University of Alberta

Heuristic Techniques for Very Fast Solution of the Mesh Spare Capacity Placement (SCP) Problem

by

Vipul Rawat

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

Department of Electrical and Computer Engineering

Edmonton, Alberta

Fall, 1996

ISBN 0-612-18316-5

University of Alberta

Library Release Form

Name of Author:	Vipul Rawat
Title of Thesis:	Heuristic Techniques for Very Fast Solution of the Mesh Spare Capacity Placement (SCP) Problem
Degree:	Master of Science
Year this Degree Granted:	1996

[illegible]

C-06, Old Colony
Pilani, Rajasthan
India 333 031

Date: 2 August 1996

Hare Krishna Hare Krishna Krishna Krishna Hare Hare

Hare Rama Hare Rama Rama Rama Hare Hare

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled *Heuristic Techniques for Very Fast Solution of the Mesh Spare Capacity Placement (SCP) Problem* submitted by *Vipul Rawat* in partial fulfillment of the requirements for the degree of *Master of Science*.

[illegible]
W. D. Grover

Mike H. MacGregor
M. H. MacGregor

[illegible]
Bruce Cockburn

[illegible]
Pawel Gburzynski

Date: Aug 2/96

Dedicated to my family and friends

Abstract

In the design of cost-optimized mesh-restorable networks, we wish to explore new span additions. In the limit, $O(N^2)$ new spans could be added to achieve a fully connected mesh architecture, where N is the number of nodes in the network. To test all possibilities, fast spare capacity placement (SCP) tools are needed because each span addition has a network-wide influence on the network survivability plan. Available SCP tools are too complex to be considered for this purpose.

This thesis comprises a study of several original SCP heuristics for identifying the best new span additions in a network. The two main principles considered are called max-latching and network reduction. The max-latching heuristics are based on the concept of placing spares by spreading the unrestorable working capacity of each span on its predefined restoration routes and then latching the maximum spare capacity values forced on each span. The network reduction heuristics reduce the network's size through low complexity network reduction algorithms. This is followed by SCP for the reduced network through the max-latching heuristics. The reduced network is then expanded to specify the SCP solution for the full network. Our results show that network reduction combined with max-latching is much faster and almost as capacity efficient as integer programming (IP) for the same problem.

Acknowledgements

I thank Dr. W. D. Grover and Dr. M. H. MacGregor for supervising this research project. I also thank Dr. W. D. Grover for financial support.

I thank TRLabs for providing resources both financial and material for the research.

Finally, I would like to thank Ashish Duggal, Demetrios Stamatelakis, Jim Slevinsky, Rainer Iraschko, Danny Li, Ping Wan, and Winnie Tsang for creating a joyful environment for this research.

Table of Contents

List of Tables

List of Figures

Symbols and Abbreviations

d	A constant network-wide node degree.
d_{avg}	The average node degree of the network.
DCA	The direct cutsets algorithm.
DCS	Digital crossconnect system.
DT	Design tightening.
H	Hop limit
ICH	Iterated Cutsets Heuristics.
IP	Integer program.
KSP	k-shortest path algorithm.
LP	Linear program.
N	Number of nodes in the network.
NP	The set of problems that a non-deterministic algorithm can solve in polynomial time.
NRG1	Network reduction greedy heuristic which maximizes S_{eq}.
NRG2	Network reduction greedy heuristic which minimizes W_{eq}.
NRRBT1	Network reduction restricted backtracking heuristic which maximizes S_{eq}.
NRRBT2	Network reduction restricted backtracking heuristic which minimizes W_{eq}.
Q_i	State i of chain-wise network reduction.
r_i	Reduced span i.
R_n	The restorability of the network.
$R_{s,i}$	The restorability of span i.
Rearrange_rel	Relations for rearranging spare links in a chain.
S	The number of spans in the network.
S_0	Spare capacity on the span joining the end nodes of a chain.
S_1	Span having maximum working capacity in a chain
S_2	Span having maximum working capacity in a chain after S_1.

S_{eq}	Spare capacity equivalent.
$S^{1}{}_{eq}$	Spare capacity equivelent of S_1.
$S_{eq}(chain)$	Spare capacity equivalent of a chain.
$S_{eq}(chains)$	Spare capacity equivelent of parallel chains.
Seq_rel	Relations for increasing $S_{eq}(chain)$.
SCP	Spare capacity placement.
SLPA	Spare Link Placement Algorithm.
SP-ring	Shared protection ring.
s_i	The number of spare links on span *i*.
T	Total spare capacity of a chain.
W	The maximum number of working links on a span in the network.
W_0	Working capacity of a span joining the end nodes of a chain.
$W_{0,eff}$	Effective working capacity of the span joining the end nodes of a chain.
W_{eq}	Working capacity equivalent.
$W^{1}{}_{eq}$	Working capacity of span S_1.
$W^{2}{}_{eq}$	Working capacity of span S_2.
$W_{eq}(chain)$	Working capacity equivalent of a chain.
$W_{eq}(chains)$	Working capacity equivalent of parallel chains.
$W_{eff}(chain)$	Effective working equivalent of a chain.
$W_{eff}(chains)$	Effective working equivelent of parallel chains.
Weq_rel	Relations for decreasing $W_{eq}(chain)$.
w_i	The working link capacities vector, the number of working links on span *i*.
w_{max}	The maximum working capacity on a span in the network.

Glossary

Non-linear spans: Spans whose spare capacity equivalent increases non-linearly with spare capacity.

Real span: An original span of the network.

Reduced network: The network obtained by running the network reduction algorithm.

Reduced span: A span which represents a subtopology or a combination of spans belonging to a subtopology in the network.

Restoration equivalent model (W_{eq},S_{eq}): Representation of a subtopology by a reduced span with working and spare capacity equivalents.

Span: Logical connection between two adjacent nodes in a network through which carrier signals flow.

Spare equivalent (S_{eq}): Spare capacity offered by a span to the rest of the network.

Working equivalent (W_{eq}): The amount of restoration capacity which a span requires from the rest of the network.

1 Introduction and Background

The issue of public network integrity is of prime importance in today's telecommunications networks [1 - 3]. High capacity fiber-optic transmission systems (FOTS) form the backbone of today's telecommunications transport networks and are highly susceptible to cuts [4,5]. According to some availability analyses [6,7], for the 200 span Telecom Canada network with an unavailability of 0.096% per span, the probability of a single span failure per year over all spans is as high as 15.86%. The probability of two or more simultaneous independent span failures is 1.6% per year. Survivability against single span cuts is therefore very important. Traffic carried by the failed spans is restored through diversely routed spare transmission capacity in the network. This restoration must be done within two seconds in order to avoid dropping the calls associated with the lost carrier signal [2,9].

Over the past decade, significant effort has been put into carri[illegible]ility restoration research and advanced methods have evolved. Restoration methods based on digital crossconnect systems (DCS) offer the prospect of transport networks that are restorable within one or two seconds [2,3,10-13]. These methods exploit the mesh-like topology of real transport networks by sharing the spare capacity of the network for the restoration of every span in the network. Since spare capacity is not dedicated for restoration of any single span, the routing plan for the failed span is determined at the time of failure. The networks that result are called *mesh-restorable networks.*

Mesh-restorable networks are more capacity efficient than their shared protection ring (SP-ring) and 1+1 diverse protection counte[illegible]arts. This difference can be a factor of three to six for typical networks [3]. Fully restorable mesh networks can approach a limiting redundancy of $1/(d_{avg} -1)$, where d_{avg} is the average node degree of the network [8], while the minimum redundancy for SP-ring and 1+1 diverse systems is 100% [3,8]. Mesh networks are also inherently more flexible in that traffic need not be accurately forecast. These networks are less sensitive to service growth and have the potential for automated service provisioning, traffic adaptation, and network surveillance [17,18]. The restoration response times for SP-rings and 1+1 diverse protection approaches are on the order of 50 msec, but the advantages of mesh-restorable networks make them a potential choice for future survivable network architectures.

Two main research areas in mesh restorable networks are fast and efficient mechanisms for restoring failures, and cost minimization through optimized placement of spare capacity in the network. This thesis deals with the placement of spare capacity for restoration of a single span failure in mesh-restorable networks.

1.1 Problem Introduction

Our aim in the present work is to efficiently address the problem of adding new spans between some node pairs of an existing mesh network. In the limit, $O(N^2)$ new span candidates could be considered, where N is the number of nodes in the network. The most efficient approach is to optimize the network topology and its working and spare capacity placement simultaneously, but this is a very difficult problem to solve. Another potential approach is to optimize working and spare capacity simultaneously for each potential new span addition, and then select the most economical design option.

A relatively simple approach is to consider topology optimization, working capacity placement, and spare capacity placement as separate sub-problems. This would involve four steps: (1) listing a set of potential new span additions; (2) routing working paths after each potential span addition; (3) generating the spare capacity placement design for the network topology obtained in (1) with the working capacity placement of (2); and then (4) calculating the total network cost associated with the new design.

However, the currently available spare capacity placement (SCP) algorithms for mesh restorable transport networks are computationally complex. To test many potential span additions, fast spare capacity and working capacity placement tools are needed. Low complexity working path routing algorithms based on shortest path routing are already available but low complexity spare capacity placement approaches are not. Furthermore, for this problem it is not required in practice that the SCP results be strictly optimal: fast and reasonably accurate SCP solutions are needed.

1.2 Research Objective

The mandate of this thesis is to explore different approaches for designing fast SCP heuristics with near-optimal spare capacity placement results. The main motivation is to develop a fast method to identify a few potential new span additions for designing efficient mesh restorable networks. The best-ranked of these potential spans can then be analysed in detail using integer programming (IP).

The main approach followed in this thesis for designing fast SCP heuristics is based on the following hypothesis: the execution time of SCP algorithms depends on network size. It could therefore be reduced by reducing the network, doing SCP for the reduced network, and then expanding the reduced network's SCP solution to specify the full network's SCP solution, as outlined in Figure 1-1.

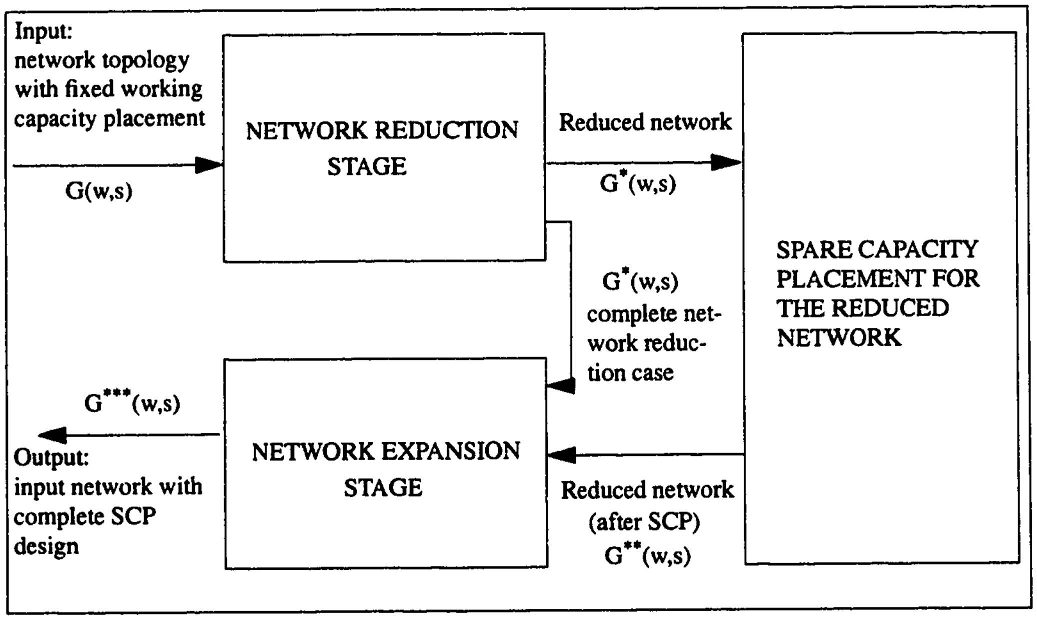


Figure 1-1: The network reduction approach for fast SCP

In Figure 1-1, the network $G(w,s)$, with fixed working capacity placement w and a null or initial default spare placement s, is first reduced to $G^{*}(w,s)$ through network reduction. SCP of $G^{*}(w,s)$ is then done, and the network obtained after the SCP is $G^{**}(w,s)$. The network $G^{**}(w,s)$ is then expanded to the original network $G^{***}(w,s)$ which specifies the complete SCP design for the network $G(w,s)$. In some cases the network

$G^{*}(w,s)$ is a completely reduced network (i.e., a network consisting of only two meta-nodes and a meta-span joining them), and in such cases $G^{*}(w,s)$ is directly expanded to $G^{***}(w,s)$ for generating a complete SCP solution.

The SCP of the reduced network, $G^{*}(w,s)$, may be done through heuristics based on the max-latching heuristics developed in this thesis, or by conventional IP methods. Max-latching heuristics involve three steps: (1) finding a suitable set of topologically feasible restoration routes for each span in the network; (2) evenly spreading the unrestored working capacity of each failed span onto its corresponding restoration routes; (3) latching the maximum spare capacity values obtained for each span to generate the SCP design for the network. This thesis tests these fast SCP heuristics against integer programming for identifying potential new span additions in a network.

1.3 Mesh Restorable Transport Networks

1.3.1 The Network

The carrier transport network is comprised of nodes and spans. *Nodes* are equipped with digital crossconnect systems (DCS), with the capability to switch carrier signals automatically. *Spans* are the logical connections between two adjacent nodes through which carrier signal(s) flow. A span consists of many *links*, where a link is an individual bi-directional DS-3 or STS-n carrier signal. In general, a link is the smallest unit of capacity which the DCS at a node manages. Links are of two types for restoration purposes: working links and spare links. *Working links* carry live traffic, while *spare links* are not in service though they are fully operational. Spare links are used for restoration of failed working links. Spans concatenated together form a *route*, and links concatenated together form a *path*. The *logical length* of a route is the number of spans present in it, and the *real length* is the sum of real distances of all its spans.

1.3.2 Span and Path Restoration

Span restoration re-routes failed working units over a set of replacement paths between the two end nodes of the failed span. *Path restoration* re-routes the failed working units over a set of replacement paths between each source and destination affected by the failure. Figure 1-2 shows an example in which the failure of span CD affects two working routes: A to F and K to G. Span restoration finds replacement path segments between nodes C and D, whereas path restoration finds end-to-end replacement paths for the demand pairs A-F and K-G.

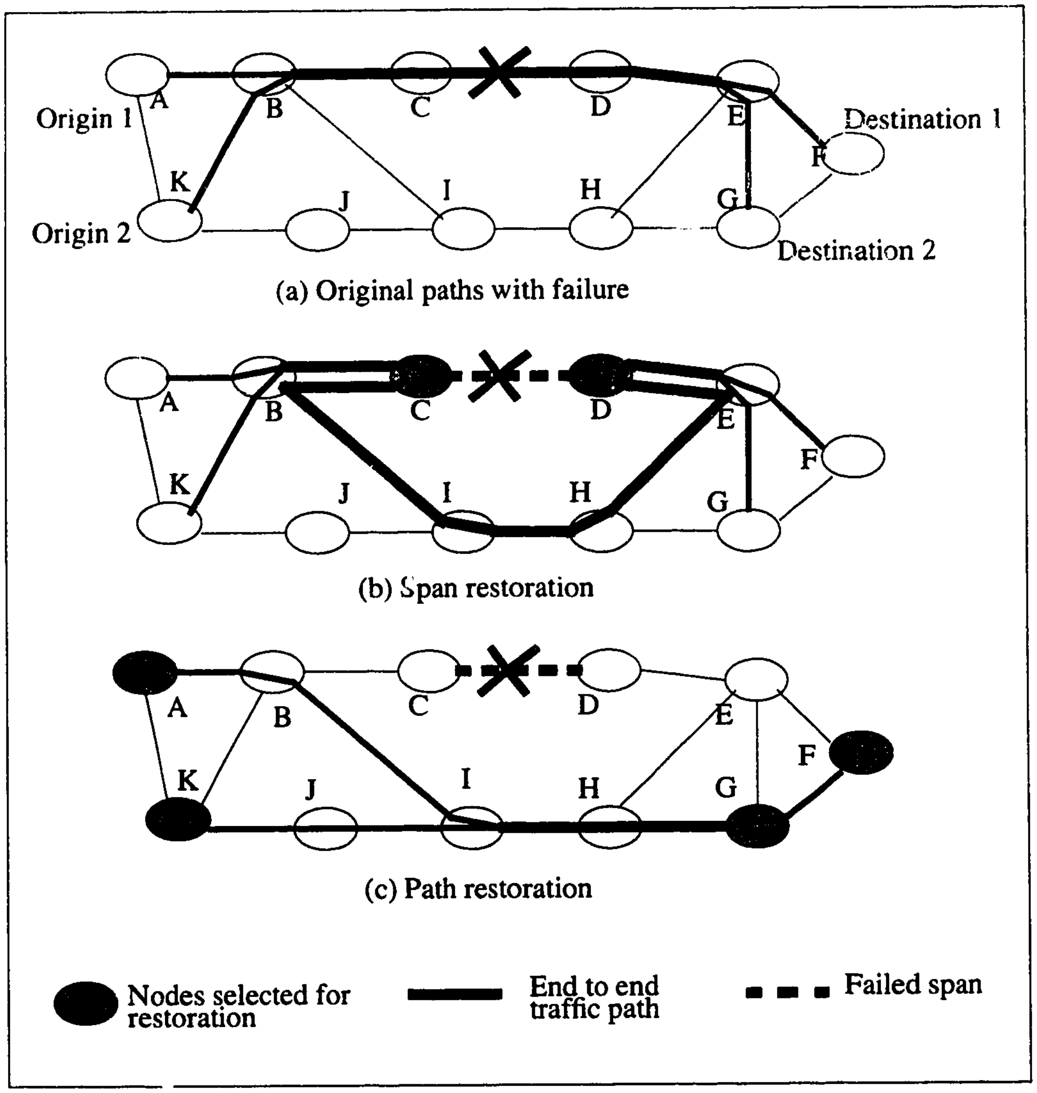


Figure 1-2: Span and path restoration

Path restoration can be more capacity efficient than span restoration as it spreads the replacement paths over a larger portion of the network, increasing the alternatives available for making efficient use of the network's spare capacity. However, path restoration is more complex to implement than span restoration as it may involve finding replacement paths for several source-destination pairs at once. Indeed the SCP problem for path restorable networks has only recently been solved [28].

This thesis is limited in scope to span-restorable networks. Future work may be able to extend the fast heuristics developed here to path-restorable networks.

1.3.3 Restoration as a Routing Problem

Restoration of the failed span's working links involves substitution of link disjoint replacement paths between the end nodes of the failed span. Each span may bear many parallel links and each link bears a carrier signal. A carrier signal completely fills a link and its corresponding replacement path in space, time, and frequency. Thus, one restoration path is dedicated completely for restoration of one failed link at a time. The restoration paths for a failure must therefore be mutually link disjoint throughout, i.e., there is no sharing of links between any two restoration paths for the same failure. Formulation of the routing problem for span restoration is discussed in detail elsewhere [2, 3].

The routing required for restoration is different from conventional packet routing in packet data networks and call routing in alternate-routing call networks [2, 3]. For restoration, each link is treated as a separate entity, therefore the transport network is a multigraph. On the other hand, packet data networks and alternate-routing call networks treat a span as a single entity by routing packets and calls nondisjointly over spans; these routing processes operate on simple graphs. There are many attributes by which restoration routing differs from packet and call routing [2, 3]; however, for this thesis, multi-graph network representations and link disjoint routing are the important distinctions from packet and call routing.

1.3.4 Routing Criteria for Span Restoration

The *maximum flow* between two nodes in a transport network is given by the capacity of the minimum cutset between the source and target node. A *cutset* is any set of spans which, if removed from the network, divide the network into two disconnected subgraphs. In our context the flow associated with any cutset is the sum of the spare capacities on its spans, excluding the *spare* capacity of the failed span. Figure 1-3 shows the cutsets between nodes 1 and 2; the maximum flow after the failure of span 1-2 is the minimum of the spare capacities of the three cutsets, i.e., the minimum of $s_{23}+s_{24}$, $s_{23}+s_{34}$, and s_{13}.

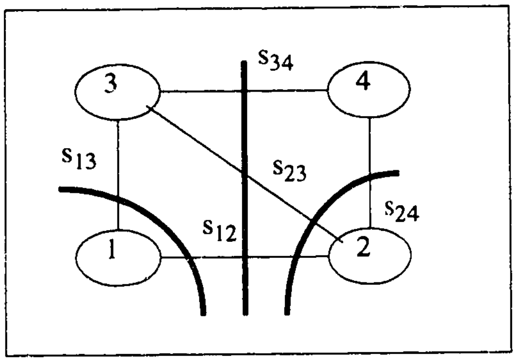


Figure 1-3: Cutsets between nodes 1 and 2

The maximum flow between the two nodes is determined by the min-cut max-flow theorem [19]. Restoration routing based on the max-flow theorem is optimal as the maximum flow calculation determines the maximum number of restoration paths available for any routing mechanism. The worst case time complexity for computing maximum flow is $O(N^3)$, N being the number of nodes in the network [19]. In addition, the maximum flow calculation does not directly yield the restoration pathset that will yield the predicted maximum flow. This must be found with a secondary computation.

Available DCS-based distributed mesh restoration methods [2,10,11,13] tend to closely emulate or reproduce the pathset generated by a *k*-shortest path (KSP) algorithm [2,12] although they do not operate algorithmically in the same manner as centralized algorithms for these problems. A classic KSP algorithm generates a pathset by selecting the shortest path, then the next shortest path that is link disjoint from the first, then the third shortest path that is link disjoint from the first two paths, and so on, until *k* paths are identified. The worst case complexity of finding *k*-shortest paths using Dijkstra's shortest path algorithm [20] is $O(k \cdot N \cdot \log(N))$ [8, 29]. KSP, therefore, is computationally simpler than maximum flow and yields a detailed pathset ready for deployment. One issue therefore becomes: what is the restoration capacity penalty if KSP is used instead of maximum flow? Previous work concludes that KSP restoration capacity is more than 99.9% of that from maximum flow in typical networks, and that this slight difference is due to a generalized trap topology [21]. The chances of occurrence of such a trap topology are low. Furthermore, when excess spare capacity arising from provisioning large modules is considered, there is expected to be virtually no penalty in practice arising from the use of KSP as opposed to maximum flow. KSP is therefore an effective and practical routing model for span restoration.

1.4 Spare Capacity Placement (SCP) for Mesh Restorable Networks

This section discusses the optimum SCP for mesh survivable networks, and summarizes some of the algorithms available. The issue which the SCP problem addresses is: *Given a restoration algorithm or a restoration routing reference criterion such as KSP, what placement of spare capacity ensures a target level of restorability with a global minimum of spare capacity?*

1.4.1 Complexity of the Optimum SCP Problem

In general, minimum total capacity is the aim of an SCP algorithm. The problem of *optimum* SCP for mesh-restorable networks is complicated because the spare capacity on one span can contribute to the survivability of many other spans, and the protection any span contributes to any other span depends on the spare capacity of all other spans. Direct search among all permutations for fully restorable networks is clearly infeasible in even the smallest problem of interest.

Some recent work has shown that the problem of finding a Hamiltonian cycle in a graph is polynomially reducible to the optimum SCP for a mesh-restorable network with one unit of working capacity on each span [8]. The problem of deciding whether there is a Hamiltonian cycle in a graph is known to be NP-complete; therefore, the problem of optimum SCP for mesh survivable networks is believed to be NP-hard (the set of problems at least as hard as NP-complete problems). *NP-complete* problems are the ones for which we suspect no algorithms exist that can guarantee their solution in polynomial time [22]. An exact solution to optimum SCP therefore, cannot be guaranteed in polynomial time. Most of the existing SCP algorithms and the approaches considered in this thesis aim for near-optimal SCP designs.

1.4.2 Performance Criteria for SCP Algorithms

Complexity, network restorability and capacity efficiency are the main performance criteria for SCP algorithms.

1.4.2.1 Complexity

The complexity of an algorithm is a measure of the execution time, memory, etc. it needs, as a function of the problem size. Complexity can be determined empirically or theoretically. Empirical analysis is used to predict the average case behaviour of the algorithm by running it on a suitable number of problem instances and observing the running time. Theoretical analysis can often be used to determine worst case behaviour. In practice, complexity is expressed in a notation which provides an upper bound to complexity within a multiplicative constant. Mathematically:

$$g(n) = O(f(n)) \tag{1.1}$$

means that there exist constants C and n_0 such that

$$|g(n)| \le Cf(n) \tag{1.2}$$

for all $n \ge n_0$. Here *f(n)* is a specific mathematical function of n such that a positive real multiple of *f(n)* exceeds the amount of resources consumed by the given algorithm. n is the problem size, and n_0 is the threshold value for the problem size beyond which equations (1.1) and (1.2) are valid. For example, if the time complexity of algorithm A is proportional to its problem size n for n greater than some threshold n_0, then the complexity of algorithm A is $O(n)$.

For fast SCP, time complexity is of primary concern. Therefore, this thesis emphasizes and seeks low-order worst case time complexity approaches for solving SCP while retaining reasonable or manageable space complexity.

1.4.2.2 Network Restorability

Restorability is the most important requirement imposed upon SCP algorithms. *Restorability of a span* is the fraction of its working links restored. For the failure of span i, if k_i is the number of available link disjoint restoration paths and w_i is the number of working links on the span before failure, the restorability of an individual span $R_{s,i}$ is defined as:

$$R_{s,i} = \frac{min(w_i, k_i)}{w_i} \quad (1.3)$$

The *overall network restorability*, R_n, is the ratio of the total number of working links restored in the network to the total working capacity of the network. It is expressed as:

$$R_n = \frac{\sum_{i=1}^{S} min(w_i, k_i)}{\sum_{i=1}^{S} w_i} = \frac{\sum_{i=1}^{S} R_{s,i} \cdot w_i}{\sum_{i=1}^{S} w_i} \quad (1.4)$$

where S is the number of spans in the network. The target network restorability for all the approaches in this thesis is 100%.

1.4.2.3 Capacity Efficiency

The capacity efficiency of a design is reflected in its redundancy:

$$Redundancy = \frac{\sum_{i=1}^{S} s_i}{\sum_{i=1}^{S} w_i} \quad (1.5)$$

where w_i and s_i are the working and spare capacities on span i, and S is the number of spans in the network. Note that comparing network spare capacity designs on the basis of redundancy is only meaningful if $\sum w_i$ is constant across the alternatives considered. This will be the norm in the following work.

