NEW TREE-BASED ALGORITHMS FOR NETWORK SPARE CAPACITY DESIGN

Yunzan Zhang

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

December 2003

Your file Votre référence
ISBN: 0-612-91156-X
Our file Notre référence
ISBN: 0-612-91156-X

Abstract

New Tree-based Algorithms for Network Spare Capacity Design

Yunzan Zhang

Survivable network design has become increasingly important due to the need for reliable communication service. One of its important components is the spare capacity design. Its main purpose is to provide cost-efficient spare capacity reservation at certain survivability level in case of predicted failures. In this thesis, we study various kinds of network survivability techniques and the corresponding algorithms. Subsequently, we introduce our two pre-planned path restoration algorithms for spare capacity design in mesh-like networks. They can get higher utilization of spare capacity and reasonable reaction time. First one is a spanning tree based algorithm with backup parents, which needs much less spare capacity than the well known hierarchical tree algorithm while the restorability is slightly higher or lower. The second algorithm is a cycle tree based algorithm with backup parents and some extra cycle edges. Simulation results show that this algorithm works much better on restorability than the other two algorithms. The time complexities of the two new algorithms are: $O(n^4)$, where n is the total number of nodes in the network. The space complexities are the same too: $O(n^3)$.

Acknowledgments

I am very grateful to my supervisor, Dr. Hovhannes A.Harutyunyan, for his guidance, encouragement and support. He inspired me to generate many good ideas found in this thesis and helped me to stay on the right track towards its completion.

I would like to give my sincere thanks to my husband for his support, my daughter for her understanding and my parents for their incessant encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the modern society, we are increasingly becoming dependent on the exchange of information. Telecommunication, Internet, on-line banking, on-line government services and E-mail system are just some examples how we use network daily. Hence, we are ever more dependent on the availability of reliable communication services and networks. To assure such reliable services, we need to design a survivable network with suitable spare capacity to enable fault recovery via rerouting of traffic in the event of a failure.

A communication network is often represented by a graph. We define as network failure, either a node failure or a link failure where the connectivity of the network is decreased and some original service connections are broken. The objective of the network survivability design is to make the probability of data loss as low as possible upon the link or node failures. In a survivable network, the original service path is referred to as the primary path. Any node failure or link failure within the primary path will break down the service connections carried on this path. The main purpose of survivability design is to find available backup path(s) for the replacement of the broken primary path upon failure(s), so that the affected service connections can be recovered. In order to make the data loss as low as possible in the restoration process, any survivability scheme must satisfy two requirements. First, it must be able to provide enough spare capacity at a certain survivability level and secondly, the restoration process must be as short as possible.

In order to improve the utilization of expensive communication facilities, communication networks are always built to share resources for serving the large numbers of communications' service requirements. Due to ever increasing social dependence on network facilities, network survivability has become a very important focus of many researchers in this domain.

In the following sections, we compare and analyze several design strategies that different schemes may employ.

1.1 Network survivability techniques

Network survivability techniques have been used to guarantee available communication services in the face of network failures, i.e., link failure and node failure. The major part of constructing a survivable network is a design of the spare topology and the corresponding capacity, which ensures enough spare capacity for the physical network restoration from a failure via traffic rerouting when the network topology is given. For example, given a mesh-like network topology and the normal traffic demands with their flow allocations, we need to resolve the following problems: how much spare capacity should be provisioned and where it should be located in order for the network to recover a specified set of failure scenarios (e.g., loss of any single link). The term mesh-like does not imply that the network topology is a full mesh, but rather that the network nodes are at least two-connected [7].

Therefore, the major interest in survivable network design has been concentrated on providing cost-efficient spare capacity reservation at a certain survivability level. The survivability level assesses the percentage of restorable network traffic upon failures [8] .

1.1.1 Restoration Process

Upon a failure, a general restoration process will invoke the following four steps [9]: fault detection, fault localization, fault notification, and fault restoration. Restoration strategies decide the responding action when a failure happens in a network. First, fault detection is responsible for deciding whether a failure exists or not and it is performed individually at all layers through the retrieving of information from monitoring or testing. Upon the detection of a fault, every layer will try to localize the fault, after the fault is detected or localized, the next step is to notify the appropriate network element to find, configure, and provision the spare capacity of the backup path. Next, the restoration process splits into two parallel processes; the repair process where the failure is isolated and repaired, and the reconfiguration process where the affected traffic flows are identified in the network nodes. The backup paths for these affected traffic flows are selected and rerouted. When the repair process is finished, the restoration process will go back to its normal state. The restoration process is also responsible for rerouting of the affected traffic from backup paths back to primary working paths.

Excluding the failure detection, which is decided by the physical layer techniques, the other steps are decided by the survivability schemes, for example, how the backup paths are selected and how the spare capacity is provisioned. In network restoration, when a failure is detected, the affected traffic flows are rerouted to backup paths that have enough spare capacity provisioned in the survivable network design phase. How to provide spare resources is a very important issue for the network survivability techniques.

1.1.2 Restoration Categories

The network restoration strategies can be classified by different criteria. We introduce some strategies related to our study as follows:

- **Pre-planned vs. Dynamic**

The criteria is according to the time when spare resources for backup paths are reserved; the pre-planned method reserves spare capacities before failure happens, while the dynamic method reserves additional spare capacity after failure happens. Pre-planned methods will have advantages on guaranteed survivability upon predicted failures. However, pre-planned resources will be wasted if none of the predicted failures happens. Meanwhile, the backup pre-planned paths are optimized for all failure scenarios, however, they might not be the optimal backup routes for the specific failure. The restoration time is gained, but the preparation for all possible failures is slow and takes up memory. Dynamic methods try to allocate the spare resources when the failure happens. In this way, better resource utilization can be achieved but the survivability assurance is risked because the requested resource might not be available when it is requested upon failures.

- **Path restoration vs. Link restoration**

According to the initial locations of the rerouting process, the protection/restoration schemes can also be categorized as path restoration and link restoration. Path restoration reroutes at the end nodes. It spreads failure influence to a larger area but it has less spare capacity requirement. At the same time, it has a slower restoration speed because of the longer reaction time. Link restoration reroutes at the nodes adjacent to the failure, has fast failure response time and significant impact in the area close to the failure. It only patches the "hole" produced by the failure and requires more spare capacity.

As showed in Figure 1, the source and destination nodes are v_1 and v_4, the original working path is $p = (v_1, v_2, v_3, v_4)$, the assumed failed link is $v_2 - v_3$. For

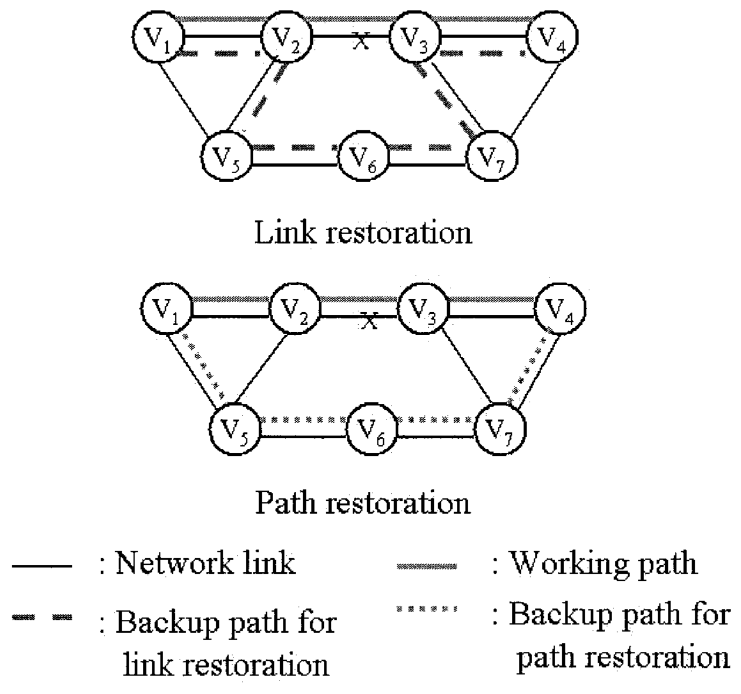


Figure 1: Link restoration and path restoration

link restoration, the nodes v_2 and v_3 are responsible for rerouting the affected traffic flows in order to keep the path away from the failed link. The spare path is $p_b = (v_1, v_2, v_5, v_6, v_7, v_3, v_4)$. In contrast, for path restoration, nodes v_1 and v_4 are responsible for restoration and rerouting of the entire path set. The spare path is $p_b = (v_1, v_5, v_6, v_7, v_4)$. In general, path restoration is known to require less spare capacity than link restoration. However, it is more complex to implement since many more nodes are involved in the restoration process [7].

- **Failure dependent vs. Failure independent**

We can further classify path restoration into failure dependent and failure independent. Failure dependent refers to the fact that path restoration depends on the failure scenario. It requires network nodes to save additional network state information achieving better utilization. Failure independent means that path restoration does not depend on failure scenarios. Hence, having less storage space and less signaling information it is easier to implement.

The selection of backup paths can be failure dependent where different failures are protected by different backup paths. A failure independent path restoration scheme requires less signaling support and is easier to employ while it might require more spare capacity than failure dependent path restoration.

- **Single layer vs. Multiple layers**

Depending on which network layer the services are restored, we have single layer restoration and multiple layer restoration. If the given restoration scheme happens on one layer, such as WDM, SONET, ATM, IP/MPLS, it is called single layer restoration. Otherwise, multiple layer restoration implies more than one layer participating in restoration design. In our work, we assume a single layer, such as the WDM or SONET layer.

- **Link failures vs. Node failures**

Which kind of failure(s) are the services designed for? One restoration scheme can be designed to resolve mainly link failures or node failures. When there is a node failure, we can consider every link connected to this failed node as invalid. So, a node failure can be equal to several link failures. Moreover, almost all the node equipment, including the switching and transport equipment, is redundant at the hardware level, which keeps the node failure probability very low. Therefore, in our study, node failures will not be considered.

1.2 Spare Capacity Design Algorithms

1.2.1 Network Design Problem

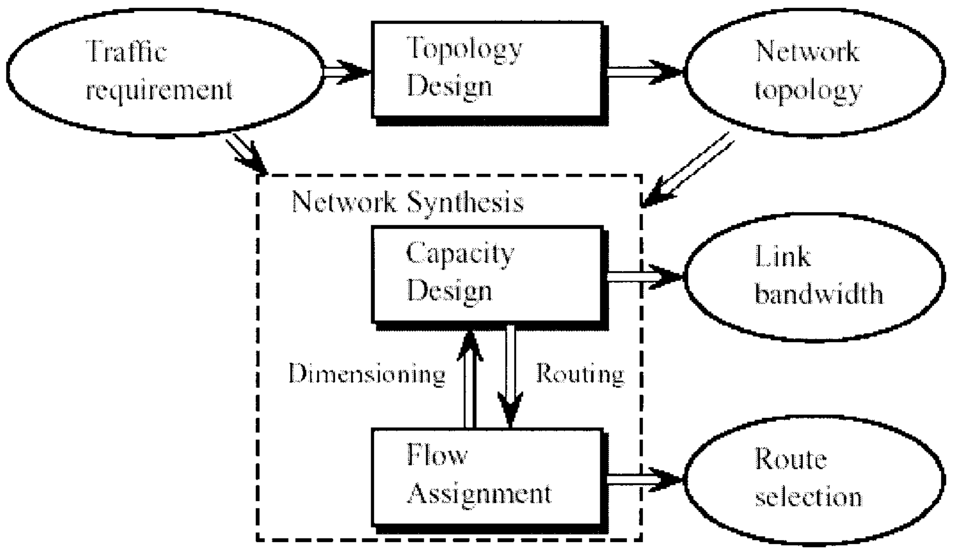


Figure 2: General network design problems

We use the Figure 2 from [8] to denote the general network design problems. Given the traffic demand locations and their Quality of Service (QoS) requirements in the network, traditional network design is responsible to allocate nodes and links with enough resources (including bandwidth, buffers, etc.) to transport all the traffic demand with guaranteed QoS requirements from the origin to the destination, as well as to minimize the total network cost. The first task is topology design, which provides the node location and link connectivity in the network topology. The next task is called network synthesis, which decides the traffic routing and resource distributing in the given network topology. The two problems faced in network synthesis are capacity design and flow assignment. Multi-commodity flow (MCF) models are used to solve these problems. Additional constraints, i.e., traffic QoS guarantees, node buffer restrictions and link capacity limits should be considered. In many cases, the MCF model is NP-hard [16]. Approximation methods are widely used to solve network design problems and the above mentioned models. Survivability requirements for the traffic introduces additional constraints into above traditional network design problems.

