score. At this point the current cycler node gives up the role and signals the next node in the series to become the cycler node. The new cycler node generates as many cycles as it can until it too is no longer able to generate useful cycles. The preconfigured cycles generated by each node alter the network's configuration as it is seen by later cycler nodes. The role of the cycler is successively assumed by all the network nodes and when the last node has terminated its role as the cycler, preconfigured cycle generation stops.

The execution time of this protocol will be many times longer than that of the SHN. This is because a statelet broadcast is required for each generated PC cycle and because of the timer delays introduced by the cycler node's sampling of its incoming statelets. However, because the DPCD protocol is being run in anticipation of a span failure event, instead of in reaction to one like the SHN, it has the luxury of being able to have a relatively long execution time.

The order in which each node assumes the role of the cycler is important because the cycler node can only generate cycles of which it is a part. Depending on the current network configuration, choosing a certain node to assume the role of the cycler node could cause the generation of cycles which are wasteful in the use of the network spare capacity. A cycler node, on which spans fall which require minimal restoration paths, could result in cycles which are larger than required or which cover spans which do not require many PC paths.

The node order with which this algorithm was used was determined by the total number of working links falling on each network node. Each network node had a total generated of the number of working links contained in the spans which fell on it. The order in which the nodes assumed the role of the cycler was generated by sorting the nodes in descending order of the calculated working links total. The reverse order was also run to evaluate any performance difference between the two orders. The node's working link total was used to determine the cycler node ordering because it seemed reasonable that if a node terminates a large number of working links then it should receive an early opportunity to form a part of the PC cycles generated.

7.2 Simulation Method

As an initial evaluation of the potential of realizing an algorithm, which could generate preconfigured cycle designs in a distributed manner, a simple simulation approach was taken. The distributed protocol was simulated in an iterative manner with the state of the current iteration being determined by only the state of the previous iteration. In each iteration, the outgoing statelet broadcast pattern, for each network node, is generated on the basis of the node's incoming statelet broadcast pattern in the previous iteration. Each node then assumes its newly calculated broadcast pattern simultaneously and in lockstep. Successive iterations are generated in a similar manner.

This simulation method approximates the case where a network's statelet insertion delay is large compared to the statelet processing delay. Insertion delay is due to the time required to transmit a statelet through a limited bandwidth communication channel while processing delay is due to the time required by the computation element present in each node to process incoming statelet events.

As an example of the iterative nature of the simulation, Figure 7.7 contains a plot of the score of the best received incoming statelet, by the current cycler node, versus the simulation iteration. In the plot, 4 distinct structures appear; each structure corresponds to the search and construction of a single PC cycle. At the left most edge of each structure, the plot can be seen to start at zero and then step up. Each step corresponds to the arrival of an incoming statelet with a superior score and an improvement to the best candidate cycle received by the cycler node. The structure then plateaus which corresponds to the cycler node's timer counting down (the timer was set to countdown 50 iterations.) When the timer counts out, the cycler initiates the construction of the cycle and resets the best received score to zero (this corresponds to the right edge of each structure.) Although, the graph goes up to only 300 iterations, the actual simulation continued for 500 more iterations. However, no more PC cycles were possible within the network after the construction of the fourth cycle and, so, the plot remains at zero after this point. Figure 7.8 shows an example of the successive displacement of the best received cycle candidate as a cycler node samples the incoming statelets generated by the evolving statelet broadcast. The statelet broadcast, from which the incoming signatures containing these cycles originate, is the broadcast represented by the left most structure in Figure 7.7. The arrival of each superior incoming signature, can be seen as a step in the left most edge of this structure. During the evolution of the statelet broadcast, 8 superior incoming statelets arrive at the cycler node (superior in the sense that their score is better than that of any previously received statelet.) Each cycle in Figure 7.8 corresponds to the cycle contained within a superior incoming statelet and the score above the cycle corresponds to the statelet's score. The cycles are ordered in the same order of arrival as the superior statelet to which each cycle corresponds. After the arrival of the 8th superior statelet, no more incoming statelets with better scores appear at the cycler. Eventually, the cycler's timer times out and the 8th cycle, as the best received cycle, is formed within the network's spare capacity.

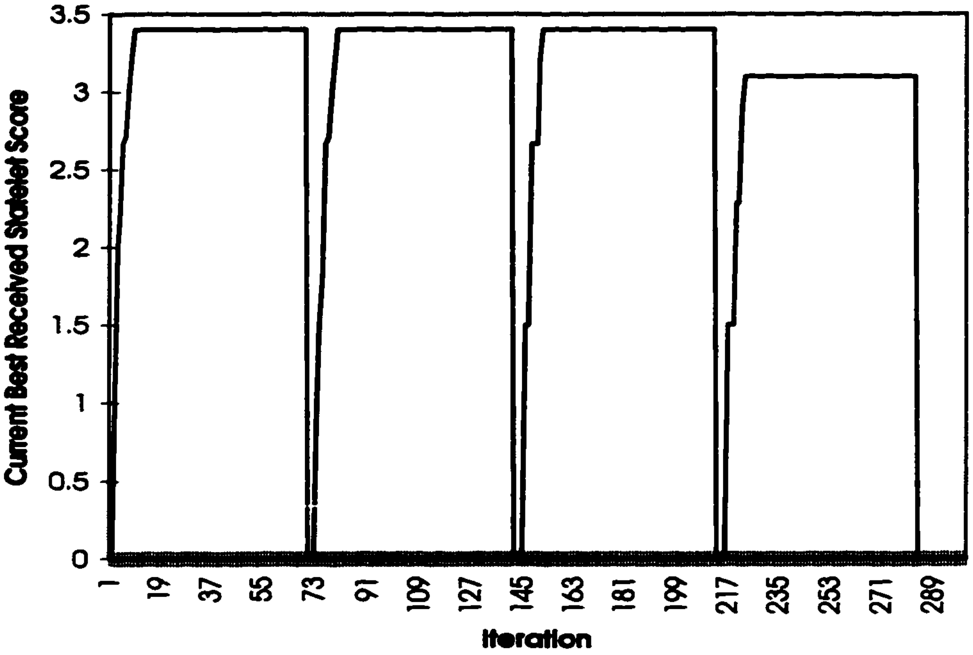


FIGURE 7.7 The Best Received Incoming Statelet Score at the Cycler Node versus Simulation Iteration when using DPCD with Node Order in Order of Decreasing Working Links in Net1

7.3 Results

Please refer to figures 7.9 and 7.10 for an example of the type of PC pattern that is generated in Net1 by the DPCD, with cycler node order sorted in order of decreasing total node working links and increasing total node working links, respectively. Table 7.1 contains the network restorability results, for the decreasing order case, over all possible span failures for restoration using only conventional KSP restoration, restoration using only pure PC paths, and 2-step restoration. Table 7.3 contains the restorability results for the increasing order case. Table 7.2 contains the total crossconnection events, for the decreasing order case, over all possible failed spans, for KSP restoration alone and for 2-step restoration. The crossconnection results for the increasing order case is presented in Table 7.4.

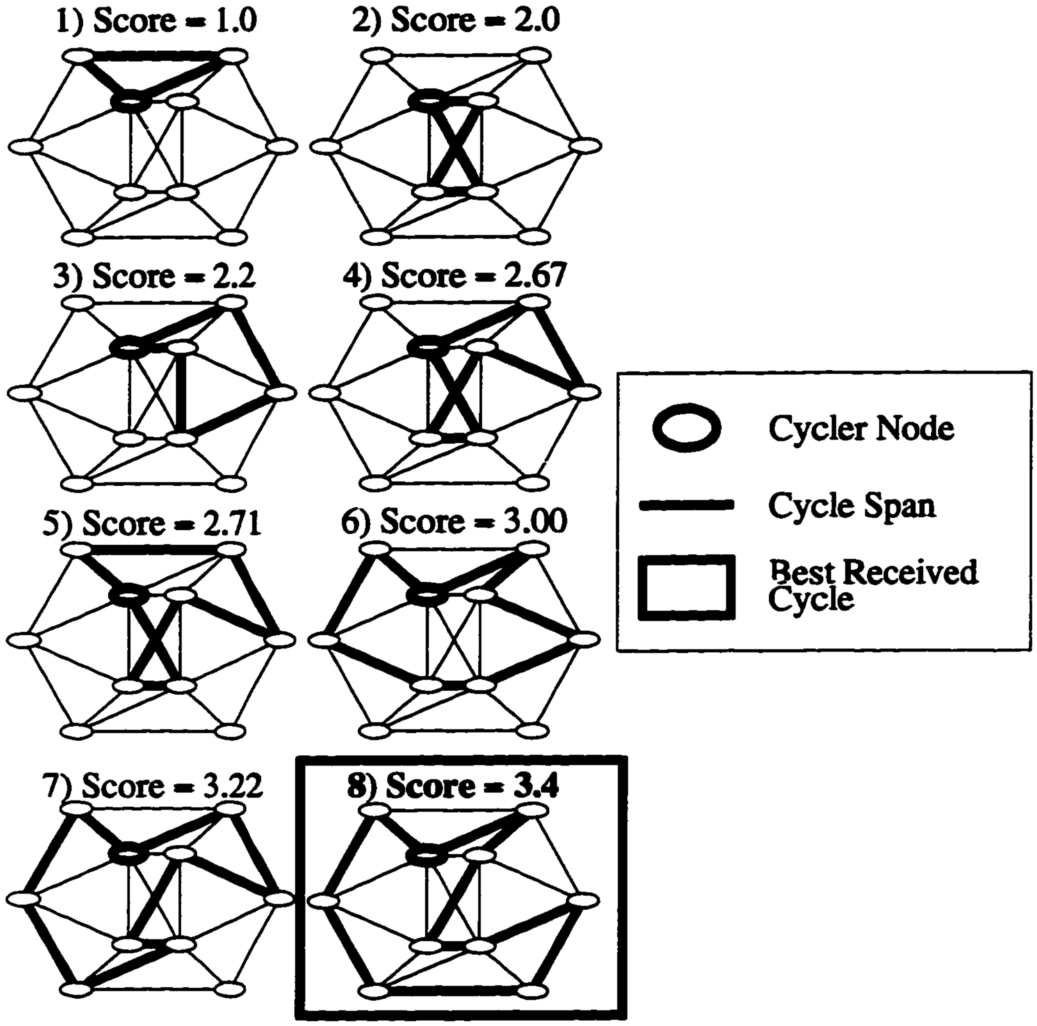


FIGURE 7.8 The Successive Displacement of the Best Scoring Cycle Candidate contained in the Received Incoming Statelets at the Cycler Node. Done for Net1 with Nodes in Decreasing Working Link Order, for the First Statelet Broadcast

Figures 7.11 and 7.12 contain examples of the PC patterns generated in Net1 by the DPCD, for decreasing and increasing cycler order, when run within the network sparing plans generated by the 100% PC cycle restorable IP method (IP2-Cycle.) Tables 7.5 and 7.7 contain the restorability results for the two orders when using IP2-cycle with the test networks. Tables 7.6 and 7.8 contain the corresponding crossconnection results.