1.4.3 Formulation of the Optimum SCP Problem

1.4.3.1 Cutset Based LP Formulation

Most approaches to solving the optimum SCP problem aim at minimizing the total spare capacity in the network while achieving a target restorability level. The SCP problem can be stated as [8]:

$$\min \sum_{j=1}^{S} s_j \tag{1.6}$$

subject to:

$$R_n = L, w_i \geq 0, s_i \geq 0 \qquad \forall i \in (0 \ldots S) \tag{1.7}$$

$$0 < L \leq 1$$

where S is the number of spans in the network; s_i and w_i are the number of spare and working links on span i; L is the desired restorability level; and R_n is network restorability. The constraint on network restorability can be written as a vector of constraint relationships pertaining to the restorability of each span individually:

$$k_i \geq L \cdot w_i \qquad \forall i \in (0 \ldots S) \tag{1.8}$$

where k_i is the number of restoration paths available for span i. The min-cut max-flow theorem states that the number of restoration paths available for a span is dictated by the min-cut associated with it. Therefore, in order to satisfy the above equation, the spare capacity of each cut associated with span i must be greater than or equal to $L \cdot w_i$. The cutset constraints can be expressed as a linear program (LP) or integer linear program (IP). The IP formulation for the SCP problem in terms of cutsets can be stated as [8]:

$$\min \sum_{j=1}^{S} s_j$$

subject to:

$$C \cdot s \geq w^* \cdot L \tag{1.9}$$

$$s_j, w_j \geq 0 \qquad \forall j \in (0 \ldots S) \tag{1.10}$$

The above equation is presented in an expanded form in Figure 1-4. ***C*** is the N_cxS cutset matrix of the network, where N_c is the number of cutsets used in the solution, and S is the number of spans in the network. ***C*** consists of cutsets corresponding to the failure of each span. Each row of ***C*** represents a cutset, and each column represents a span. The spans in the cutset are represented by 1's and those absent are represented by 0's. ***s*** is a vector of length S, and each element s_i represents the spare capacity on a span. $\mathbf{w}^*$ is a length N_c vector of working capacity, where each w_i in w^* appears as many times as there are cutsets for constraining the restoration capacity of span i. w_i appears as the j-th element of w^* if the j-th row of C represents a cutset constraint for ensuring flow for w_i.

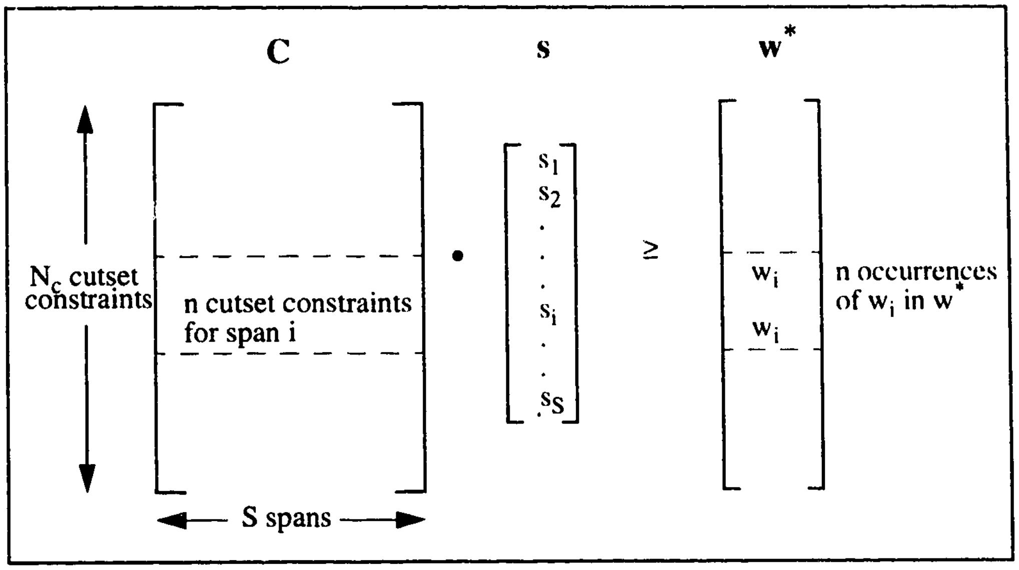


Figure 1-4: Description of the elements of constraint set

An important consideration in this formulation is that the number of cutsets N_c in a network grows exponentially with the network size N. In a fully connected network, the maximum number of cutsets is $O(2^N)$. Of these 2^N combinations, half of the combinations are not distinct as each assignment is repeated twice. Also, in two cases, one of the sub-networks is devoid of nodes. Therefore, the maximum number of cutsets is $(2^{N-1} -1)$, but this is still $O(2^N)$.

The above approach is called the *direct cutsets algorithm* (DCA) [8]. DCA involves finding all the cutsets of the network to form the constraint set for LP, minimizing total spare capacity in the network and then rounding off the real-valued spare capacities obtained after the LP run. The complexity associated with the first step is $O(2^N)$, as $O(2^N)$ iterations are needed to find all the constraints. Therefore, this step alone makes DCA exponential in time complexity. The simplex method is a well known method to solve the LP problem, and its worst case complexity is also exponential. Some methods for solving LP problems in polynomial time are available [22] but these methods are very complex. The complexity of rounding the real-valued spare capacity for each span is negligible as compared to that of generating the constraint set. The complexity associated with this step is $O(S \cdot N^3)$, where N^3 is due to max-flow [15], which must be run S times. IP can be used instead of LP in step 2 and 3 but it is NP-hard. Therefore, the complexity of the DCA algorithm is exponential, and this complies with the NP-hard nature of the optimum SCP problem.

1.4.3.2 Flow-Assignment Based IP Formulation

The constraint set for LP or IP-based SCP approaches can also be specified in terms of flow constraints that are based on a suitable subset of all distinct topologically feasible or "eligible" restoration routes. Such approaches involve two main steps. First all distinct restoration routes must be found for each span failure, subject to a suitable hop limit, distance limit, or other practical criterion for defining "eligible" routes. The constraint set is then generated based on these routes such that each span failure is fully restorable under the aggregate flow assignment to each of the eligible routes. Then IP is run to satisfy this constraint set with a minimum total spare capacity in the network. Linear programming can be used instead of integer programming to find a lower bound, but may involve fractional, unrealizable flow assignments on routes. An IP formulation using the flow assignment approach for 100% span restorable network was first reported by Herzberg [24] and developed further by Iraschko et al. [25].

The complexity of this formulation is exponential as the complexity of finding all distinct restoration routes for all failed spans and solving the IP are both exponential. $O(2^S)$ distinct restoration routes are possible in a network of S spans. A suitable hop limit can be used to limit the size of the restoration routeset for all spans [24]. Limiting the number of restoration routes for each failed span also reduces the number of constraints and variables for the IP problem, and in most cases this reduces the execution time of IP. The complexity of finding all restoration routes for S spans, up to a hop limit of H, is

$O(S(d_{avg} - 1)^H)$ [24]. In addition, solving the IP is exponential, and therefore the worst case complexity of this approach is also exponential. However, this approach does yield details of the actual paths used to restore each span failure, which is helpful when evaluating the performance of mesh restoration algorithms.

1.4.4 Prior Work on SCP Heuristics

Since optimum SCP for mesh survivable networks is NP-hard, there is no polynomial time algorithm which can guarantee an optimal solution. Therefore, heuristic approaches which have polynomial complexity and generate near-optimal solutions have been considered for solving the SCP problem. Over the past eight years, significant effort has been put into designing heuristics for generating near-optimum SCP designs. This section summarizes some of the available heuristic approaches.

The *Iterated Cutset Heuristics* (ICH) formulates SCP as an LP problem subject to constraints based on a subset of cutsets of the network [8,13,23]. The initial constraint set for ICH consists of only the incident cutsets of every node. An LP is executed for SCP, subject to this constraint set. If the resulting network is not fully restorable, one bottleneck cutset for each unrestored span is identified and added to the previous constraint set. The LP is then executed with the augmented constraint system. After a few iterations, the network typically becomes fully restorable with a near-optimal SCP solution.

The three primary components of ICH are executing the LP, augmenting the constraint set, and iterating these two steps. The execution time of the LP is dependent upon the number of constraints. ICH starts with a constraint set of size $O(S)$. At each subsequent step, at most S additional constraints are added, representing the case where none of the spans were restorable, and all these constraints can have no more than S spans. Therefore, the size of the constraint set for iteration i is $O(iS^2)$. In general, only a few iterations are needed to achieve full restorability for typical networks and the SCP results are very close to the optimal values [8,23]. In the worst case, $O(2^N)$ iterations are needed to generate a fully restorable SCP design. Thus, the worst case complexity of ICH is exponential.

The *Spare Link Placement Algorithm* (SLPA) is a greedy algorithm which makes successive improvements in maximizing restorability with minimum additional redundancy by adding, subtracting and redistributing spare capacity [8, 14, 23]. It consists of two phases: forward synthesis and design tightening. The objective of forward synthesis is to attain a target restorability with the minimum total spare capacity. It does so by adding one link (add_1), a combination of two links (add_2) or one full restoration path (add_path) in a way that maximizes network restorability. At each step, the operation involving the least number of links is preferred, i.e., add_1 is preferred over add_2, which is preferred over add_path. Design tightening is used to remove redundant spare capacity from the network through three steps: add0_sub1, add1_sub2, and add2_sub3, where addn_subn+1 involves adding *n* spare capacity links and removing *n+1* links from the network.

Three variants of SLPA implementations were developed and characterized [8,23]. These are SLPA Dijkstra, SLPA short, and SLPA path-table. SLPA Dijkstra and SLPA path-table perform all the forward synthesis and design tightening operations discussed above, while SLPA short omits the add2_sub3 operations. For a typical 60-node 120-span network, SLPA may require up to one million calculations of network restorability [8]. SLPA Dijkstra uses a binary min-heap implementation of Dijkstra's shortest path algorithm of $O(N \cdot \log(N))$ complexity [8,29], to find the *k*-shortest restoration paths for evaluating network restorability. SLPA path-table speeds up the restorability calculations by maintaining a pre-processed path-table which contains all topologically possible restoration paths between pairs of nodes in the network. Only the restoration paths with length less than or equal to a specified hop limit are included in the path-table.

The worst case complexities of SLPA Dijkstra, SLPA short, and SLPA path-table are $O(W^2 \cdot S^5 \cdot N \cdot \log(N))$, $O(W^2 \cdot S^3 \cdot N \cdot \log(N))$, and $O(W \cdot S^7 \cdot d_{max}^{RPL})$ respectively [8], where *W*, *S*, and *N* are the number of working links, number of spans, and number of nodes respectively. SLPA path-table requires $O(S \cdot RPL \cdot d_{max}^{RPL})$ memory locations to store the path-table in the manner outlined [8]. SLPA Dijkstra and SLPA path-table, on average place 5% more spare capacity than their ICH counterparts, while SLPA short places 2% more spare capacity than SLPA Dijkstra and SLPA path-table as it does not use add2_sub3 [8].

All the SCP heuristics available in the literature, to our knowledge, aim at achieving a near-optimal solution, and are still computationally time-consuming for the large, up to 200 node networks we would now like to work on. The primary aim of this thesis is therefore to design a fast SCP algorithm for use in identifying potential new span additions in the network. The existing SCP approaches are too complex to be used for testing $O(N^2)$ span additions in the network in a reasonable time.

1.5 Outline

Chapter 2 presents the max-latching SCP heuristics. In its introductory sections, the chapter discusses the idea of max-latching, the restoration routeset, and the classification of max-latching heuristics. The chapter then outlines the general characteristics of redundancy, execution time, and memory usage as a function of H for all max-latching heuristics. Six heuristics are analysed in terms of complexity, capacity efficiency, and execution time. These heuristics are also compared with IP. An extra step of design tightening for improving the capacity efficiency of these heuristics is also considered, and the capacity efficiency obtained after this step is also compared with IP.

Chapter 3 presents $O(N^2)$ complexity network reduction and expansion algorithms which are meant to improve the execution times for the max-latching SCP heuristics. The chapter discusses three methods for network reduction. These are fixed topology identification and substitution, greedy reduction and backtracking. All these methods reduce subtopologies that are common in real transport networks and whose efficient SCP can be done through these algorithms.

Chapter 4 combines the network reduction methods with the max-latching heuristics. These combinations are then compared with individual max-latching heuristics and with IP.

Chapter 5 presents an example of ranking new span addition candidates in optimizing a network's topology. Some max-latching heuristics and combined network reduction - max-latching heuristics are used to identify the best span from a set of potential spans. The relative ranking of the potential spans is also compared with that of IP.

Chapter 6 summarizes the results.

2 Max-latching SCP Heuristics

This chapter explores max-latching heuristics for spare capacity placement in mesh restorable networks. These heuristics are based on the idea of restoring the network through spreading each span's working capacity individually over its restoration routeset, generating a set of spare values, s_i, implied by each span cut individually, and then "latching" the single maximum spare capacity value generated for each span over this series of single-span restoration assessments.

Several max-latching heuristics can be designed following this basic principle depending on the manner in which routing for a span is done given the s_i values forced by prior test spans. The *restoration routeset* considered for each span consists of all distinct, topologically feasible restoration routes with maximum length limited by a specified *hop limit*. The required restoration capacity of a failed span is spread evenly over its restoration routes. The routes in a routeset for each span are sorted by the number of hops, to emulate the KSP restoration routeset. The *unrestorable capacity* of a failed span is the difference between its working capacity and the number of restoration paths available through the spare capacity in the network after the span's failure. Max-latching heuristics will choose spans one by one in some specified or random order. If each span uses spare capacity that has been placed for the restoration of previously failed spans, then the order in which spans are chosen by the heuristic affects the final SCP design of the network. These are called *order-dependent heuristics*. Heuristics that neglect the spare capacity present in the network while placing spares for a span are independent of the order in which spans are tested.

The performance of the max-latching heuristics depends in part on the hop limit used to delimit the length of routes in the restoration routeset of each span. Variations in network redundancy, execution time, and memory usage as a function of hop limit are observed and analysed for some metropolitan area and long distance networks to characterize the max-latching heuristics.

Max-latching heuristics will achieve 100% restorability, but result in some excess spare capacity in the network. The capacity efficiency of the SCP designs obtained from max-latching heuristics can be improved by selectively removing excess spare capacity from these designs while maintaining full network restorability. This step is called *design tightening* (DT) and is conceptually the same procedure as was the part of the SLPA algorithm described in Section 1.4.4. This chapter also compares the capacity efficiency results for max-latching heuristics with and without design tightening.

The next two sections discuss the generation and organization of the restoration routeset and the methods for computing network restorability. The concepts developed in these sections serve as background for this chapter.

2.1 Restoration Routeset for the Max-latching Heuristics

The restoration routeset used for the max-latching heuristics consists of all distinct, loop-free routes with lengths less than or equal to the specified hop limit, H. This type of routeset is chosen because it consists of a large number of routes providing the required route diversity for spreading the working capacity of a failed span as evenly as possible. Figure 2-1 presents the data structure we will use for representing the restoration routeset of a span. The restoration routes for each span are arranged by increasing length, and simple arrays of pointers are used for storing these routes. For example, in Figure 2-1, the first route contains only spans 1 and 3.

The problem of finding all distinct, loop free routes involves a search through a combinatorial space. The *brute force approach* would be to try all possible combinations of spans, and check whether they form a feasible route. This is highly inefficient. Efficient search techniques are needed which avoid examining the cases that are not valid. One efficient search technique for this problem is called backtracking.

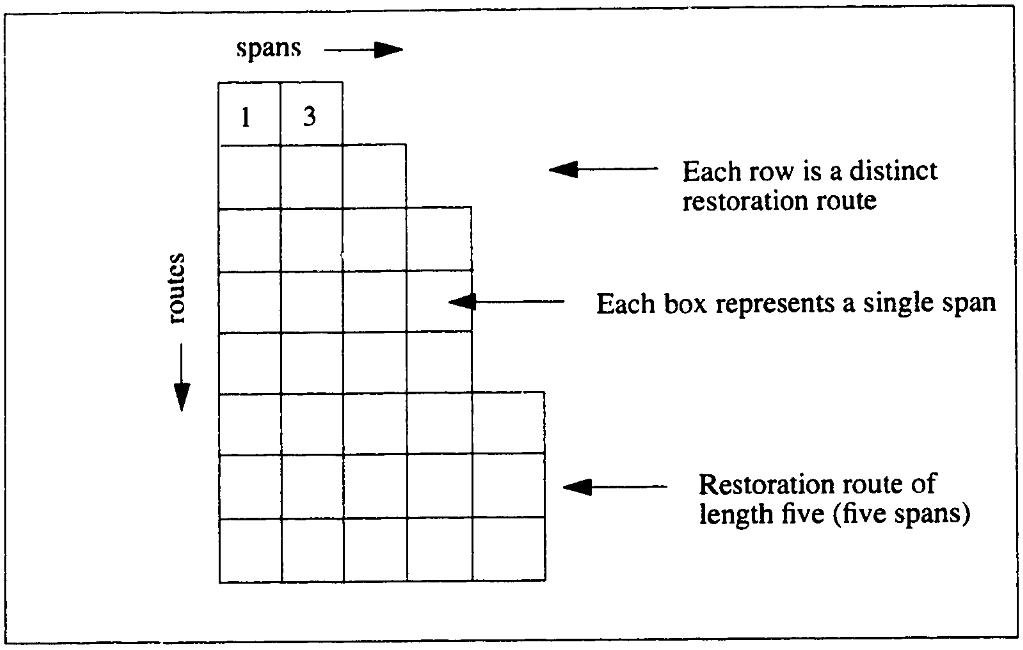

spans
1
3
routes
Each row is a distinct restoration route
Each box represents a single span
Restoration route of length five (five spans)

Figure 2-1: Organization of the restoration routeset of a span

Backtracking is essentially a depth-first search technique, with bounding functions to determine when to stop searching down a particular branch. The solution to a general backtracking search must be expressible as an n-tuple $(x_1, x_2, ..., x_n)$, where the x_i are chosen from some finite set U_i. The basic idea is to build up the solution vector one component at a time, and test whether the vector being formed has any chance of success. The major advantage of this method is that if the partial vector $(x_1, x_2, ..., x_n)$ cannot lead to a solution, then all the vectors descending from it can be ignored.

In our problem of finding all distinct restoration routes for a given span, x_i is the i-th span in a route, U_i is the set of spans incident with the same node as x_i and x_{i-1}, and n is the number of spans in a route, $n \leq H$. The network is viewed as a graph with nodes as vertices and spans as edges. Spans and nodes are arranged in the form of an adjacency list, and the restoration routes are found by traversing the adjacency list. The solution vector is built one x_i at a time, starting with x_0, where x_0 is a span incident on either node adjacent to the failed span. The starting node is called the *source node* and the other node of the failed span is the *destination node.* After the addition of span x_i, the length of the vector is tested, and loops are checked for. The length of the vector must be less than H if it does not represent a complete restoration route. The partial route is loop-free if no x_i appears twice. If any of these conditions are violated, then x_i is dropped, and the search backtracks to another branch by choosing one of the other possibilities in U_i. If all the possibilities in U_i have been tested, then the algorithm backtracks to the node common to x_{i-2} and x_{i-1}, drops x_{i-1}, retains the remaining $(x_0,\ldots\ldots, x_{i-2})$ elements, and tests the remaining possibilities in U_{i-1}. If all the possibilities in U_{i-1} have been exhausted, then the algorithm backtracks to the node common to x_{i-2} and x_{i-3}.

The algorithm returns a solution vector if a restoration path is formed by the addition of x_i. It then searches through the other spans in U_i, while retaining the vector $(x_0,\ldots\ldots, x_{i-1})$. If all the spans in U_i have been tried, the algorithm backtracks to the node common to spans x_{i-1} and x_{i-2}. The search continues until all the potential possibilities have been tested.

Figures 2-2 and 2-3 and Table 2-1 present an example to illustrate how backtracking is used for finding restoration routes for span 7. Table 2-1 summarises all the events that occur during the search and Figure 2-3 presents the search tree. In Figure 2-3, edges represent span numbers and node numbers represent the order in which the graph is traversed. Nodes labelled "B" in Figure 2-3 are the nodes at which the algorithm backtracks.

Let the source node be 1 and the destination node be 2. The backtracking algorithm starts the search at the source node. Only span 0 can be chosen for U_0. The search then moves to node 3. At node 3, there are three spans in U_i: spans 1, 2, and 4. The algorithm picks one of these at random. If span 1 is chosen, then a restoration route (0,1) is formed. The search then backtracks to node 3 and chooses one of spans 2 and 4. If span 4 is picked, the search reaches node 5, where two choices exist. If span 3 is chosen, then a loop consisting of spans 2, 3, and 4 is formed. Therefore, this branch is discarded. The only choice left at node 5 is span 5, which results in the restoration route (0,4,5,6). The search then backtracks to node 6, then to node 5, and finally to node 3, as all the choices at nodes 6 and 5 have been tried. At node 3, the only choice left is span 2. One of the routes through span 2 ends up in a loop, while the other forms restoration route (0,2,3,5,6). The search finally backtracks to node 1 and ends there.

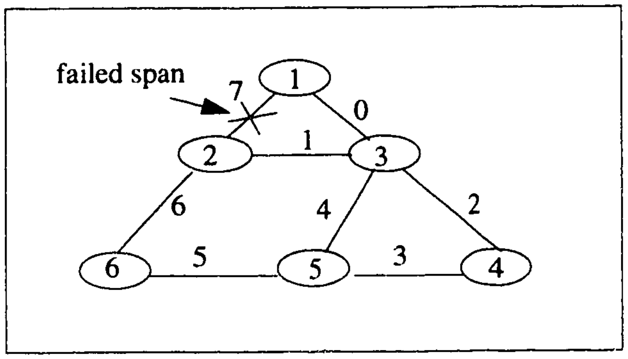


Figure 2-2: An example of backtracking algorithm for finding restoration routes

Table 2-1: Events for finding restoration routes for span 7

Path	Restoration route formed	Comment
0-1	0-1	Backtrack to node 3
0-4-3-2	X	Backtrack to node 5
0-4-5-6	0-4-5-6	Backtrack to node 3
0-2-3-4	X	Backtrack to node 5
0-2-3-5-6	0-2-3-5-6	Backtrack to node 1

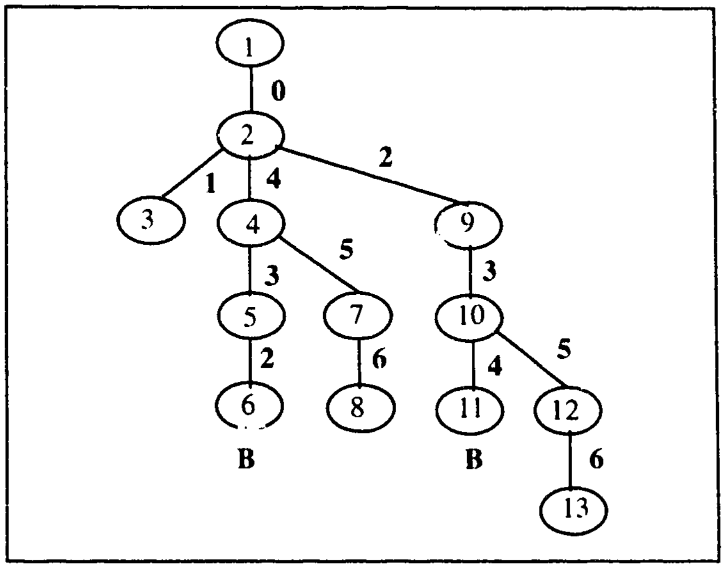


Figure 2-3: Search tree generated during backtracking

2.1.1 Complexity

The number of restoration routes for a span depends on the average node degree, network topology, and the hop limit used to restrict the lengths of the routes. Consider the case where all the nodes in the network have the same degree, d. There are $(d-1)$ possible initial choices for all restoration routes in the search tree. At each level down the search tree, the number of possibilities increases exponentially by a factor of $(d-1)$. Therefore, with a hop limit of H, there are $(d-1)^H$ choices. In the worst case, all the nodes are traversed. This condition occurs above some minimum hop limit H_0. The maximum number of routes possible is $O(d_{avg}{}^H)$, where d_{avg} is the average node degree. Therefore, the worst case complexity of finding the restoration routeset for all spans is $O(S \cdot H \cdot d_{avg}{}^H)$ in a network with S spans, provided $H \geq H_0$. Storing the routeset in the manner outlined above requires $O(S \cdot H \cdot d_{avg}{}^H)$ space, where each route requires at most H bytes of storage.

2.2 Network Restorability Computation

There are two simple methods for computing network restorability, R_n. They both involve calculating each span's restorability $R_{s,i}$ (equation 1.4). The most complex step in the $R_{s,i}$ calculation is in finding the restoration paths. Note that the restoration pathset is not the same as the routeset. A pathset is realized using the routes of a routeset, where a single route can host more than one path.

One choice is to use k-shortest paths for computing $R_{s,i}$ [8]. The restoration paths given an s_i assignment can be found with $O(N \cdot \log(N))$ complexity by using Dijkstra's algorithm. Computing R_n is of $O(S \cdot N \cdot \log(N))$ complexity in this case.

Alternatively, the restoration routeset for each span can be used for finding all available restoration paths for each span. In the worst case, all $O({d_{avg}}^H)$ restoration routes are checked for each span. The maximum length of each route is H, therefore the complexity of calculating $R_{s,i}$ is $O(H \cdot {d_{avg}}^H)$ so that calculating R_n is $O(S \cdot H \cdot {d_{avg}}^H)$. Real transport networks have d_{avg} between 2.5 and 4.5, and the restoration paths have hop limits generally greater than 4. Therefore, this method is more complex than the one based on Dijkstra's algorithm.

In this work, network restorability is used only for finding span-order sequences for some order-dependent heuristics and for design tightening, and is calculated using Dijkstra's algorithm.

2.3 Order-Independent Max-latching Heuristics

These heuristics place spare capacity by spreading the lost working capacity of each failed span evenly over its restoration routes, without considering the restoration paths already available through the spare capacity placed for other spans. The maximum spare capacity attained on each span, after placing spares for all spans, is latched as the design value s_i for that span. Since spare capacity placement for the restoration of each span is independent of every other span, the order in which spans are chosen for restoration does not affect the final SCP design in these approaches.

The order independent heuristic, *Heuristic0*, arbitrarily chooses spans according to their numbering in the network. Each span is made restorable by assigning one unit of spare capacity to all its available restoration routes starting with the shortest route, and iterating until the span is fully restored. The spare capacity on each span in the network is set to the maximum of its previous spare capacity, and the value found by placing spares for span *i*.

Figure 2-4 presents an example of Heuristic0 and illustrates the max-latching concept in general. Figure 2-4(a) shows a network with working and spare capacities on each span in the format (working, spare). Three restoration routes are available for the restoration of each span, except span 1-3 which has four restoration routes. Span 0-1 has seven units of working capacity which can be restored by the three routes shown in Figure 2-4(b). Route 0-2-1 carries three paths, while routes 0-2-3-1 and 0-2-4-3-1 carry only two paths. Note that the flow assignments are nominally levelled over the eligible routes but that equal whole numbers on each route are not always feasible. Figure 2-4(c) presents the spare capacity on each span after placing spares for span 0-1. Figure 2-4(d) shows the restoration routing for span 2-4, which involves placing two paths on routes 2-3-4 and 2-1-3-4, and one path on route 2-0-1-3-4. Figure 2-4(e) shows the SCP for restoration of span 2-4. The spare capacity on each span in Figure 2-4(f) is the maximum of the spare capacity on the corresponding spans in Figure 2-4(c) and Figure 2-4(e). Figures 2-4(g) and 2-4(h) show the restoration routing and SCP for span 2-3, and Figure 2-4(i) shows the state of network after the SCP of all three spans. In summary, each span in turn is allowed to "force" the s_i values that it requires in isolation for its restoration. A complete SCP design is the maximum of the s_i values forced on every span.