Inherited from the traditional network design phase, survivable topology design has specific requirements. A common requirement is try to achieve 100% restorability in case of a single failure, it demands that at least the rest of the network is connected after any single link(or node) failure. This requirement is defined as two-link(node) connectivity. A two-link(node) connected topology has at least two link(node)-disjoint paths between any source-destination pairs [8]. In addition to the survivable topology design, the survivable requirements of the traffic need extra resources to reroute the affected traffic upon failure scenarios. In this thesis, we focus on spare capacity design algorithms.

The fastest possible scheme for restoration is to reroute the signals over two physically disjoint paths (1+1 DP) and do the selection of the surviving signal at the receiver. This can be cost-effective for some very large point-to-point demands in some metropolitan area networks, but in general requires an investment of over 100% redundancy in terms of bandwidth-distance product consumed. Therefore, schemes that provide some form of protection source sharing are of interest. The SONET bi-directional line-switched ring (BLSR) is today perhaps the most widely used protection sharing structure, which is a kind of the ring-based scheme. In a BLSR the ring protection bandwidth is shared over all spans of the same ring that is often more efficient than 1+1 DP [11].

"Mesh-restorable" networks, on the other hand, can be much more capacity-efficient than ring-based networks in terms of the total spare capacity in a spare capacity design. Here, "mesh-restorable" refers to the ability of the rerouting mechanism to exploit an arbitrary topology in a mesh-like (as opposed to ring-like) way through diverse rerouting [11]. In a ring, the spare capacity protects only spans on the same ring. But in mesh-like network restoration flows may follow many diverse paths, network-wide, using smaller amounts of spare capacity on each route. The mesh-like restoration path set is generally different for each failure and adaptive to the actual spare capacity available on each span. Each unit of spare capacity in a mesh-like network is shared and re-usable in many more ways than in ring-based networks. Thus, rings use very simple mechanisms and therefore are simple, fast, but relatively capacity-inefficient. Mesh-like networks can be far more capacity-efficient, but have considerably longer restoration times.

1.2.2 Pre-planned Network Capacity Design Algorithms

Pre-planned methods for network capacity design have been well discussed in the literature. In centralized pre-planning, a path set for restoration is centrally calculated, then either downloaded in advance or on-demand to each node when needed. In distributed pre-planning each node develops either a network database image or local action results from the information the node knows. In either case, it is only that the computation of what to do that is "pre-planned"; each node must still execute a list of required connection actions in real time for its role in the overall restoration of a specific failure. In contrast, the idea in pre-configuration of mesh-like spare capacity, is to have the required cross-connections between spare channels already made in advance of failure. Ideally, for the fastest possible restoration the mesh-like network would be both pre-planned and pre-connected. But the premise is that the spare paths are fixed, what we called failure independent (described in Section 1.1.2). For the failure dependent method we do not know which spare connections will be needed before failures as in our new tree based algorithms. Furthermore, for different layers the emphasis is different. Higher layer does not need to care for the physical connection in a path, knowing there exists a path is enough, but lower layer should know every physical connection. In our work, we emphasize the pre-planned capacity design, regardless of whether the spare connections are pre cross-connected or on demand. The following are some typical kinds of pre-planned spare capacity design algorithms.

- **Tree-based algorithms**

A tree is a set of connected nodes having only one path between any pair of nodes. A spanning tree is a tree which covers all network nodes. When a weight is associated with each link, a maximum weight spanning tree is a tree for which the sum of the weights is a maximum. The spare topology with a spanning tree modal can provide one backup path between any pair of nodes, which means that there is a spare path for any possible working path. It can cover any link failure which is not the case on the tree edges.

One approach to generate pre-planned trees is based on maximum weight spanning trees in the network spare capacity, where the weight can be based on the number of times a spare link can contribute to the restoration of other links or some other meanings.

The hierarchical tree algorithm is described well in [1, 2] and we will introduce it briefly in chapter 2. Another approach to the tree-based restoration algorithms is the multi-tree algorithm. We introduce briefly an algorithm called "redundant trees" which creates redundant trees on arbitrary node-redundant or link-redundant networks. On the contrary, the hierarchical tree algorithm and our new tree based algorithms try to crate one tree for the whole graph. However, the purpose of the "redundant trees" algorithm is to establish two link or node disjoint trees for every source node. As a result, each node is connected to the common root of the two trees (that is the source node) by at least one of the trees in case of a node or link failure. Of course, not for every network graph we can create such redundant trees, this algorithm is applicable to the node-redundant or link-redundant graphs [10].

Actually, the construction of the two link or node disjoint trees for every source node needs cycle-oriented and tree-based concepts, so we prefer to put it in the combination of tree-based and cycle-oriented algorithm.

- **Cycle-oriented algorithms**

This method is based on the formation of the pre-planned cycles. The final spare allocation will include many closed cycles to form the required spare paths. Cycle-oriented pre-configuration remains fundamentally a mesh-like restorable network technology in terms of its capacity efficiency and its functional differences from self-healing rings.

One noted strategy is called *p-cycle*, with the motivation to achieve ring-like restoration time while remaining the desirable capacity efficiency of the mesh-restorable alternative. The method of *p-cycle* is based on the formation of closed paths (elementary cycles in graph theoretic terms), called *p-cycle*s, in the spare topology of a mesh-restorable network. They are formed in advance of any failure, out of the previously unconnected spare capacity units of a restorable network. Despite similarity to the rings, both use a cycle on the network graph for their topology, *p-cycle* are unlike BLSR/OSPR, UPSR/OPPR (or FDDI) logical rings because they protect both on-cycle and straddling failures. *P-cycle* spare capacity design takes little or no more capacity than a corresponding link-restorable mesh-like network [5].

A linear integer program (IP) was formulated for the design of *p-cycle* based restorable networks. First, the set of all simple distinct cycles up to some limiting size is generated from the network topology. The IP then generates an optimal *p-cycle* plan by choosing the number of copies of each elemental cycle on the network graph, to be configured as a *p-cycle.* For detailed description, please refer to [4, 5].

For the time complexity, because of IP or genetic algorithm, we cannot give an upper bound. However, its time complexity is much higher than that of our new tree based algorithms and the hierarchical tree algorithm. At the same time, from the theory, its average restorability should be higher than that of the basic plain tree-based algorithms, which is the most important reason for introducing the cycle-oriented concept to our cycle tree-based algorithm (we will explain it in Chapter 4).

- **Other algorithms**

Generally, using some models (like MCF mentioned above) to formulate spare capacity design problems, the objective is to minimize the total spare capacity required for the restoration from failure scenarios, and to get the highest restorability (100% maximum). But the resulting Integer Programming(IP) formulation is NP-hard [8]. Many heuristic methods are used to approach the optimum. In [8], there are some introductions relating this kind of algorithms, such as LP relaxation method, branch and bound based heuristics and genetic algorithms.

1.2.3 Our Goal

The two new tree-based algorithms proposed in this thesis use pre-planned and path restoration methods. The spanning tree based algorithm generates a maximum spanning tree with backup parents table, which needs much less spare capacity than the hierarchical tree algorithm while the restorability slightly bigger or smaller. The cycle tree based algorithm with backup parents and some extra cycle edges is going to produce a basic spanning tree topology with backup parents and some extra cycle edges, which form some cycles with the original spanning tree edges. It can get much higher restorability than the other two algorithms presented in this thesis, with very reasonable increased spare capacity. The most important objective is to achieve as large as possible restorability with the least spare capacity used.

We introduce two concepts here: link-disjoint and node-disjoint. Link-disjoint represents two paths with no shared link, that is, every link presented on one path never appears on the other one. Otherwise, node-disjoint implies to two paths with no shared node. Based on the theory, for every working path if there is a spare path which is link-disjoint with the working path, then we can say that the spare topology can protect any single link failure. Similarly, if there exists another node-disjoint path for every working path, then the one node failure should be restored. Actually, for an arbitrary network graph, it is not only hard to know whether there exists all link(node)-disjoint paths for all working paths, but is also very difficult to get such paths. The two new tree based algorithms we introduce later do not attempt to find the link-disjoint paths, instead, they construct the spare trees regardless of the original working paths. In addition, they use backup parents and cycle edges to gain another spare path(s) in case of the failed link being nested in the original spare path. From time complexity analysis and simulation results, these algorithms need much less running time and get very good restorability at the same time.

In our work, one traffic demand between each pair of nodes in the network was assumed as general. The allocated working capacity for each network was determined using the shortest path routing. The required spare capacity is calculated to provide the fault tolerance for any single link failure.

1.2.4 Organization

In the following chapters, we will introduce our two new algorithms and the hierarchical tree algorithm. The main idea of the well-known hierarchical tree algorithm will be presented in Chapter 2. In Chapter 3 we will give a detailed description of the new spanning tree algorithm with backup parents, especially on how to use backup parents to find backup spare paths. We will calculate the time complexity at the same time. The most important contribution of this thesis will be revealed in Chapter 4, introducing the cycle tree based algorithm. It will be shown how to obtain the cycle edges and how they can help the algorithm to reach higher restorability. In Chapter 5, the simulation results and comparisons with 5 typical topology models will be presented. Finally, we will state our conclusion in Chapter 6.

Chapter 2

Hierarchical Tree Algorithm

2.1 Definitions of the Terms

Formally, any network can be expressed as a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Two nodes $u \in V$ and $v \in V$ are *adjacent* if there is an edge $e \in E$, such that $e = (u, v)$. We use $u - v$ to denote the edge. We also say node u or v is a *neighbor* of another node. A node in the graph represents a station in a network, and an edge a connection between two stations, which creates a link between that two nodes. A path p in a graph $G = (V, E)$ is a sequence of nodes of the form $p = (v_1, v_2...v_n), (n \geq 2)$, in which each node v_i is adjacent to the next node. Obviously, the path p is also a sequence of edges or links. (v_i, v_j) is used to denote a source-destination pair of nodes. The length of a path is the number of edges in the path. A graph $G = (V, E)$ is said to be *connected* if there is a path between any two nodes of G.

We call a path on the tree between any two nodes *spare path*, the edges which form the original tree *tree edges*, the edges which connect the backup parents to the tree *backup edges*, the edges which form some cycles with the original tree edges *cycle edges*. The spare paths get from the backup parents or cycle edges are called *backup spare paths*. *Work capacity* denotes the assigned capacity for working path, while *spare capacity* means the unused capacity by any working path. The *restorability* is the percentage of restorable network traffic upon failures. In our work, we use the average fraction $\frac{restorable\ traffic\ capacity}{affected\ traffic\ capacity}$ to denote the term *restorability*, over all possible single link failures. For a pre-planned restoration algorithm, this measures effectiveness of a pre-planned method in providing immediate restoration without requiring any on-demand computation for dynamic restoration.

2.2 Algorithm Description

In this section, we introduce a well-known pre-planned tree-based algorithm, called "hierarchical tree algorithm" [1, 2]. The algorithm is based on the distribution of node identification message, called *tree ID* labels, which include the ID labels itself and the *capacity* to the backbone.

The tree ID labels indicate the position of the node in the hierarchical tree. The *capacity* field indicates the available protection capacity for this node by the path towards the root node. The node with the largest average link capacity is selected as the root node. The procedure starts from a pre-selected root node that passes the IDs to its children, and the same process is repeated at each node. Allegedly, each root node starts with *tree ID* label of 1, then it would distribute labels 1.1, 1.2 and the rest, to its neighbors with the link capacity between the root node and each child node. The node with ID 1.2, for example, will in turn distribute 1.2.1, 1.2.2 and the rest, to its neighbors with the corresponding updated capacity, excluding the new parent node 1.2. Through the tree-growing procedure, each node keeps record of the ID label message. When a node receives a new *tree ID* label, the algorithm compares the capacity field in the new label to its current capacity. It replaces the current ID with the newly arrived one if the new one provides larger link capacity to the backbone, that is, to the root node. The old ID is kept as backup parent(s) in a record look-up table. Subsequent to using the backup parents to form other valid backup paths, the final valid spare network includes not only the tree link themselves, but also the needed links used for the backup connection.

As the tree ID labels are exchanged among the nodes, each node recognizes its position on the tree and its respective parent node. At the same time, each backup parent's list is established. It is important to note that each node would generate new ID labels only if its current ID labels had changed because of a newly received ID or a change in topology. Therefore, once the hierarchical tree is formed, no new ID labels are generated anymore, and the algorithm is completed.