In addition, figures 7.13-7.20 give counts of the number of crossconnection events required at each node to form a restoration pathset, for each span failure, in test network

Net2, when using only on-demand KSP restoration and when using 2-step restoration. The figures present the results corresponding to the original sparing and the sparing generated by IP2-Cycle for both decreasing and increasing total working link cycler ordering. The crossconnect totals for each node appear sequentially within the space, on the x-axis, corresponding to each span failure.

The countdown timer, used by the cycler node when sampling incoming statelets, was set to time out after 50 iterations in all test networks.

Table 7.1: Network Restorability using Standard KSP and PC Restoration using the DPCD with Cycler Order in Descending Total Node Working Links Order

Network	KSP Restorability (%)	PC Restorability (%)	2-step Restorability (%)
Net1	100	93.66	93.66
Net2	96.87	95.01	99.43
Net3	100	81.73	94.53
Net4	100	85.12	94.93
Net5	100	92.20	99.50

Table 7.2: Total Network Crossconnect Events Results using Standard KSP and PC Restoration using the DPCD with Cycler Order in Descending Total Node Working Links Order

Network	Xpts Closed KSP	Xpts Closed PC	Xpts Opened PC	Total Xpt Events PC
Net1	310	0	174	174
Net2	3984	122	2220	2342
Net3	13787	1917	7011	8928
Net4	84389	8269	43277	51546
Net5	8529	455	3985	4440

Table 7.3: Network Restorability using Standard KSP and PC Restoration using the DPCD with Cycler Order in Increasing Total Node Working Links Order

Network	KSP Restorability (%)	PC Restorability (%)	2-step Restorability (%)
Net1	100	84.51	84.51
Net2	96.87	85.33	96.87
Net3	100	82.86	96.29
Net4	100	84.91	94.57
Net5	100	86.13	98.86

Table 7.4: Total Network Crossconnect Events Results using Standard KSP and PC Restoration using the DPCD with Cycler Order in Increasing Total Node Working Links Order

Network	Xpts Closed KSP	Xpts Closed PC	Xpts Opened PC	Total Xpt Events PC
Net1	310	0	162	162
Net2	3984	420	2278	2698
Net3	13787	1666	6595	8261
Net4	84389	7812	44074	51886
Net5	8529	1074	4190	5264

Table 7.5: Network Restorability using Standard KSP and PC Restoration using the DPCD with Cycler Order in Descending Total Node Working Links Order with the Sparing Plans generated by IP2-Cycle

Network	KSP Restorability (%)	PC Restorability (%)	2-step Restorability (%)
Net1	100	94.37	97.18
Net2	100	99.15	100
Net3	100	90.02	97.12
Net4	100	96.84	99.00
Net5	100	92.42	96.44

Table 7.6: Total Network Crossconnect Events Results using Standard KSP and PC Restoration using the DPCD with Cycler Order in Descending Total Node Working Links Order with the Sparing Plans generated by IP2-Cycle

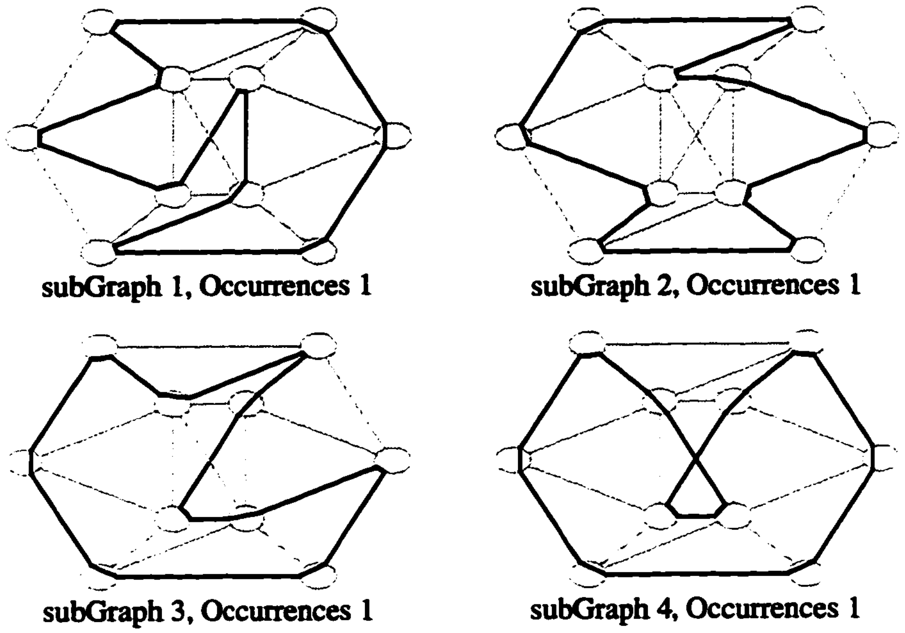
Network	Xpts Closed KSP	Xpts Closed PC	Xpts Opened PC	Total Xpt Events PC
Net1	291	9	203	212
Net2	4176	20	2080	2100
Net3	20744	1074	6783	7857
Net4	86048	1762	42292	44054
Net5	13727	382	3828	4210

Table 7.7: Network Restorability using Standard KSP and PC Restoration using the DPCD with Cycler Order in Increasing Total Node Working Links Order with the Sparing Plans generated by IP2-Cycle

Network	KSP Restorability (%)	PC Restorability (%)	2-step Restorability (%)
Net1	100	83.10	95.07
Net2	100	98.72	100
Net3	100	94.44	97.94
Net4	100	96.23	98.22
Net5	100	86.86	97.49

Table 7.8: Total Network Crossconnect Events Results using Standard KSP and PC Restoration using the DPCD with Cycler Order in Increasing Total Node Working Links Order with the Sparing Plans generated by IP2-Cycle

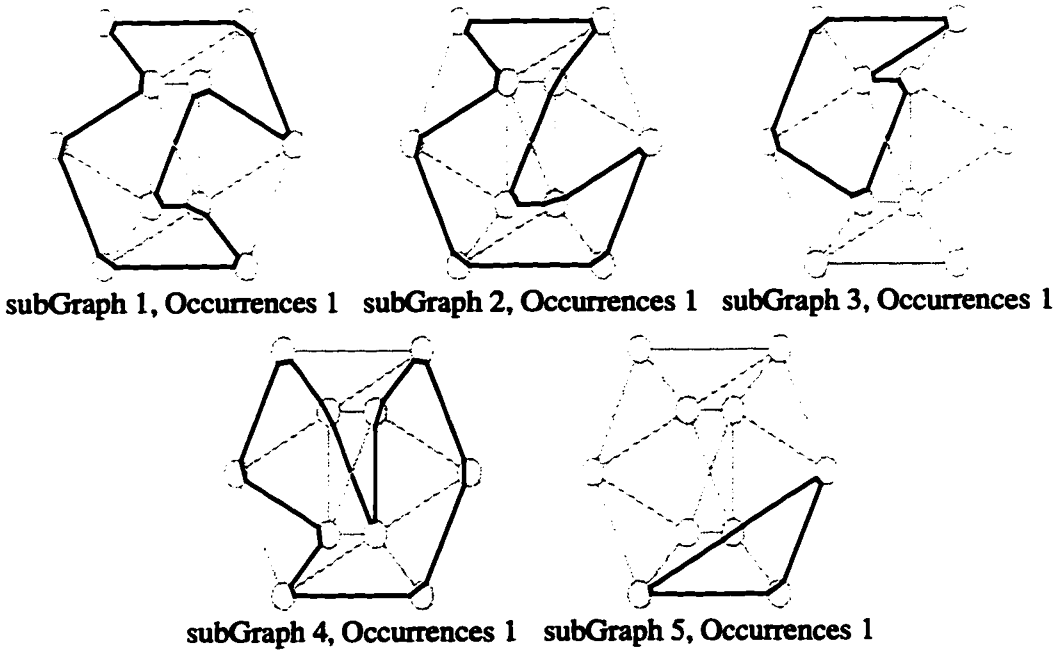
Network	Xpts Closed KSP	Xpts Closed PC	Xpts Opened PC	Total Xpt Events PC
Net1	291	44	199	243
Net2	4176	32	2120	2152
Net3	20744	558	6836	7394
Net4	86048	2222	42020	44242
Net5	13727	2044	4757	6801