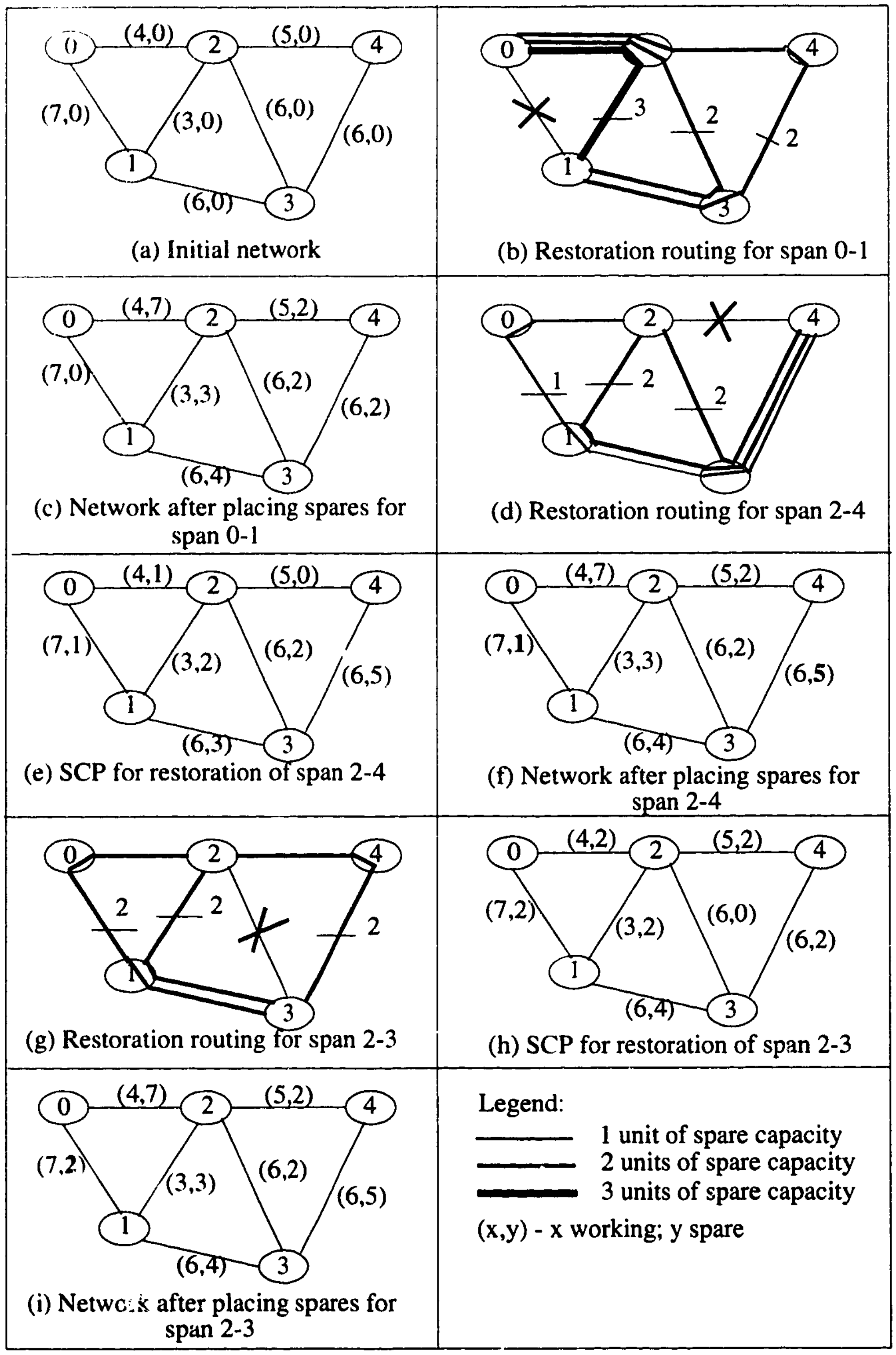

(4,0)
(5,0)
(7,0)
(3,0)
(6,0)
(6,0)
(6,0)
(a) Initial network
3
2
2
(b) Restoration routing for span 0-1
(4,7)
(5,2)
(7,0)
(3,3)
(6,2)
(6,2)
(6,4)
(c) Network after placing spares for span 0-1
1
2
2
(d) Restoration routing for span 2-4
(4,1)
(5,0)
(7,1)
(3,2)
(6,2)
(6,5)
(6,3)
(e) SCP for restoration of span 2-4
(4,7)
(5,2)
(7,1)
(3,3)
(6,2)
(6,5)
(6,4)
(f) Network after placing spares for span 2-4
2
2
2
(g) Restoration routing for span 2-3
(4,2)
(5,2)
(7,2)
(3,2)
(6,0)
(6,2)
(6,4)
(h) SCP for restoration of span 2-3
(4,7)
(5,2)
(7,2)
(3,3)
(6,2)
(6,5)
(6,4)
(i) Network after placing spares for span 2-3
Legend:
1 unit of spare capacity
2 units of spare capacity
3 units of spare capacity
(x,y) - x working; y spare

Figure 2-4: An example of an order- independent max-latching heuristic

SCP designs obtained through order-independent heuristics are expected to be less capacity-efficient than order-dependent heuristics as these heuristics neglect the existing spare capacity in the network while placing spares for each span. This thesis, therefore, uses an order-independent heuristic primarily only for the development and illustration of max-latching concepts, and then concentrates on order-dependent heuristics.

2.3.1 Complexity of Heuristic0

The complexity of finding the restoration routeset for S spans, subject to a hop limit H, is $O(S \cdot H \cdot d_{avg}^{H})$, where d_{avg}^{H} is the maximum number of restoration routes of up to H hops. The restoration routeset for each span is found only once, and routing and SCP for all spans is done after the restoration routeset has been found. In the worst case, all restoration routes for all spans are involved in restoration. Therefore, the complexity of evenly spreading working demand on the restoration routes for one span is $O(d_{avg}^{H})$. After placing spares for each span, the maximum spare capacity on every span in the network is latched. The complexity of max-latching the spare capacity values on S spans is $O(S)$. Therefore the overall complexity of Heuristic0 is $O(S \cdot H \cdot d_{avg}^{H})$.

2.4 Order-Dependent Max-latching Heuristics

Unlike order-independent heuristics, order-dependent heuristics force new spares only for the portion of the working capacity on each span that remains unrestorable after considering the spares previously placed in the network. That is, the sparing for each span is affected by the spare capacity that has been forced by previous spans. Therefore the order in which spans are treated affects the final SCP design. The following steps are followed for each span i:

Step 1: Find the number of available restoration paths, $NAPaths_j$, using the current spare capacity in the network for each route j, starting with the shortest route.

Step 2: Find the remaining unrestored working capacity, $Wunrest_i$, where

$$Wunrest_i = w_i - \sum_j NAPaths_j.$$

Step 3: Find the number of paths, $NPaths_j$, placed on each route j by evenly spreading the original working capacity, w_i.

Step 4: Spread $Wunrest_i$ on all the restoration routes, starting with the shortest route, in the following way:

```
WHILE ($Wunrest_i > 0$)
IF($NAPaths_j < NPaths_j$)
Begin
    Add $Minimum(Wunrest_i, (NPaths_j - NAPaths_j))$ paths to route $j$;
    Decrement $Wunrest_i$ by $Minimum(Wunrest_i, (NPaths_j - NAPaths_j))$;
End
ELSE increment $j$;
END WHILE;
```

Step 5: Latch the maximum spare capacity values forced on each span in the network.

Consider an example in which w_i is 6, the number of restoration routes is 3, and $NAPaths_j$ for the three routes are 3, 2, and 0. For this case, $Wunrest_i$ is 1 and $NPaths_j$ for each route is 2. Therefore, routes 1 and 2 already carry at least as many paths as they would be given by spreading the pathset for this failure. Hence, $Wunrest_i$ is added to the last route.

Figure 2-5 repeats the example of Figure 2-4 for the case of an order-dependent max-latching heuristic. Here span 0-1 is spared first, followed by span 2-4, and span 2-3. There is no spare capacity in the network initially. Figure 2-5(b) shows the restoration paths for span 0-1, and Figure 2-5(c) shows the corresponding s_i values. For the sparing of span 2-4, two paths are already feasible using route 2-3-4, so only three units of working capacity are unrestorable. These are made restorable through forcing new s_i values on the two routes shown in Figure 2-5(d). The maximum spare capacity on each span after placing spares for span 2-4 is latched, and is shown in Figure 2-5(f). Figure 2-5(g) shows all six restoration routes (dotted lines) available for the restoration of span 2-3 through the previously placed spare capacity. Figure 2-5(h) presents the SCP for span 2-3, and Figure

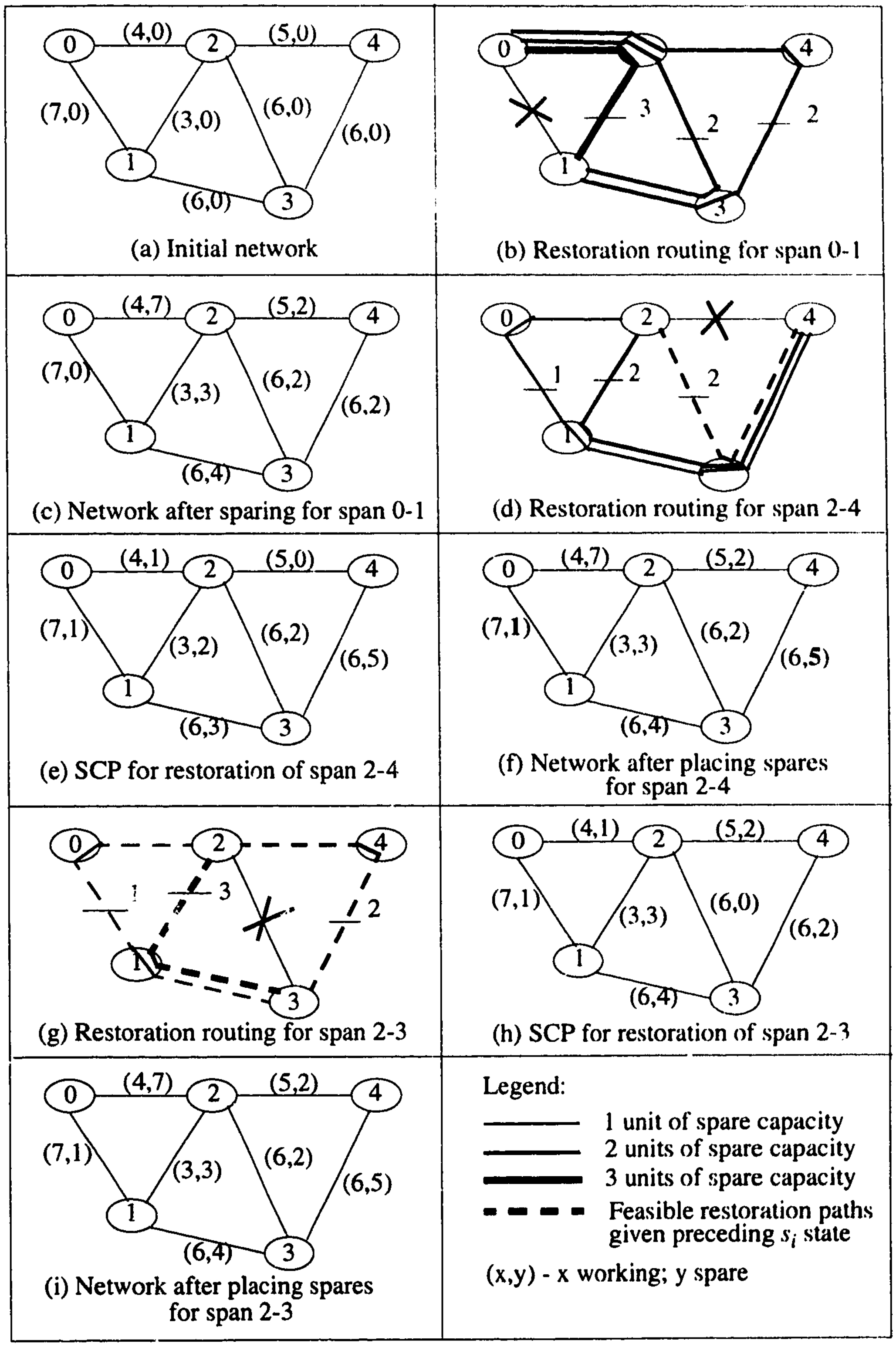

(a) Initial network
(b) Restoration routing for span 0-1
(c) Network after sparing for span 0-1
(d) Restoration routing for span 2-4
(e) SCP for restoration of span 2-4
(f) Network after placing spares for span 2-4
(g) Restoration routing for span 2-3
(h) SCP for restoration of span 2-3
(i) Network after placing spares for span 2-3
Legend:
1 unit of spare capacity
2 units of spare capacity
3 units of spare capacity
Feasible restoration paths given preceding s_i state
(x,y) - x working; y spare

Figure 2-5: An example of an order- dependent max-latching heuristic

2-5(i) shows the max-latching results for each span, after latching the spare capacities on each span in Figure 2-5(f) and 2-5(h). The overall SCP result obtained in this example uses one less unit of spare capacity than Heuristic0 (see Figure 2-4).

The SCP of order-dependent heuristics depends not only on the working capacity distribution but also on the order in which spans are chosen for restoration. For a network consisting of S spans, S! distinct span-order sequences are possible. Therefore it is not feasible in practice to test all possible sequences. For example, in a small network with 23 spans, if testing each sequence takes 0.1 seconds, then it will take 8.192 e+11 years to test just 1% of the total number of sequences. In addition, there is no guarantee that the type of sequence which is best for one particular working capacity distribution is also best for all other working capacity distributions. We therefore explore a limited number of sequences based on reasoned strategies to try to find those that result in the most capacity-efficient design.

2.4.1 Alternative P[illegible]ciples for Choosing Span Sequences

Several heuristics can be designed to determine the order in which spans should be chosen for placing spares. These can be put into three classes:

(1) Class A: Spans are ordered from greatest to least contribution towards network restorability R_n. Three subclasses are possible:

(1) A1: The sequence is found by failing each span individually, and calculating R_n for its SCP alone.

(2) A2: This sequence consists of spans ordered from greatest to least working capacities. In general, spans with high working capacities have a greater contribution towards R_n than spans with lower working capacity.

(3) A3: This sequence is determined by iteration as the SCP plan is developed. At each iteration, the span which contributes the greatest increase in R_n is chosen, and its sparing is allowed to force the s_i values in the network under design. Iterations are performed until the network becomes 100% restorable.

The heuristics associated with sequences A1, A2, and A3 are denoted HeuristicA1, HeuristicA2, and HeuristicA3, respectively.

(2) Class B: Spans are ordered by how much they individually force R_n but only the sequence of increasing working capacity is considered. This is the converse of A2, and is named B1.

(3) Class C: These sequences have the first one or two spans pre-specified, while the rest of the spans in the sequence are named in random order. The heuristics associated with this class try more than one simply obtained partly random sequence, and select the least redundant SCP design attained through these sequences. The two heuristics considered are:

(a) C1: This heuristic tries S sequences with the first span distinct in each sequence. The rest of the $(S-1)$ spans are chosen in random order.

(b) C2: This heuristic considers $S * (S-1)$ sequences with each sequence having the first two spans distinct from the other sequences. The order of the rest of the $(S-2)$ spans is random.

2.4.1.1 Time Complexity

Unlike Heuristic0, finding the span restoration sequence in order-dependent heuristics can be very complex. The complexity associated with finding available restoration paths for each span is $O(S \cdot H \cdot d_{avg}^{H})$, which is also the same as the complexity for spreading unrestored working capacity on the restoration routes.

The complexity of finding sequences A1 and A3 depends on that of the R_n computation. For A1, the contribution to R_n for each span is computed only once, therefore the overall complexity of finding this sequence is $O(S^2 \cdot N \cdot \log(N))$. In the case of HeuristicA1, finding available restoration paths and spreading the unrestored working capacity for each span has the maximum complexity among all steps, therefore the complexity of HeuristicA1 is $O(S \cdot H \cdot d_{avg}^{H})$.

The complexity for selecting a span for sparing in each iteration for A3 is $O(S \cdot H \cdot d_{avg}^{H})$. In the worst case, S iterations are performed for finding the A3 sequence, therefore the complexity associated with HeuristicA3 is $O(S^2 \cdot H \cdot d_{avg}^{H})$.

The complexity of finding sequences A2 and B2, through the quicksort algorithm is $O(S^2)$. Therefore, HeuristicA2 and HeuristicB2 are of $O(S \cdot H \cdot d_{avg}{}^{H})$ complexity.

HeuristicC1 and HeuristicC2 involve $O(S)$ and $O(S^2)$ iterations respectively. The complexity of these heuristics is dominated by finding the available restoration paths and spreading the unrestored working of each span. Therefore, the complexity of HeuristicC1 is $O(S^2 \cdot H \cdot d_{avg}{}^{H})$, and of HeuristicC2 is $O(S^3 \cdot H \cdot d_{avg}{}^{H})$.

2.5 Characteristics of the Max-latching Principle

Considerable insight about the performance of the max-latching heuristics is attained by studying how they depend on the hop limit used to specify the restoration routeset for each span in the network. This section explores the dependency of redundancy, execution time, and memory usage on hop limit for each of the max-latching heuristics considered in this thesis. We use Heuristic0 as an example for max-latching heuristics but the ideas developed here are valid for the other max-latching heuristics.

For testing the max-latching heuristics, four networks (Appendix A) were chosen on the basis of three parameters: (1) network average node degree (d_{avg}), where the degree of a node is the number of spans incident on it; (2) size of the network, measured by the number of nodes and spans present in the network; (3) the range of the minimum restoration path length available for all spans in the network. Table 2-2 presents these details for the test networks.

Net1 is a metropolitan area network, and it is a common example quoted in the literature. Net2 is another metropolitan area model based on a Canadian city. Net3 and Net4 are long haul networks.

Table 2-2: Networks Investigated

Networks	d_{avg}	Number of spans	Number of nodes	Range of minimum restoration path length for all spans	Total number of working links
Net1	3.733	23	11	2 to 2	1252
Net2	3.1	31	20	2 to 3	4112
Net3	3.93	59	30	2 to 4	27702
Net4	3.057	81	53	2 to 7	2389

2.5.1 Redundancy - Hop limit Characteristics

The ideal behaviour of redundancy as a function of hop limit for all max-latching heuristics is that of the flow-based IP formulation. Figure 2-6 shows ideal redundancy vs. hop limit characteristics for Net2, obtained by the reference IP formulation described later.

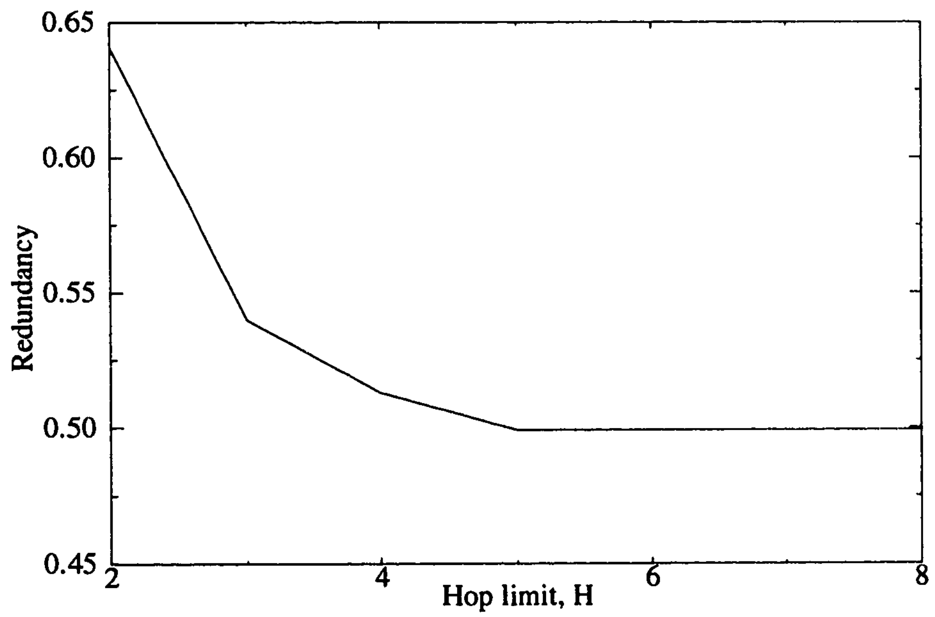


Figure 2-6: Ideal redundancy - hop limit characteristics for Net1

Redundancy decreases monotonically with increasing hop limit because more paths enable a more efficient SCP solution. In effect, the sharing of the network's spare capacity improves with hop limit as longer paths allow more spans to take part in the restoration of other spans in the network.

Although sharing of spare capacity increases with hop limit, effective sharing or overlapping of forced s_i values for max-latching heuristics may actually increase, decrease, or remain constant with hop limit. Effective sharing of the network's spare capacity is the measure of minimum total spare capacity needed to achieve full restorability, and it is maximum in the case of optimum SCP designs. For the IP formulation, effective sharing increases monotonically with hop limit. This is the ideal reference behaviour.

2.5.1.1 General Characteristics of the Max-latching Principle

Figures 2-7 to 2-10 present graphs of the actual design redundancy vs. hop limit for Heuristic0, for Net1, Net2, Net3, and Net4 respectively. The redundancy in each case decreases initially with an increase in hop limit, then remains minimum or near-minimum up to some higher hop limit value, and then in most cases increases with further increase in hop limit, finally saturating at high hop limit values. Based on this observation, the graphs can be classified into four main regions:

(1) Region I: Initial region in which redundancy decreases with hop limit.

(2) Region II: Region in which redundancy is minimum or close to the minimum value.

(3) Region III: Region after region II in which redundancy increases with hop limit.

(4) Region IV: Region in which redundancy remains constant with an increase in hop limit.

Regions I to IV for the four test networks are shown in Table 2-3. The behaviour of Net4 is different from that of the other test networks as its redundancy decreases monotonically with hop limit.

Table 2-3: Redundancy vs. hop limit for the four test networks

Networks	Region I	Region II	Region III	Region IV
Net1	2,3	4,5	6,7,8	> 8
Net2	4,5	6,7	8-16	>16
Net3	4,5	6,7	8-11	>11
Net4	7,8	>8	X	>13

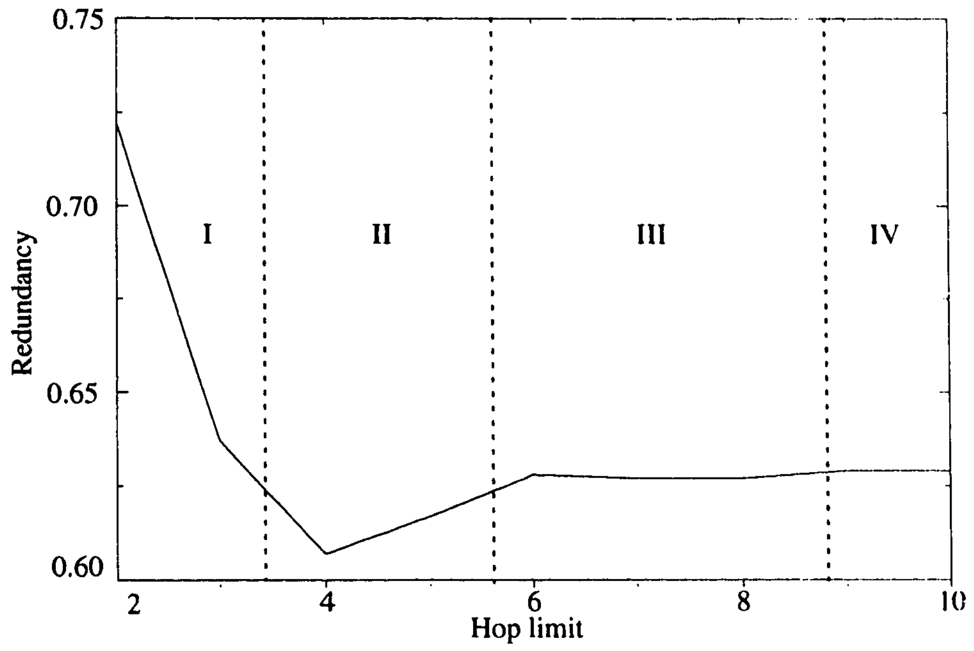


Figure 2-7: Redundancy vs. hop limit for Net1

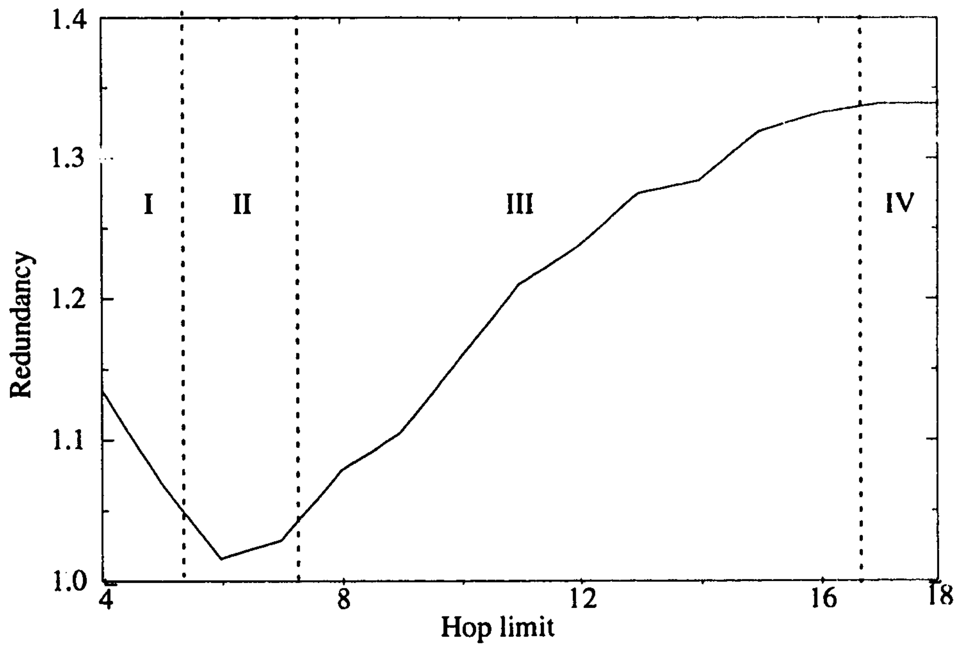


Figure 2-8: Redundancy vs. hop limit for Net2

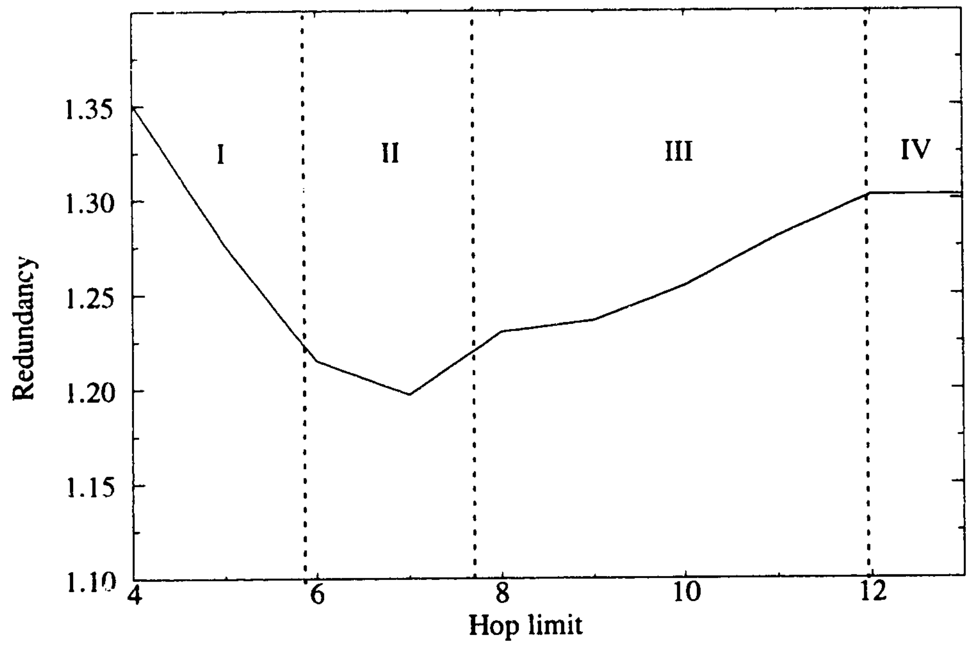


Figure 2-9: Redundancy vs. hop limit for Net3

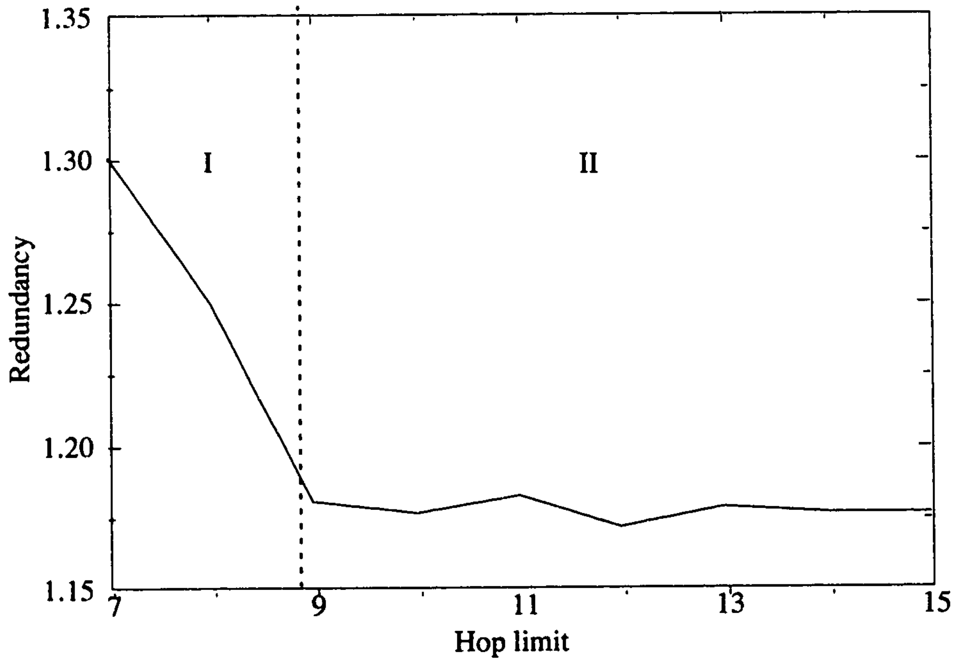


Figure 2-10:Redundancy vs. hop limit for Net4

By diagnosing the reasons for these behaviours, we gain understanding of the max-latching principle, which will later be reflected in our use of practical max-latching heuristics. In region I, the restoration of each span is highly localized due to short restoration paths. Each span depends on very few other spans for its restoration, which forces the spans in the network to have excessively high spare capacities. As the restriction on hop limit is eased, the working capacity on each span is spread over more spans in the network. Therefore, the spare capacity on each span contributes to the restoration of a larger number of spans than before. This increases the sharing of the network's spare capacity so that network redundancy decreases with an increase in hop limit in this region. As a diagnostic, Figure 2-11 shows the increase in the average number of spans taking part in the restoration of a span as a function of hop limit for Net2. This is due to an increase in the average restoration path length with hop limit, as presented in Figure 2-12 for Net2. Average restoration path length is the total length of all the paths used in restoration of all spans divided by the total working capacity of the network.