From the definition of the spanning tree, the hierarchical tree is also a spanning tree. The time complexity to construct the hierarchical tree is $O(n^3)$, where n denotes the number of nodes in the graph. Detailed analysis of its time complexity is in [2]. The methods of spare capacity assignment and restorability calculation are the same as in the following spanning tree-based algorithm. The time complexity of which equals to $O(n^4)$. In total, the running time for the whole algorithm is $O(n^4)$.

It is a distributed algorithm, although the hierarchical tree can be generated centrally. We can use it as link restoration or path restoration, according to which source-destination pair of nodes have to be rerouted on the tree. In our work, it is used as a path restoration algorithm. For the path restoration method, we can use different backup path for different pair of nodes in case of the same link failure.

Concerning the way of using the backup parents to get valid backup spare paths, we will describe it amply in Section 3.1, since all of the three tree-based algorithms use backup parents to gain backup spare paths.

Chapter 3

New Spanning Tree Based Algorithm

In this chapter we give a typical maximum spanning tree-based algorithm, which can achieve very close restorable percentage level compared to the above hierarchical tree algorithm, but saves much more spare capacity. Making of the algorithm involves two parts: the spanning tree construction and spare capacity assignment & restorability calculation, which are introduced in the following two sections.

3.1 Spanning Tree Construction

As introduced in Section 1.2.2, a maximum weight spanning tree is a tree for which the sum of the weights is a maximum. One way to generate pre-planned trees is based on maximum weight spanning trees, where the weight is based on the number of times the spare links can contribute to the restoration of other links, although the weights can be the spare capacities on the edges, or the work capacities, or some other numbers with special meanings. A heuristic method can be used to get the tree. For details, please refer to [11]. However, because of the time complexity we use another method instead. First, we get each shortest path as the spare path upon every one link failure, then calculate the total needed spare capacity according to the spare paths and apply this figure as the associated weight for every link, and lastly, build the maximum spanning tree. Another approach is to simply put the work capacity on the link as its weight, then construct the maximum spanning tree according to the weight. Using these two methods with some examples, we realize very close restorability. We select the second one, in our following simulation algorithms, because the first one needs much more time and space.

Once the link/edge weights are assigned, Prim-Jarnik's algorithm is used to find a maximum weight spanning tree. A single copy of this tree is then configured as a pattern (unit link capacity topology) in the network spare capacity design process. For any single link failure, we use this pattern to find any existing spare path, and give the needed spare capacity as the spare capacity on the link. Finally, after considering every possible single link failure, the maximum needed spare capacity on link will be the final designed spare capacity.

If the network spare topology is already given, it can be used as the basic spare graph. We can select our maximum spanning tree from the all possible spare links with the spare capacity limit as their assigned link weight. Then, we use this tree as the spare topology pattern to allocate spare capacity on every tree edge. In this way we put the tree link with necessary and enough spare capacity to get higher restorability. If only the network topology itself is available, then this topology will be used as the pattern to finish the tree creation and capacity assignment. In this thesis, we state the problem where no spare resource has been allocated in the given network topology. However, in order to compare the results between having or not the spare topology, we have several simulation results presented in Table 15 and Table 16 using existing spare topology[1]. Comparing the results when using the network topology in Table 17 and Table 18, we know that the restorability using designed spare topology is higher than that of using the network topology itself, because of the well designed spare topology structure by some other algorithm using IP and Genetic methods.

As we do in hierarchical tree algorithm, we also use backup parents as the backup selection for other available spare paths. Unlike the hierarchical tree algorithm, the table "backup parents" is established after the construction of the spanning tree. The table keeps the adjacent nodes for every node, including their parent node and children nodes used in the tree. Only the nodes that lose their connection to the tree need to switch to backup parents for other possible valid backup paths. The same method for using backup parents, with hierarchical tree algorithm, maximum three nodes in the affected original spare path are used to check their backup parents. These are: the source node, the second node and the node just being prior to the

[1]These three examples of the networks are from [7] with well designed spare topology.

failed link. After considering all the possibilities, some additional edges, used for backup connections, will be added to the basic tree structure. The final structure for the spare capacity allocation is an extended tree with more than $n-1$ edges. The algorithm used to create backup edges using backup parents is included in the next part: spare capacity assignment and restorability calculation, we will introduce the method in the next section.

To create a "backup parents" table, for each node, we scan the whole network graph according to the node number sequence, and select maximum *backup_num* (defined by the software) adjacent nodes to the table "backup parents" as backup nodes. The main difference, compared to the hierarchical tree algorithm in using the backup parents' table, is the sequence of every backup parents' set. In the spanning tree algorithm, the sequence is in terms of the natural node sequence; whereas in hierarchical tree algorithm is according to the ascending of capacity to the root.

3.2 Spare Capacity Assignment & Restorability Calculation

At this point, the spanning tree and the table of "backup parents" have already been created; we should consider the spare capacity assignment and the percentage of the restorable traffic according to this assignment. How do we assure that enough spare capacity is assigned to every spare link? First, upon each possible failed link, we

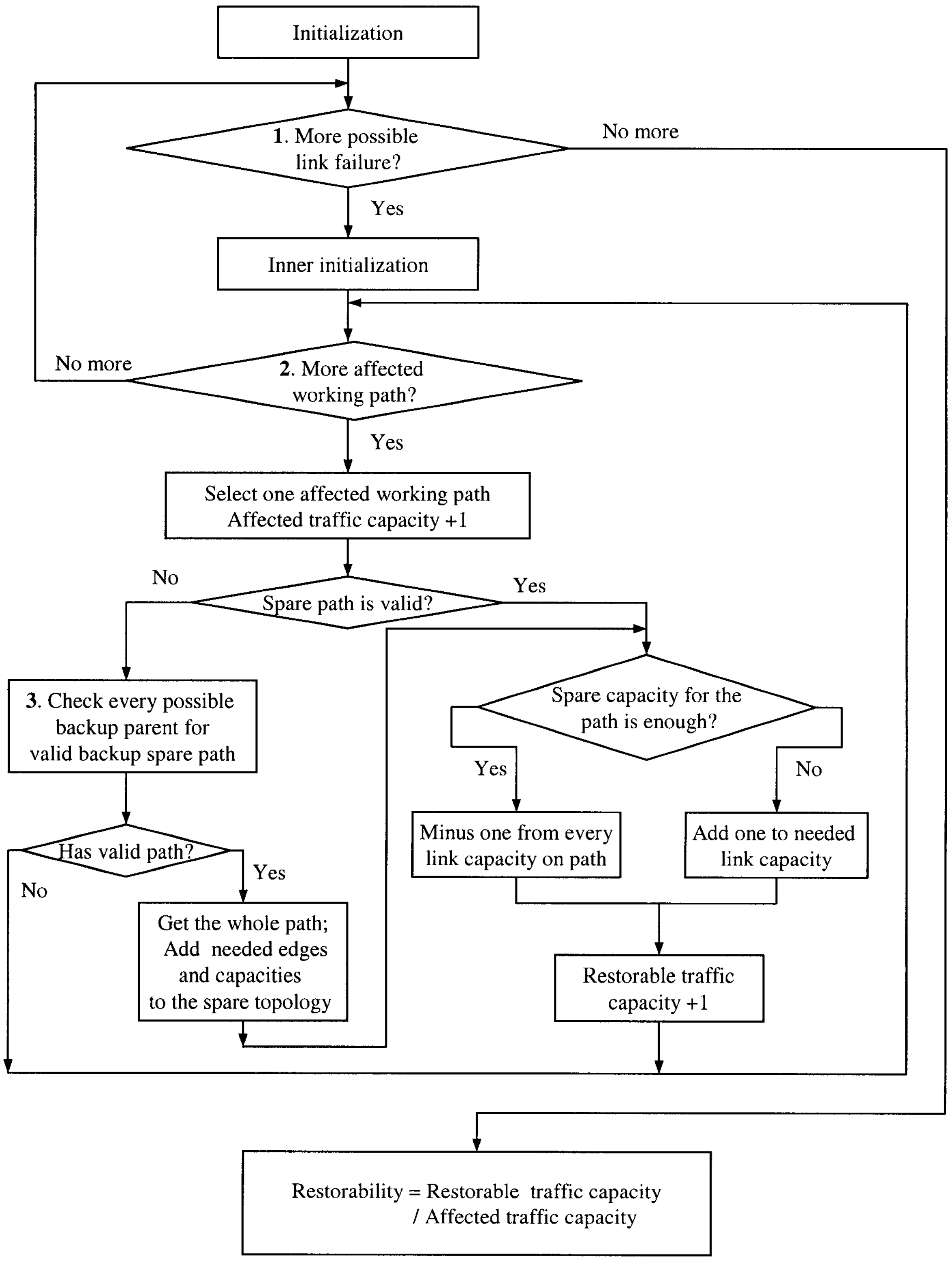

Initialization
1. More possible link failure?
No more
Yes
Inner initialization
No more
2. More affected working path?
Yes
Select one affected working path
Affected traffic capacity +1
No
Spare path is valid?
Yes
3. Check every possible backup parent for valid backup spare path
Spare capacity for the path is enough?
Yes
No
Has valid path?
Yes
No
Minus one from every link capacity on path
Add one to needed link capacity
Get the whole path; Add needed edges and capacities to the spare topology
Restorable traffic capacity +1
Restorability = Restorable traffic capacity / Affected traffic capacity

Figure 3: Flow chart of the spare capacity assignment and restorability calculation

calculate the just enough spare capacity on every tree edge and the backup edge. Second, the maximum needed spare capacity on the tree edges and backup edges will be selected as the final assigned spare capacity. This can assure sufficient spare capacity if there exists a spare path on the tree based spare topology. This spare capacity assignment process is dealing together with the restorability calculation as shown in Figure 3.

Let us see how the potential backup spare path can be derived from the table "backup parents". Suppose a link failure happened, for a source-destination pair of nodes the working path and the original spare path are both affected so we need to search the table "backup parents" to find another spare path. In our algorithm, as mentioned above, we check maximum three nodes' backup parents on the spare path, and try to find another valid backup spare path. If for any of the three nodes, one of its backup parents has a valid spare path to reach the destination node; then the path from the source node to this node, plus the edge between this node and its backup parent, plus the spare path from its backup parent to the destination node, is the valid backup spare path for this given failure.

We use Figure 4 as an example. To simplify, we just show only one backup parent for each usable nodes. From the spare path $p_s = (v_1, v_2, v_3, v_4, v_5, v_6)$ with the broken link $v_4 - v_5$, we cannot use p_s to reach the destination node v_6. We should check backup parents of node v_1, v_2 and v_4. Using v_1's backup parent v_{10}, the backup spare path $p_b = (v_1, v_{10}, v_3, v_4, v_5, v_6)$ includes the broken link $v_4 - v_5$, so it is useless. Using v_2's backup parent v_{11}, the backup spare path $p_b = (v_1, v_2, v_{11}, v_{12}, v_5, v_6)$ can

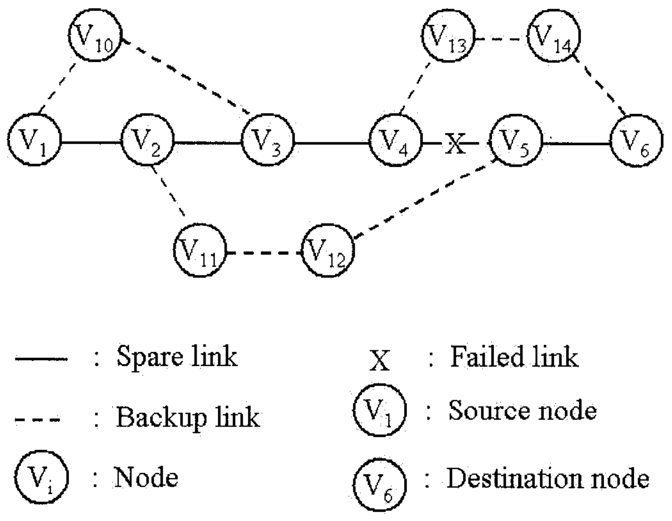


Figure 4: Spare path with backup parents

reach the destination node v_6, so we do not need to check v_4's backup parents. At last, we have found another valid backup spare path $p_b = (v_1, v_2, v_{11}, v_{12}, v_5, v_6)$ for source-destination pair (v_1, v_6), on condition that $v_4 - v_5$ is the failed link.

As we mentioned before, the backup spare path is failure dependent, that is, for different failure, the backup spare path maybe different. Compared to the plain spanning tree algorithm (before using backup parents), the use of backup parents improves the restorability very much. This conclusion can be confirmed by the simulation results in Section 5.2.1. Although the increased restorability is different on different network models, for every model, after using the backup parents, the restorability is always much higher than before. For example, in one model the difference even reaches the average of 40%.