subGraph 1, Occurrences 1 subGraph 2, Occurrences 1

subGraph 3, Occurrences 1 subGraph 4, Occurrences 1

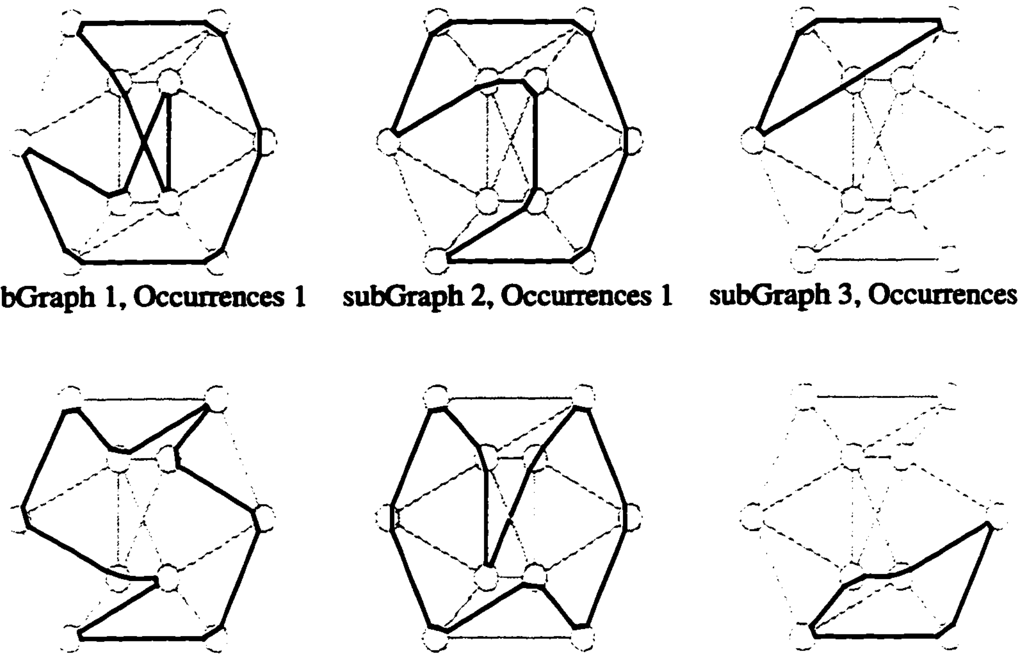
FIGURE 7.9 Example of the Patterns which Result in Net1 using the DPCD with Cycler Order in Descending Node Total Working Links



subGraph 1, Occurrences 1 subGraph 2, Occurrences 1 subGraph 3, Occurrences 1

subGraph 4, Occurrences 1 subGraph 5, Occurrences 1

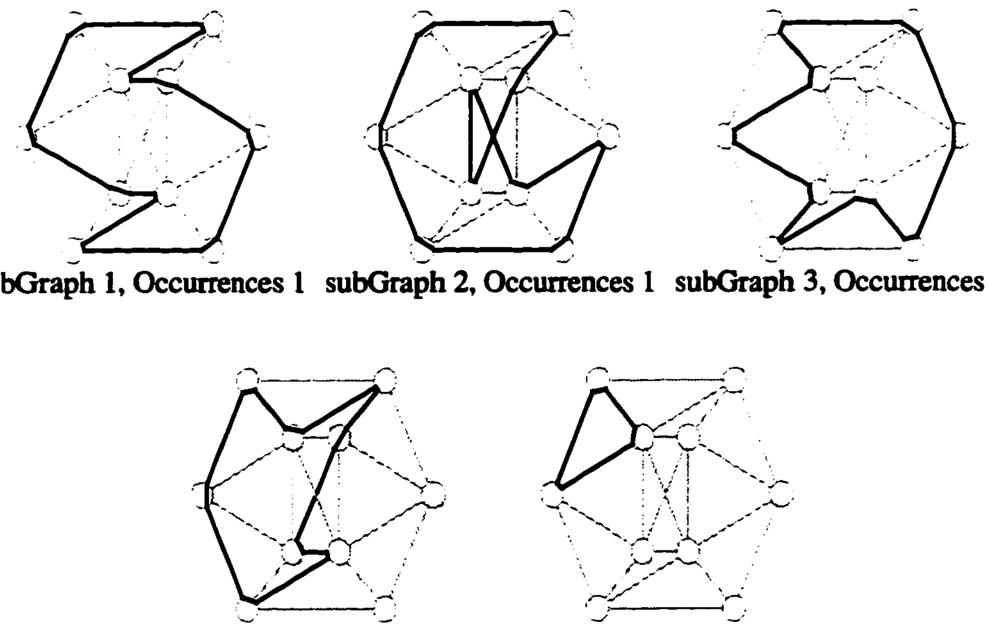
FIGURE 7.10 Example of the Patterns which Result in Net1 using the DPCD with Cycler Order in Increasing Node Total Working Links



subGraph 1, Occurrences 1 subGraph 2, Occurrences 1 subGraph 3, Occurrences 1

subGraph 4, Occurrences 1 subGraph 5, Occurrences 1 subGraph 6, Occurrences 1

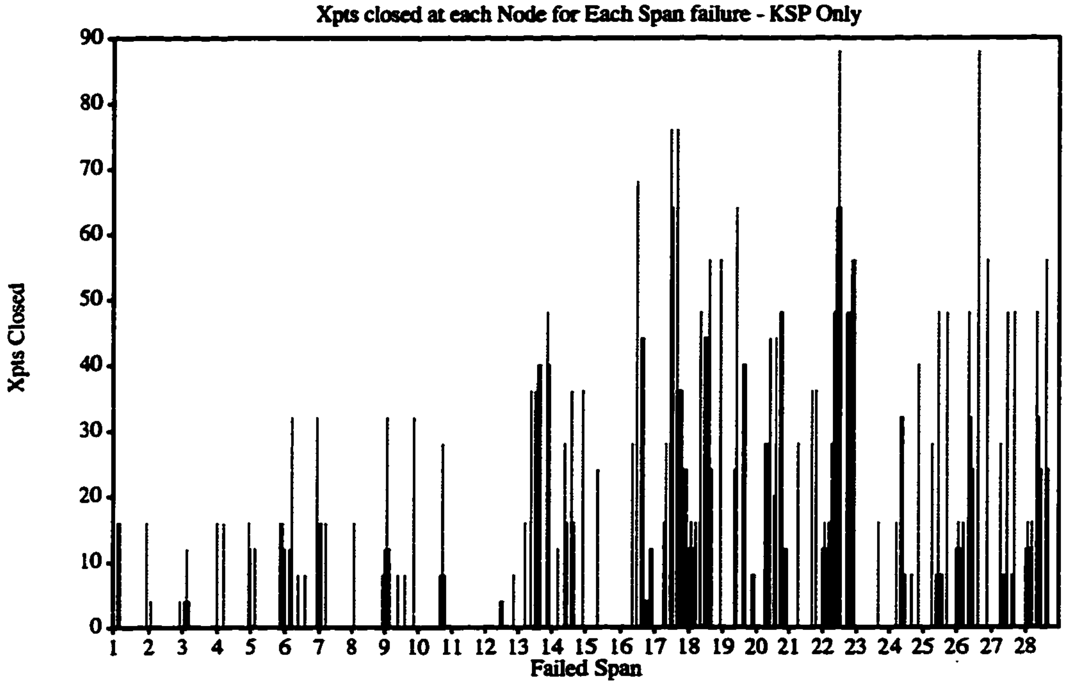
FIGURE 7.11 Example of the Patterns which Result in Net1 using the DPCD with Cycler Order in Descending Node Total Working Links with the Sparing Plans generated by IP2-Cycle



subGraph 1, Occurrences 1 subGraph 2, Occurrences 1 subGraph 3, Occurrences 1

subGraph 4, Occurrences 1 subGraph 5, Occurrences 1

FIGURE 7.12 Example of the Patterns which Result in Net1 using the DPCD with Cycler Order in Increasing Node Total Working Links with the Sparing Plans generated by IP2-Cycle



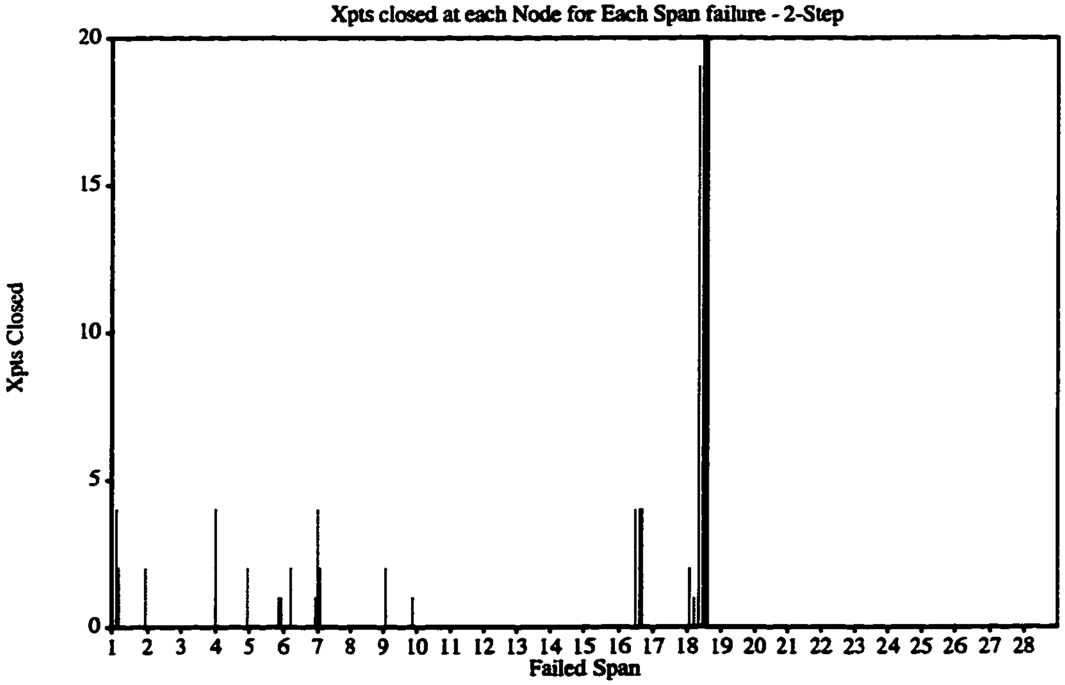


FIGURE 7.13 Crossconnections Closed at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using the DPCD with Cycler Nodes in Decreasing Total Working Link Order. For Restoration using only Conventional KSP Restoration and for 2-Step Restoration.