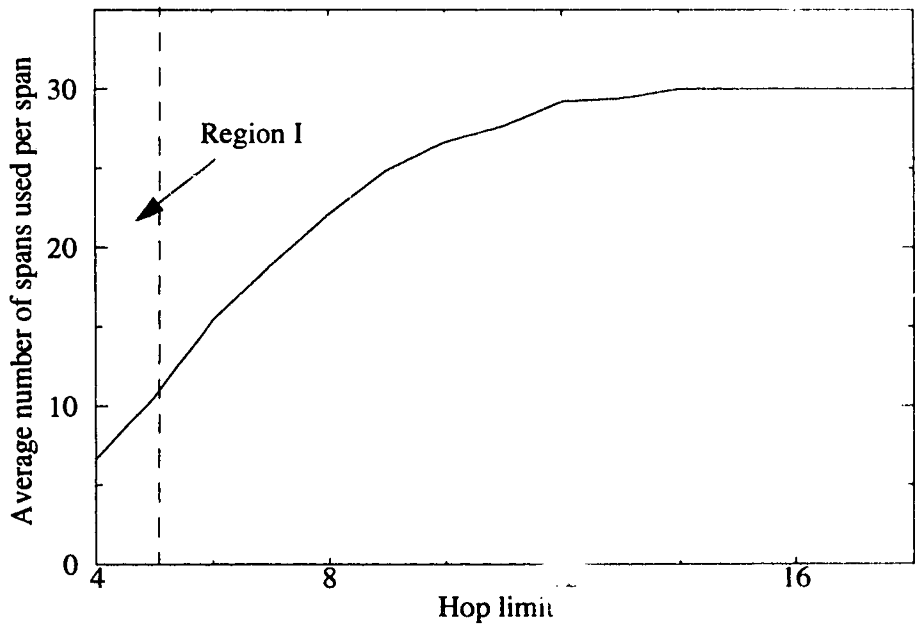


Figure 2-11: Average number of spans used per span for restoration as a function of hop limit for Net2

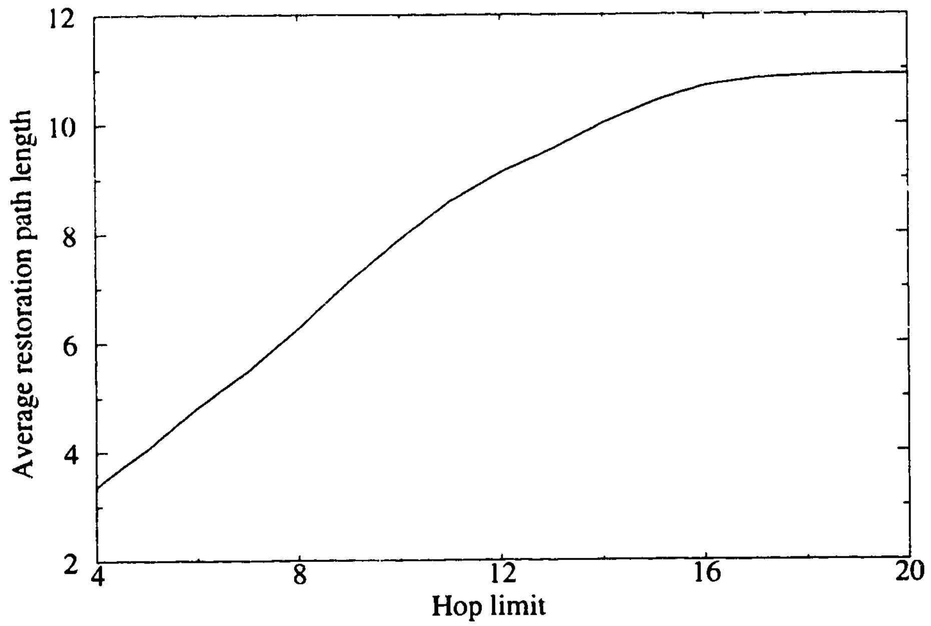


Figure 2-12: Average restoration path length as a function of hop limit for Net2

In region II the redundancy is minimum or near-minimum. In this region the effective sharing of the spare capacity in the network is maximum. In general, this region is confined to either a single hop limit value, or a range of two to three hop limit values. For ideal cases this region extends up to infinity. This can be thought of as the characteristic range in which restoration routes travel away from a failed span. In this region, the forced s_i values for each span are most generally similar to each other. In other words, the restoration pathsets have some general maximum interference, commonality, and reuse amongst themselves at the region II hop limits.

In region III, each span is forcing too far away from itself to efficiently "overlap" with the forced s_i values of other spans, and this contributes towards the over-restorability of the network. *Over-restorability* of a network is the ratio of the maximum number of restoration paths available for all the spans in the network to the total working capacity of the network. Figure 2-13 shows over-restorability as a function of hop limit for Net2. Note that redundancy and over-restorability follow the same trend in this region.

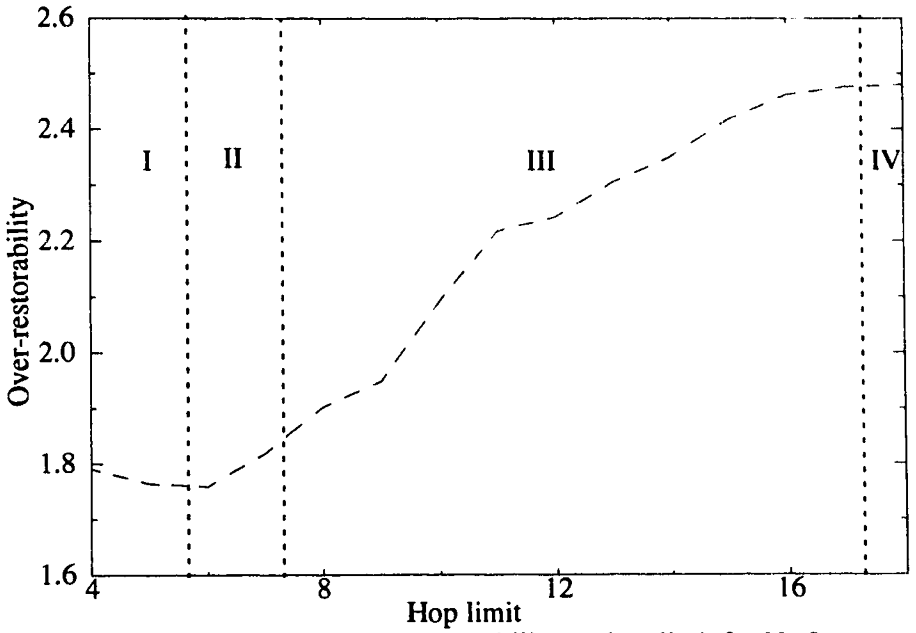


Figure 2-13: Over-restorability vs. hop limit for Net2

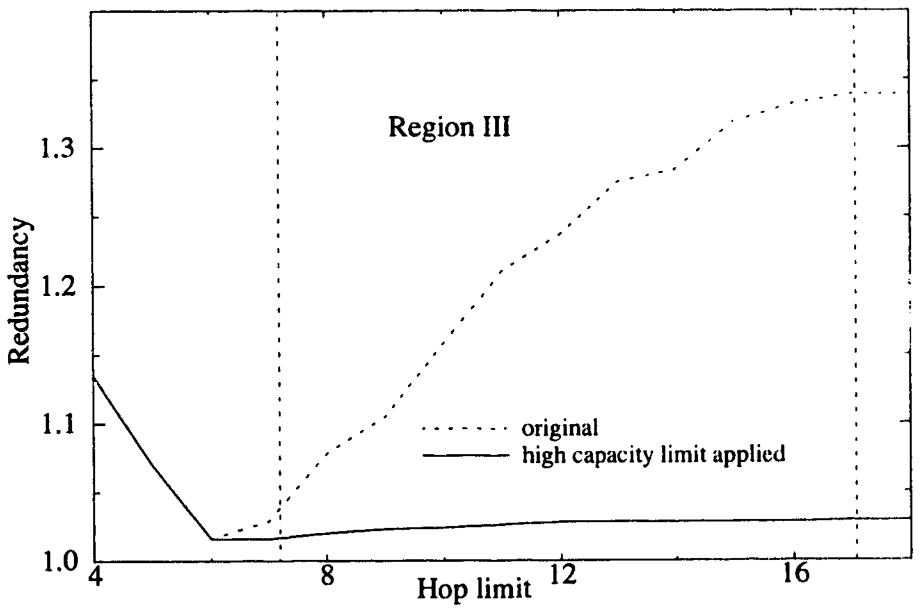


Figure 2-14: Comparison of the modified and the original hop limit characteristics for Net2

The redundancy in region III, for the cases which have high excess redundancy, can be decreased in general by restricting spans with high working capacities from using long restoration routes. Figure 2-14 presents a graph of redundancy vs. hop limit for Net2 where we restrict the top ten working capacity spans from using restoration routes of length greater than six hops. It is evident from Figure 2-14 that the high excess redundancy in this region was due to the spans which had high working capacities spreading and forcing more widely than desired.

Region IV is the saturation region in which there is no increase in redundancy with hop limit. This region occurs after the working capacity on every span has been spread to the maximum extent possible, such that the new routes added to the pathset by further increasing the hop limit are never contributors to the maximum forced s_i values.

The effective sharing of the network's spare capacity also depends on the distribution of working capacity in the network. Two different working capacity distributions were tried on Net4, and Figure 2-15(a) and 2-15(b) display graphs of redundancy vs. hop limit for these distributions. Each distribution was observed for different scalings of w_i. The first distribution is (1a) that used in Figure 2-10; (1b) that used in Figure 2-10 with the working capacity of each span multiplied by 10; (1c) Figure 2-10 distribution with the working capacity of each span multiplied by 100. The second distribution is random, and is comprised of two scalings of w_i. In (2a) the total working capacity is nearly ten times of that in (1a), and in (2b) the total working capacity is nearly 100 times of that in (1a).

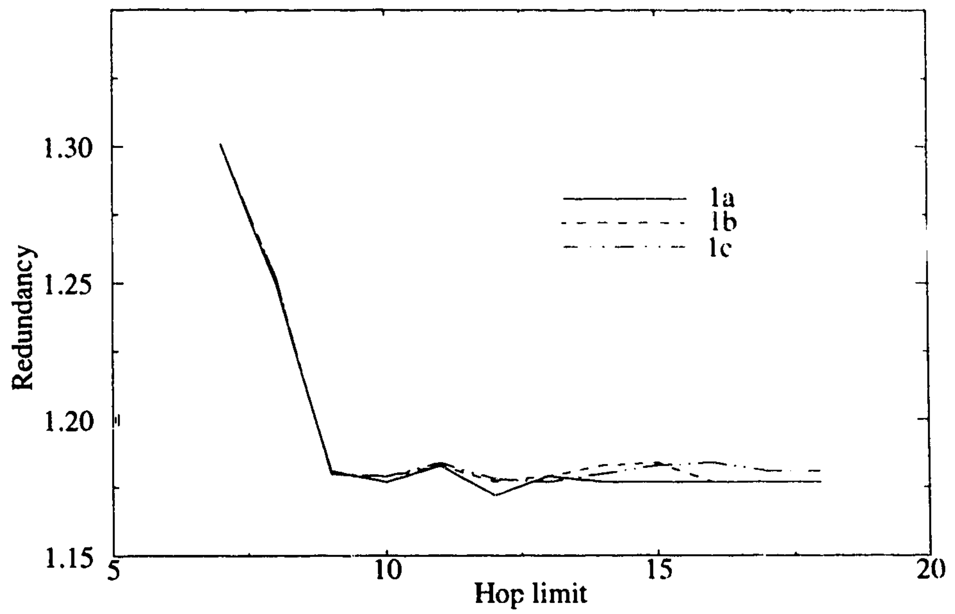


Figure 2-15(a): Redundancy vs. hop limit for three scalings of the same working capacity distribution, for Net4

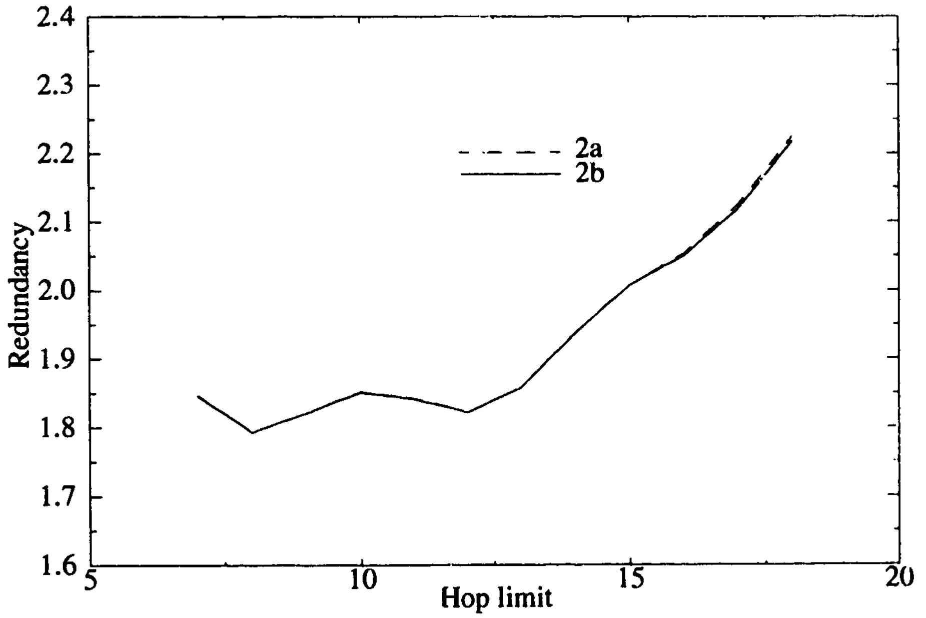


Figure 2-15(b): Redundancy vs. hop limit for two scalings of the same working capacity distribution, for Net4

It is evident from Figures 2-15(a) and 2-15(b) that redundancy of the SCP designs generated through max-latching methods depends more on the working capacity distribution than on the total working capacity in the network, because there is very little change in the curves for different scalings of the same distribution, while there are large differences between distributions.

Figure 2-16 presents a general graph of redundancy vs. hop limit for max-latching heuristics. Redundancy as a function of hop limit falls in the region between the two extreme curves a-d and b-c. The nature of the graph depends on the working capacity distribution and the max-latching heuristics used for SCP. Span order dependent heuristics differ from Heuristic0 only in that they consider restoration paths already available in the network before restoring each span. Therefore, SCP by these heuristics also depends on the working capacity distribution in the network. Hence the general nature of the dependence of redundancy on hop limit for these heuristics is expected to be the same as that of Heuristic0.

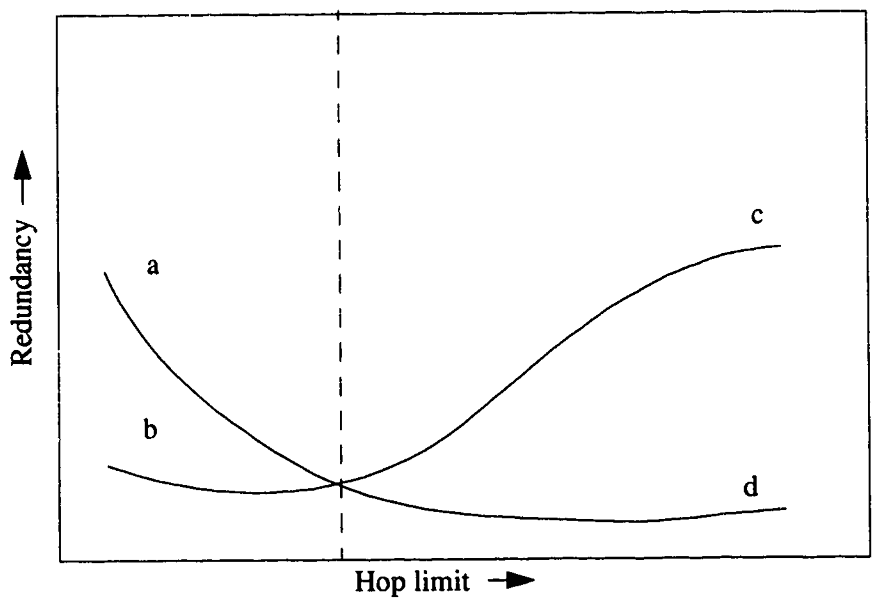


Figure 2-16: General redundancy vs. hop limit curves for max-latching heuristics

2.5.2 Execution Time as a Function of Hop Limit

The time complexity of the max-latching heuristics depends on:

(1) Generating the complete restoration routeset for all spans in the network. For a network of S spans, the complexity of finding a restoration routeset with hop limit H is $O(S \cdot H \cdot d_{avg}^{H})$.

(2) Finding the order in which spans are made restorable. The complexity of this step depends on the individual heuristic. For Heuristic0 the order is fixed, and therefore the complexity of this step is $O(1)$.

(3) Making each span restorable by placing spare capacity on its restoration routes. The complexity of this step is $O(S \cdot H \cdot d_{avg}^{H})$, as in the worst case all restoration routes are traversed.

(4) Max-latching the spare capacities on each span in the network, after the SCP for each span. This step is of $O(S)$ complexity.

The time complexity of a max-latching heuristic depends on steps 2 and 3, and is at least $O(S \cdot H \cdot d_{avg}^{H})$. Hence the execution time of all max-latching heuristics increases exponentially with hop limit. Figure 2-17 shows the graph of execution time vs. hop limit for Heuristic0 in the case of Net4, which validates this prediction. Therefore, it is imperative to consider only small hop limit values for fast SCP heuristics. This is not a problem in practice as in most practical cases a hop limit of 10 would be an upper limit to manageability. And in many other networks H = 5 or 6 will be fully adequate.

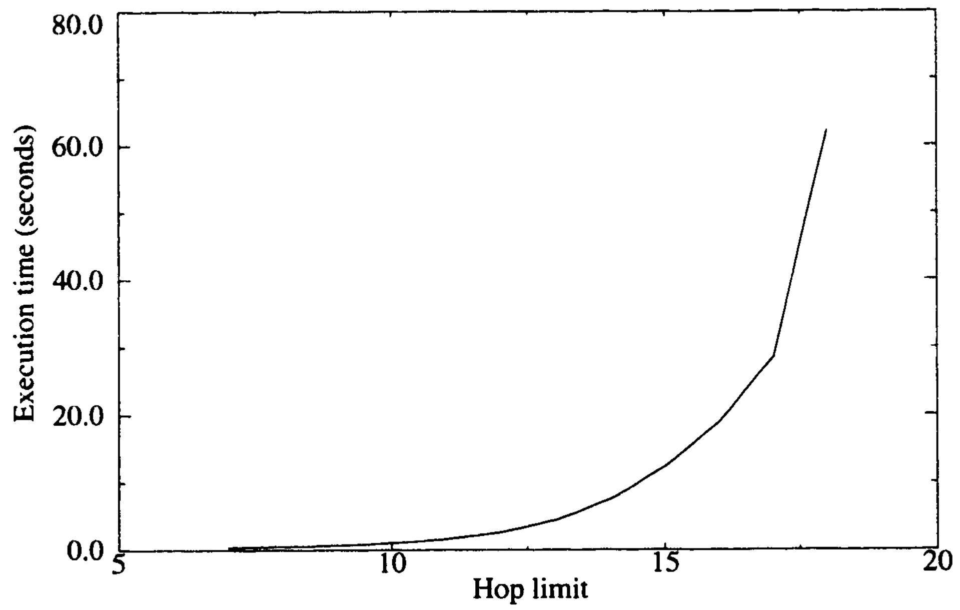


Figure 2-17: Execution time vs. hop limit for Heuristic0 in case of Net4

2.5.3 Memory Usage as a Function of Hop Limit

The memory used by one restoration route is at most H bytes, hence the space required by $O(d_{avg}^{H})$ restoration routes of a span is $O(H \cdot d_{avg}^{H})$. For a network of S spans, this approach requires $O(S \cdot H \cdot d_{avg}^{H})$ memory locations to store the routeset. This restricts fast SCP heuristics to small hop limit values. Figure 2-18 shows memory usage as a function of hop limit for Net4.

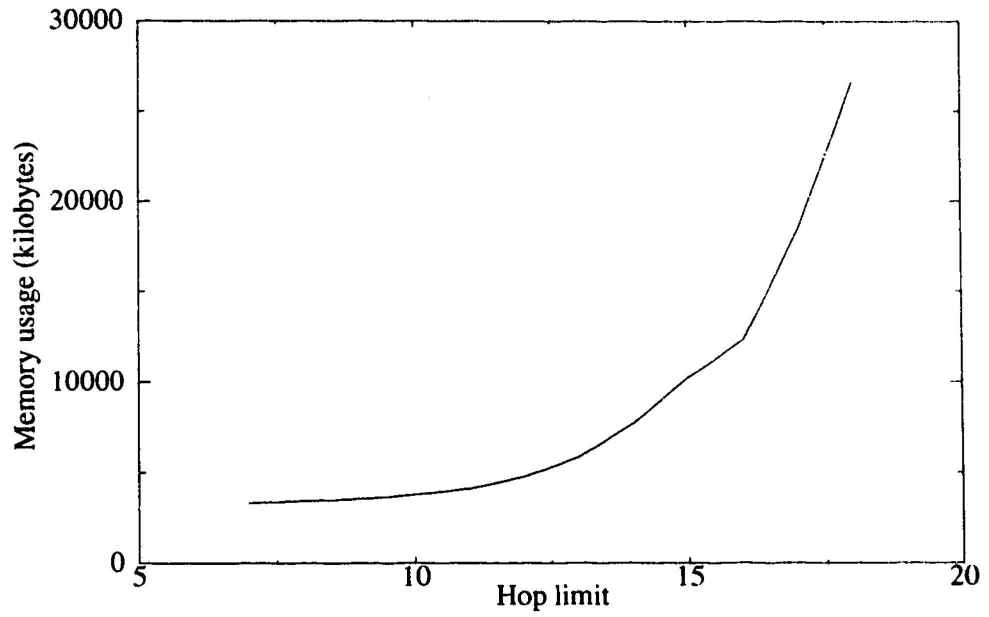


Figure 2-18: Memory usage vs. hop limit for Net4

2.6 Comparison of the Heuristics

Capacity efficiency, execution time, and memory usage all depend on the hop limit chosen for the restoration routeset for all spans in the network. Execution time and memory usage increase exponentially with hop limit, as established in sections 2.5.2 and 2.5.3. Therefore, for fast SCP heuristics, unnecessarily high hop limits should be avoided. The results for Heuristic0 in section 2.5 show that the minimum or near-minimum redundancy region usually starts within three hops of the minimum hop limit. The ***minimum hop limit*** is the value at which the corresponding restoration routeset guarantees 100% restorability. This section presents the graphs of redundancy vs. hop limit for the order-dependent heuristics in case of the four test networks, along with execution times and memory usage.