To calculate the restorability, as defined in Section 2.1, through the whole spare capacity assignment procedure, for each possible link failure, we should check every affected working path to get the number of the affected paths and of the restorable paths. After considering every single link failure, we have the average percentage of restorable traffic as the restorability for the whole network. In our algorithm, we always keep the maximum needed spare capacity for every possible single link failure, so at last the values stored in the spare capacity matrix are the final spare capacities. They include spare capacity for the original spanning tree edges and the backup edges used for backup parent's connection.

3.3 Time Complexity & Space Complexity

The spanning tree algorithm is similar to the new cycle tree algorithm that will be described in Chapter 4, except for the application of cycle edges. Similarly, based on the time complexity analysis in Section 4.4, the time complexities for the tree construction and table "backup parents" making process are the same: $O(n^3)$, although, the new cycle tree based algorithm adds a little more time complexity to obtain the cycle edges. Besides this part, spare capacity assignment is another very important portion in the whole algorithm. As shown in Figure 3, only the marked parts act on the total time complexity as follows.

Part 1 : the running time is $O(m) = O(n^2)$.

Part 2 : the upper bound is $O(n^2)$.

Part 3 : because the maximum number of the backup parents is defined by the software, and the actual number is restrained by the node degree, we consider its contribution a constant.

Because of the part 2 being nested in part 1, together they need $O(n^4)$ running time. So, the time complexity of spare capacity assignment process is $O(n^4)$. Adding the tree construction part, the time complexity of the whole algorithm is $O(n^3) + O(n^4) = O(n^4)$.

As to the space complexity, we need the network connection matrix of $O(n^2)$, the tree connection matrix of $O(n^2)$ and the backup parents table of $O(n)$ for the tree construction process. In total, for this part, the space complexity is $O(n^2)$. In addition, for the second part, the spare capacity assignment and restorability calculation, assuming the maximum length of any path between any source-destination pair is $n - 1$, the memory needed to record working paths for every pair of nodes is $O(n^3)$, and for the spare paths is $O(n^3)$. So we have $O(n^3)$ space complexity for this process. Finally, we deduce that the upper bound of space complexity for spanning tree algorithm is $O(n^3)$.

Chapter 4

The Cycle Tree Based Algorithm

In this section, we present the new algorithm that we named cycle tree based algorithm for spare capacity design. It can guarantee higher restorability than the spanning tree algorithm described above. The new cycle tree algorithm is made of a spanning tree with backup parents and some extra *cycle edges* that form some cycles with the original spanning tree edges. We call the path that forms a cycle *cycle path.* The table "backup parents" keeps the unused adjacent edges for every node. When the nodes lose their connection to the tree, they can be switched to backup parents for other possible valid backup spare paths. More significantly, we add some redundant edges, which are called *cycle edges*, to the basic spanning tree to achieve higher restorability with very little additional time complexity and reasonable increased spare capacity. Here, as mentioned before, we assume one traffic demand between each pair of nodes in the network.

In the following two sections we will introduce the two main parts of the algorithm: tree construction procedure and spare capacity assignment & restorability calculation procedure.

4.1 Tree Construction

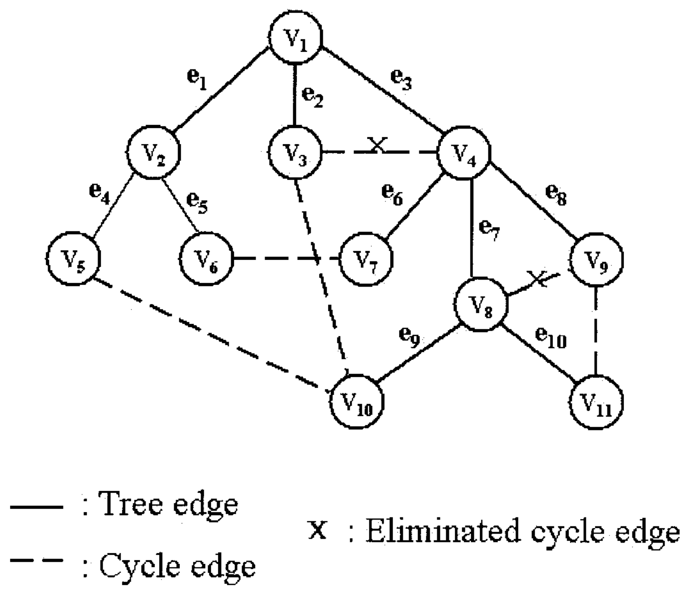


Figure 5: Tree edges and cycle edges on the spanning tree

Our cycle tree algorithm is like the spanning tree algorithm described in Chapter 3, excluding the application of the cycle edges. The construction of the new tree can be described as follows:

1. Find the shortest working path for every pair of nodes, using Dijkstra's algorithm, then get the connection cost table by calculating every link cost on every working path. We assume that the cost of every link on the traffic path is equal to 1.

2. Get the tree with backup parents and some extra cycle edges.

2.1 Sort the edges in a descending order to the edge array, according to the cost on every link.

2.2 If one of the following ending conditions happen: there are no more edges in the sorted edge array, or the number of edges in the tree edges set is $n-1$ and every tree edge is included in at least one cycle path (being composed of the tree edges and cycle edges together), go to **2.4**. For the second ending condition, it means that every edge on the tree can be protected by at least one cycle path for reaching some destination nodes. As an example, the tree edges and the cycle edges in Figure 5 satisfy the second ending condition mentioned previously. In this example graph, all tree edges are protected by at least one cycle path as follows:

◦ e_1, e_5, e_6 and e_3 are protected by the cycle path $(v_1, v_2, v_6, v_7, v_4, v_1)$ because of adding the cycle edge $v_6 - v_7$.

◦ e_2, e_9, e_7 and e_3 are protected by $(v_1, v_3, v_{10}, v_8, v_4, v_1)$ because of the cycle edge $v_3 - v_{10}$.

◦ e_7, e_8 and e_{10} are protected by $(v_4, v_8, v_{11}, v_9, v_4)$ because of the cycle edge $v_{11} - v_9$.

◦ e_1, e_4, e_9, e_7 and e_3 are protected by $(v_1, v_2, v_5, v_{10}, v_8, v_4, v_1)$ because of the cycle edge $v_5 - v_{10}$.

2.3 Take out the edge with the largest weight from the sorted edge array, check whether there will be a cycle with the selected edge and the other edges already on the tree. If there is no cycle, then add this edge to the tree edges set; otherwise, check whether this edge is already included in a bigger cycle, if the answer is "no", then we can add this edge as a cycle edge. Go to **2.2**.

2.4 Eliminate redundant cycle edges.

When a cycle edge is included in a bigger cycle, then it is redundant and can be eliminated. As the example showed in Figure 5, edge $v_3 - v_4$ is included in large cycle formed by another cycle edge $v_3 - v_{10}$ and the tree spare path $(v_3, v_1, v_4, v_8, v_{10})$, so it is eliminated. In the same way, cycle edge $v_8 - v_9$ is taken out. The remained cycle edges maybe not the optimal choice to generate as many as possible cycle paths from the cycle edges and tree edges, which will be researched further in the future.

3. Get spare paths for every pair of nodes just from the tree edges, which represent the original spare paths.

4. Get the table of backup parents for every node.

Check all adjacent nodes and edges, record maximum number of nodes (defined by the software) as backup parents.

At this time, it is important to mention that the cycle paths used in step 2 are formed by the tree edges and only one cycle edge, for its simpleness. However, we may use more than one cycle edge in a cycle path in creating the backup spare paths in the next part described in the following section. When the process is completed, we cannot assure that every tree edge is included in at least one cycle path (i.e., due to the lack of enough cycle edges), this shows that there is no guarantee that all pair of nodes are protected by these cycle paths. We will do more research in the future on cycle edge selection and ending conditions to get more powerful cycle edges.

From all the tree edges and the cycle edges, we can get some cycles using BFS (Breadth First Search)-like algorithm. In our algorithm, this part is included in the process "spare capacity assignment and restorability calculation". Its detailed description will be given in the next section.

4.2 Spare Capacity Assignment & Restorability Calculation

Following the obtaining of the spare tree topology, the table "backup parents" and the cycle edges, the next step is the spare capacity assignment and restorability calculation. These two functions are dealt within one procedure.

When we calculate the restorability, upon any possible single link failure, we should check whether there exists another available spare path for any affected working path. If the answer is "yes", then the traffic on this working path is protected by the spare topology. The spare path can be derived from the tree edges, or the tree edges and the cycle edges, and even from the backup parents. Finally, using the definition described in Chapter 2, we have the restorability for our new cycle tree based algorithm. Initially, we should assign enough spare capacity on every edge that appeared in the spare topology. Similarly, as the new spanning tree based algorithm described in Chapter 3, in addition to the process showed in Figure 3, this new algorithm adds cycle edge processing. The procedure of obtaining the cycle edges has already been described in the above section, and assigning enough capacity on these tree edges and backup edges is similar to the spanning tree algorithm described also in Section 3.2. Here, we put emphasis on how to get good and enough cycles from the tree edges and the cycle edges, and assign spare capacity on these edges.

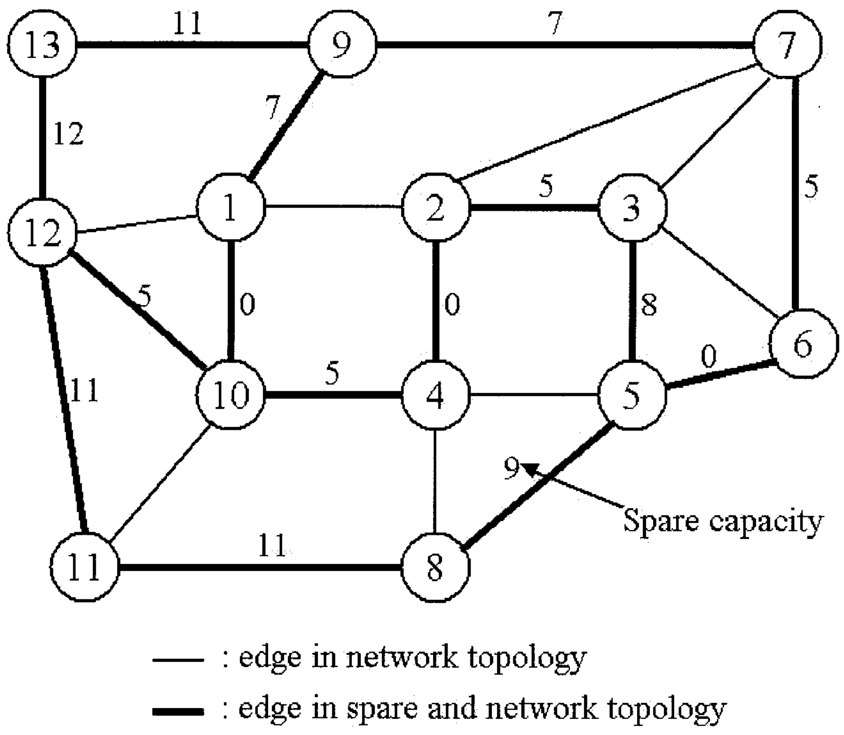
From the cycle edges and the tree edges we should get as many cycles as possible. Within one cycle, every pair of nodes can have two link-disjoint paths, which means that we can protect any single link failure between these pairs of nodes. If every pair of nodes appears at least in one cycle provided by the cycle edges and the tree edges, then every pair of nodes can be protected from any single link failure, then 100% restorability has been achieved. So more cycle paths imply higher restorability. But for an arbitrary network, we cannot guarantee that it has enough cycle edges, and every pair of nodes on the graph can have these backup spare paths. However, if the pair of nodes has the backup spare paths, it means that it can cover any single failure. That is the most important reason why we add these cycle edges to the spanning tree topology.

In our cycle tree based algorithm we use these cycle edges as backup spare edges to generate backup spare paths. We do not use these cycles directly as spare paths but as backup spare paths due to the fact that the length of paths gained from these cycles is usually longer than of the ones from the plain spanning tree. Hence, they will spend more spare capacity. Additionally, the cycle-oriented algorithms usually consume much more running time and are more complex than the tree based algorithms. Therefore, we try to combine them to gain both merits. However, we cannot guarantee 100% restorability.