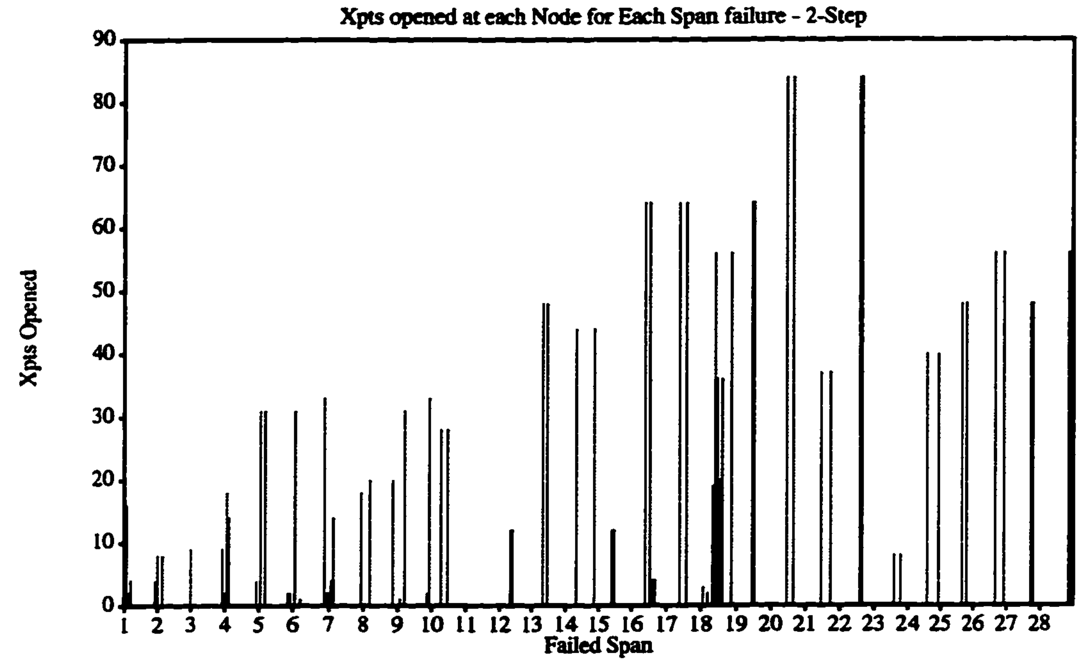
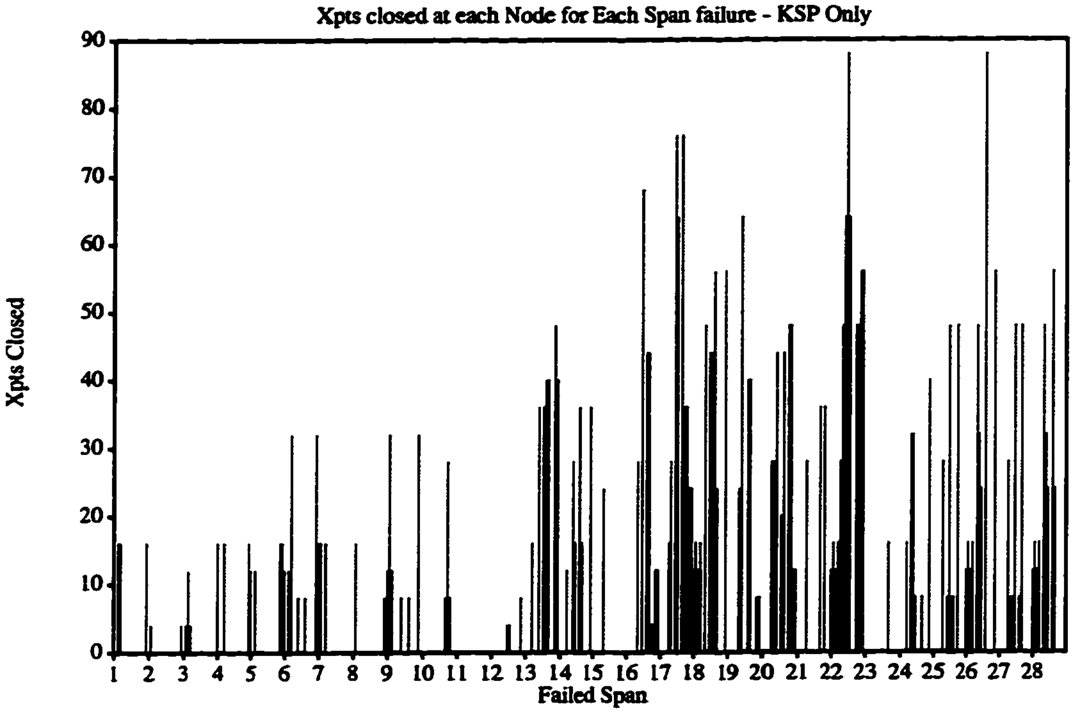


FIGURE 7.14 Crossconnections Opened at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using the DPCD with Cycler Nodes in Decreasing Total Working Link Order. For Restoration using 2-Step Restoration.

7.4 Conclusions

For the original network sparing plans, the performance of the distributed cycle generator at times approached that of the IP based PC cycle design (it equals the performance for Net1 using the descending ordering) but, as the distributed algorithm is based on a heuristic, there was no guarantee of optimal cycle generation. Although, the overall performance did not exactly equal that of the IP PC designs, the results were still quite good. Using the descending order, the distributed cycle generator generated PC restorability scores which ranged between 82% and 95%. The overall restorability using PC with leftover KSP restoration varied between 94% and close to 100%. Also, the crossconnect results showed the result of using a simple pattern as the PC plan's building block; the total number of crossconnection opens was quite low for all test networks. In addition, the total number of crossconnection closures was significantly lower for restoration using PC paths compared to KSP restoration alone, in all the test networks.



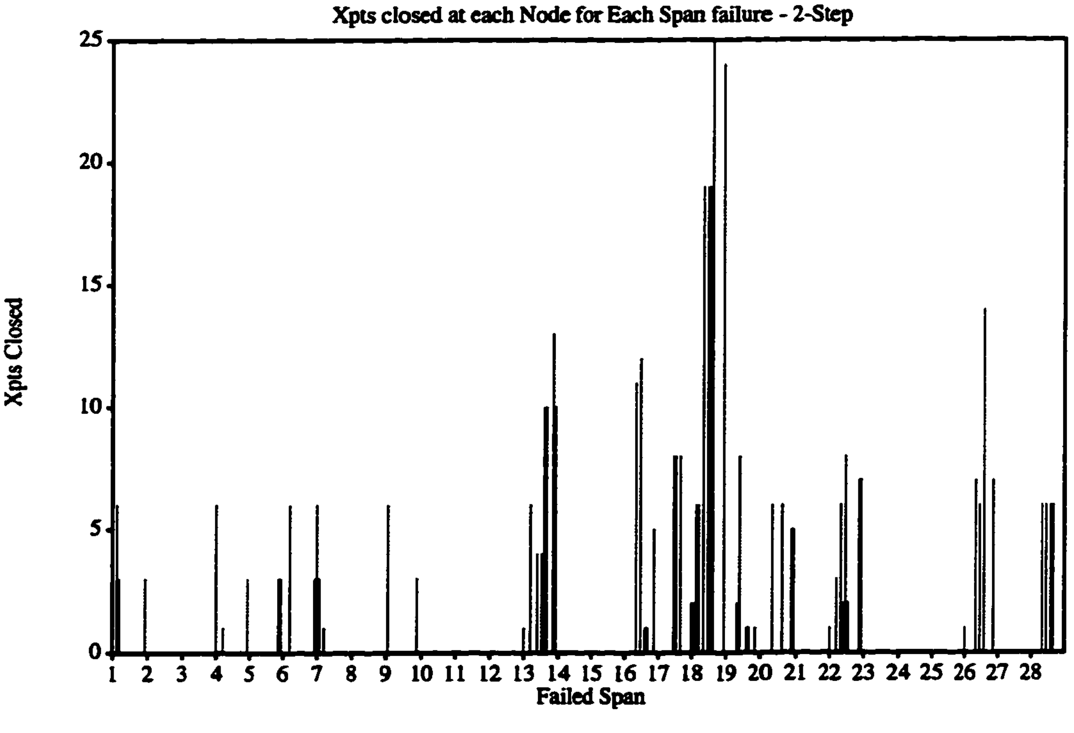


FIGURE 7.15 Crossconnections Closed at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using the DPCD with Cycler Nodes in Increasing Total Working Link Order. For Restoration using only Conventional KSP Restoration and for 2-Step Restoration.