2.6.1 Capacity Efficiency

The graphs of redundancy vs. hop limit for all six heuristics are shown in Figures 2-19 to 2.22. It is evident from these graphs that the best sequence belongs neither to class A nor to class B. HeuristicC2 performed the best in all four cases, followed by

HeuristicC1. HeuristicC2 tried *S * (S-1)* sequences, and chose the result corresponding to the least redundant SCP design. The first five spans of the best sequence among these *S * (S-1)* sequences were different in each case. Therefore a sequence that performs well for one hop limit will not necessarily do so for other hop limits. A collection of sequences considered together are more likely to perform better at all hop limit values than a single sequence. Figures 2-19 to 2-22 also show less dependence of HeuristicC1 and HeuristicC2 on working capacity distribution than the single sequence heuristics. Ideally redundancy decreases with hop limit. Any deviation from the ideal reflects the effects of sequence dependence on the working capacity distribution and network topology

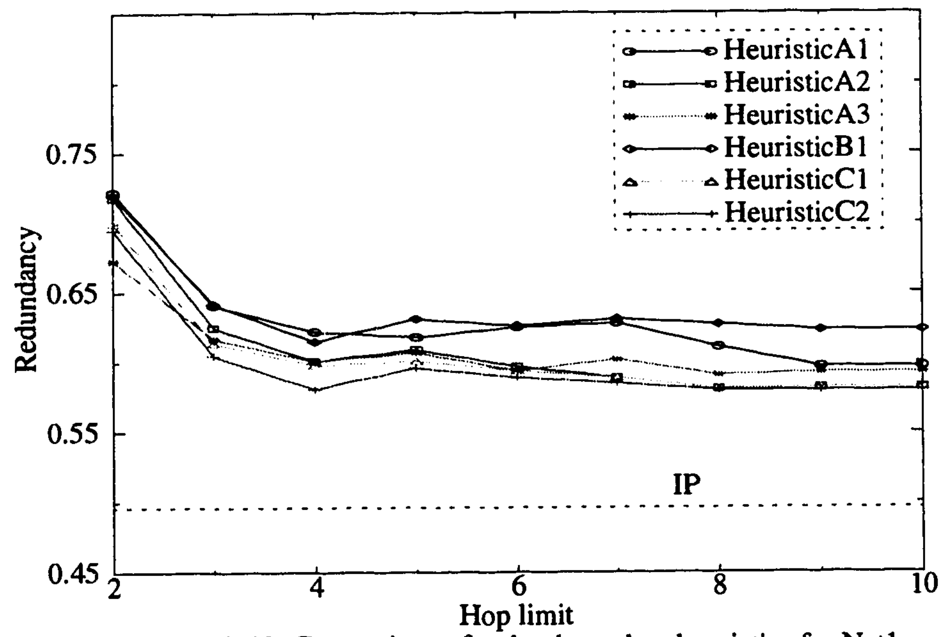


Figure 2-19: Comparison of order dependent heuristics for Net1

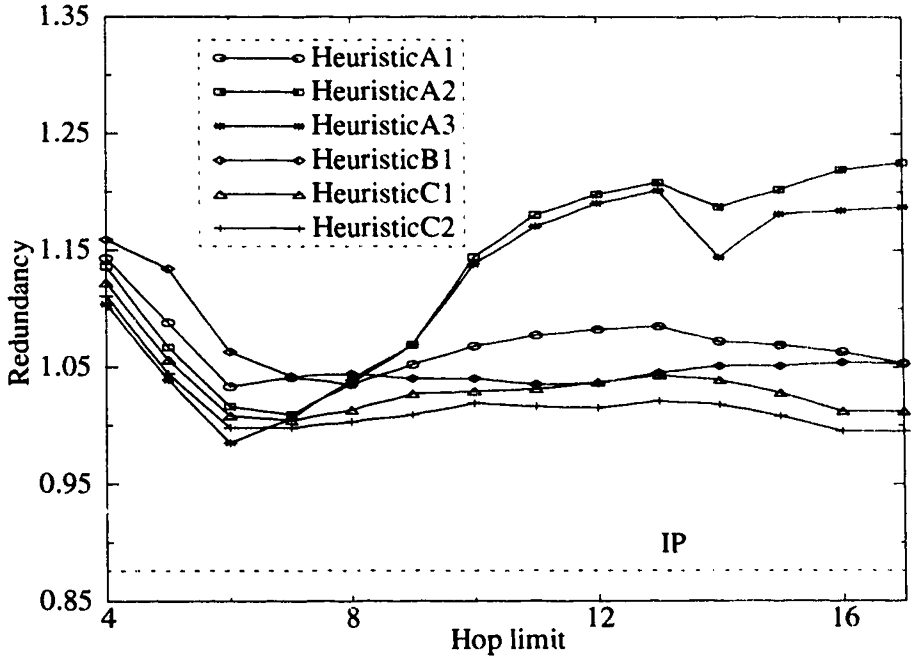


Figure 2-20: Comparison of order dependent heuristics for Net2

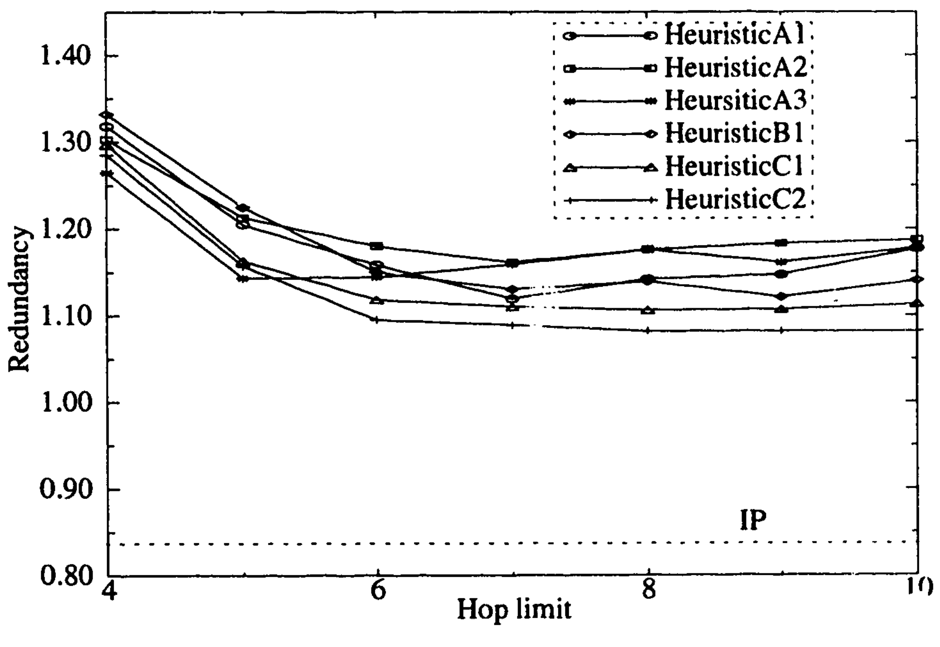


Figure 2-21: Comparison of order dependent heuristics for Net3

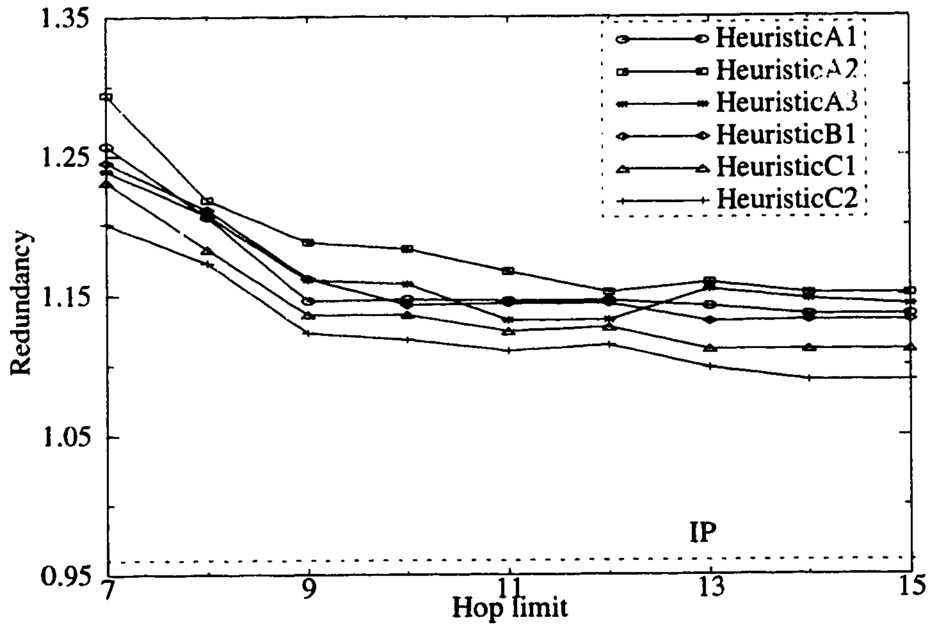


Figure 2-22: Comparison of order dependent heuristics for Net4

All the sequences considered exhibit the general redundancy - hop limit characteristics of max-latching heuristics discussed in section 2.5. Table 2-4 presents the range of region II for all the heuristics in all the test networks. Region II is defined by the range of hop limits over which redundancy values are minimum. In most cases, region II starts within three hops of the minimum hop limit needed for 100% restoration. Therefore, small hop limit values can be used effectively with these heuristics.

Table 2-4: Range of region II for the considered heuristics

Network	Min. Hop limit	A1	A2	A3	B1	C1	C2
Net1	2	>8	>5	>3	>3	>3	>3
Net2	4	6 - 9	6 - 7	6	>6	>5	>5
Net3	4	6 - 8	6 - 9	5 - 6	6 - 8	>5	>5
Net4	7	>8	>11	11 -12	>9	>10	>9

HeuristicC1 and HeuristicC2 are of special interest as they are more capacity-efficient, and seem to be less sensitive to the working capacity distribution than the other heuristics. Tables 2-5 and 2-6 compare the region II redundancy values of these two heuristics with respect to IP. IP results were arrived at by first generating a flow-assignment based constraint set using all distinct routes up to the specified hop limit and then minimizing the total spare capacity using CPLEX 3.0 ™ optimization software. In Table 2-5, IP results were obtained for region II hop limits. These hop limits are practical from the implementation point of view because long restoration routes cannot be used in real time restoration. Table 2-6 uses large hop limits for IP. The hop limits used for Net1, Net2, Net3, and Net4 are 10, 20, 9, and 16 respectively.

Table 2-5: Comparison of HeuristicC1 and HeuristicC2 with IP at practical hop limits

Network	Hop limit	IP redundancy	HeuristicC1		HeuristicC2	
			Redundancy	% Excess redundancy w.r.t. IP	Redundancy	% Excess redundancy w.r.t IP
Net1	6	0.4992	0.589	17.99	0.585	17.18
Net2	6	0.919	1.008	9.68	0.998	8.6
Net3	6	0.899	1.110	23.47	1.089	21.13
Net4	11	0.993	1.124	13.19	1.110	11.78

Table 2-6: Comparison of HeuristicC1 and HeuristicC2 results with those of IP using large hop limits

Network	IP redundancy	HeuristicC1		HeuristicC2	
		Redundancy	% Excess redundancy w.r.t. IP	Redundancy	% Excess redundancy w.r.t. IP
Net1	0.4992	0.589	17.99	0.585	17.18
Net2	0.875	1.008	15.2	0.998	14.06
Net3	0.841	1.110	31.98	1.089	29.49
Net4	0.963	1.124	16.71	1.110	15.26

According to Table 2-5, for region II hop limits, the SCP results of HeuristicC1 and HeuristicC2 are within 25% of those of IP at practical hop limits. Also, the redundancy for HeuristicC1 and HeuristicC2 are within 2% of each other. In Table 2-6 the redundancy for these heuristics for Net1, Net2, and Net3 are still within 20% of the IP while that of Net3 is within 32% of the IP using large hop limits. The capacity efficiency of these designs can be improved through design tightening. This is discussed in section 2.7.

Among the heuristics, HeuristicA2 and HeuristicB1 are of least complexity. Table 2-7 compares redundancy for these heuristics with that of IP for practical hop limits. The redundancy for these heuristics is greater than those of HeuristicC1 and HeuristicC2.

Table 2-7: Comparison of HeuristicA2 and HeuristicB1 with IP at practical hop limits

Network	Hop limit	IP redundancy	HeuristicA2		HeuristicB1	
			Redundancy	% Excess redundancy w.r.t. IP	Redundancy	% Excess redundancy w.r.t. IP
Net1	6	0.4992	0.589	17.99	0.626	25.4
Net2	6	0.919	1.016	10.55	1.063	15.67
Net3	6	0.899	1.18	31.25	1.15	27.92
Net4	11	0.993	1.152	16.01	1.144	15.2

2.6.2 Execution Time and Memory Usage

The execution time of all the heuristics increases exponentially with hop limit. The minimum redundancy region, region II, of the redundancy - hop limit graph starts at practical hop limit values. Table 2-8 presents execution times corresponding to a region II hop limit value. Execution times (CPU seconds) for all these cases was measured on a 150 MHz DEC Alpha 3000 ^{TM} workstation with 40Mb main memory and 406Mb of swap space.

HeuristicA2 and HeuristicA1 are the fastest, while HeuristicA3 and HeuristicC2 are the slowest. For small networks like Net1 and Net2, all heuristics took less than 40 seconds but for large networks like Net3 and Net4, HeuristicA3 and HeuristicC2 took over 20 minutes. Therefore, HeuristicA3 and HeuristicC2 are not well suited as fast SCP tools for large networks. On the other hand, all other heuristics took less than 40 seconds, hence these heuristics could be useful for fast SCP applications.

Table 2-8: Execution time for heuristics at practical hop limits

Network	Hop limit	Execution time (sec)					
		A1	A2	A3	B1	C1	C2
Net1	6	3.74	0.5	12.9	0.5	2.10	36.75
Net2	6	2.74	0.21	14.2	0.26	1.18	15
Net3	6	11.6	0.97	121	0.99	11.02	451
Net4	11	24.23	2.92	1300	3.02	33.8	1800

Memory usage of all the heuristics is dominated by the memory required for storing the restoration routeset. For the hop limit values in Tables 2-7 and 2-8, the maximum memory usage for all test networks was found to be less than 1Mb.

2.6.3 Conclusions

Six order-dependent heuristics based on three classes of sequences were tested on four test networks. The dependence of redundancy on hop limit for all heuristics was similar to that of the general characteristics of max-latching. All heuristics attained their individual minimum or near-minimum redundancy values at hop limits within four hops of the minimum.

The best sequence for all the test networks belonged to Class C. Within this class of heuristics, however, the sequence that performed the best at one hop limit for a network did not do so at other hop limits. This makes it difficult to generalize about a single best max-latching span ordering principle.

Class C heuristics were observed to be less dependent than classes A or B on the working capacity distribution and network topology. Graphs of redundancy vs. hop limit for class C heuristics were observed to be near-monotonically decreasing in all cases. Furthermore, these heuristics were the most capacity-efficient of all those considered. On the other hand, the heuristics based on class A and class B sequences showed high dependence on the working capacity distribution as their results were sometimes highly susceptible to hop limit.

In class C, HeuristicC2 was found to be 2% less redundant than HeuristicC1 but it is too complex to be used as a fast SCP heuristic for large networks. HeuristicC1 was observed to take less than 40 seconds for a large 53 node network. Therefore it is a potential heuristic for fast SCP.

Redundancy for HeuristicC1 and HeuristicC2 was within 25% of that of IP for practical hop limits without any further processing to remove excess redundancy. The redundancy for these heuristics was also compared with that of IP using large hop limits and was found to be within 20% for Net1, Net2 and Net4, and within 32% for Net3.

2.7 Design Tightening to Reduce Excess Redundancy

Design tightening improves the capacity efficiency of an SCP design by eliminating and, in some cases, rearranging spare capacity in the network while maintaining full network restorability. Simple eliminations of spare ca[illegible]nvolve removing spare capacity from a span where possible without decreasin[illegible]rrangement of spare capacity in the network involves replacing spares on *n* spa[illegible]ituting *m (m < n)* spares on other spans, while maintaining 100% restorability. [illegible]vious work has used add(n)_sub(n+1) type rearrangements [8], where *n* units of spare capacity are added and *(n+1)* are removed from the network. add(n)_sub(n+1) involves $O(S^{(2n-1)})$ computations of K_n [8]. Apart from add0_sub1, all other rearrangements are too complex to be practical for fast SCP. For example, if one calculation of R_n takes 0.1 seconds, then add1_sub2 will take nearly 3.47 hours for a 50 node network, while add0_sub1 will take 5 seconds. Therefore, only add0_sub1 type simple eliminations of spare capacity are considered here.

Add0_sub1 removes surplus spare capacity from the network where possible without decreasing R_n. R_n is calculated after each such step. The overall complexity of removing excess spare capacity from the whole network using add0_sub1 is

$O(w_{max} \cdot S^2 \cdot N \cdot \log(N))$, where w_{max} is the maximum number of working links on a span.

In general, max-latching methods may place surplus sparing on any given span. Add0_sub1 finds the exact number of surplus spare links on a span. The maximum spare capacity on each span can be w_{max}. The number of iterations needed per span to find the exact number of surplus spare links is $log(w_{max})$ in the worst case. Thus, the complexity of tightening a network through add0_sub1 using Dijkstra's algorithm is $O(S^2 \cdot \log(w_{max}) \cdot N \cdot \log(N))$.

2.7.1 Performance of Max-latching heuristics with Design Tightening

Figures 2-23 to 2-26 are graphs of redundancy vs. hop limit for HeuristicA2, HeuristicC1, and HeuristicC2, along with those obtained after add0_sub1 type design tightening.

For Net1, Net3, and Net4, for all the heuristics, the redundancy obtained after design tightening is nearly the same for all hop limits. Also, the redundancy values obtained for all heuristics after design tightening are quite close to each other. For Net2, the graphs of redundancy vs. hop limit for HeuristicC1 and HeuristicC2 have nearly the same behaviour, while that of HeuristicA2 shows a gradual increase with hop limit. This is because of high redundancy at high hop limits for HeuristicA2. add0_sub1 is not able to remove all excess spare capacity at high hop limits due to a high degree of sharing of the network's spare capacity for restoration among all spans. To further detect and remove excess redundancy add1_sub2 searches, etc., are needed but not warranted as a practical matter.

The fraction of excess spare capacity in the network that can be removed by add0_sub1 eliminations depends on the extent of sharing of the network's spare capacity. If sharing is low, then the chance of removing excess spare capacity from a given span without affecting the restorability of the other spans in the network is high. If sharing is high, it is less likely that spare capacity can be removed from a span without affecting the restorability of all the dependent spans.

At low hop limits, the sharing of spare capacity is low. Therefore, most of the surplus spare capacity units can be eliminated through add0_sub1 eliminations. At high hop limits, the degree of sharing is high, and therefore add0_sub1 eliminations are not able to remove some of the redundant spare capacity units. In Figures 2-23 to 2-26, redundancy at low hop limits after design tightening is minimum or very close to minimum. At high hop limits, if the redundancy is higher than that of low hop limits, then the redundancy after design tightening is also expected to be higher than that of low hop limits, as for HeuristicA2 for Net2.

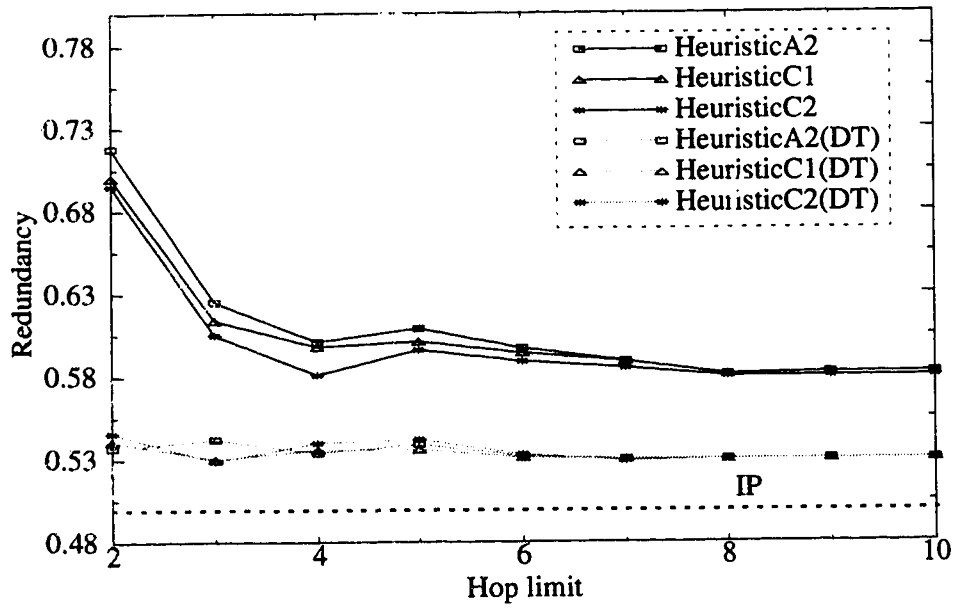


Figure 2-23: Comparison of heuristics with and without DT for Net1

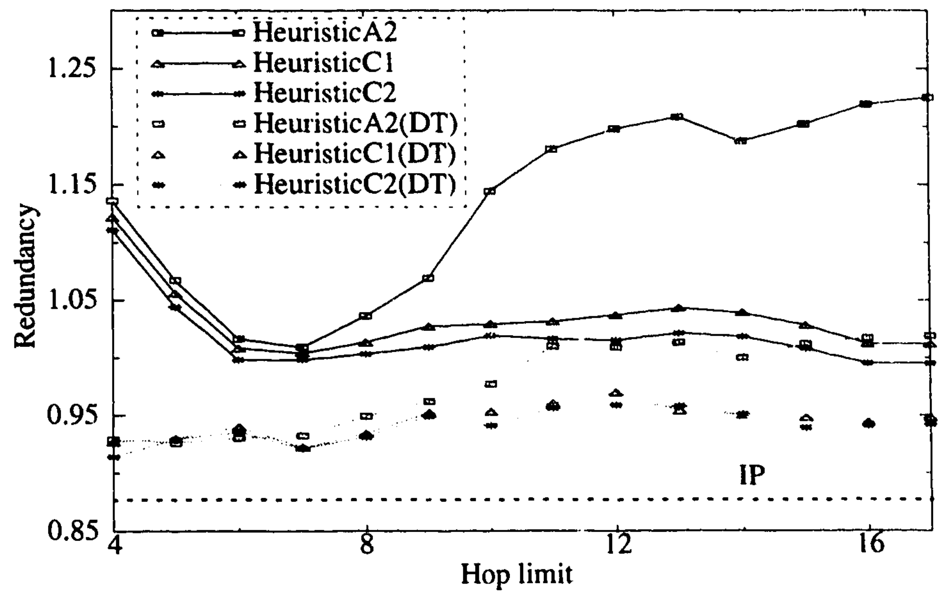


Figure 2-24: Comparison of heuristics with and without DT for Net2

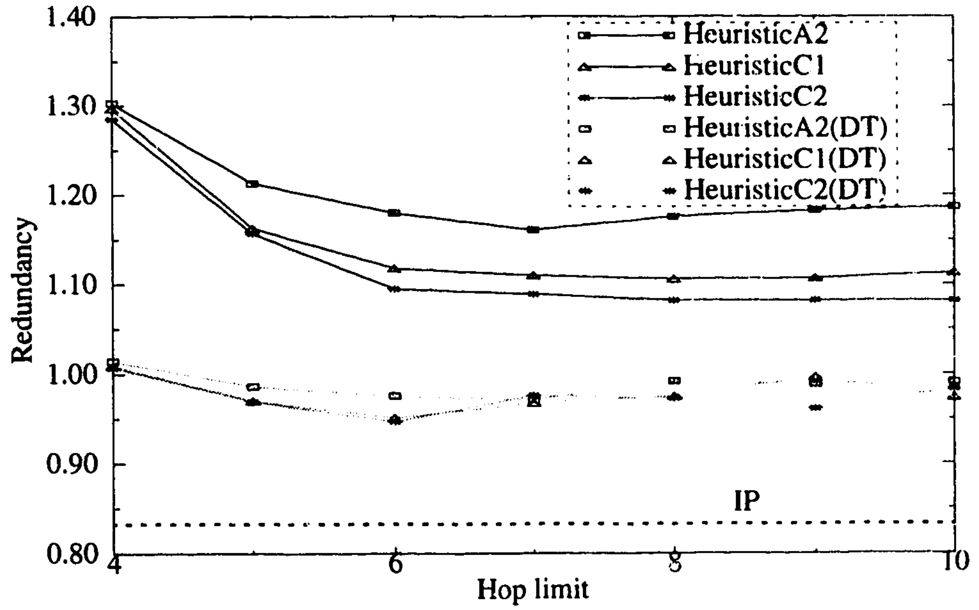


Figure 2-25: Comparison of heuristics with and without DT for Net3

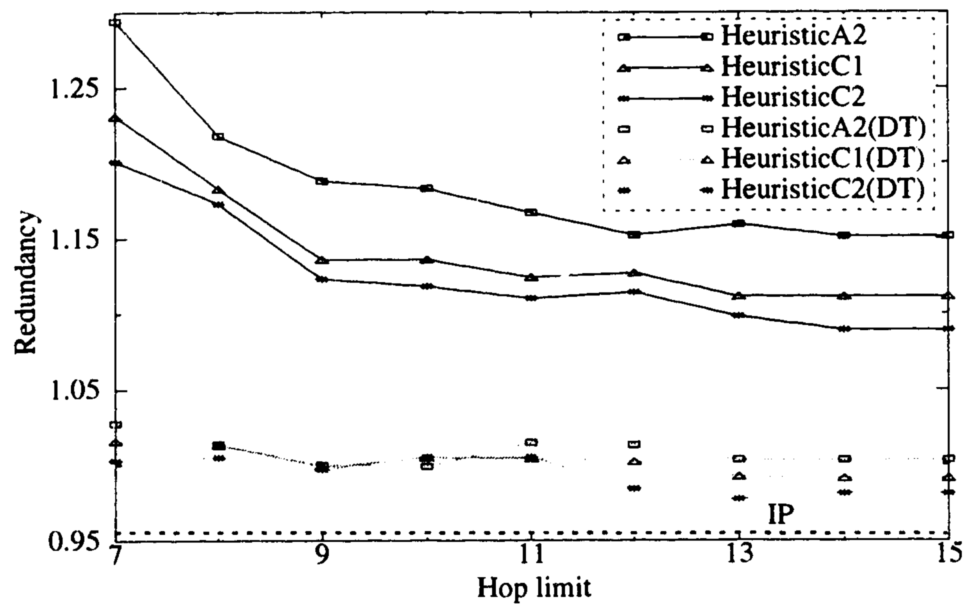


Figure 2-26: Comparison of heuristics with and without DT for Net4

Design tightening, in the case of the large networks Net3 and Net4, increased the capacity efficiency of all the heuristics. Tables 2-9 and 2-10 compare redundancy for HeuristicA2, HeuristicC1 and HeuristicC2 obtained after DT with that of IP. Table 2-9 uses practical hop limits for IP and all three heuristics. Table 2-10 compares the redundancy for these heuristics with those of IP using large hop limits. IP values for Tables 2-9 and 2-10 are the same as those in Tables 2-5 and 2-6 respectively.

Table 2-9: Comparison of heuristics after DT with IP at practical hop limits

Network	Hop limit	IP redun-dancy	HeuristicA2		HeuristicC1		HeuristicC2	
			Redun -dancy	% Excess redun-dancy w.r.t IP	Redun -dancy	% Excess redun-dancy w.r.t IP	Redun -dancy	% Excess redun-dancy w.r.t IP
Net1	6	0.4992	0.532	6.57	0.53	6.17	0.533	6.77
Net2	6	0.919	0.93	1.2	0.922	0.32	0.921	0.22
Net3	6	0.899	0.975	8.45	0.951	5.78	0.947	5.34
Net4	11	0.993	1.014	2.11	1.004	1.11	1.005	1.21

Table 2-10: Comparison of heuristics after DT with those of IP using large hop limits

Network	IP redun-dancy	HeuristicA2		HeuristicC1		HeuristicC2	
		Redun-dancy	% Excess redun-dancy w.r.t IP	Redun-dancy	% Excess redun-dancy w.r.t IP	Redun-dancy	% Excess redun-dancy w.r.t IP
Net1	0.4992	0.532	6.57	0.53	6.17	0.533	6.77
Net2	0.875	0.93	6.28	0.922	5.37	0.921	5.26
Net3	0.841	0.975	15.93	0.951	13.08	0.947	12.6
Net4	0.963	1.014	5.3	1.004	4.25	1.005	4.36

Table 2-9 shows an improvement in redundancy after design tightening for all three heuristics. All heuristics are within 9% of the IP results. Also the results for Net1, Net2, and Net3 are within 7% of those of IP using large hop limits, while those of Net3 are within 16% as shown in Table 2-10. The results for HeuristicA2 are close to that of HeuristicC1 and HeuristicC2, therefore the improvement in capacity efficiency in its case is even more than that of HeuristicC1 and HeuristicC2. Table 2-11 shows percentage gain in redundancy after design tightening for all three heuristics.