How to use the cycle edges and the tree edges to form as many cycles as possible? First, we generate the simple cycles just between any cycle edge and the spare path between these two nodes, then get the two backup spare paths for every pair of nodes on these cycles. Second, we use BFS (Breadth First Search) based algorithm to find more cycle paths. We will not record whether one node has been searched already, so one node may be searched many times. The intent is to reach the source node to get a cycle, of course other cycles within one cycle path are avoided. When one branch reaches the root, it will stop searching, but the others will continue. Every node will be the root to begin one search process. After all the search process are finished, we have already recorded all possible cycles they have created. Later, we can obtain the all possible backup spare paths from these cycles. Some pairs of nodes may exist on more than one cycle, we just select one pair of backup spare paths. To assure to get more possible cycles, we use the above BFS based algorithm two times according to the serial number of nodes ascending and descending. Here we can show some example cycle paths from Figure 5, $p = (v_1, v_3, v_{10}, v_8, v_4, v_1)$ is a cycle, from it, we have two link-disjoint backup spare paths for the pair of nodes (v_1, v_{10}): $p_{b1} = (v_1, v_3, v_{10})$ and $p_{b2} = (v_1, v_4, v_8, v_{10})$. Similarly, we can get other link-disjoint backup spare paths for any pair of nodes on this cycle.

Before the process of spare capacity assignment and restorability calculation, we have already got the all possible backup spare paths gained from the cycles. Subsequently, similarly as described in the spanning tree algorithm in Figure 3, for every affected working path, the algorithm first checks the original spare path we got from the tree edges, if it works, then this pair of nodes is protected. Otherwise, the related backup parents should be checked in order to find a valid backup spare path. At last, if the above two steps do not work, we will check whether it has backup spare paths gained from the cycles, in case that the answer is positive, the one which does not include the failed link is selected; or if the two backup spare paths are all valid, the shorter one is chosen to consume less spare capacity. This pair of nodes is protected too. Of course, as we do in spanning tree algorithm, when a valid spare path or backup spare path is used, the algorithm should check whether the every link spare capacity is enough for the path, if not, we will add capacity for the path to be useful. As a result, we have the number of protected/restorable path (restorable traffic capacity) and the number of affected path (affected traffic capacity), and finally the restorability has been achieved. At this point, we select to check backup spare paths using backup parents first, then the paths from cycle edges. The reason for doing this is that the backup spare paths using backup parents are usually shorter than paths using cycle edges and therefore consume less spare capacity.

4.3 An Example



13
9
7
1
2
3
12
6
10
4
5
11
8
11
7
12
7
5
5
5
0
0
8
0
11
5
9
Spare capacity
11
: edge in network topology
: edge in spare and network topology

Figure 6: Network topology and spare topology connections

We use the first network with 13 nodes in Table 16 as an example to show the difference of spare capacity assignment among the three algorithms. The results are in Figure 6 and Figure 7. To simplify, we just show the results using an existing spare topology.

In our algorithm, for the basic spare tree topology (without using backup parents or cycle edges), the paths between every pair of nodes are bi-directional, so the spare capacities for the two directions on these tree edges are the same. Nevertheless, for

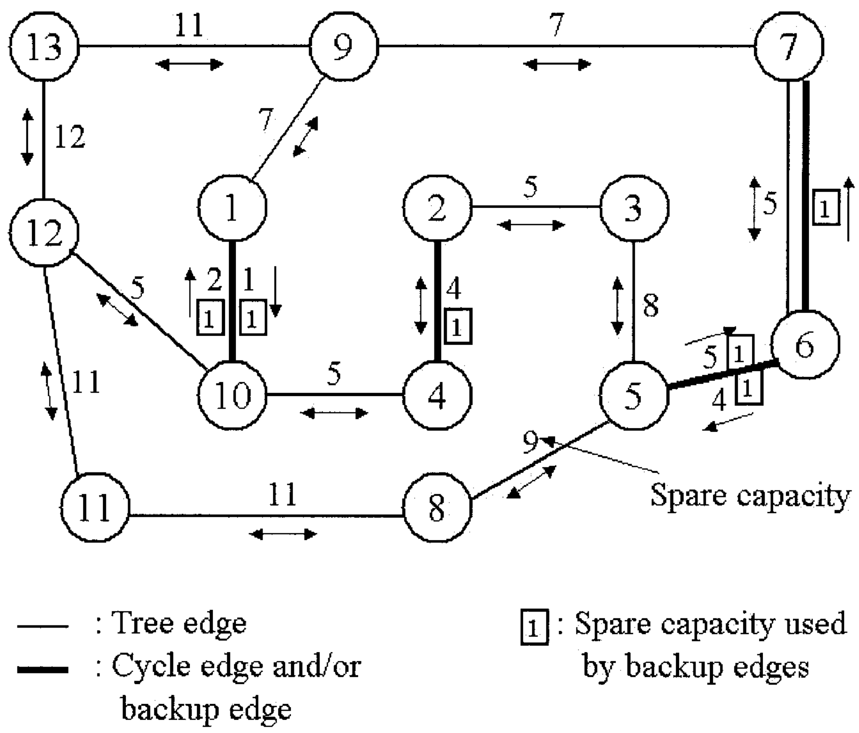

11
7
12
7
5
5
8
5
2
1
1
1
4
1
5
1
5
1
4
1
11
5
9
Spare capacity
11
13
9
7
1
2
3
12
6
10
4
5
11
8
: Tree edge
: Cycle edge and/or backup edge
1 : Spare capacity used by backup edges

Figure 7: Spare capacity assignment

backup edges, the backup spare paths are unidirectional due to the employment of different backup parent for the two reverse paths. Therefore, the spare capacities for the two reverse directions on these backup edges may not be the same. We use arrows to denote spare capacities of the corresponding direction.

Figure 6 presents the original network topology and spare topology. At the same time, the numbers on every spare link denote the spare capacity before using backup parents and cycle edges. The number 0 means that this edge is not used as a spare tree edge. For the basic tree topology and corresponding spare capacities of this example, the results of all of the three algorithm are the same. The spare capacities are for both directions. Adding them up for both directions, we get the total spare capacity of 192, as shown in Table 16.

Figure 7 presents the spare capacity assignments of the basic tree edges, the backup edges and the cycle edges for the spanning tree algorithm and the cycle tree algorithm, after applying backup parents or backup parents and cycle edges. The results for the hierarchical tree algorithm and the spanning tree algorithm are the same, because of the same tree structure and backup edges. The numbers on every tree edge denote the basic spare capacity for this tree edge. Also, the numbers on cycle edges or backup edges imply the total spare capacity needed for this edge. Therein, the spare capacity on each needed edge used only by the backup parents is also shown in the figure. By adding them respectively we can compute the total spare capacity used by the spanning tree algorithm and the hierarchical tree algorithm, which totals 199, and of the cycle tree algorithm totals 213. The difference is used by the cycle edges, which is 14.

4.4 Time and Space Complexity

Let n denotes the number of nodes, and m denotes the number of edges of graph G. As we know, $m = O(n^2)$ in the worst case. The following numbers of steps are consistent with the steps in Section 4.1.

Step 1, the running time is $O(n^3)$.

Step 2.1, the upper bound is $O(m \cdot \log m) = O(n^2 \log n)$.

Step 2.2 to 2.4, the running time is $O(m \cdot (n + n)) = O(n^3)$.

Step 3, the bound is $O(n^3)$.

Step 4, the time is $O(n^2)$.

All added up, the time complexity for tree construction is $O(n^3)$.

We turn to the procedure "spare capacity assignment and restorability calculation" at this point. Compared to the similar procedure in spanning tree algorithm, based on the algorithm described in Section 4.2, except the time complexity in spanning tree algorithm, only the cycle generation process dominates the increased time complexity. Let us see how much it can contribute. To check all possible cycles from one root, we need $O(n^3)$ running time, every node should be as root once, totally, the time complexity for the cycle generation process is $O(n^4)$. As for the running time of the process of getting backup spare paths from the cycles, compared to the running time of cycle generation procedure, it can be ignored. In total, adding the tree construction part, and the similar part for "spare capacity assignment and restorability calculation" in spanning tree algorithm, the whole algorithm time complexity is $O(n^3) + O(n^4) + O(n^4) = O(n^4)$.

Here, we stress on the algorithm itself, and just give the very loose upper bound for the time complexity. There are many methods to give more precise analysis to get more exact running time, i.e. "amortized analysis", "accounting method", "potential method" [22].

Based on the above time complexity analysis, the new cycle tree based algorithm has the same upper bound running time as the spanning tree algorithm : $O(n^4)$, although the cycle tree algorithm had just added some more time complexity to obtain the cycle edges and get backup spare paths from the cycle edges and the tree edges. However, from the simulation results, the restorability of the cycle tree algorithm is much higher than of the spanning tree algorithm and the hierarchical tree algorithm.

Concerning the space complexity, for the tree construction process, excluding the edge connection matrix of $O(n^2)$, the tree edge connection matrix of $O(n^2)$, the backup parents table of $O(n)$ as used in spanning tree algorithm, additionally we need another matrix to record the cycle edges of $O(n^2)$. Entirely, for this part, the space complexity is $O(n^2)$. For the spare capacity assignment and restorability calculation part, we also assume that the maximum length of these paths is $n-1$, the same as in spanning tree algorithm, and the memory for recording working paths and spare paths is $O(n^3)$. We need to add memory to record the all possible cycle paths among the tree edges and cycle edges, and the possible two backup spare paths for every pair of nodes. It needs $O(n^3)$ space for each. Adding them up, we have $O(n^3)$ space complexity for the spare capacity assignment and restorability calculation process. Finally, the upper bound of space complexity for the cycle tree based algorithm is $O(n^3)$.

Chapter 5

Simulation and Comparison

5.1 Topology Models

For a new algorithm related to networks, one of its most important question is whether it works well on real networks. In some cases, a model could impact the performance of an algorithm greatly. In our simulation, we consider two classes of network generators. The first category is random graph generators. One of the most commonly used models for generating random networks is represented by Waxman generator [19]. The model "pure random graph" used in our simulation is generated randomly by the generator "GT-ITM" (we will explain this and other generators later). Another random network model we used is AS-level (Autonomous System, compared to Internet Router-Level (RL) topology) model, where ASs are nodes and edges that represent peering relationships between ASs. It is also generated randomly, by the generator

"Inet". For illustrating the use of existing spare topology, we take advantage of three examples from Adel Al-Rumaih [7], also these example networks are generated randomly. When using the pure random model, the most important parameter is the probability of an edge between any pair of nodes.

The second category is the structural generators such as Transit-stub and Tiers generators [18, 19] which reflect the structure of the Internet. We use the figure from [18] to give an example of an Internet structure. Transit-stub creates a number of top-level transit domains within which nodes are connected randomly. Attached to each transit domain are several similarly generated stub domains. Additional stub-to-transit and stub-to-stub links are added randomly based upon a specified parameter. WAN, MAN and LAN, stand for Wide Area Network, Metropolitan Area Network and Local Area Network respectively, make up the graphs created by the Tiers model. Tiers uses a somewhat different procedure to create network topology. First, it creates a number of top level networks (WAN), to each of which are attached several intermediate tier networks (MAN). Similarly, several LANs are randomly attached to each intermediate tier network. Within each tier (except the LAN), Tiers uses a minimum spanning tree to connect all the nodes, then adds additional links in order of increasing inter-node Euclidean distance. LAN nodes are connected using a star topology, additional inter-tier links are added randomly based upon a specified parameter. From their generating methods, these models cannot be guaranteed to be at least two-connected for every node (mesh-like). We used both Transit-stub and Tiers models for our simulation, and the results are described in the following tables.