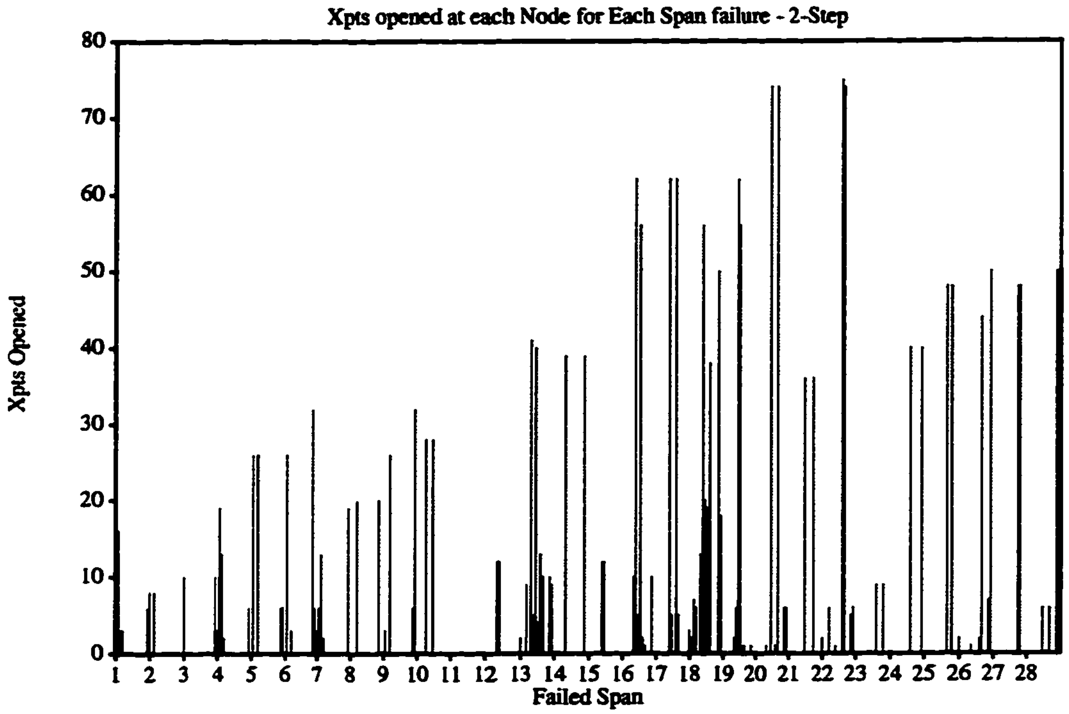
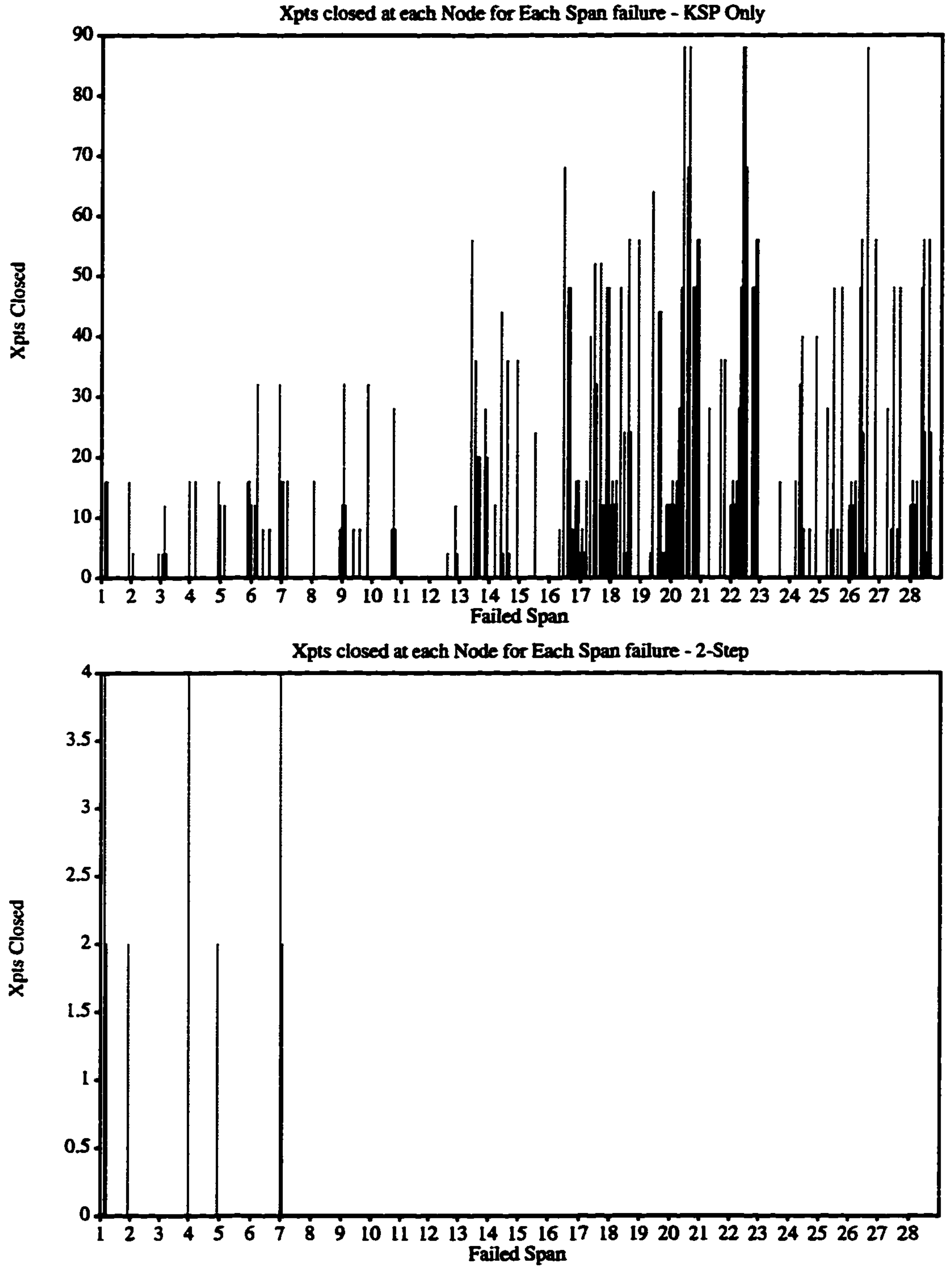


FIGURE 7.16 Crossconnections Opened at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 using the DPCD with Cycler Nodes in Increasing Total Working Link Order. For Restoration using 2-Step Restoration.

The performance of the DPCD, when run using the sparing generated for the test networks using IP2-Cycle, was in general better than when run using the original sparing, with improvements in PC restorability, 2-step restorability and total crossconnection event numbers. However, it did not achieve, although it did approach, the performance of the original PC cycle plan which was generated by the IP.

As expected the performance of the descending order execution was, in general, better than that of the increasing order execution producing better PC restorability scores and PC plus leftover KSP restorability scores for 4 out of the 5 test networks when run in the test network's original sparing plans (decreasing order was better than increasing order in 3 of the 5 test networks when using the sparing plans generated using IP2-cycle.) The algorithms dependency on the order in which the network nodes assume the role of the cycler nodes is shown by the variation in the two orders scores.


Xpts closed at each Node for Each Span failure - KSP Only

Xpts closed at each Node for Each Span failure - 2-Step

FIGURE 7.17 Crossconnections Closed at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 (as Spared using IP2-Cycle) using the DPCD with Cycler Nodes in Decreasing Total Working Link Order. For Restoration using only Conventional KSP Restoration and for 2-Step Restoration.

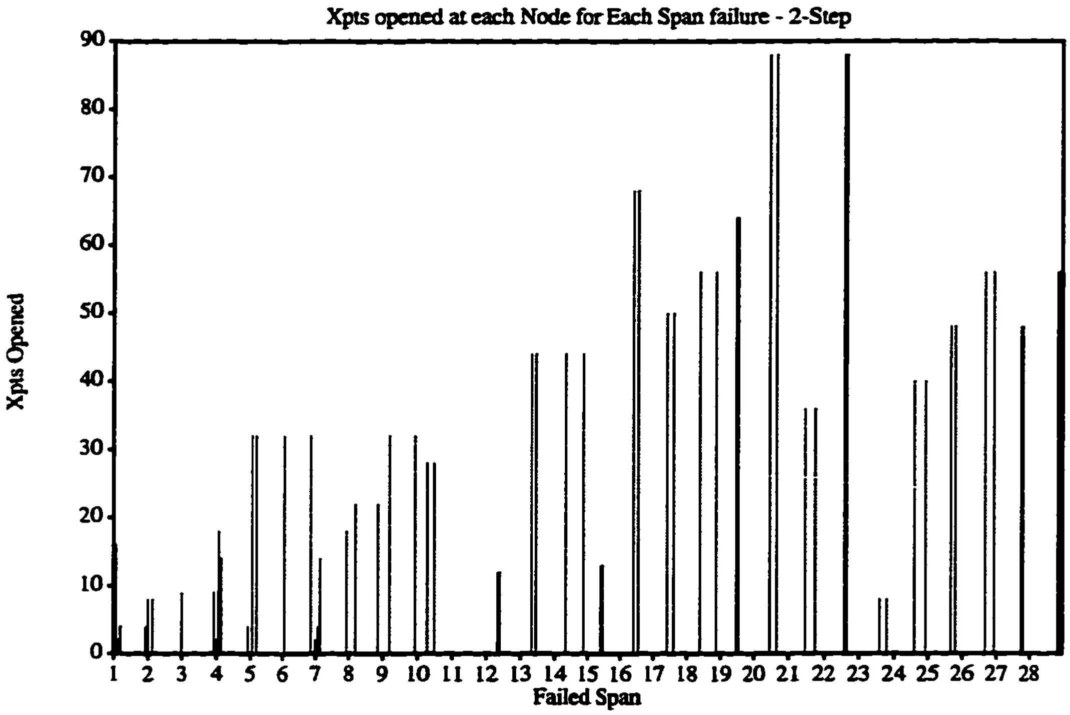
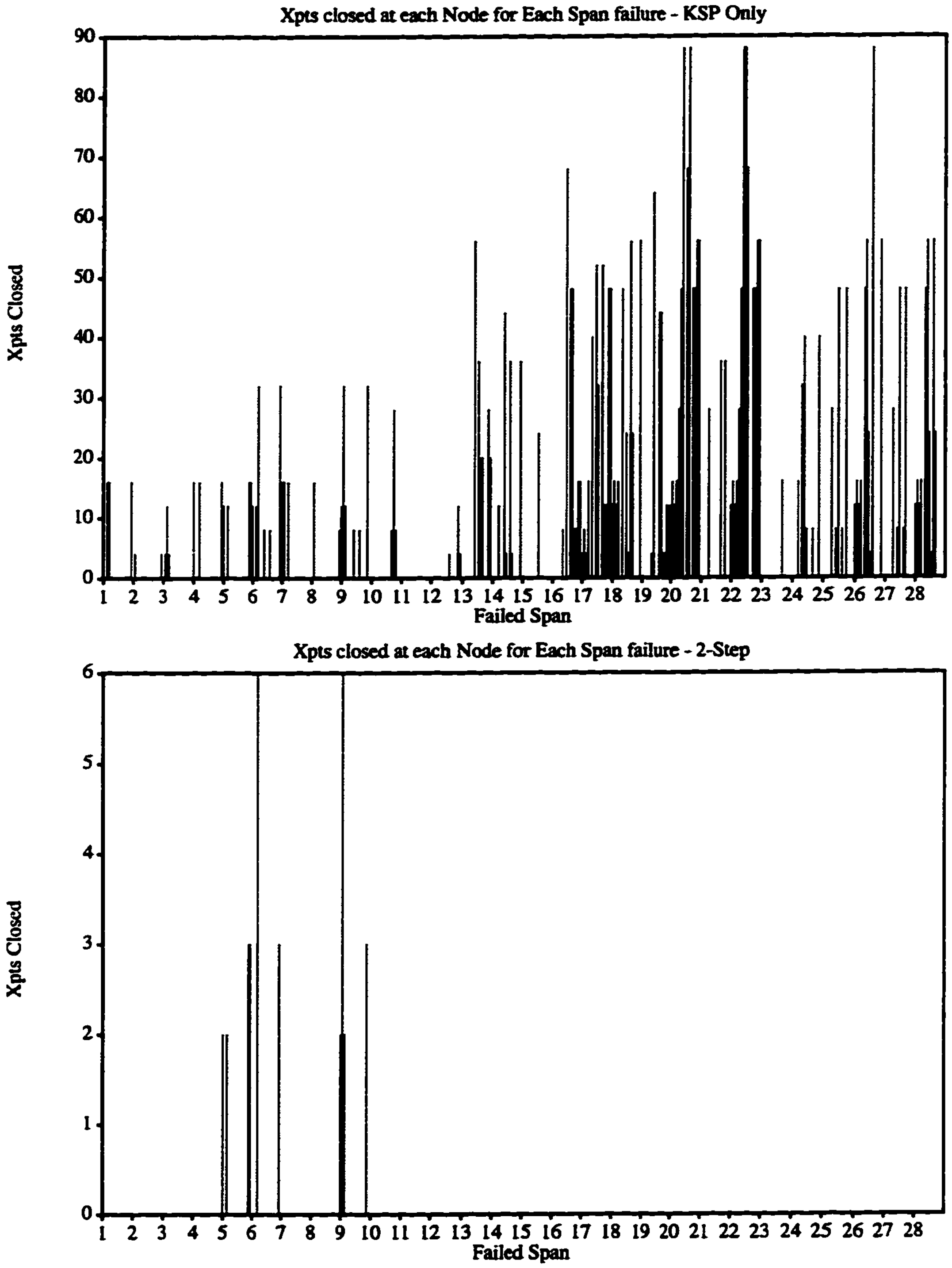