Table 2-11: Decrease in redundancy for HeuristicA2, HeuristicC1, and HeuristicC2 after DT

Network	HeuristicA2 % decrease after DT	HeuristicC1 % decrease after DT	HeuristicC2 % decrease after DT
Net1	11.132	11.132	9.75
Net2	10.41	9.327	8.36
Net3	21.52	16.72	14.99
Net4	14.85	12.251	10.45

The time taken by the add0_sub1 design tightener for HeuristicA2, HeuristicC1, and HeuristicC2 designs was found to be nearly the same in all cases. The execution time was also observed to be independent of the hop limit used by the max-latching heuristics. Table 2-12 presents the execution times at practical hop limits for all three heuristics and IP.

Table 2-12: Execution times for DT

Network	Hop limit	Heuristic A2 (sec)	Heuristic C1 (sec)	Heuristic C2 (sec)	IP	
					generation of constraint set (sec)	CPLEX^{TM} run (sec)
Net1	6	2.23	2.18	2.56	1.6	10.05
Net2	6	4.09	4.13	3.6	0.8	1.00
Net3	6	70.2	67.55	78.4	3.5	45
Net4	11	43.47	45.67	49.6	6.83	226

For all three heuristics, design tightening took less than 80 seconds. Therefore, design tightening can be used with max-latching heuristics for fast and efficient SCP.

2.8 Software Implementation

All the max-latching heuristics are coded using the "C" programming language. The code consists of a separate routine for finding the restoration routeset and for each heuristic. The executable version of this code is started by typing **scp** at the UNIX prompt. The command line arguments for this tool are:

scp { input network file with fixed w_i }

{ output network file to hold final s_i assignments }

{ max-latching heuristics used }

{ hop limit for the max-latching heuristics }

{ output log file name }

The output network file contains the final SCP design, and the output log file contains information about the final network design, including the number of spare, working, and total links, total real distance, redundancy, hop limit, and the best and the worst sequences in case of HeuristicC1 and HeuristicC2.

2.9 Summary

This chapter introduced and explored some heuristics for fast SCP based on the "max-latching" principle. All the heuristics involve three main steps: (1) generation of the restoration routeset consisting of all distinct, topologically eligible, restoration routes for each span; (2) spreading the unrestorable working capacity of each span over its restoration routes to force the networks s_i values; (3) latching the maximum spare capacity values on each span. Two types of max-latching heuristics - order independent and order dependent - were discussed. These two types of heuristic differ from each other at step (2). This chapter discussed order-independent heuristics for conceptual development only, while order-dependent heuristics were discussed in detail, as they are more capacity-efficient.

Three classes of order-dependent max-latching heuristics were considered. Class A heuristics (HeuristicA1, HeuristicA2, and HeuristicA3) place spares for spans in decreasing order of their contribution towards network restorability, while class B (HeuristicB1) sorts spans by increasing working capacity. Class C heuristics (HeuristicC1 and HeuristicC2) use a group of partially random sequences as opposed to only one sequence. HeuristicC1 uses S sequences with the first span distinct in each sequence, while HeuristicC2 uses $S * (S-1)$ sequences with the first two spans distinct in each sequence. In all cases the minimum or near-minimum redundancy was observed within four hops of the minimum possible hop limit value.

Three performance parameters: redundancy, execution time, and memory usage, were studied at different hop limit values for four transport networks. In the ideal case, redundancy decreases monotonically with hop limit but for max-latching heuristics, the variation in redundancy with hop limit deviated from ideal at high hop limit values. Therefore, a general graph of redundancy vs. hop limit for max-latching heuristics was developed.

The time complexity for steps (1) and (3) is $O(S \cdot H \cdot d_{avg}{}^{H})$ and $O(S \cdot H \cdot d_{avg}{}^{H})$ respectively. The complexity of step (2) depends on the complexity of finding the span order. For heuristics in which steps (1) and (3) dominate the time complexity, execution time increases exponentially with hop limit. Memory usage for all max-latching heuristics is dominated by the memory occupied by the restoration routeset. Using the simple storage scheme outlined, $O(S \cdot H \cdot d_{avg}{}^{H})$ memory locations are required to store the routeset.

The dependence of redundancy on hop limit characteristics for max-latching heuristics is a function of the order in which spans are restored and of the working capacity distribution in the network. For some working capacity distributions, redundancy decreases monotonically with hop limit, while for others it first decreases and then increases monotonically.

The graphs of redundancy vs. hop limit for all the heuristics showed that class C gives the best results. Also, class A and B sequences were dependent on the working capacity distribution of the network. On the other hand, class C heuristics were more capacity-efficient and less working capacity distribution dependent than the other two classes. HeuristicC2 was the most capacity efficient followed by HeuristicC1.

In class A, HeuristicA2 has the least complexity, $O(S \cdot H \cdot d_{avg}{}^{H})$, where S, H, and d_{avg} are the number of spans, hop limit, and average node degree, respectively. HeuristicB1 has the same complexity as HeuristicA2, while HeuristicC1 and HeuristicC2 have complexities $O(S^2 \cdot H \cdot d_{avg}{}^{H})$ and $O(S^3 \cdot H \cdot d_{avg}{}^{H})$ respectively. HeuristicC2 is too complex to be considered for fast SCP.

The redundancy values of SCP designs obtained for practical hop limit values for HeuristicA2, HeuristicB1, HeuristicC1, and HeuristicC2 were compared with those of IP. The results of HeuristicC1 and HeuristicC2 were within 23% of those of IP using practical hop limits for all cases, while HeuristicA2 and HeuristicB1 were within 32%. The redundancy results for HeuristicC1 and HeuristicC2 were also compared with those of IP using large hop limits and were found to be within 20% for all networks except Net3. For Net3, the results were within 32%. The redundancy for all class A and class B heuristics were very high, therefore these heuristics cannot be considered for SCP.

Design tightening, which involves simple elimination of surplus spare capacity from the SCP designs while maintaining 100% restorability, was used to improve the capacity efficiency of the SCP designs. Designs from heuristics HeuristicA2, HeuristicC1, and HeuristicC2 were tightened. The results at practical hop limits for all three heuristics were within 2% of each other.

At practical hop limits, after design tightening, the redundancy values for HeuristicA2, HeuristicC1 and HeuristicC2 were within 9% of those of IP using practical hop limits. The redundancy results for these heuristics were also compared with those of IP using large hop limits and were found to be within 7% of IP for Net1, Net2, and Net4 while for Net3 the results were within 16%.

The complexity of design tightening is $O(S^2 \cdot \log(w_{max}) \cdot N \cdot \log(N))$, where N is the number of nodes and w_{max} is the maximum working capacity on a span in the network. Therefore, design tightening can be used for fast and efficient SCP, especially with HeuristicA2, HeuristicB1, and HeuristicC1.

3 Network Reduction and Expansion

This chapter discusses $O(N^2)$ complexity approaches for network reduction and expansion, where N is the number of nodes in the network. These approaches are used to speed up the execution of SCP algorithms. Figure 1-1 shows how the network reduction and expansion stages can be used in conjunction with a SCP algorithm.

The problem for network reduction can be stated as: *Given a network, is it possible to reduce it to the maximum extent possible, such that the worst case complexity of the algorithm used for reduction is $O(N^2)$, and the total spare capacity of the reduced portion of the network is close to optimal?*

This chapter considers three approaches for network reduction. These are (i) fixed topology identification and substitution, (ii) greedy reduction, and (iii) backtracking. All the approaches involve three main steps: (1) identification of subtopologies; (2) SCP for the identified subtopology; (3) reduction of the subtopology by replacing it with a reduced span having equivalent spare and working capacities as viewed from the remaining surrounding network.

Fixed topology identification and substitution identifies and reduces predefined, fixed (canonical) subtopologies in the network. All subtopologies are identified and reduced in a single step. On the other hand, *greedy reduction* and *backtracking* are more general approaches as these identify and reduce a large range of subtopologies in multiple steps. In each step, a *subtopology of the network* consisting of a chain is identified and reduced. A *chain* is a concatenation of degree two nodes. This chapter presents the fixed topology identification and substitution for conceptual completeness and emphasizes greedy reduction and backtracking in detail.

The second step, spare capacity placement for the subtopology, is different for the fixed topology method than for the other two. A fixed topology can be spared by solving a fixed system of constraints. The chain subtopologies dealt with by greedy reduction and backtracking, on the other hand, appear in a wide variety of forms and require an algorithmic approach to sparing.

After the SCP of the reduced network, the spare and working capacity equivalents of some reduced spans may change. The network expansion stage modifies the SCP of these subtopologies to accommodate the changes in spare and working capacity equivalents. This stage is different in detail for all three approaches.

This chapter is organized into five sections. Sections 3.1, 3.2, and 3.3 discuss fixed-topology identification and substitution, greedy reduction, and backtracking respectively. Section 3.4 specifies how the heuristics are run, while section 3.5 summarises the chapter.

3.1 Fixed Topology Identification and Substitution

Fixed topology identification and substitution identifies and reduces a predefined set of subtopologies. The predefined set of subtopologies are those common in real transport networks, and for which an efficient SCP can be done through elementary equations.

The most complex step in this method is that of directly identifying whole subtopologies of given predefined forms. The identification algorithm visits each node, and looks for any predefined subtopology around that node. If a subtopology is identified, it is immediately reduced to an equivalent span.

All the subtopologies considered in this work have at most two nodes connecting them to the rest of the network. These two end nodes have degree of more than 2. The complexity of identifying a subtopology at a given node in the network is *O(N)* because in the worst case *O(N)* nodes are checked for identifying a subtopology in its proximity.

Figure 3-1 shows the predefined set of fixed subtopologies, consisting of a chain, two four node topologies, and some five node subtopologies. The end nodes of the subtopologies, nodes 1 and 2, may be adjacent. The subtopologies may also have only one node connecting them to the network.

For subtopologies with high overall connectivity, only an IP can guarantee an optimal SCP as all the cutset constraints must be solved simultaneously. Fortunately these subtopologies are not very common in real transport networks. For example, none of the four test networks contains the subtopology of Figure 3-2, which is the complete graph of four nodes named K4. The chances of occurrence of higher order subtopologies such as K5 or K6 are even lower. Therefore, only K4 is considered.

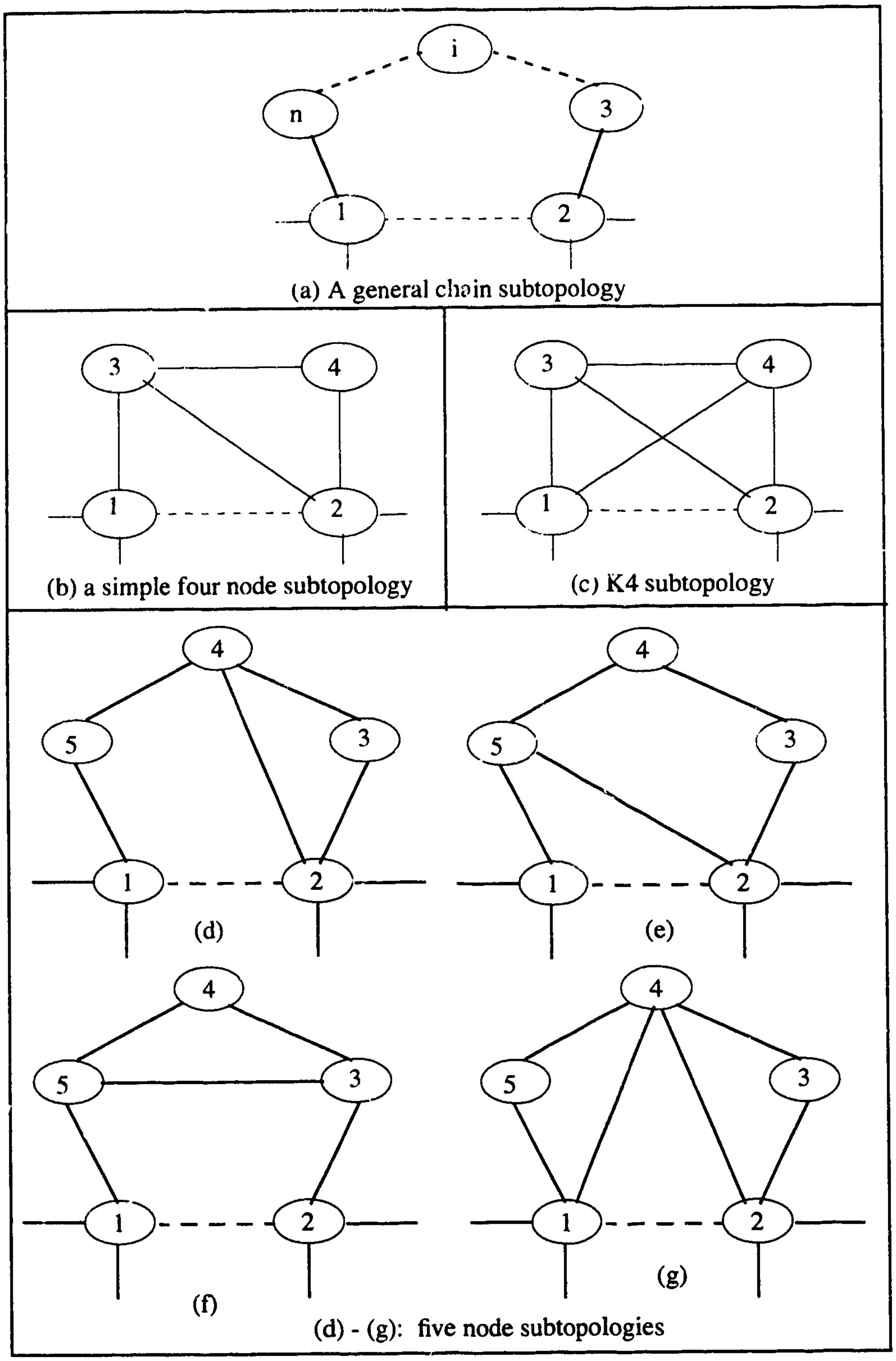

i
n
3
1
2
(a) A general chain subtopology
3
4
1
2
(b) a simple four node subtopology
3
4
1
2
(c) K4 subtopology
4
5
3
1
2
(d)
4
5
3
1
2
(e)
4
5
3
1
2
(f)
4
5
3
1
2
(g)
(d) - (g): five node subtopologies

Figure 3-1: The predefined set of subtopologies

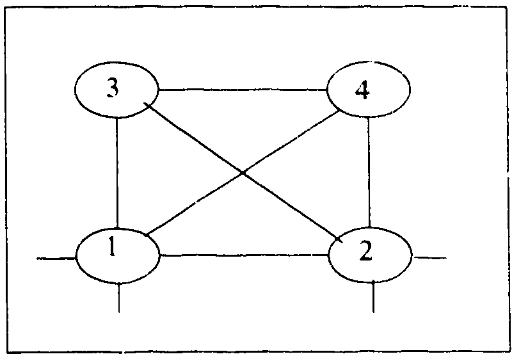


Figure 3-2: Example of a subtopology (between 1-2) having a high degree of internal connectivity

3.1.1 SCP Calculations for Subtopologies

The chain is the simplest subtopology. Figure 3-3 shows a chain consisting of three spans with w_i working links and s_i spare links. For 100% restoration, the spare capacities on spans 1, 2, and 3 are given by:

$$s_1 \geq max(w_2, w_3) \tag{3-1}$$

$$s_2 \geq max(w_1, w_3) \tag{3-2}$$

$$s_3 \geq max(w_1, w_2) \tag{3-3}$$

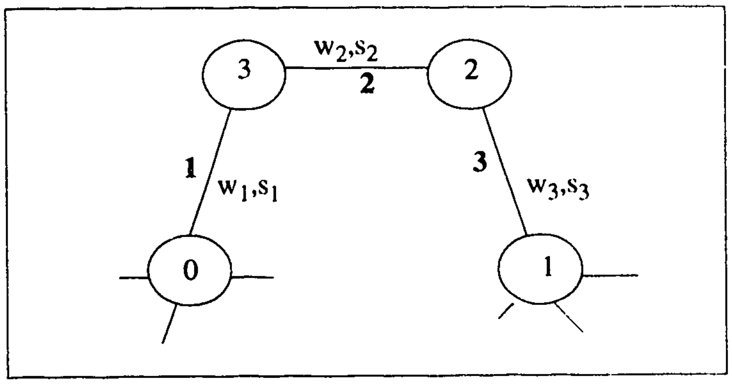


Figure 3-3: A chain subtopology

For a chain of n spans, the above equations can be extended to:

$$s_i \geq max(w_j) \qquad for\ i,j = 1\ to\ n,\ and\ i \neq j \tag{3-4}$$

The SCP of the other subtopologies involves solving a system of equations. The main objective is to minimize the total spare capacity of a subtopology. The secondary objectives are to minimize the *working capacity equivalent*, W_{eq}, and to maximize its *spare capacity equivalent*, S_{eq}. The S_{eq} of a subtopology is the spare capacity offered by it to the rest of the network between the end nodes connecting it to the rest of the network. Its W_{eq} is the working capacity not restored internally by the subtopology. The SCP equations are solved such that after SCP a subtopology is 100% restorable if W_{eq} working links are made restorable by spares in the rest of the network.

For example, consider the simple four node network of Figure 3-4. If this subtopology is allowed to depend on the rest of the network for restoration of some of its working links then only cutsets 1, 2, and 3 need to be considered.

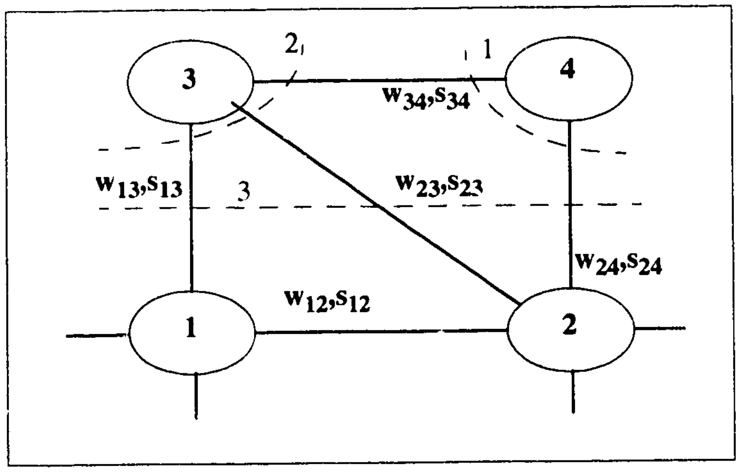


Figure 3-4: A simple four node subtopology connected to the remaining network at nodes 1 and 2

The working and spare capacities on each span are w_{ij} and s_{ij}, where i and j are the end nodes of a span. The equations for full restorability considering cutsets 1, 2, and 3 are:

Cutset 1:

$$s_{34} \geq w_{24} \tag{3.5}$$

$$s_{24} \geq w_{34} \tag{3.6}$$

Cutset 2:

$$s_{13} + s_{23} \geq w_{34} \tag{3.7}$$

$$s_{13} + s_{34} \geq w_{23} \quad (3.8)$$

$$s_{23} + s_{34} \geq w_{13} \quad (3.9)$$

Cutset 3:

$$s_{13} + s_{23} \geq w_{24} \quad (3.10)$$

$$s_{13} + s_{24} \geq w_{23} \quad (3.11)$$

$$s_{23} + s_{24} \geq w_{13} \quad (3.12)$$

The equations for cutsets 2 and 3 can be rewritten by combining (3.7) with (3.10), (3.8) with (3.11), and (3.9) with (3.12):

$$s_{13} + s_{23} \geq max\,(w_{34}, w_{24}) \quad (3.13)$$

$$s_{13} + min\,(s_{24}, s_{34}) \geq w_{23} \quad (3.14)$$

$$s_{23} + min\,(s_{24}, s_{34}) \geq w_{13} \quad (3.15)$$

The S_{eq} and W_{eq} of the subtopology in Fig. 3-4, viewed between nodes 1 and 2 are:

$$S_{eq} = s_{12} + min\,(s_{13}, s_{23} + min\,(s_{24}, s_{34})\,) \quad (3.16)$$

$$W_{eq} = max\,(\Delta w_{ij}) \text{ for } i,j = 1 \text{ to } 5, \text{ and } i \neq j \quad (3.17)$$

where Δw_{ij} is the working capacity of span i-j that is not internally restorable within the subnet.

Now let $w_{12} = 10$, $w_{13} = 100$, $w_{23} = 150$, $w_{24} = 100$, and $w_{34} = 50$. Equations (3.5) and (3.6) give the minimum values for s_{24} and s_{34} as 50 and 100 respectively. Substituting the w_{ij}'s into equations (3.13) to (3.15), we get:

$$s_{13} + s_{23} \geq 50 \quad (3.13a)$$

$$s_{13} + min\,(s_{24}, s_{34}) \geq 150 \quad (3.14a)$$

$$s_{23} + min\,(s_{24}, s_{34}) \geq 100 \quad (3.15a)$$

The approach we use is to incrementally increase the values of one variable or a combination of variables so that all constraints are satisfied while using the minimum total number of spares. The best variable or combination of variables is chosen at each step. In cases where there are several equally good possibilities, the choices which also increase the S_{eq}

of the subtopology are considered. If many such possibilities exist for increasing S_{eq}, then the choice that benefits mostly the constraints requiring the maximum spare capacity is preferred over others as doing so may eventually require fewer spares. It is very likely that in the process of satisfying these constraints, any constraints requiring less spare capacity will also be satisfied.

For example, combinations of s_{13} with s_{23}, and s_{13} with s_{24} can be used to increase the S_{eq} of Figure 3-4. Of these two combinations, the latter appears in equation (3.14a) which requires the maximum spare capacity among all constraints. Therefore the values of s_{13} and s_{24} are increased to 50 and 100 respectively, and all three constraints are satisfied. Recall that cutset 1 dictates $s_{34} \geq 100$ and $s_{24} \geq 50$. Figure 3-5 shows the final s_{ij} values. The S_{eq} of the subtopology is 50 (equation 3.16). The results of this case-specific SCP formulation were tested for all 4! possible relative rankings of w_{ij} values in the subtopology of Figure 3-5 and were found to be optimal.

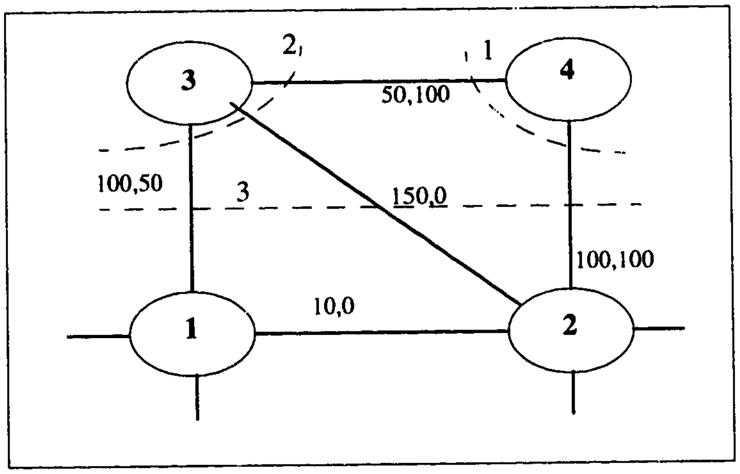


Figure 3-5: Final SCP internal design of the 4-node (1-2) subtopology

Figure 3-6 shows an example of a subtopology whose SCP can be solved by satisfying the constraints from a single cutset. Figure 3-6(a) shows all the cutsets that need to be solved for SCP, while Figure 3-6(b) shows the flow-equivalent single cutset representation. In this system of constraints, each variable s_{ij} appears in all but one constraint. Therefore, any set of constraints can be satisfied by incrementing the values of at most two variables which appear in all unsatisfied constraints.

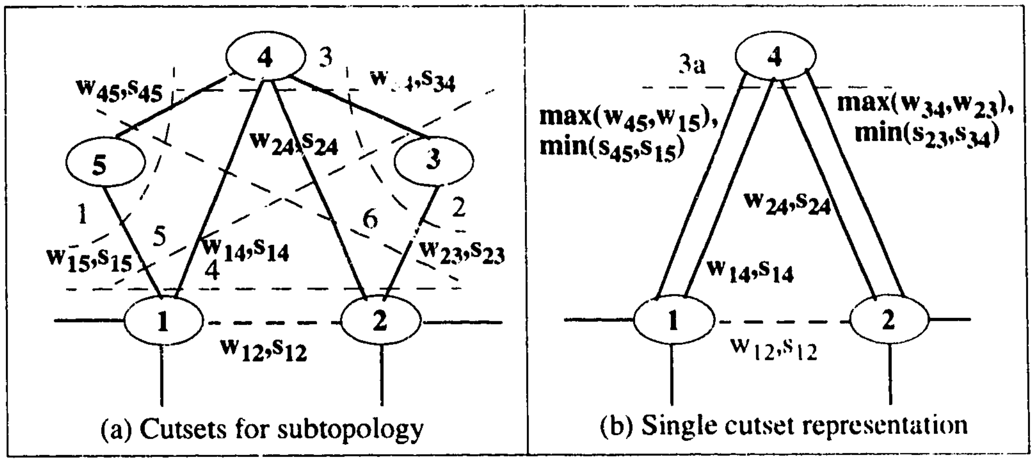

4
3
5
1
2
6
w_{45},s_{45}
w_{34},s_{34}
w_{24},s_{24}
w_{15},s_{15}
w_{14},s_{14}
w_{23},s_{23}
w_{12},s_{12}
(a) Cutsets for subtopology
3a
max(w_{45},w_{15}), min(s_{45},s_{15})
max(w_{34},w_{23}), min(s_{23},s_{34})
(b) Single cutset representation

Figure 3-6: An example of a subtopology whose SCP can be reduced to solving a system of three constraints

This method can also be used to solve a system of constraints determined by more than one cutset, but the SCP for such cases may not be optimal. This is because the sharing of variables among the constraints is low. Hence the values of many variables must be increased as opposed to at most two variables. For example, in the case of the three constraints given by equations (3.18) to (3.20), there are no variables common to two constraints. To satisfy all three constraints simultaneously, the values of at least three variables must be increased. For this case, 27 such combinations are possible, and therefore it is hard to find a combination of increments which eventually leads to an efficient result.

$$s_1 + s_2 + s_3 \geq w_1 \tag{3.18}$$

$$s_4 + s_5 + s_6 \geq w_2 \tag{3.19}$$

$$s_7 + s_8 + s_9 \geq w_3 \tag{3.20}$$

3.1.2 Network Expansion

Network expansion is done after the network has been reduced and spared. During the sparing of the reduced network, a reduced span may be required to supply more spares to the network than placed during reduction. The network expansion stage modifies the SCP of the subtopology represented by such a span, increasing its S_{eq} to the required value.