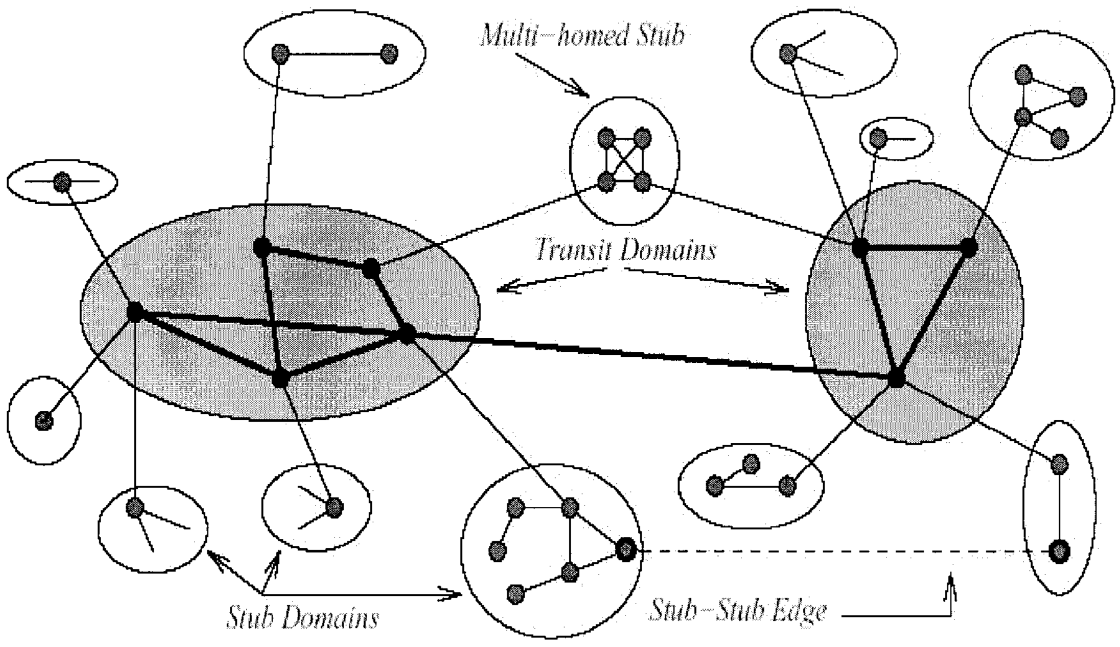


Figure 8: Example of Internet domain structures

The network simulator "NS-2" has a set of topology generators for network simulations. In NS-2 one may create a topology using one of the four methods: Inet, GT-ITM, Tiers and other topology generators. To compare our two new algorithms with others, four different network topology models from NS-2 are considered: Inet, GT-ITM pure random, GT-ITM Transit-stub and Tiers. The generator is available on the Web [21].

At this point, we introduce the generators from NS-2 we used in our simulations. First, we use the "Inet" from University of Michigan to generate an AS-level representation of the Internet that just provides the connectivity information. It generates random networks with characteristics similar to the Internet. We should mention here that the original generator can just generate large scale Internet topology which is larger than 3037 nodes. Because of the memory and the speed of our computer, we expect to do the simulations with several hundreds of nodes. Therefore, from the networks generated by Inet, we selected the subset and add some connections to maintain a connected graph. The main parameters are as follows:

- N, the total number of nodes in the topology
- k, the fraction of degree-one nodes.
- sd, the seed to initialize the random number generator

The generator "GT-ITM" developed by Georgia Institute of Technology can create flat random graphs and two types of hierarchical graphs: N-level and Transit-stub. We use two of them: flat random and the Transit-stub. In our simulations the pure random model from the flat random model selections is used to generate pure random graphs. Using this model generator, the other main parameters are the total number of nodes and the probability of an edge between any two nodes. When using the Transit-stub model, the user could employ the following parameters:

- The number of transit domains and the number of stub domains connected to each transit node.
- The number of transit-stub and stub-stub edges.
- The number of nodes in the transit domains and the stub domains.

• The probability of an edge between each pair of nodes in the transit domains and stub domains.

The generator Tiers created by Matthew B. Doar is used to generate tier graphs. The major parameters chosen for this model are:

• N_W, the number of WANs; N_M, the number of MANs per WAN; and N_L, the number of LANs per MAN. To simplify, the number N_W is set as 1 to generate one connected WAN graph.

• SN_W, the number of nodes in a WAN; SN_M, the number of nodes per MAN; and SN_L, per LAN.

Accordingly, the total number of nodes in the graph is given by $N = SN_W + N_M \times SN_M + N_M \times N_L \times SN_L$, with $N_W = 1$.

5.2 Comparison of Simulation Results

5.2.1 Terms and Simulation Results

For the three tree-based algorithms used in simulation only one traffic demand between each pair of nodes in the network was assumed. The terms used in the tables of simulation results are defined as follows: N denotes the total number of nodes in the graph. *Average degree* means the average number of adjacent nodes per node within the whole network graph. *Before backup spare paths* and *after backup spare paths* denote respectively the results before and after using backup spare paths gained from the backup parents and cycle edges. *Restorability* is the percentage of the working bandwidth that could be restored in case of a single failure, defined as before: $\frac{restorable\ traffic\ capacity}{affected\ traffic\ capacity}$. *Work capacity* is defined as the sum of work bandwidth on every link using the shortest path routing. Similarly, *spare capacity* is defined as the sum of spare capacity on every spare link. We have used 5 topology models to do the simulation. The detailed descriptions of the models are given in the last section and [18, 19, 20].

Altogether, we generate 4 topologies using Inet generator; 2 kinds of topologies for 100 nodes and 2 for 200 nodes. Using GT-ITM generator, we have 4 kinds of topologies for pure random model and 4 for transit-stub model, including 2 topologies for 100 nodes and 2 for 200 nodes. And we use Tiers generator to get 2 topologies with one for 106 nodes and the other for 200 nodes. For every topology we generate 10 example graphs to do the simulations. Adding the three examples from [7], we have used 15 topologies, that is 143 networks as examples to do our simulations. The following tables are some of the simulation results, with the 2 Inet topologies, 3 GT-ITM pure random topologies, 1 transit-stub topology, 1 tiers topology and the topologies from [7].

The meaning of the abbreviations in the following tables are:

N : Number of nodes in the graph

Av. : Average

H.T. : Hierarchical Tree algorithm with backup parents

S.T. : Spanning Tree algorithm with backup parents

S.T.C. : Cycle Tree algorithm with backup parents and some cycle edges

Graph Model : AS-level Internet topology (generated by Inet)								
N	Av. Degree	Work Capacity	Restorability					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
100	3.62	19058	23.86%	23.86%	23.86%	62.22%	62.03%	68.86%
100	3.92	18792	22.49%	22.49%	22.49%	61.63%	61.34%	74.90%
100	3.58	19096	25.01%	25.01%	25.01%	62.94%	62.62%	76.15%
100	3.74	18604	20.87%	20.87%	20.87%	60.66%	60.54%	68.73%
100	3.84	18992	21.66%	21.66%	21.66%	61.17%	60.91%	73.03%
100	3.7	19096	20.52%	20.52%	20.52%	60.48%	60.35%	73.82%
100	3.86	18782	17.95%	17.95%	17.95%	59.09%	59.08%	67.19%
100	3.76	18994	24.02%	24.02%	24.02%	62.40%	62%	68.65%
100	3.94	18816	17.15%	17.15%	17.15%	58.79%	58.64%	67.11%
100	3.9	18928	32.00%	32.00%	32.00%	66.56%	66.26%	71.37%
Av.	3.79	18916	22.55%	22.55%	22.55%	61.59%	61.39%	70.98%

Table 1: AS-level Internet topology networks with 100 nodes – restorability

Graph Model : AS-level Internet topology (generated by Inet)								
N	Av.	Work	Spare Capacity					
			Before backup spare paths			After backup spare paths		
	Degree	Capacity	H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
100	3.62	19058	1742	1742	1742	4281	2859	4297
100	3.92	18792	1560	1560	1560	4323	2738	5662
100	3.58	19096	2158	2158	2158	4716	3354	6694
100	3.74	18604	1948	1948	1948	4602	2841	4823
100	3.84	18992	1624	1624	1624	4580	2859	6076
100	3.7	19096	1536	1536	1536	4365	2989	6454
100	3.86	18782	1352	1352	1352	3987	2644	4480
100	3.76	18994	1824	1824	1824	4364	3073	4540
100	3.94	18816	1196	1196	1196	4104	2420	4463
100	3.9	18928	3094	3094	3094	5547	4046	5317
Av.	3.79	18916	1803	1803	1803	4487	2982	5281

Table 2: AS-level Internet topology networks with 100 nodes – spare capacity

Graph Model : AS-level Internet topology (generated by Inet)								
N	Av. Degree	Work Capacity	Restorability					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
200	3.2	84774	21.28%	21.28%	21.28%	61.24%	61.25%	78.21%
200	3.54	83902	22.74%	22.74%	22.74%	61.95%	61.98%	88.90%
200	3.06	83582	22.54%	22.54%	22.54%	61.80%	61.81%	81.46%
200	3.2	83112	24.15%	24.15%	24.15%	63.13%	63.16%	77.79%
200	3.2	83470	22.74%	22.74%	22.74%	62.32%	62.34%	82.60%
200	3.46	83252	25.12%	25.12%	25.12%	63.50%	63.49%	73.17%
200	3.38	82664	17.08%	17.08%	17.08%	59.07%	59.02%	90.60%
200	3.5	85132	26.80%	26.80%	26.80%	64.11%	64%	79.54%
200	3.5	82614	18.29%	18.29%	18.29%	59.42%	59.47%	75.45%
200	3.38	84666	29.10%	29.10%	29.10%	65.56%	65.59%	77.12%
Av.	3.34	83717	22.98%	22.98%	22.98%	62.21%	62.22%	80.48%

Table 3: AS-level Internet topology networks with 200 nodes – restorability

Graph Model : AS-level Internet topology (generated by Inet)								
N	Av. Degree	Work Capacity	Spare Capacity					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
200	3.2	84774	8144	8144	8144	19726	14045	33224
200	3.54	83902	7760	7760	7760	19810	13292	47279
200	3.06	83582	7894	7894	7894	19908	12991	38740
200	3.2	83112	10420	10420	10420	21053	15582	32759
200	3.2	83470	9752	9752	9752	20497	15490	40732
200	3.46	83252	9816	9816	9816	20484	14300	24592
200	3.38	82664	5734	5734	5734	18014	11380	51556
200	3.5	85132	9236	9236	9236	20638	14253	30194
200	3.5	82614	6724	6724	6724	19986	12471	29928
200	3.38	84666	12664	12664	12664	23100	17139	28536
Av.	3.34	83717	8814	8814	8814	20322	14094	35754

Table 4: AS-level Internet topology networks with 200 nodes – spare capacity

Graph Model : Pure Random (generated by GT-ITM)								
N	Av. Degree	Work Capacity	Restorability					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
100	8.92	23086	72.56%	72.56%	72.56%	88.16%	87.66%	90.68%
100	7.72	24468	73.97%	74.01%	74.01%	88.79%	88.22%	91.14%
100	9.28	22660	73.96%	74.07%	74.07%	88.66%	88.27%	90.82%
100	5.5	28676	67.30%	67.33%	67.33%	84.22%	83.77%	86.65%
100	4.62	31088	63.60%	63.60%	63.60%	81.31%	81.49%	84.57%
100	6.5	26116	70.30%	70.30%	70.30%	86.58%	86.11%	90.13%
100	6.24	26722	71.29%	71.29%	71.29%	86.95%	86.81%	90.18%
100	6.54	26100	72.80%	72.74%	72.74%	87.72%	87%	89.90%
100	6.18	26832	70.21%	70.04%	70.04%	86.29%	86.04%	88.93%
100	5.56	28580	66.39%	66.32%	66.32%	84.06%	83.79%	86.69%
Av.	6.71	26433	70.24%	70.23%	70.23%	86.27%	85.92%	88.97%

Table 5: Pure random topology networks with 100 nodes – restorability

Graph Model : Pure Random (generated by GT-ITM)								
N	Av. Degree	Work Capacity	Spare Capacity					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
100	8.92	23086	4436	4436	4436	5873	5475	5899
100	7.72	24468	5376	5398	5398	6515	6391	6839
100	9.28	22660	4124	4146	4146	5238	5046	5430
100	5.5	28676	6778	6796	6796	8518	8308	8734
100	4.62	31088	7510	7510	7510	10120	10065	10533
100	6.5	26116	5742	5742	5742	7349	7148	7932
100	6.24	26722	6362	6362	6362	7766	7632	8130
100	6.54	26100	5582	5620	5620	6898	6748	7159
100	6.18	26832	6232	6210	6210	7518	7446	7835
100	5.56	28580	7102	7126	7126	9165	8985	9391
Av.	6.71	26433	5924	5935	5935	7496	7324	7788