FIGURE 7.18 Crossconnections Opened at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 (as Spared using IP2-Cycle) using the DPCD with Cycler Nodes in Decreasing Total Working Link Order. For Restoration using 2-Step Restoration.



Xpts closed at each Node for Each Span failure - KSP Only

Xpts closed at each Node for Each Span failure - 2-Step

FIGURE 7.19 Crossconnections Closed at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 (as Spared using IP2-Cycle) using the DPCD with Cycler Nodes in Increasing Total Working Link Order. For Restoration using only Conventional KSP Restoration and for 2-Step Restoration.

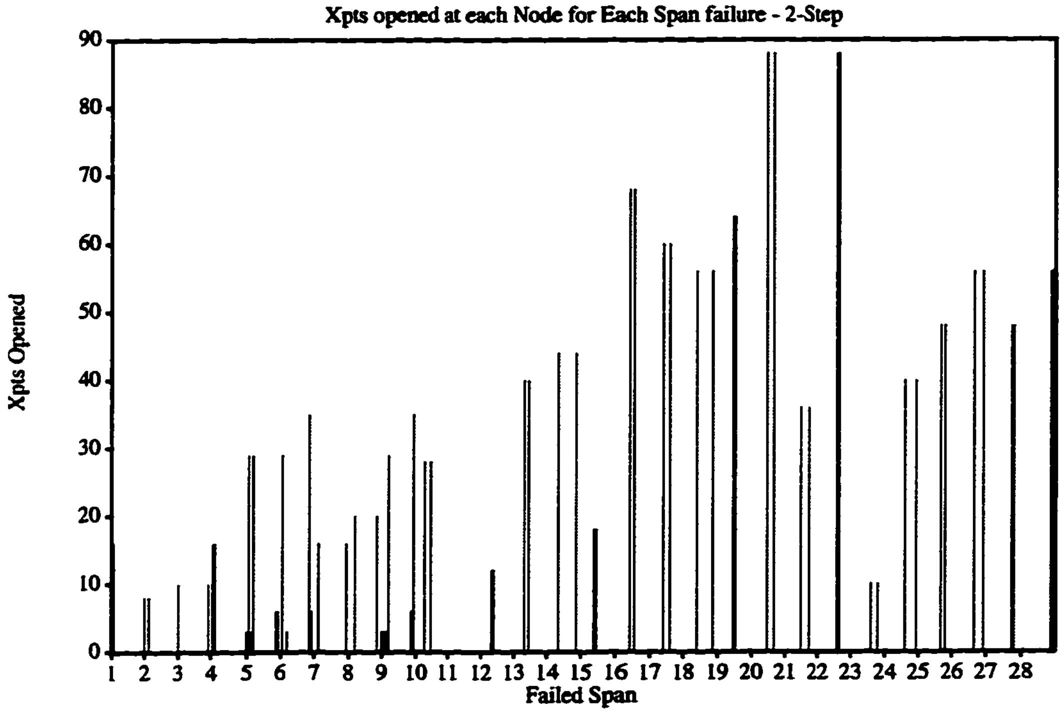


FIGURE 7.20 Crossconnections Opened at each Node to Form the Restoration Pathsets for Each Span Failure when using the PC plan formed within Net2 (as Spared using IP2-Cycle) using the DPCD with Cycler Nodes in Increasing Total Working Link Order. For Restoration using 2-Step Restoration.

8. Summary

This thesis explored the performance of a number of different subgraph types when applied to the preconfigured restoration of mesh-type transport networks. The motivation behind preconfigured restoration is that current DCS implementations have crossconnection formation times which may not permit a span failure to be restored quickly enough to avoid large scale consequences when using an on-demand restoration method. Preconfiguration attempts to pre-form crossconnections within the network spare capacity in best average-case anticipation of span failures. In the event of a span failure, preconfigured restoration paths may be removed from the pre-formed spare link structures by opening preconfigured crossconnections, and may then be applied to the restoration of the disrupted working signals. If there are not sufficient immediately available preconfigured restoration paths, additional restoration paths may be scavenged from the remaining pruned and unallocated spare capacity using an on-demand restoration method. By preconfiguring spare capacity, the number of crossconnections which must be dynamically closed, to form the restoration pathset of any span failure, is reduced and should, therefore, result in a speedup of restoration times.

In chapter 3, two related heuristics were evaluated, which generate preconfigured trees using a network's spare capacity plan. In the test networks, network restorability, when using preconfigured restoration alone, was in the range of 30-40% while the overall restorability, when using 2-step restoration, was very close to, if not, 100%. The total number of crossconnections which were asserted to form the 2-step restoration pathsets, of all span failures, was 50-70% of the number required for conventional on-demand KSP restoration alone. The tree pattern type has no constraint on the nodal degree which it may have. This requires a technical ability which present DCS machines do not have; the ability to directly connect a single port card to multiple other port cards. However, the PC tree was still evaluated to evaluate its potential contribution to preconfigured restoration. The unconstrained nodal degree of the tree pattern resulted in a large number of preconfigured crossconnections within the network, as it was assumed that all of a tree's links, meeting at a node, would be fully crossconnected (i.e. there would exist a crossconnection joining each pair of tree links.) This also generated a relatively large number of crossconnections which would have to be opened to access the preconfigured paths contained with the PC trees. As discussed in chapter 2, whether or not this would be a technical problem depends on the method by which a DCS forms multiple crossconnections to a single port card and on the relative speed of closing and opening a crossconnection.

In chapter 4, two genetic algorithm based heuristics, were allowed to form preconfigured plans which contain a heterogeneous mixture of preconfigured subgraph variants. Each GA evolves a one unit capacity preconfigured pattern, at each iteration, although multiple copies of the pattern may be taken. The first GA (GA1) formed PC patterns based on a span level of representation while the second GA (GA2) formed PC patterns based on a crossconnection level of representation. In the test networks, both GAs generated PC restorabilites from 75-90% and overall 2-step restorabilities of 90-100%. When using the PC plan generated by GA1, the total number of crossconnect assertions to form the 2-step restoration pathsets were 15-25% of the total required when using on-demand KSP restoration alone. With GA2, this was only 11-23% of the total required when using on-demand KSP restoration. However, as with PC trees, the patterns which were generated by the GAs were not constrained in their nodal degree. GA1 used a span based representation to form PC patterns in which all links of a common pattern, which met a node, are assumed to be fully connected, i.e., a crossconnection is implied between all pairs of pattern links meeting at a node. GA2, however, used a crossconnection based representation which does not require that every pair of pattern links, at a node, have a crossconnection. As a result GA1 had a significantly larger number of crossconnections to open (or prune), to effect 2-step restoration, than did GA2. Again, as with PC trees, whether, the number of crossconnections to open is an issue or not depends on the DCS implementation. The most important result is the demonstration of 4 to 5-fold reduction in the crosspoints to be asserted in realtime to affect restoration. This may be a significant speed-up for network restoration with slow crossconnects, if one considers that the pruning workload is not of a realtime concern.

Chapter 5 introduces two integer program techniques which form PC plans containing cycles. The first integer program technique (IP1-Cycle) generates a PC cycle plan using a network's defined sparing plan and a pre-defined cycle set. The resulting PC plan is optimal, in that it minimizes the number of uncovered working links within the network, within the limits imposed by the network's pre-existing sparing plan and the richness of the pre-defined cycle set. The second integer program (IP2-Cycle) generates a PC cycle plan and a network spare capacity plan based on a network's given working link placement plan and a pre-defined cycle set. The resulting PC plan is 100% restorable using only PC paths and is optimal, within the limits imposed by the richness of the cycle set provided for the IP, in the sense that the total physical distance of the spare fibre is minimized. This results in an economical savings while achieving 100% preconnected readiness for restoration. IP1-Cycle generated PC plans with restorabilities, using only preconfigured paths, in the range of 75-96%. The lower value of 75% occurred in test network Net4 which was run with a severely limited pre-defined cycle set, due to the size and complexity of the network. The remaining networks were all greater than 90%. IP1-Cycle's 2-step restorabilites ranged between 93 and 100%. The total crossconnection closures required to form the 2-step pathsets, of all span failures, was, excluding the most severely constrained test network (Net4), in the range of 0-4% of the total generated when using on-demand KSP restoration alone. Net4 required a total number of crossconnection closures which was 24% of that required by on-demand KSP.