A simple method to increase the S_{eq} of a subtopology is to increase the S_{eq} associated with its shortest route, for example route 1-2 in Figure 3-5. A general network expansion routine was devised to increase the S_{eq} of the shortest route for all subtopologies.

A more advanced approach in network expansion is to consider reducing W_{eq} while increasing S_{eq}. The implementation of this approach is subtopology specific. A reduction in W_{eq} may make some spare capacity redundant elsewhere in the network. This increases the subsequent gain from design tightening.

3.1.3 Limitations of Fixed Topology Identification and Substitution

The complexity of fixed topology identification and substitution depends on the complexity of identifying subtopologies. The complexity of identifying a single subtopology at a given node is $O(N)$ because in the worst case $O(N)$ nodes around every node in the network are checked for identifying a subtopology in its proximity. The identification algorithm searches all N nodes in the network, therefore its complexity is $O(N^2)$. However, it is difficult to implement an identification algorithm that identifies all target subtopologies in the network especially if there is a large number of fixed subtopologies to consider.

An efficient and practical SCP for subtopologies is limited to those whose SCP is reducible to solving a system of constraints where each constraint arises from only a single cutset. For other cases, an efficient SCP cannot be guaranteed. A large number of subtopologies do not fall in this category, such as the subtopology of Figure 3-7. In addition, each subtopology has a specific SCP. Therefore, the implementation becomes difficult as the size of the predefined set of subtopologies increases.

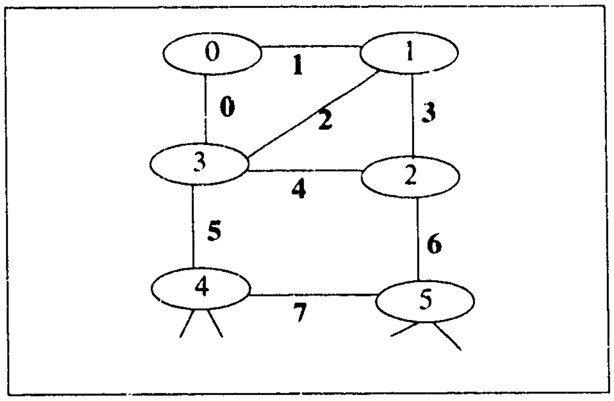


Figure 3-7: An example of a subtopology whose SCP involves solving interrelated cutset equations

3.2 A Greedy Method for Successive Chain-wise Reduction

In real transport networks, subtopologies consisting of chains of degree 2 nodes are reasonably common. For example, Net1 contains three and Net2 contains four significant chain-reducible subtopologies, as shown in Figures 3-8 and 3-9. Therefore, identification and reduction of these chain subtopologies is a potential approach for network reduction.

Large optimization problems can sometimes be solved by breaking them into small manageable sub-problems or stages and solving them in a certain order. At each stage a large number of feasible solutions are possible. The greedy method works by picking the optimum solution at each stage. For example the forward synthesis (FS) phase of the SLPA algorithm [8] is "greedy". The practical advantage of greedy algorithms is that they are usually straightforward, easy to understand and easy to code. Their main disadvantage is that there is no guarantee that making locally optimal improvements at each stage will yield a globally optimal solution.

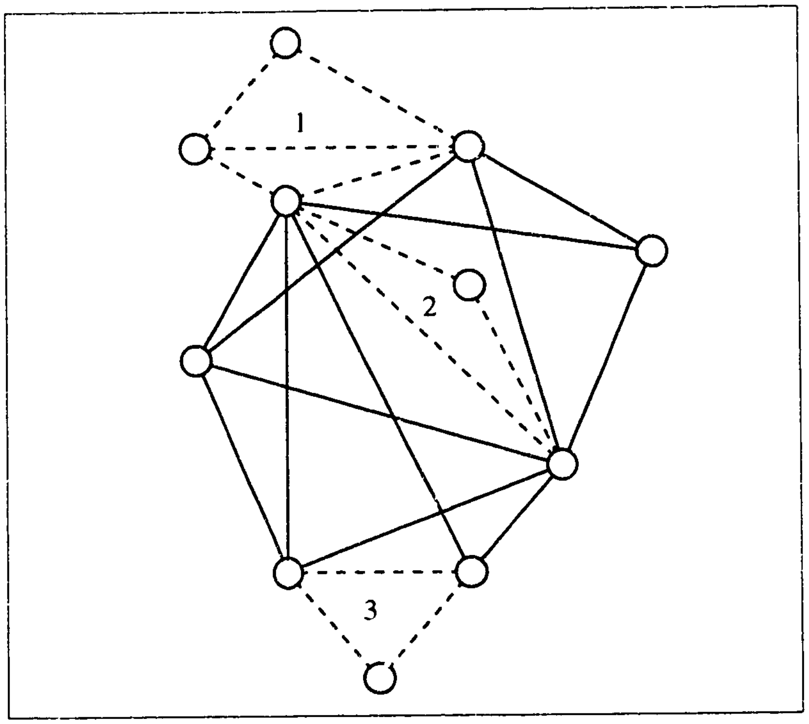


Figure 3-8: Chain-reducible subtopologies in Net1

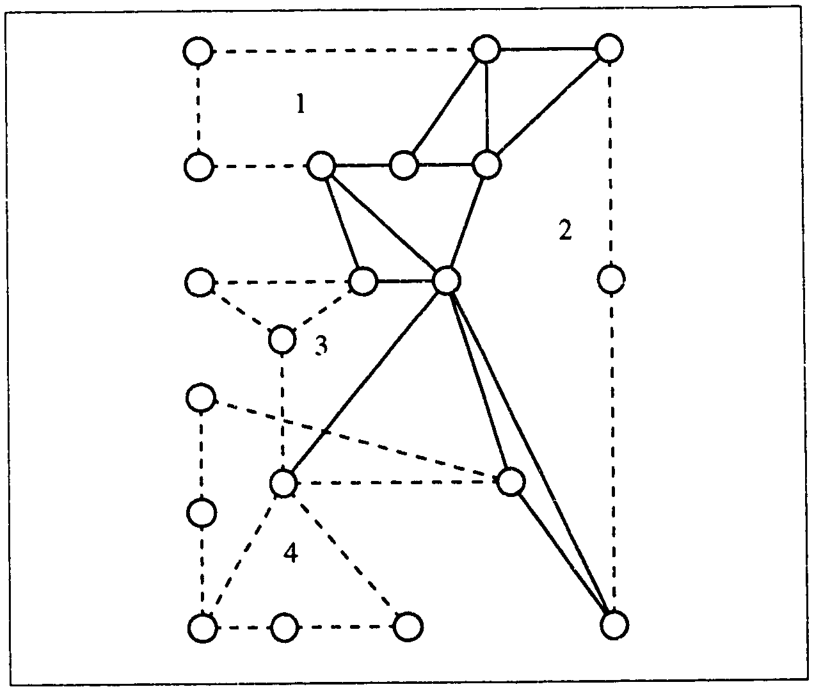
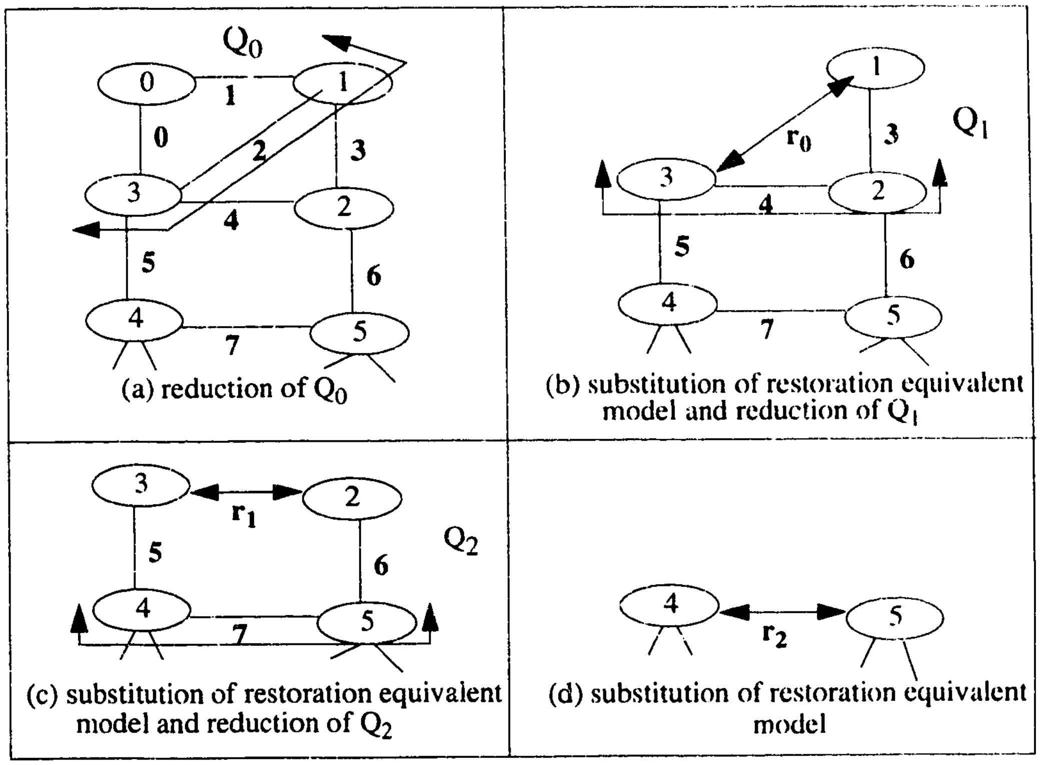


Figure 3-9: Chain-reducible subtopologies in Net2

The greedy method can be used to reduce and place spares for all subtopologies that consist of chains. Let P be the problem of placing spares for a subtopology consisting of chains. Problem P can be broken into a series of subproblems $(Q_0, Q_1, \ldots, Q_n)$, where Q_i identifies a part of the subtopology and places spares for it. In order to solve Q_i, the output of Q_{i-1} is needed. Figure 3-10 shows how subtopologies can be collapsed through a series of reductions.



Q_0
0
1
2
3
4
5
6
7
(a) reduction of Q_0
r_0
Q_1
(b) substitution of restoration equivalent model and reduction of Q_1
r_1
Q_2
(c) substitution of restoration equivalent model and reduction of Q_2
r_2
(d) substitution of restoration equivalent model

Figure 3-10: A greedy method for chain-wise successive network reduction

In Figure 3-10(a), state Q_0 involves reduction of the subtopology formed by span 2 and the chain of spans 0 and 1. Reduced span r_0 is the subtopology's restoration equivalent model. The result is a similar subtopology consisting of span 3 and the chain of spans r_0 and 4 (Figure 3-10(b)). Reducing the chain in state Q_1 is therefore similar to reducing the chain in state Q_0. Also, Q_1 depends on the results of Q_0 and can be dealt with only after Q_0. Reduced span r_1 represents the restoration equivalent model of the topology considered in state Q_1. After reducing the chain in state Q_1, Q_2 is reduced (Figure 3-10(c)).

The greedy method is used to find the SCP for each state by optimizing three parameters. These are the total spare capacity, T, of the chain of Q_i, the working capacity equivalent, W_{eq}, of Q_i, and the spare capacity equivalent, S_{eq}, of Q_i. At each state, T is minimized. After finding the minimum total sparing, we maximize the S_{eq} of the chain and minimize its W_{eq} because a reduced span with a higher S_{eq} value can contribute more towards the restoration of the rest of the external network, and a reduced span with a lower W_{eq} requires less support from the rest of the network. The example presented in Figure 3-11 shows the need for minimizing W_{eq}. Reduced span r_0 in Figure 3-11 forces spans 0, 1, and 2 to have 100 spare links. If the W_{eq} of r_0 can be reduced then the sparing of spans 0, 1, and 2 can also be reduced.

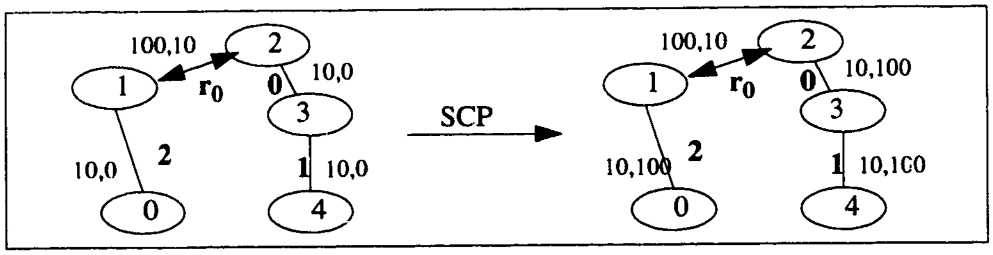


Figure 3-11: An example showing the importance of minimizing W_{eq} at each state

After minimizing T, we cannot always increase S_{eq} and decrease W_{eq} simultaneously. The optimal solution at state Q_i is somewhere between the maximum S_{eq} and minimum W_{eq} designs but this solution can be known only in some subsequent state. In such cases, the optimum solution at state Q_i can be realized only by backtracking from some state Q_{i+j} ($j > 0$) to Q_i, and re-working the SCP from state Q_i and onwards. The backtracking approach is very complex and is discussed in detail in Sec. 3.3. The greedy approach in such cases is to maximize S_{eq} or minimize W_{eq}. This thesis discusses two greedy heuristics, NRG1 and NRG2. When we must choose between S_{eq} and W_{eq}, NRG1 always maximizes S_{eq} while NRG2 always minimizes the W_{eq} of a chain. The primary objective of both these heuristics is to minimize T at each state Q_i.

3.2.1 Overall Greedy Reduction Algorithm

Each step of reduction involves three main operations:

(1) Identification of a chain and the span joining its end nodes (if present).

(2) SCP for the identified chain.

(3) Substitution of a restoration equivalent model for the reduced span representing this chain, previously identified parallel chains, and the span joining the end nodes.

Networks may have several chain-reducible subtopologies that can be collapsed through the greedy method. Each of these subtopologies is reduced separately. Figure 3-12 shows the overall algorithm and its interface with the max-latching heuristics for SCP of the reduced network. The method first picks an arbitrary node of degree two. It then attempts to reduce the subtopology associated with this node. A subtopology is reduced in a loop of n steps $(Q_0,..., Q_n)$, and each iteration of this loop starts in state Q_i. Spans belonging to reduced subtopologies are removed from the network and the spare capacity assigned to these spans is stored. The nodes belonging exclusively to these subtopologies are also removed from the network. The reduced network is obtained after the network reduction algorithm has gone through all the nodes. This reduced network is then passed on to the max-latching SCP heuristics for its SCP. During the SCP of the reduced network, the S_{eq} of some reduced spans may change. The network expansion stage is used to reflect these changes while expanding the reduced network back to its original form. In some cases the network reduction algorithm reduces the whole network to a single span, and for these cases the network expansion stage simply outputs the SCP design achieved during reduction.

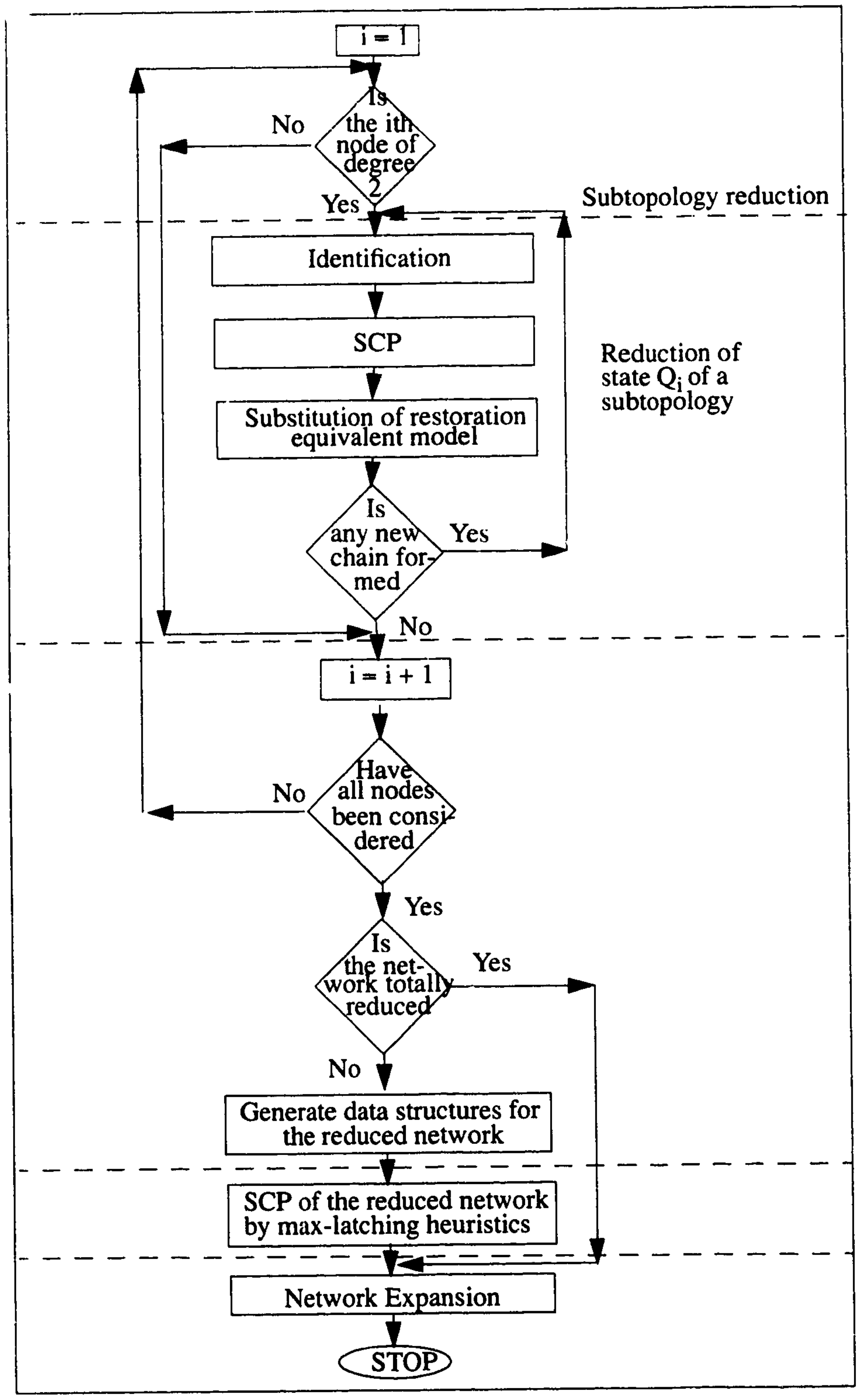

i = 1
Is the ith node of degree 2
No
Yes
Subtopology reduction
Identification
SCP
Substitution of restoration equivalent model
Reduction of state Q_i of a subtopology
Is any new chain formed
Yes
No
i = i + 1
Have all nodes been considered
No
Yes
Is the network totally reduced
Yes
No
Generate data structures for the reduced network
SCP of the reduced network by max-latching heuristics
Network Expansion
STOP

Figure 3-12: Overall greedy algorithm

3.2.2 Representing Reduced Spans

Reduced spans represent at least one chain and a span joining the end nodes of the chains, if one is present. Depending on whether there are reduced or real spans in the chain, and whether the end nodes of the chain are adjacent to each other at their interface to the surrounding network, the reduced spans are of four types:

1. TYPE1: this reduced span consists of only one chain, and the end nodes of the chain are adjacent to each other. In Figure 3-13 the span formed is TYPE1 if the end nodes 0 and 1 are adjacent. The chain can have reduced as well as real spans.

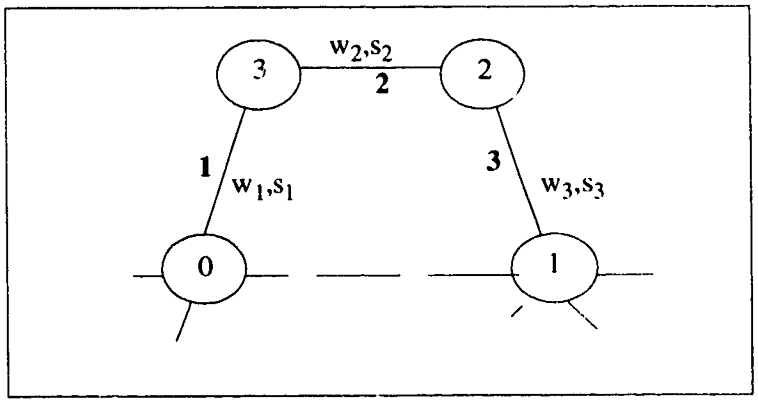


Figure 3-13: A simple subtopology

2. TYPE2: this reduced span consists of only one chain of real spans, and the end nodes of the chain are not adjacent to each other. e.g., Figure 3-13 shows a TYPE2 subtopology if spans 1, 2, and 3 are real and nodes 0 and 1 are not adjacent.
3. TYPE3: this reduced span consists of only one chain which contains at least one reduced span. The end nodes of the chain are not adjacent to each other. In Figure 3-13 if at least one of spans 1,2, or 3 is a reduced span, and the end nodes 0 and 1 are not adjacent, then the subtopology is TYPE3.
4. TYPE4: this reduced span consists of more than one chain in parallel with each chain having the same end nodes. The end nodes may or may not be adjacent to each other as shown in Figure 3-14.

Real spans are also called TYPE0.

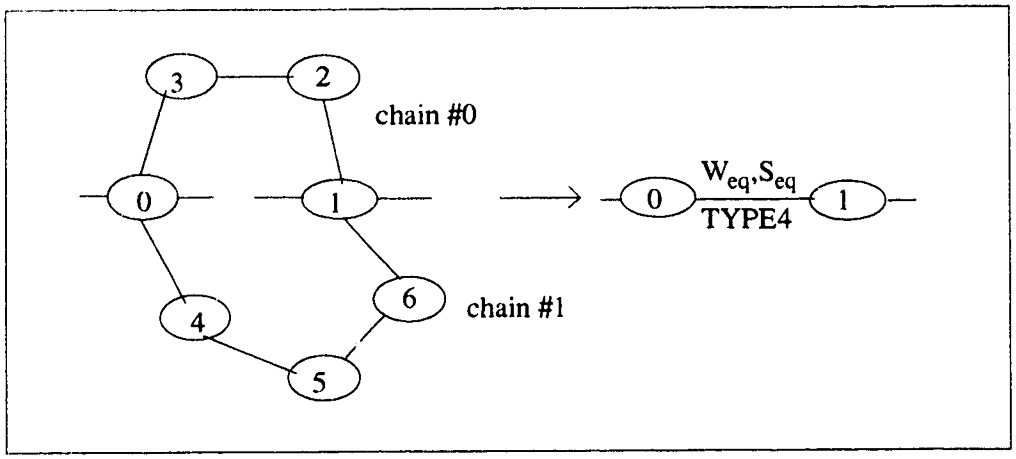


Figure 3-14: A TYPE4 reduced span

3.2.2.1 Restoration Equivalent Model

The restoration equivalent model of a reduced span is made up of two parameters S_{eq} and W_{eq}. In the case of a TYPE1 reduced span, the end nodes of the chain are connected by a real span. The restoration equivalent model for the reduced span plus the span connecting the end nodes is:

$$S_{eq} = S_{eq}(chain) + S_0 \qquad (3.21)$$

$$W_{eq} = max(W_{eq}(chain) - S_0, W_0 - S_{eq}(chain)) \qquad (3.22)$$

where S_0 is the spare capacity and W_0 is the working capacity of the span joining the end nodes of the chain. S_{eq}*(chain)* and W_{eq}*(chain)* are the spare and working equivalents of the chain, and are determined by:

$$S_{eq}(chain) = min(s_i) \qquad i = 1, ..., n \qquad (3.23)$$

$$W_{eq}(chain) = max(w_i) \qquad i = 1, ..., n \qquad (3.24)$$

where n is the number of spans in the chain. Equations (3.23) and (3.24) also give the S_{eq} and the W_{eq} of TYPE2 and TYPE3 spans. The W_{eq} in equation (3.22) can be expressed as:

$$W_{eq} = max(W_{eff}(chain), W_{0,eff}) \qquad (3.25)$$

where $W_{eff}(chain)$ is the effective working equivalent for the chain and $W_{0,eff}$ is the effective working equivalent for the span joining the end nodes of the chain. The effective working equivalents $W_{eff}(chain)$ and $W_{0,eff}$ are the internally unrestored portion of the working equivalents $W_{eq}(chain)$ and W_0:

$$W_{eff}(chain) = W_{eq}(chain) - S_0 \tag{3.26}$$

$$W_{0,eff} = W_0 - S_{eq}(chain) \tag{3.27}$$

In case of a TYPE4 reduced span, more than one chain is present in parallel and the common end nodes of the chains may or may not be adjacent. The restoration equivalent model is:

$$S_{eq} = \sum_{i=1}^{nc} S_{eq}(chain)_i \tag{3.28}$$

$$W_{eq} = max\left(W_{eq}(chain)_i - \sum_{\substack{j=1 \\ j \neq i}}^{nc} S_{eq}(chain)_j \right) \tag{3.29}$$

where nc is the number of chains in the TYPE4 span. Equation (3.29) can also be expressed as:

$$W_{eq} = max(W_{eff}(chain)_i) \tag{3.30}$$

where $W_{eff}(chain)_i$ is the effective working equivalent of chain i.

If the end nodes of the chains are adjacent then the restoration equivalent model including the connecting span becomes:

$$S_{eq} = S_{eq}(chains) + S_0 \tag{3.31}$$

$$W_{eq} = max(W_{eq}(chains) - S_0, W_0 - S_{eq}(chains)) \tag{3.32}$$

or

$$W_{eq} = max(W_{eff}(chains), W_{0,eff}) \tag{3.33}$$

where S_0 is the spare capacity and W_0 is the working capacity, $W_{0,eff}$ is the working capacity equivalent of the span joining the end nodes of the chains, and $W_{eff}(chains)$ is the working capacity equivalent of the chains.