Table 6: Pure random topology networks with 100 nodes – spare capacity

Graph Model : Pure Random (generated by GT-ITM)								
N	Av.	Work	Restorability					
			Before backup spare paths			After backup spare paths		
	Degree	Capacity	H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
100	3.54	37812	54.74%	54.41%	54.41%	73.72%	73.89%	76.49%
100	3.48	37124	51.97%	52.23%	52.23%	72.48%	72.49%	75.59%
100	3.3	37124	56.10%	56.19%	56.19%	74.27%	74.30%	77.87%
100	3.5	37454	51.07%	51.23%	51.23%	72.94%	72.66%	75.94%
100	3.34	38946	50.75%	50.62%	50.62%	70.18%	69.91%	72.90%
100	3.64	35670	56.91%	56.91%	56.91%	74.21%	74.49%	77.43%
100	3.06	44450	52.77%	52.77%	52.77%	70.09%	70.09%	73.43%
100	3.36	39664	55.34%	55.34%	55.34%	73.76%	73.76%	76.72%
100	3.26	38330	51.41%	51.64%	51.64%	69.85%	69.60%	72.08%
100	3.4	38312	53.53%	53.53%	53.53%	71.78%	71.57%	74.75%
Av.	3.39	38489	53.46%	53.49%	53.49%	72.33%	72.28%	75.32%

Table 7: Another pure random topology networks with 100 nodes – restorability

Graph Model : Pure Random (generated by GT-ITM)								
N	Av.	Work	Spare Capacity					
			Before backup spare paths			After backup spare paths		
	Degree	Capacity	H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
100	3.54	37812	10876	10436	10436	16101	15579	16013
100	3.48	37124	9524	9824	9824	13847	13660	14104
100	3.3	37124	11790	11750	11750	15415	15358	15753
100	3.5	37454	9620	9792	9792	13895	13910	14323
100	3.34	38946	10394	10414	10414	14779	15022	15412
100	3.64	35670	9116	9116	9116	12530	12271	12663
100	3.06	44450	13776	13776	13776	20006	19910	20497
100	3.36	39664	11552	11552	11552	15217	15396	15798
100	3.26	38330	10040	10124	10124	15916	15600	15920
100	3.4	38312	10040	10040	10040	14646	14484	14876
Av.	3.39	38489	10673	10682	10682	15235	15119	15536

Table 8: Another pure random topology networks with 100 nodes – spare capacity

Graph Model : Pure Random (generated by GT-ITM)								
N	Av. Degree	Work Capacity	Restorability					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
200	8.43	107634	79.36%	79.29%	79.29%	90.79%	90.95%	92.61%
200	7.19	115130	76.65%	76.65%	76.65%	89.49%	89.42%	90.94%
200	7.95	110852	77.07%	77.08%	77.08%	89.80%	89.79%	91.26%
200	9.77	102148	79.86%	79.89%	79.89%	91.10%	90.99%	92.38%
200	6.4	121188	75.16%	75.19%	75.19%	88.59%	88.62%	90.45%
200	7.94	110676	78.51%	78.50%	78.50%	90.43%	90.44%	92.06%
200	7.44	113048	78.10%	78.07%	78.07%	90.34%	90.20%	91.86%
200	9.65	102244	79.15%	79.14%	79.14%	90.82%	90.75%	92.16%
200	8.74	106170	78.18%	78.19%	78.19%	90.22%	90.22%	92.02%
200	7.13	114910	73.32%	73.35%	73.35%	87.72%	87.80%	89.33%
Av.	8.06	110400	77.54%	77.53%	77.53%	89.93%	89.92%	91.51%

Table 9: Pure random topology networks with 200 nodes – restorability

Graph Model : Pure Random (generated by GT-ITM)								
N	Av. Degree	Work Capacity	Spare Capacity					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
200	8.43	107634	18940	19122	19122	22462	22694	23842
200	7.19	115130	21372	21372	21372	25871	25266	26274
200	7.95	110852	19270	19274	19274	24191	23404	24361
200	9.77	102148	17162	17112	17112	20161	19826	20775
200	6.4	121188	24334	24336	24336	29948	29438	30591
200	7.94	110676	20004	20140	20140	23513	23448	24497
200	7.44	113048	21240	21164	21164	26363	26154	27192
200	9.65	102244	16254	16230	16230	19793	19273	20282
200	8.74	106170	18780	18790	18790	22553	22044	23235
200	7.13	114910	21616	21564	21564	27454	27008	27941
Av.	8.06	110400	19897	19910	19910	24231	23856	24899

Table 10: Pure random topology networks with 200 nodes – spare capacity

Graph Model : Transit-stub (generated by GT-ITM)								
N	Av.	Work	Restorability					
			Before backup spare paths			After backup spare paths		
	Degree	Capacity	H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
100	5.3	43010	13.74%	13.74%	13.74%	48.18%	48.07%	48.88%
100	5.06	45084	14.63%	14.63%	14.63%	47.24%	47.23%	48.35%
100	4.48	48230	6.45%	6.45%	6.45%	30.18%	30.18%	30.82%
100	4.58	47648	13.23%	13.23%	13.23%	45.75%	45.75%	46.76%
100	4.54	44402	7.91%	7.91%	7.91%	40.38%	40.38%	41.15%
100	5.22	45840	14.28%	14.28%	14.28%	37.43%	37.43%	38.16%
100	5.12	44642	11.34%	11.34%	11.34%	42.20%	42.02%	43.09%
100	4.22	45642	16.75%	16.75%	16.75%	46.05%	45%	46.33%
100	4.64	44854	12.45%	12.45%	12.45%	42.43%	42.45%	43.09%
100	4.44	46790	7.20%	7.20%	7.20%	38.65%	38.65%	39.58%
Av.	4.76	45614	11.80%	11.80%	11.80%	41.85%	41.74%	42.62%

Table 11: Transit-stub topology networks with 100 nodes – restorability

Graph Model : Transit-stub (generated by GT-ITM)								
N	Av.	Work	Spare Capacity					
			Before backup spare paths			After backup spare paths		
	Degree	Capacity	H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
100	5.3	43010	9952	9952	9952	24627	22792	23016
100	5.06	45084	9538	9538	9538	27004	24999	25350
100	4.48	48230	4508	4508	4508	15967	15645	15882
100	4.58	47648	10350	10350	10350	26350	25883	26205
100	4.54	44402	4086	4086	4086	22526	20867	21092
100	5.22	45840	7696	7696	7696	19903	19220	19473
100	5.12	44642	5456	5456	5456	21761	20647	21011
100	4.22	45642	10336	10336	10336	24764	23905	24163
100	4.64	44854	10448	10448	10448	23908	23019	23172
100	4.44	46790	4948	4948	4948	24195	23526	23792
Av.	4.76	45614	7732	7732	7732	23101	22050	22316

Table 12: Transit-stub topology networks with 100 nodes – spare capacity

Graph Model : Tiers– WAN-MAN-LAN (generated by Tiers)								
N	Av. Degree	Work Capacity	Restorability					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
200	2.42	279168	9.34%	9.34%	9.34%	36.74%	36.74%	36.86%
200	2.49	264082	7.09%	7.09%	7.09%	38.19%	38.19%	38.25%
200	2.4	271346	10.01%	10.01%	10.01%	37.38%	37.38%	37.47%
200	2.06	350758	1.12%	1.12%	1.12%	15.10%	15.10%	15.11%
200	2.21	294898	7.76%	7.76%	7.76%	23.71%	23.71%	23.80%
200	2.41	321978	4.04%	4.04%	4.04%	31.13%	31.13%	31.17%
200	2.61	264890	15.08%	15.08%	15.08%	44.37%	44.37%	44.51%
200	2.46	267124	7.01%	7.01%	7.01%	35.40%	35.40%	35.49%
200	2.59	293428	8.60%	8.60%	8.60%	35.21%	35.21%	35.31%
200	2.35	317328	3.40%	3.40%	3.40%	26.63%	26.63%	26.66%
Av.	2.40	292500	7.35%	7.35%	7.35%	32.39%	32.39%	32.46%

Table 13: Tiers topology networks with 200 nodes – restorability

Graph Model : Tiers– WAN-MAN-LAN (generated by Tiers)								
N	Av. Degree	Work Capacity	Spare Capacity					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
200	2.42	279168	59440	59440	59440	214112	209845	209980
200	2.49	264082	56528	56528	56528	234047	231706	231789
200	2.4	271346	74892	74892	74892	221625	209356	209495
200	2.06	350758	28582	28582	28582	273606	273606	273617
200	2.21	294898	64242	64242	64242	201318	191358	191468
200	2.41	321978	58910	58910	58910	271973	270527	270577
200	2.61	264890	83844	83844	83844	199553	199167	199374
200	2.46	267124	42322	42322	42322	203768	199720	199826
200	2.59	293428	71578	71578	71578	219155	217841	217991
200	2.35	317328	27218	27218	27218	227022	224249	224274
Av.	2.40	292500	56756	56756	56756	226618	222738	222839

Table 14: Tiers topology networks with 200 nodes – spare capacity

Graph Model : Three examples (with existing spare topology)								
N	Av. Degree	Work Capacity	Restorability					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
13	3.54	324	71.61%	71.61%	71.61%	80.56%	80.56%	98.77%
17	3.65	640	75.00%	75.00%	75.00%	80.94%	80.78%	100.00%
20	3.7	966	73.29%	73.29%	73.29%	79.92%	79.92%	99.79%
Av.	3.63		73.30%	73.30%	73.30%	80.47%	80.42%	99.52%

Table 15: A set of example networks with existing spare topology – restorability

Graph Model : Three examples (with existing spare topology)								
N	Av. Degree	Work Capacity	Spare Capacity					
			Before backup spare paths			After backup spare paths		
			H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
13	3.54	324	192	192	192	199	199	213
17	3.65	640	342	326	326	350	334	371
20	3.7	966	552	552	552	569	569	639
Av.	3.63							

Table 16: A set of example networks with existing spare topology – spare capacity

Graph Model : Three examples (without spare topology)								
			Restorability					
N	Av.	Work	Before backup spare paths			After backup spare paths		
	Degree	Capacity	H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
13	3.54	324	43.21%	43.21%	43.21%	74.69%	74.69%	89.82%
17	3.65	640	45.00%	45.00%	45.00%	74.53%	74.53%	86.56%
20	3.7	966	47.00%	47.00%	47.00%	79.30%	79.30%	90.06%
Av.	3.63		45.07%	45.07%	45.07%	76.17%	76.17%	88.81%

Table 17: The same set of example networks without existent spare topology – restorability

Graph Model : Three examples (without spare topology)								
			Spare Capacity					
N	Av.	Work	Before backup spare paths			After backup spare paths		
	Degree	Capacity	H. T.	S. T.	S.T.C.	H. T.	S. T.	S.T.C.
13	3.54	324	126	126	126	181	181	207
17	3.65	640	238	238	238	351	336	368
20	3.7	966	360	360	360	545	526	569
Av.	3.63							

Table 18: The same set of example networks without existent spare topology – spare capacity

5.2.2 Comparison between Spanning Tree Based Algorithm and Hierarchical Tree Algorithm

We have used 5 models to do the simulations. In Inet model, from the Table 1 – Table 4, we compared our new spanning tree based algorithm with the hierarchical tree algorithm, concluding that two restorabilities are the same for *before backup spare paths.* In addition, for *after backup spare paths*, the restorability of spanning tree based algorithm is very slightly higher or lower than the hierarchical tree algorithm. This can be explained by their similar tree structures. However, we notice that their spare capacities are much different for *after backup spare paths.* In one set of examples the spanning tree algorithm can save an average of 33.5% of spare capacity compared to the hierarchical tree algorithm. This is because the different sequence of the backup parents brings on the different length in backup spare paths. In the hierarchical algorithm, the order of the backup parents is according to the spare capacity to the root. Whereas, in the spanning tree algorithm, we just select the backup parents in term of the serial numbers of nodes. But in the graph generated from the generators mentioned above, the sequence of the serial numbers is usually determined by the degree of the nodes, but not precisely. That is, the node 0 has the highest degree, the node 1 takes second place, $\cdots$, and so on. Consequently, from this analysis, we concluded that selecting the backup parents with higher degree is much more efficient on spare capacity, because they usually bring shorter backup spare paths than the nodes with lower degree.

The example results of GT-ITM pure random model are present in Table 5 – Table 10. From these results, we see that the restorabilities of the two algorithms are very similar (average difference is about 0.1%), no matter *before backup spare paths* or *after backup spare paths*. At the same time, although the needed spare capacities in spanning tree algorithm *before backup spare paths* are almost a little more than that in hierarchical tree algorithm, they are all a little less *after backup spare paths* with the same slight lower restorability. These also demonstrate that the spanning tree algorithm can economize some more spare capacity than the hierarchical algorithm *after backup spare paths*. Most importantly, we use this model to compare the restorabilities themselves with different average degrees, the details are in Section 5.2.3.

For the model GT-ITM transit-stub, the results are in Table 11 and Table 12. *Before backup spare paths*, the restorability and the spare capacity of the two algorithms are the same. *After backup spare paths*, although the restorability of the spanning tree algorithm is on average 0.11% lower than that of hierarchical tree algorithm, it economizes about 4.5% spare capacity.