IP2-Cycle generated restorabilites, using only preconfigured restoration paths, which were 100%. In addition, because only PC paths were used there were no crossconnection closures required to form the restoration pathset. However, the spare capacity required to make a network fully PC restorable was greater, compared to the original fully sparing which was fully restorable using only on-demand KSP restoration, for 3 of the 5 test networks. 3 to 18.5% excess sparing distance was required for these networks, although if the constrained network (Net4) is excluded, the range of excess sparing becomes 3-9%. The cases where less sparing was required is attributed to the IP, which generated the original 100% KSP restorable sparing of the networks, being constrained in the set from which it could form restoration paths (in a manner similar to the constraint of the IP1-Cycle and IP2-Cycle's cycle set.)

For both IPs, the total number of crossconnections pruned, to form the 2-step restoration pathsets of all span failures, was relatively low compared to the previously evaluated

PC patterns. This is due to the cycle's simple structure (all nodes in a cycle are degree 2.) In addition, a cycle is an example of a pattern class which can be formed by any existing DCS implementation without any modification.

Chapter 6 establishes theoretical upper limits to the number of preconfigured paths per preconfigured pattern link which the tree and cycle subgraph types can provide. It was found that a cycle, spanning a certain number of nodes, could potentially provide twice as many immediately useful PC paths per spare link consumed than a tree, spanning the same number of nodes. In addition, the maximum number of PC paths which can be provided per pattern link was established for any possible configuration of pattern and this value was found to match the value found for the cycle.

Motivated by the results of chapter 5 and 6, chapter 7 presents and tests a distributed preconfigured cycle design (DPCD) protocol. The DPCD is based on the statelet processing rules of the Selfhealing Network (SHN) protocol but is adapted significantly to discover the most useful cycle in the network spares, on each iteration, in a non real-time preplanning orientation. The overall performance of the PC plans generated by the DPCD did not, in general, equal that of the plans generated by IP1-Cycle. However, as the DPCD is a heuristic, it was not expected that they would. The comparative performance of the DPCD was quite good, however, sometimes equal to the IP generated results, in terms of PC restorabilities (80-95%) and 2-step restorabilities (94-99%). The total crossconnection closures required to form the 2-step pathsets was in the range of 0-15% of the total required to form the on-demand KSP pathsets. In addition, due to the simple structure of a cycle, the total number of crossconnections which were opened was relatively low. When the DPCD was run using the sparing plans generated by IP2-cycle, it generated PC plans with good PC restorabilities (90-99%) and 2-step restorabilities (96-100%). Also, the total crossconnection closures required to form the 2-step restoration pathset were quite low (0.5-5% of the total number of closures required to form the KSP restoration pathset.)

Of all the evaluated pattern types, the cycle seems clearly to offer the most advantage when used in PC restoration. It can form PC plans which offer a high restorability when using only PC restoration to restore all span failures. In addition, because of the cycles simple structure (all of a cycle's nodes are degree 2) it can be directly implemented using existing DCS implementations, and requires the minimum of subsequent pruning workload. Another benefit of the cycle's simple structure is that, in the event of a span failure, to extract the useful PC paths only the failed span's end nodes need prune any preconfigured crossconnections which means that no signalling is required to inform other network nodes of the failure. This could translate into a speed advantage when compared against other PC pattern types.

8.1 Future Research

This work presented a first attempt at a distributed preconfigured cycle design protocol. Future work could modify the protocol to improve its effectiveness and more accurately model its behaviour using discrete event simulation.

Research into the speed-up of DCS crossconnect times would also be of benefit to preconfigured restoration. Although, preconfigured restoration was conceived to compensate for the slow crossconnect time of current DCS implementations, it would still be useful if fast DCS implementations came to be; with a suitably fast DCS implementation combined with a PC restoration plan, one can begin to see how the restoration times of dedicated capacity restoration methods such as SHR and APS could be approached by a mesh-based restoration method.

In closing this thesis, it should again be emphasized that the work contained within was done at a graph theoretical level. Pure crossconnection event totals were tabulated for the various methods presented within the thesis. Although, this data is useful for establishing that preconfigured restoration has a great deal of potential for speeding up the restoration of failures in mesh networks, it would be of great interest to perform a discrete event simulation of PC restoration based on an actual DCS implementation to establish how much of a speed-up there is compared to conventional on-demand restoration.

Bibliography

[1]. J. C. McDonald, "Public Network Integrity - Avoiding a Crisis in Trust," *IEEE Journal on Selected Areas in Communications*, Vol. 12, No. 1, pp. 5-12, January 1994.

[2]. W. D. Grover, Chapter 11 of *Telecommunications Network Management Into the 21st Century*, IEEE Press, 1994, pp. 337-413, edited by S. Aidarous and T. Plevyak.

[3]. W. D. Grover, "The selfhealing network: A fast distributed restoration technique for networks, using digital cross-connect machines," *Proc. IEEE Globecom'87*, pp. 1090-1095, 1987.

[4]. W. D. Grover, "Method and apparatus for self-healing and self provisioning networks," U.S. Patent No. 4,956,835, September 11, 1990.

[5]. D. A. Dunn, W. D. Grover, M. H. MacGregor, "Comparison of k-shortest paths and maximum-flow routing for network facility restoration," *IEEE J-SAC Integrity of Public Telecommunications Networks*, Vol. 12, No. 1, pp. 88-99, January 1994

[6]. J. Sosnosky, "Service applications for SONET DCS distributed restoration," *IEEE Journal on Selected Areas in Communications*, Vol. 12, No. 1, pp. 59-68, January 1994.

[7]. SR-NWT-002514: "Digital cross-connect systems in transport network survivability," *Bellcore*, 1, January 1993

[8]. T.-H. Wu, H. Kobrinski, D. Ghosal, T. V. Lakshman, "The impact of SONET digital cross-connect system architecture on distributed restoration," *IEEE Journal on Selected Areas in Communications*, Vol. 12, No. 1, pp. 79-87, January 1994.

[9]. W. D. Grover, M. H. MacGregor, "Potential for spare capacity preconnections to reduce crossconnection workloads in mesh-restorable networks," *Electronics Letters*, Vol. 30, No. 3, February 3, 1994, pp. 194-195.

[10]. M. H. MacGregor, W. D. Grover, K. Ryhorchuk, "Optimal spare capacity preconfiguration for faster restoration of mesh networks," accepted for publication by *Journal of Network and Systems Management*

[11]. R. Iraschko, M. H. MacGregor, W. D. Grover, "Optimal Capacity Placement for Path Restoration in Mesh Survivable Networks," *Proc. ICC'96*, pp. 1568-1574, 1996.

[12]. R. Brualdi, *Introductory Combinatorics*, Englewood Cliffs, NJ: Prentice Hall, 1992.

[13]. G. J. E. Rawlins, Introduction to *Foundations of Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann Publishers, 1991, pp. 1-10, edited by G. J. E. Rawlins

[14]. D. Beasley, D. R. Bull, R. R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals," *URL - ftp://alife.santefe.edu/pub/USER-AREA/EC/GA/papers/ over93.ps.gz*, 1993

[15]. A. L. Corcoran, R. L. Wainwright, "LibGA: A User-Friendly Workbench for Order-Based Genetic Algorithm Research," *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing (SAC 93)*, Indianapolis, Indiana, Feb. 14-16, 1993.

[16]. R. R. Iraschko, *Path Restorable Networks*, Ph. D. Dissertation, University of Alberta, Spring 1997

[17]. D. Stamatelakis, "Source Code Listings for Programs Used in M. Sc. Thesis on Preconfiguration of Mesh Network Spare Capacity", *TRLabs Internal Report TR-96-13*, 1996

Appendix A: Test Network Standard Network Interface Files (SNIF)

Network: Net1

Program: combined working and spare capacity placement span restoration

Node	Xcoord	Ycoord
0	20	100
1	80	100
2	0	50
3	40	75
4	60	75
5	100	50
6	40	25
7	60	25
8	20	0
9	80	0

Span	NodeA	NodeB	Distance	Working	Spare
1	0	1	1.000000	6	3
2	0	2	1.000000	6	3
3	0	3	1.000000	6	3
4	1	3	1.000000	7	2
5	1	4	1.000000	6	2
6	1	5	1.000000	7	2
7	2	3	1.000000	7	2
8	3	6	1.000000	7	0
9	3	7	1.000000	8	1
10	3	4	1.000000	7	1
11	4	6	1.000000	4	3
12	4	7	1.000000	6	1
13	4	5	1.000000	7	1
14	2	8	1.000000	7	2
15	2	6	1.000000	6	2
16	6	8	1.000000	7	1
17	6	7	1.000000	6	2
18	7	8	1.000000	7	2
19	7	9	1.000000	8	1
20	5	7	1.000000	7	2
21	8	9	1.000000	5	4
22	5	9	1.000000	5	4

Network: Net2

Program: combined working and spare capacity placement span restoration

Node	Xcoord	Ycoord
0	60	75
1	40	80
2	70	80
3	50	70
4	10	40
5	10	55
6	45	50
7	30	47
8	45	40
9	60	30
10	30	30
11	10	20
12	50	10
13	15	70
14	70	50

Span	NodeA	NodeB	Distance	Working	Spare
1	0	1	9.000000	20	12
2	0	2	6.000000	8	16
3	0	14	21.000000	16	4
4	1	2	14.000000	28	4
5	1	3	6.000000	32	32
6	1	13	11.000000	48	16
7	2	14	16.000000	20	12
8	3	13	8.000000	24	16
9	3	14	11.000000	48	16
10	4	7	7.000000	28	28
11	4	11	7.000000	0	28
12	5	6	5.000000	12	28
13	5	7	7.000000	76	24
14	5	13	10.000000	44	48
15	6	7	11.000000	24	0
16	6	8	11.000000	68	64
17	6	9	11.000000	100	32
18	6	13	8.000000	104	8
19	7	8	1.000000	64	68
20	7	10	7.000000	136	40
21	7	11	13.000000	64	20
22	9	10	5.000000	88	88
23	9	12	23.000000	16	0
24	9	14	10.000000	40	56

Span	**NodeA**	**NodeB**	**Distance**	**Working**	**Spare**
25	10	12	13.000000	48	48
26	10	14	19.000000	112	0
27	11	12	8.000000	48	48
28	13	14	11.000000	88	24

Network: Net3

Program: combined working and spare capacity placement span restoration

Node	**Xcoord**	**Ycoord**
0	100.000000	100.000000
1	0.000000	0.000000
2	0.000000	80.000000
3	50.000000	80.000000
4	70.000000	80.000000
5	0.000000	100.000000
6	0.000000	20.000000
7	20.000000	50.000000
8	100.000000	0.000000
9	30.000000	80.000000
10	70.000000	100.000000
11	0.000000	60.000000
12	20.000000	25.000000
13	60.000000	60.000000
14	0.000000	40.000000
15	40.000000	60.000000
16	50.000000	0.000000
17	75.000000	25.000000
18	20.000000	0.000000
19	100.000000	60.000000

Span	**NodeA**	**NodeB**	**Distance**	**Working**	**Spare**
1	0	4	80.000000	177	65
2	0	10	110.000000	21	59
3	0	19	45.000000	71	124
4	1	6	40.000000	51	59
5	1	12	65.000000	153	36
6	1	18	64.000000	94	95
7	2	5	95.000000	33	118
8	2	9	83.000000	118	117
9	3	4	30.000000	103	242
10	3	9	23.000000	287	197
11	3	10	76.000000	87	45
12	4	10	76.000000	133	14
13	4	13	26.000000	321	163
14	5	10	118.000000	117	118
15	6	14	30.000000	51	59
16	7	11	51.000000	72	171
17	7	12	72.000000	202	203
18	7	15	20.000000	273	73
19	8	13	76.000000	213	36

Span	NodeA	NodeB	Distance	Working	Spare
20	8	17	76.000000	70	95
21	8	19	95.000000	124	118
22	9	13	30.000000	236	20
23	9	15	20.000000	215	183
24	11	15	33.000000	171	130
25	12	13	110.000000	321	84
26	12	16	60.000000	95	94
27	12	17	108.000000	35	59
28	13	15	20.000000	326	158
29	13	17	122.000000	131	23
30	14	17	80.000000	51	59
31	16	18	65.000000	17	95

Network: Net4

Program: combined working and spare capacity placement span restoration

Node	Xcoord	Ycoord
0	27662.000000	58733.000000
1	36839.000000	59453.000000
2	40193.000000	51952.000000
3	59028.000000	51473.000000
4	45300.000000	48681.000000
5	36839.000000	45327.000000
6	26544.000000	44810.000000
7	31358.000000	32213.000000
8	39315.000000	39740.000000
9	53040.000000	41815.000000
10	60304.000000	44847.000000
11	64071.000000	37623.000000
12	50965.000000	35670.000000
13	62094.000000	28058.000000
14	53441.000000	30961.000000
15	48094.000000	28886.000000
16	40193.000000	25933.000000
17	32545.000000	19089.000000
18	40272.000000	16914.000000
19	48094.000000	18512.000000
20	51893.000000	19353.000000
21	57272.000000	23060.000000
22	57512.000000	17315.000000
23	61578.000000	24480.000000
24	65811.000000	23221.000000
25	65650.000000	17716.000000
26	59652.000000	14602.000000
27	47375.000000	10850.000000
28	47454.000000	15240.000000
29	33568.000000	12448.000000

Span	NodeA	NodeB	Distance	Working	Spare
1	0	1	10.000000	786	786
2	0	2	98.000000	726	332
3	0	6	21.000000	1118	454
4	1	3	75.000000	786	786
5	1	2	166.000000	123	0
6	3	10	34.000000	1273	276
7	2	4	42.000000	1215	332
8	2	3	88.000000	55	510
9	5	6	78.000000	1003	510

Span	NodeA	NodeB	Distance	Working	Spare
10	6	7	56.000000	526	431
11	2	6	45.000000	209	608
12	4	5	50.000000	144	948
13	5	8	50.000000	1458	592
14	4	10	10.000000	604	910
15	4	9	45.000000	1524	595
16	9	10	99.000000	0	69
17	10	11	23.000000	1255	432
18	9	11	56.000000	31	671
19	11	12	89.000000	0	515
20	11	13	45.000000	1335	228
21	7	9	91.000000	32	431
22	9	12	10.000000	1509	769
23	8	12	10.000000	1219	962
24	12	14	10.000000	787	962
25	8	14	34.000000	18	453
26	8	16	56.000000	59	49
27	7	17	134.000000	332	0
28	7	16	122.000000	108	0
29	16	17	111.000000	0	0
30	15	16	102.000000	11	10
31	16	18	104.000000	3	49
32	14	15	106.000000	118	108
33	15	20	67.000000	118	108
34	13	23	45.000000	164	81
35	13	21	23.000000	1538	1307
36	13	14	11.000000	788	1307
37	23	24	100.000000	49	164
38	18	19	100.000000	7	49
39	18	27	66.000000	170	1
40	18	29	67.000000	114	98
41	17	18	29.000000	196	98
42	17	29	44.000000	326	228
43	28	29	46.000000	278	228
44	26	27	32.000000	85	85
45	25	26	77.000000	12	85
46	22	27	67.000000	86	85
47	21	24	79.000000	184	79
48	22	25	55.000000	189	85
49	19	28	23.000000	279	228
50	19	20	11.000000	853	1415
51	8	17	45.000000	326	228
52	8	19	66.000000	1692	1121
53	8	9	99.000000	0	79
54	7	8	100.000000	195	95

Span	NodeA	NodeB	Distance	Working	Spare
55	22	28	10.000000	109	228
56	20	21	10.000000	1036	1307
57	24	25	34.000000	33	104
58	21	22	30.000000	307	230
59	20	22	40.000000	21	0

Network: Net5
Program: combined working and spare capacity placement span restoration

Node	Xcoord	Ycoord
0	167.751000	43.902400
1	193.225000	67.479700
2	175.881000	55.284600
3	191.599000	95.122000
4	149.051000	94.579900
5	18.157200	99.187000
6	18.970200	49.864500
7	80.216800	85.094900
8	15.447200	7.317070
9	9.214090	139.024000
10	139.566000	173.984000
11	7.046070	78.319800
12	164.499000	63.956600
13	126.287000	37.398400
14	68.834700	176.152000
15	66.124700	75.067800
16	149.051000	117.886000
17	169.106000	102.439000
18	164.770000	121.680000
19	162.602000	80.758800
20	81.571800	127.642000
21	46.612500	127.642000
22	15.718200	18.699200
23	47.425500	106.504000
24	63.956600	100.271000
25	51.761500	85.636900
26	88.888900	189.431000
27	104.878000	66.395700
28	117.886000	78.048800
29	110.569000	150.678000
30	106.233000	85.365900
31	78.048800	71.002700
32	71.273700	189.702000
33	122.764000	60.433600
34	111.653000	37.398400
35	136.856000	59.349600
36	153.930000	151.491000
37	70.189700	43.902400
38	149.593000	132.249000
39	33.333300	173.442000
40	140.108000	188.618000
41	28.726300	82.926800

Node	Xcoord	Ycoord
42	107.317000	173.713000
43	123.306000	189.160000
44	39.837400	6.775070
45	39.837400	48.238500
46	52.303500	176.152000
47	94.579900	66.937700
48	162.331000	7.317070
49	79.674800	109.756000
50	42.547400	75.880800
51	104.336000	49.322500
52	30.000000	45.000000

Span	NodeA	NodeB	Distance	Working	Spare
1	0	2	5.000000	11	13
2	0	1	76.000000	27	13
3	1	2	46.000000	13	13
4	1	12	209.000000	32	48
5	20	38	299.000000	73	62
6	3	4	168.000000	22	22
7	4	12	214.000000	29	29
8	4	19	16.000000	22	22
9	5	41	91.000000	9	13
10	1	48	349.000000	62	56
11	6	45	36.000000	12	11
12	6	52	36.000000	11	12
13	7	15	29.000000	13	21
14	7	49	1.000000	21	21
15	8	22	12.000000	2	3
16	8	44	30.000000	3	3
17	12	35	250.000000	43	42
18	9	39	2560.000000	70	65
19	10	40	114.000000	4	4
20	10	43	35.000000	4	4
21	36	38	1480.000000	65	70
22	9	21	585.000000	46	70
23	11	22	118.000000	61	62
24	11	41	79.000000	48	54
25	5	11	100.000000	13	9
26	0	13	414.000000	26	14
27	13	34	305.000000	15	1
28	14	26	23.000000	6	7
29	14	32	25.000000	13	13
30	14	42	32.000000	15	10
31	14	46	53.000000	20	13
32	15	37	148.000000	37	23

Span	NodeA	NodeB	Distance	Working	Spare
33	15	49	31.000000	24	21
34	15	50	7.000000	35	8
35	16	17	48.000000	11	12
36	16	18	28.000000	12	11
37	16	20	210.000000	40	29
38	4	16	1011.000000	25	29
39	16	38	243.000000	29	11
40	17	18	20.000000	6	12
41	3	19	721.000000	14	22
42	10	36	79.000000	35	11
43	20	21	373.000000	91	70
44	13	35	35.000000	5	25
45	28	49	88.000000	38	21
46	22	44	30.000000	3	3
47	23	24	8.000000	11	11
48	23	25	2.000000	11	11
49	23	49	19.000000	27	19
50	23	50	36.000000	19	27
51	24	25	1.000000	7	11
52	27	33	112.000000	37	42
53	27	47	10.000000	28	42
54	27	51	14.000000	11	16
55	28	30	43.000000	2	22
56	28	31	76.000000	43	22
57	29	36	80.000000	33	62
58	10	29	89.000000	19	27
59	29	46	348.000000	43	62
60	22	31	513.000000	63	35
61	30	31	76.000000	43	22
62	30	49	88.000000	37	21
63	31	47	71.000000	56	42
64	33	34	12.000000	6	22
65	33	35	207.000000	36	55
66	37	45	139.000000	19	19
67	22	37	740.000000	42	21
68	39	46	79.000000	36	62
69	40	43	98.000000	1	4
70	9	41	173.000000	62	61
71	10	42	377.000000	17	8
72	45	50	166.000000	19	19
73	26	42	49.000000	7	7
74	34	47	146.000000	14	21
75	47	51	14.000000	16	16
76	22	48	653.000000	96	62
77	45	52	7.000000	11	12

Span	NodeA	NodeB	Distance	Working	Spare
78	21	48	309.000000	90	62
79	32	39	75.000000	13	13