Table 13 and Table 14 show the results of examples expressed by model Tiers. *Before backup spare paths*, the restorability and the spare capacity of the two algorithms are the same. *After backup spare paths*, the two average restorabilities are almost the same (the restorability of spanning tree algorithm is 0.01% higher than that of hierarchical tree algorithm), but the spanning tree algorithm save an average of 1.7% more spare capacity than the other one.

Regarding the three examples from another paper [7], because the number of nodes is not the same, we do not give the average spare capacity. These models are mainly used for the comparison between the network with or without existing spare topology. As we have already mentioned in Section 3.1, using the existing spare topology, we can get higher restorability, no matter *before backup spare paths* or *after backup spare paths.* In addition, for these three examples, with the existing spare topology, there is only one example just *after backup spare paths*, the restorability of the spanning tree algorithm is a little lower than that of hierarchical tree algorithm and the spare capacity is also smaller. For the other conditions, with or without existing spare topology, *before backup spare paths* or *after backup spare paths*, the restorability of the spanning tree algorithm is the same as that of the hierarchical tree algorithm, with the same or less spare capacity.

Generally, with almost the same restorability, the spanning tree algorithm always saves more spare capacity in comparison to the other one. This means that if the capacity is a big issue in a given network, then our new spanning tree algorithm is the better one to use, since it can reach almost the same restorability using much less spare capacity.

5.2.3 Comparison between Cycle Tree Algorithm and Hierarchical Tree Algorithm, Spanning Tree Algorithm

For every model, the cycle tree based algorithm and the spanning tree algorithm are all using the maximum spanning tree method as their basic mode to generate the basic tree topology, so *before backup spare paths*; the restorability and the spare capacity for each of them are the same. The comparisons between the spanning tree algorithm and the hierarchical tree algorithm are already done in Section 5.2.2. Therefore, all comparisons in the following paragraph are for *after backup spare paths.* Because the cycle tree based algorithm expands the new spanning tree algorithm with some cycle edges, the restorability is always no less than that of the spanning tree algorithm. And for all the results, the spare capacity of cycle tree algorithm is always more than that of the spanning tree algorithm.

In Inet model, the restorability of cycle tree based algorithm is always much higher than that of the hierarchical tree algorithm. In one topology, the difference achieves 27%. The difference reaches an average of 18% in this set of topologies. Nevertheless, simultaneously, the spare capacity needed is much more than the hierarchical tree algorithm. Compared to the increase of 18% in restorability, the average difference in spare capacity between them is about 76%. Compared to the spanning tree algorithm, the increased spare capacity even reaches 154%. This is a 200 nodes network. For another example topology with 100 nodes, the increased restorability is 10%, with the increased spare capacity 18%. Compared to the spanning tree algorithm, the spare capacity has increased by 77% with the same 10% higher restorability. The increased spare capacity is strongly dependent on the network topology itself and the size of the network. From the above analysis, with the restorability increased, there is more need for spare capacity. The reason is that the backup spare paths generated by the cycle edges and the tree edges usually have much longer path lengths, especially for larger graph.

From those results in Table 5 – Table 10 for GT-ITM pure random model, the restorability of the cycle tree algorithm is always higher than that of hierarchical tree algorithm. The differences of restorability in the three kinds of topologies, between the cycle tree algorithm and the other two, are at the same level; with the average of 2.5%. The spare capacity is a little larger too; the average increased amount is about 2.9%. In this kind of model, compared to the hierarchical tree algorithm and the spanning tree algorithm, the restorability of the cycle tree algorithm is always higher and proportional to the increased spare capacity.

We use this model to indicate that the network redundancy can have a great effect on the restorability. In our simulation network examples, the average degree is used to indirectly reflect the network redundancy. From the theory and our simulation results, for higher redundancy network topologies with the same size and same kind of model, using any of the three algorithms, we can usually get higher restorability than in lower redundancy networks. Obviously, this happens because more redundant edges can generate more efficient tree topology, more backup connections and more cycle paths, therefore we can get more efficient spare paths and more valid backup spare paths.

For the model GT-ITM transit-stub , the results are in Table 11 and Table 12. The restorability of the cycle tree algorithm is 0.77% higher than that of the hierarchical tree algorithm, furthermore it saves about 3.4% more spare capacity. In this model the cycle tree algorithm needs less spare capacity than the hierarchical tree algorithm with a little higher restorability. Compared to the spanning tree algorithm, the cycle tree algorithm use about 1.2% more spare capacity to reach 0.88% higher restorability, which sounds very reasonable.

The results of examples expressed by model Tiers are shown in Table 13 and Table 14. The spare capacity of the cycle tree algorithm is less than that of the hierarchical tree algorithm about 1.67%, with 0.05% higher restorability. Compared to the spanning tree algorithm, the cycle tree algorithm uses about 0.05% more spare capacity to reach 0.05% higher restorability. This is very similar with the above transit-stub model, because they have similar hierarchical structures.

For the three examples from another paper [7], excluding the comparison in the above section, we conclude that the restorability using cycle tree algorithm is always much higher than that of the other two algorithms, with very little increased spare capacity. On average, the increased restorability is 19% for existent spare topology, and 12.6% for non existent one. Compared to the restorability, the increased spare capacity is more economical than of other models. In one example, it just consumes 6% more spare capacity to get 18% higher restorability than that of the hierarchical tree algorithm, using existing spare topology.

Commonly, although the difference of restorability among the three algorithms may be more or less, and the restorability itself may be higher or lower, the cycle tree based algorithm has always higher restorability than the other two algorithms, with very reasonable increased spare capacity.

Chapter 6

Conclusions and Future Work

In this thesis, we introduced two new spare capacity design algorithms: spanning tree based algorithm with backup parents, and cycle tree based algorithm with backup parents and some cycle edges. In addition to the hierarchical tree algorithm that we used for comparison, altogether there are three algorithms presented in this thesis. All of them can be used as pre-planned path restoration algorithms.

From the analysis in the preceding sections, the time complexities for the two new algorithms introduced in this thesis are the same as of the hierarchical tree algorithm: $O(n^4)$. The space complexity of the three algorithms is the same too: $O(n^3)$.

The spanning tree based algorithm with backup parents needs much less spare capacity than the hierarchical tree algorithm, which can sometimes economize nearly 40% spare capacity. Nevertheless, the difference between the two restorabilities is minimal. So if the spare capacity is a big issue in a given network, then the new spanning tree algorithm is the best one to use among the three algorithms.

The cycle tree based algorithm with backup parents and some cycle edges has not only the merit the other two have, nonetheless, it derives some good ideas from the cycle-oriented algorithms which make it stronger than the other two. With the same upper bound of time and space complexity, although the difference of restorability among the three algorithms may be bigger or smaller, and the restorability itself may be higher or lower, for every model, the restorability of the cycle tree based algorithm is always higher than of the other two algorithms. In one topology, the difference reaches about 27%. When compared to the higher restorability, the increased spare capacity is very reasonable.

Furthermore, we can get much advantage in using backup parents in all of the three algorithms. From the simulation results, we always can get much higher restorability after using backup parents than before using them. From the results of the spanning tree algorithm and the hierarchical tree algorithm, we can see the effect of only using backup parents. In one topology the difference can achieve amazing figure: 46%.

We have used 5 topology models for the simulation. The restorability and the spare capacity can be very different for various network models. From the simulation results, the two new algorithms work very well on mesh-like random models, such as pure random model and AS-level Internet model.

For some hierarchical models, such as transit-stub model and Tiers model [18], we notice that our new algorithms do not work as well as the above networks generated randomly, although the cycle tree based algorithm provides a little better restorability than the other two tree-based algorithms. In future research, we hope to modify it to suit this kind of model in order to get sub-spanning tree with backup parents and cycle edges for every sub-structure, then get another spanning tree with every sub-spanning tree as one node.

Using the cycle edges in the cycle tree based algorithm, causes much higher restorability than the other two algorithms. The selection of cycle edges and cycle paths can play a very important role in the algorithm; good cycle edges and cycle paths bring much higher increase of the restorability. In our future work, we will do further research on constructing more useful cycle edges in getting more powerful cycle paths.

Bibliography

[1] Shahram Shah-Heydari and Oliver Yang, A Tree-based algorithm for protection/restoration in optical mesh networks, *CCECE*, pp. 1169-1174, May 2001, Canada.

[2] Yuza Zhang, Shahram Shah-Heydari and Oliver Yang, Complexity Analysis of the Hierarchical Tree Algorithm, 21st *Biennial Symposium on Communications*, pp. 46-50, 2002.

[3] Lech Szymanski and Oliver W. W. Yang, Spanning tree algorithm for spare network capacity, *CCECE*, pp. 447-543, May 2001, Canada.

[4] Wayne D. Grover and Demetrios Stamatelakis, Cycle-Oriented Distributed Preconfiguration: Ring-like Speed with Mesh-like Capacity for Self-planning Network Rrestoration, *1998 IEEE International Conference on Communication (ICC'98)*, Vol.1, pp. 537-543, 1998.

[5] Demetrios Stamatelakis and Wayne D. Grover, IP Layer Restoration and Network Planning Based on Virtual Protection Cycles, *IEEE Journal on Selected*

Areas in Communications, Vol.18, NO.10, pp. 1938-1949, 2000.

[6] Rainer R. Iraschko, M. H. MacGregor, and Wayne D. Grover, Optimal Capacity Placement for Path Restoration in STM or ATM Mesh-Survivable Networks, *IEEE/ACM Transactions on Networking*, Vol.6, No.3, pp. 325-336, June 1998.

[7] Adel Al-Rumaih, David Tipper, Yu Liu, and Bryan A. Norman, Spare Capacity Planning for Survivable Mesh Networks, *Networking 2000*, LNCS 1815, pp. 957-968, 2000.

[8] Yu Liu, Space Capacity Allocation: Model, Analysis and Algorithm, *Ph.D dissertation*, University of Pittsburgh, 2001.

[9] Yufeng Xin, Topology Design of Large-Scale Optical Networks, *Ph.D dissertation*, North Carolina State University, 2002.

[10] Muriel Médard, Steven G. Finn, Richard A. Barry, and Robert G. Gallager, Redundant Trees for Preplanned Recovery in Arbitrary Vertex-Redundant or Edge-Redundant Graphs, *IEEE/ACM Transactions on Networking*, Vol. 7, No. 5, pp. 641-652, October 1999.

[11] D. Stamatelakis and W. D. Grover, Network Restorability Design Using Preconfigured Trees, Cycles, and Mixtures of Pattern Types, *TRlabs Technical Report TR-1999-05*, October, 2000.

[12] Bülent Yener, Yoram Ofek and Moti Yung, Design and Performance of Convergence Routing on Multiple Spanning Trees, *Proc. Globecom'94*, Vol.1, pp. 169-175

[13] Eytan Modiano and Aradhana Narula-Tam, Survivable Lightpath Routing: A New Approach to the Design of WDM-Based Networks, *IEEE Journal on Selected Areas in Communications*, Vol. 20, No. 4, pp. 800-809, MAY 2002.

[14] L. Nederlof, K. Struyue, C. Shea, H. Misser, Y. Du, and B. Tamayo, End-to-end survivable broadband networks, *IEEE Communications Magazine*, pp. 63-70, 1995.

[15] A. Srikitja, D. Tipper, and D. Medhi, On providing survivable services in the next generation internet, *Proceedings of IEEE Military Communications Conference*, Atlantic, NJ, October 1999.

[16] Dorit S. Hochbaum, *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, 1995.

[17] Cheng-Shong Wu and Shi-Wei Lee, Modeling, Algorithms and Analysis of Survivable VP Planning in ATM Networks, *IEICE Trans. Commun.*, Vol. E82-B, No. 4, pp. 591-599, April 1999.

[18] Ken Calvert, Matt Doar and Ellen W. Zegura, Modeling Internet Topology, *IEEE Communications Magazine*, June 1997.

[19] B. M. Waxman, Routing of Multipoint Connections, *IEEE Journal on Selected Areas in Communication*, 6(9):1617-1622, December 1988.

[20] H.Tangmunarunkit, R.Govindan, S.Jamin, S.Shenker, and W.Willinger, Network Topology Generators: Degree based vs. Structural, *ACM SIGCOMM'02*, August 2002, Pittsburgh, Pennsylvania, USA.

[21] The Network Simulator ns-2: Topology Generation, *http://www.isi.edu/nsnam/ns/ns-topogen.html*

[22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Introduction to Algorithms, Second Edition, *The MIT Press*, 2001.