3.2.2.2 Non-linearity in Reduced Spans

Obviously the S_{eq} of real spans increases linearly as actual spare links are added to the span. But this is not true in general for reduced spans because of serial interrelationships between spans which may mean that adding spares to a span increases S_{eq} up to a point, until a bottleneck arises somewhere else clamping the effect of further additions to the first span considered. Consider the chain shown in Figure 3-15. The S_{eq} of the chain is 30 as drawn but it can be increased up to 50 by adding 20 units of spare capacity on span 0. After the S_{eq} of the chain has been raised to 50, three units of spare capacity are needed for each unit increase in S_{eq}. The behaviour of S_{eq} as additional spare links, X, are invested is given by the following equations:

$$S_{eq} = 30 + X \qquad 0 \le X \le 20 \tag{3.34}$$

$$S_{eq} = 50 + \left\lfloor \frac{X-20}{3} \right\rfloor \qquad 21 \le X \tag{3.35}$$

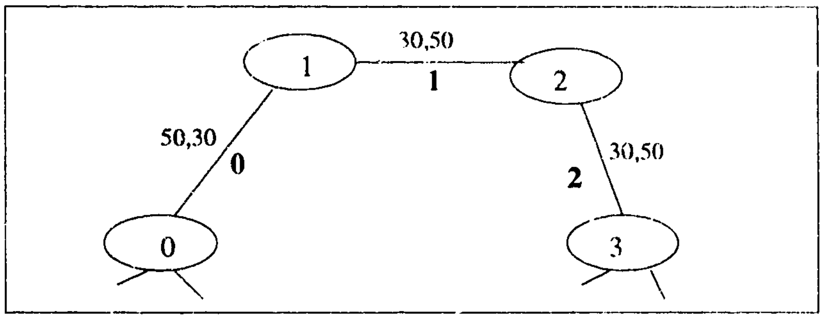


Figure 3-15: Illustrating the non-linearity in S_{eq} of reduced spans as spares are added

Also, unlike real spans, the W_{eq} of reduced spans may decrease with the addition of spare capacity. Consider the subtopology in Figure 3-16. This subtopology can be represented by a TYPE1 span and its W_{eq} as drawn is 50 (equation 3.22). The W_{eff}*(chain)* and $W_{0,eff}$ from equations (3.26) and (3.27) are 50 and 10 respectively. Therefore, the W_{eq} of 50 for this reduced span is due to W_{eff}*(chain)* (equation 3.25). W_{eff}*(chain)* can be decreased by one unit down to a value of 10 for each unit increase in span 3's spare capacity. After this, a further decrease in W_{eq} is possible only by adding two spare capacity units: one each on spans 0 and 3. Therefore, the W_{eq} of some reduced spans can be decreased by adding spares that make the network internally self- restorable, and this decrease is non-linear. The behaviour of W_{eq} for the subtopology of Figure 3-16 is given by:

$$W_{eq} = 50 - X \qquad X \le 40 \tag{3.36}$$

$$W_{eq} = max\left(10 - \left\lfloor \frac{X-40}{2} \right\rfloor, 0\right) \qquad 40 < X \tag{3.37}$$

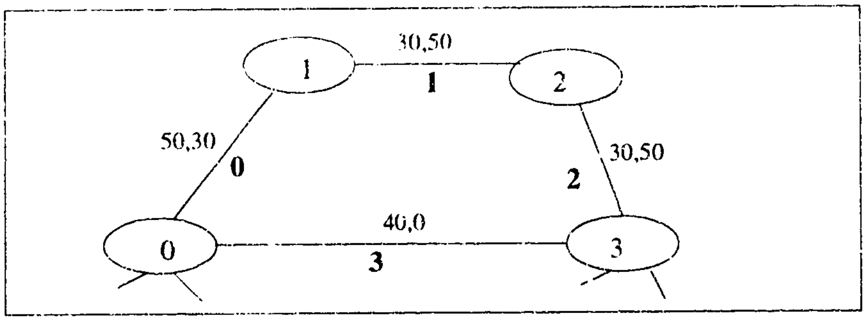


Figure 3-16: Illustrating non-linearity in W_{eq} of reduced spans as spares are added

The non-linear representation of the span includes not only W_{eq} and S_{eq} but also the information about how W_{eq} is decreased and S_{eq} is increased.

3.2.3 Operations on Reduced Spans

Three types of operations can be performed on reduced spans. These operations are increasing S_{eq}, decreasing W_{eq}, and removing spare capacity. The greedy method uses only the first two operations. Removing spare capacity from reduced spans is used in the backtracking method discussed later.

3.2.3.1 Increasing S_{eq}

The S_{eq} of a particular reduced span RSPAN is increased in increments of one through spare capacity addition. While increasing the S_{eq} of RSPAN, the possibility of decreasing its W_{eq} is also analysed. If the number of spare links needed to increase the S_{eq} of a span is the same as that required to simultaneously decrease its W_{eq}, then the W_{eq} is also decreased. The steps involved in increasing the S_{eq} of RSPAN depend on its type:

(a) TYPE1

The S_{eq} of this span can be increased by adding one link to the span joining the end nodes of its chain. S_{eq} can also be increased by decreasing W_{eq}; the decision to do so is made only when this can be done by adding one spare link. The steps for decreasing the W_{eq} of a TYPE1 span are discussed in the next section.

(b) TYPE2

Two steps are involved in increasing the S_{eq} of this type of span:

(1) Find the set of all spans, U, in the chain whose spare capacity is the minimum in the chain. The size of this set is also the number of spares needed to increase the S_{eq} of RSPAN by 1.

(2) Increase the spare capacities of all spans in U by 1.

Consider the subtopology in Figure 3-15. The reduced span representing this chain is of TYPE2 with an S_{eq} of 30. The S_{eq} of this span can be increased by incrementing the spare capacity of span 0.

(c) TYPE3

The steps involved in increasing S_{eq} are:

(1) Find the set of all spans, U, in the chain whose spare capacity is the minimum in the chain.

(2) Find the total spare capacity, T_1, needed to increase the S_{eq} of all spans in U.

(3) Let U_1 be the set of all spans in U having W_{eq} equal to the W_{eq} of RSPAN. If any of the spans in U_1 is real or if the W_{eq} of a reduced span in U_1 cannot be decreased then T_2 is set to infinity, else the total spare capacity, T_2, needed to decrease the W_{eq} of U_1 while increasing their S_{eq} is found. Let T_3 be the spare capacity needed to increase the S_{eq} of the span in $\{U - U_1\}$.

(4) if T_2+T_3 is equal to T_1 then decrease the W_{eq} of U_1 while increasing its S_{eq} and increase the S_{eq} of $\{U - U_1\}$, else increase the S_{eq} of U.

(d) TYPE4

The S_{eq} of a TYPE4 span that consists of a span joining the end nodes of its parallel chains is increased differently from one that has no such span. If there is a span joining the end nodes, S_{eq} can be increased by increasing S_0. S_{eq} can also be increased by decreasing W_{eq}; this is done if it is possible to [illegible] so by adding one spare link. The steps for decreasing the W_{eq} for this case are discussed in the next section.

In case of a TYPE4 span which has no span joining its end nodes, first the minimum number of spares needed to increase its S_{eq} is found. Then the possibility of reducing W_{eq} is analysed. First, all chains whose $W_{eff}(chain)$ is less than the W_{eq} of RSPAN are checked. If any of these chains requires the same number of spares to increase $S_{eq}(chain)$ as is required to increase the S_{eq} of RSPAN then $S_{eq}(chain)$ is increased. This will decrease the W_{eq} of RSPAN.

3.2.3.2 Decreasing W_{eq}

As for S_{eq}, the W_{eq} of RSPAN is decreased one unit at a time. W_{eq} is decreased only through the addition of spare capacity which may also increase S_{eq}. The steps involved in decreasing the W_{eq} of RSPAN depend on its type:

(a) TYPE1

Depending on the values of $W_{eff}(chain)$ and $W_{0,eff}$, there are three cases (see equation 3.25):

(1) $W_{eff}(chain) > W_{0,eff}$: S_0 is increased to decrease $W_{eff}(chain)$.

(2) $W_{eff}(chain) < W_{0,eff}$: S_{eq} of the chain is increased to decrease the $W_{0,eff}$.

(3) $W_{eff}(chain) = W_{0,eff}$: Either:

(i) Find the number of spares needed to decrease the W_{eq} of the chain while increasing its S_{eq}, or

(ii) Find the number of spares needed to increase S_0 and the S_{eq} of the chain.

If the number of spares needed for (i) is less than (ii) then it is implemented else (ii) is implemented. The S_{eq} of RSPAN increases by 2 units if (ii) is chosen.

(b) TYPE2

The W_{eq} of a TYPE2 span cannot be decreased because it consists only of real spans.

(c) TYPE3

The W_{eq} of this RSPAN can be decreased only when the W_{eq} of all the spans in the chain which have their W_{eq} equal to the W_{eq} of RSPAN can be reduced. If any of these spans is real then W_{eq} cannot be decreased. The steps involved are:

(1) Find the set U of reduced spans having W_{eq} equal to RSPAN's W_{eq}.

(2) Decrease the W_{eq} of all spans in U.

(d) TYPE4

The W_{eq} of a TYPE4 span having a span joining its end nodes is determined using equation (3.33). Depending on the values of $W_{eff}(chains)$ and $W_{0,eff}$, there are three cases:

(1) $W_{eff}(chains) > W_{0,eff}$: S_0 is increased.

(2) $W_{eff}(chains) < W_{0,eff}$: The S_{eq} of the chain requiring the minimum spare capacity among all the parallel chains is increased.

(3) $W_{eff}(chains) = W_{0eff}$: Three subcases must be checked:

(i) Find the number of spares needed to increase S_0 and the S_{eq} of a chain having its $W_{eff}(chain)$ equal to the W_{eq} of RSPAN. If more than one such chain exists then choose the chain that requires the minimum spare capacity of all such chains.

(ii) Of all chains having $W_{eff}(chain)$ less than the W_{eq} of RSPAN, find the one requiring the minimum number of spares to increase S_{eq}.

(iii) Find the number of spares needed to simultaneously increase the S_{eq} and decrease the W_{eq} of a chain having $W_{eff}(chain)$ equal to the W_{eq} of RSPAN. If more than one such chain exists then choose the chain which requires the minimum spare capacity among all such chains.

The option requiring the minimum spare capacity is chosen. In case of a tie (i) is preferred as the S_{eq} of RSPAN increases by 2 units in this case.

The W_{eq} of a TYPE4 span which does not have a span joining its end nodes is given by equation (3.30) and it can be decreased in the following ways:

(1) If there is more than one chain with $W_{eff}(chain)$ equal to the W_{eq} of RSPAN, then find the minimum number of spares needed to increase the S_{eq} of two such chains.

(2) Of all chains with $W_{eff}(chain)$ less than the W_{eq} of RSPAN, find the one requiring the minimum number of spares to increase S_{eq}.

(3) Find the number of spares needed to simultaneously increase the S_{eq} and decrease the W_{eq} of a chain having W_{eq} equal to the W_{eq} of RSPAN. If more than one such chain exists then choose the chain that needs the minimum number of spares.

Out of these three, the option requiring the minimum number of spares is chosen. In case of a tie, (1) is preferred over (2) and (3) as S_{eq} increases by 2 units in case of (1).

3.2.4 Two Greedy Heuristics for Chain-wise Successive Network Reduction

This section presents an example to illustrate and contrast two greedy heuristics or chain-wise successive network reduction, NRG1, NRG2 (see Figure 3-17). The primary aim of these heuristics is to minimize the total spare capacity, T, at every state Q_i. After minimizing T, these heuristics try to simultaneously maximize S_{eq} and minimize W_{eq} of the subnetwork of state Q_i. Because it is not always possible to do so, NRG1 maximizes S_{eq} while NRG2 minimizes W_{eq}.

State Q_0 in Figure 3-17(a) involves identification of chain 1, SCP of the spans in chain 1, and substitution of the restoration equivalent model for the topology formed by chain 1 and span 3. Equation (3-4) is used to find the spare capacities on the spans, and equations (3-21) to (3-22) are used to find the S_{eq} and W_{eq} for the reduced span r_0. After the substitution of span r_0, chain 2 is formed which gives rise to state Q_1 (Figure 3-17(b)).

In state Q_1 the algorithm first minimizes the total spare capacity of chain 2. In chain 2, reduced span r_0 has the maximum W_{eq}, thus setting the spares on all other spans in the chain. Hence, the total spare capacity chain 2 can be decreased if the W_{eq} of span r_0 can be decreased.

At the same time, the S_{eq} of span r_0 must be increased to 60 (equation 3-4) for 100% restorability of chain 2. While increasing the S_{eq} of span r_0, the possibility of simultaneously reducing its W_{eq} is also considered. In order to reduce its W_{eq} by one, three units of spare capacity are needed, and in doing so three units of spare capacity are saved in chain 2, one each on spans 4, 5, and 6. No additional spare capacity is needed to do this. The other option is to increase the S_{eq} of r_0 by putting spare capacity on span 3, but doing so does not decrease the W_{eq} of r_0, so this option is not as good as the previous one. Hence, chain 2 can be made restorable with minimum sparing by reducing the W_{eq} of r_0 to 140. The result is shown in Figure 3-17(c).

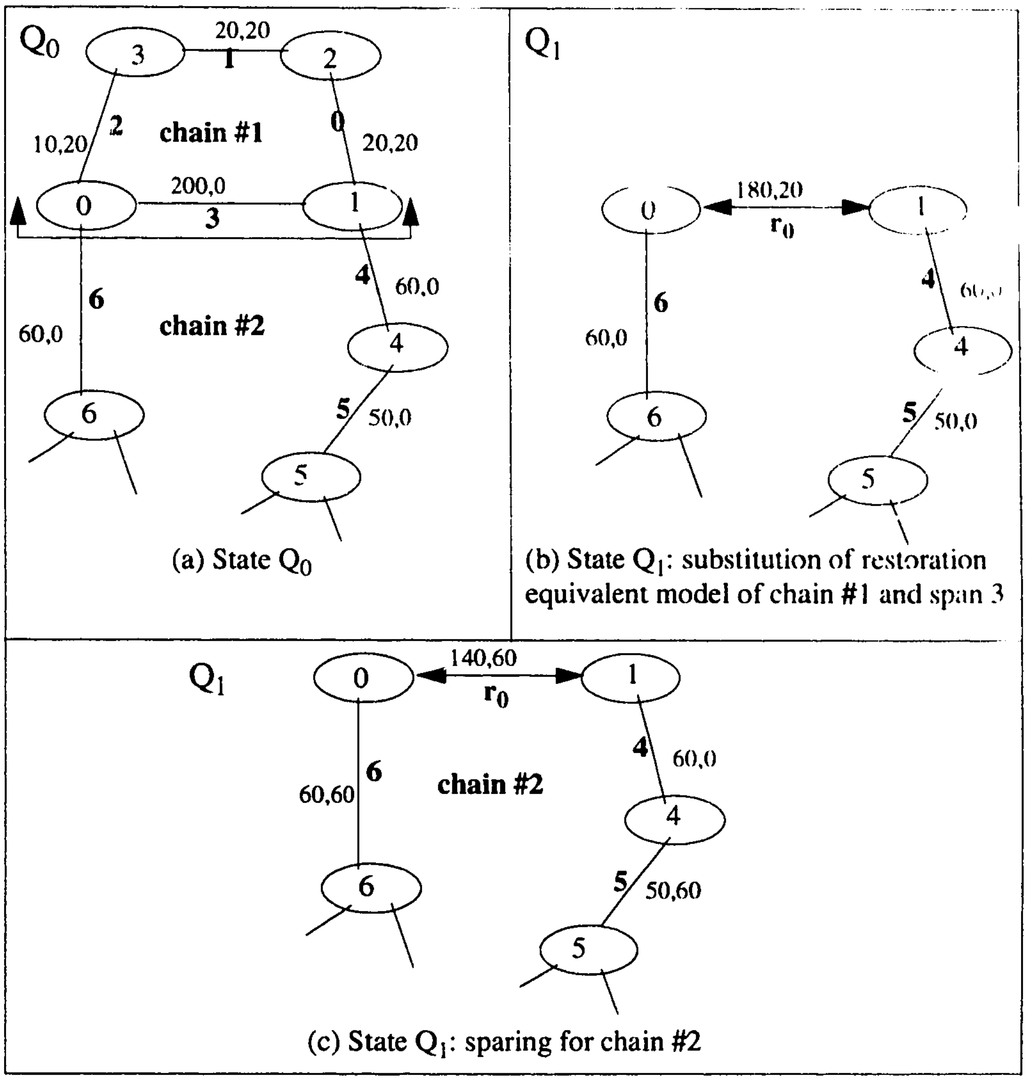


Figure 3-17: An example illustrating NRG1 and NRG2 greedy heuristics

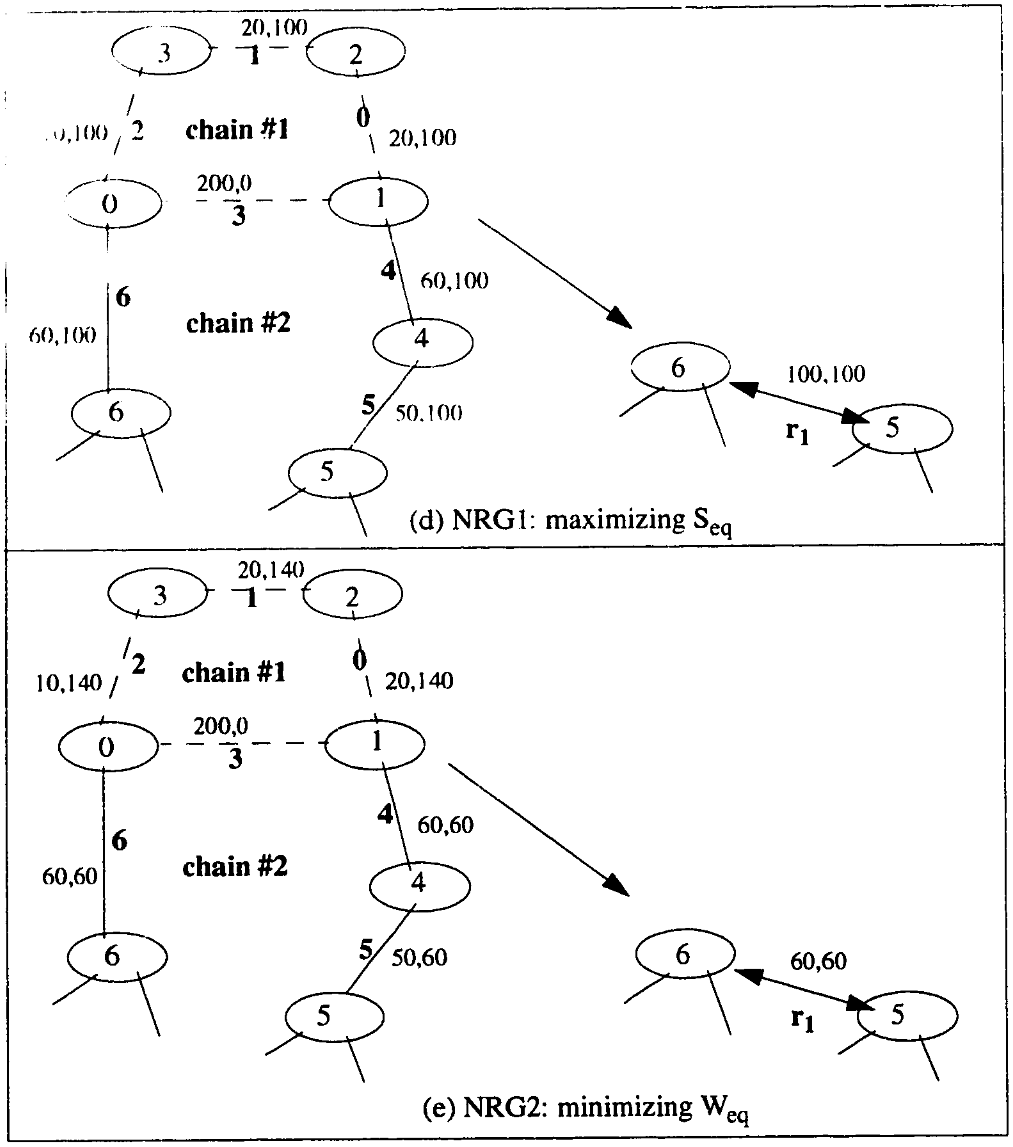


Figure 3-17: An example illustrating NRG1 and NRG2 greedy heuristics (cont'd)

Both greedy heuristics now try to simultaneously maximize the S_{eq} and minimize the W_{eq} of chain 2. If spare capacity is now assigned to all the spans in chain 2 using equation (3-4), then the S_{eq} of the chain becomes 60, by equation (3-21). The S_{eq} of the chain can be increased to 100 by decreasing the W_{eq} of r_0 to 100. This operation involves no additional spare capacity. The SCP design is shown in Figure 3-17(d) which is also the maximum S_{eq} design generated by NRG1. Reduced span r_1 is the restoration equivalent

Q_l. NRG2 tries to minimize the W_{eq} of the chain but at the cost of decreasing S_{eq}. The minimum W_{eq} design is obtained by further reducing span r_0's W_{eq} to 60; this reduces the S_{eq} of the design to 60 as shown in Figure 3-17(e). Both NRG1 and NRG2 use the same total sparing in this case, and in fact many designs with the same total spare capacity are possible between these two designs.

3.2.5 SCP for Identified Chains

The first step in each iteration of the greedy algorithm is identification of the longest chain present in state Q_i. If TYPE2 and TYPE3 reduced spans are present in the chain, they are expanded into the spans they represent. This simplifies the algorithm, as only TYPE0, TYPE1, and TYPE4 spans must be dealt with in the next two steps.

The second step is to place spares for each chain found by the identification step. Equation [illegible] is used for this purpose. The SCP for the chain is actually decided by only [illegible] spans. Let U be the set of spans in the chain. Let S_l be the span having the maximum working equivalent in U, and let S_2 be the span having the maximum working equivalent in $\{U\text{-}S_l\}$. Then the spare capacity of all the spans in the chain, except S_l, is equal to the W_{eq} of S_l, and that of S_l is equal to the W_{eq} of S_2. For example, S_l is span 1 and S_2 is span 2 for the chain in Figure 3-18.

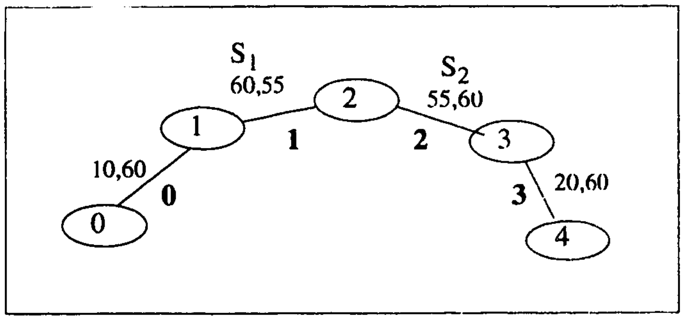


Figure 3-18: An example showing SCP of a chain.

Based on whether S_l and S_2 are real or reduced, four cases are possible:

(a) Both S_l and S_2 are real.

(b) S_l is real and S_2 is reduced.

(c) S_1 is reduced and S_2 is real.

(d) Both S_1 and S_2 are reduced.

The algorithm is different for each case. In case (a) the SCP for the chain's spans is direct. In the other cases the possibility of reducing W^1_{eq} and W^2_{eq} is checked before assigning spare capacity to the chain's spans. The first priority is always minimizing the total spare capacity of the chain and this is done by reducing W^1_{eq} and/or W^2_{eq}, if the spare capacity required to do so is less than the spare capacity needed to accommodate their equivalent working capacities.

(a) S_1 and S_2 are real

Since S_1 and S_2 are real, their W_{eq} cannot be reduced. Spare capacity is assigned directly to the chain's spans using equation (3-4). The SCP for this case is same for both NRG1 and NRG2.

(b) S_1 is real and S_2 is reduced

The algorithm for this case involves three steps. In step 1, the spare equivalents of all the spans in the chain, except S_1, are made equal to W^1_{eq} by using equation (3-4), if they are less than that. Since this may reduce W^2_{eq}, in step 2, S_2 is chosen again from $\{U\text{-}S_1\}$. In step 3, the spare capacity of S_1, S^1_{eq}, is made equal to W^2_{eq} if it is less than that. Since the W_{eq} of the chain cannot be reduced in this case, the SCP is the same for both NRG1 and NRG2.

(c) S_1 is reduced and S_2 is real

The SCP for this case is done in three steps:

(1) assign spare capacity to all the chain's spans except S_1 and S_2, and make their S_{eq} equal to W^2_{eq} if they are less than this.

(2) reduce W^1_{eq} by assigning spare capacity to S_1. This is done by first finding the amount of spare capacity, n_1, needed to reduce W^1_{eq} by 1 and then finding the amount of spare capacity, n_2, saved in the chain if W^1_{eq} is reduced. If the cost of reducing W^1_{eq} is less than or equal to the benefit obtained, $n_1 \leq n_2$, then we do so.

There is a difference between NRG1 and NRG2 if $n_1 = n_2$ and S^1_{eq} is greater than or equal to W^2_{eq}.

In NRG1, W^1_{eq} is reduced further only when S^1_{eq} is greater than or equal to W^1_{eq}. NRG2 only reduces W^1_{eq} when doing so reduces the overall W_{eq} of the reduced span formed from the present chain. This is because the S_{eq} of the chain also decreases when its W_{eq} is dec[illegible]ased. [illegible]duc[illegible]d unconditionally if a TYPE3 reduced span is formed as the overall W_{eq} of this span is same as that of S_1. For TYPE1 and TYPE4 reduced spans W^1_{eq} is reduced only until the W_{eq} of the present chain is greater than or equal to the W_{eq} of the reduced span. If a TYPE1 reduced span is formed then W_{eq} is reduced only when $W_{eff}(chain)$ is greater than $W_{0,eff}$. In case of a TYPE4 span, $W_{eff}(chain)$ of this chain is reduced only when it is more than the $W_{eff}(chain)$ of the other chains and $W_{0,eff}$ (if present).

(3) assign spare capacity to all spans in the chain. Make the S_{eq} of all spans except S_1 equal to W^1_{eq} if they are less.

(d) Both S_1 and S_2 are reduced spans

In this case, both W^1_{eq} and W^2_{eq} are variable. Therefore, assigning spare capacity directly to the spans in the chain may not give the minimum total spare capacity solution for the chain. Let U be the set of all spans in the chain. Let U_1 be the set of S_1 and the spans in the chain having $W_{eq} = W^1_{eq}$. Let U_2 be the set consisting of S_2 and the spans in the chain having $W_{eq} = W^2_{eq}$. A span in U_1 forces the spare capacity of all other spans in the chain. If there is more than 1 span in U_1, then they also force each other's spare capacities. Therefore, we decrease the W_{eq} of spans in U_1 if the number of spares needed to do so is less than the number of spares saved.

After decreasing the W_{eq} of spans in U_1, the W_{eq} of spans in U_2 is decreased. The spans in U_2 force the spare capacity of the spans in U_1 only when U_1 consists of the single span S_1. The W_{eq} of spans in U_2 is decreased only when doing so requires less than the number of spares saved by increasing the S^1_{eq}.

Spare capacity is assigned to all the spans in the chain only if the W_{eq} of neither the spans in U_1 nor the spans in U_2 can be decreased. The SCP for this case is done in three steps; the first two steps first minimize the total spare capacity of the chain, then simultaneously maximize its S_{eq} and minimize its W_{eq}, and are the same for both NRG1 and NRG2. If possible, the third step also maximizes the S_{eq} and minimizes the W_{eq} simultaneously, and then either maximizes the S_{eq} or minimizes the W_{eq} of the chain. This step is different for NRG1 and NRG2.

(1) *reduction of the W_{eq} of spans in U_1*: Spare capacity in the chain is saved on spans in $\{U-U_1\}$ and also on spans in U_1 as spans in U_1 force each other's spare capacities. The W_{eq} of each span in U_1 is reduced until the number of spares (n_1) needed to reduce it by one is greater than the number of spares (n_2) saved in the chain.

If n_1 is equal to n_2 then the S_{eq} of the chain is maximized and W_{eq} is minimized simultaneously. The S_{eq} of the chain is maximum when the S_{eq} of all spans in U is equal to or greater than W^1_{eq}.

After decreasing the W_{eq} of U_1's spans by 1, the W_{eq} of other spans in the chain may become equal to W^1_{eq}. If any of these spans are real then the algorithm moves to step (3) for direct assignment of spare capacity. If none of these spans are real then they are included in U_1 and step (1) is repeated until reduction of the W_{eq} of the spans in U_1 is no longer of benefit.

Consider the subtopology presented in Figure 3-19 which consists of two TYPE1 spans with restoration equivalent models (W_{eq},S_{eq}) of (100,10) and (170,30) (Figure 3-19(b)). Therefore, S_1 is the (170,30) span, and S_2 is the (100,10) span. Here, n_1 is 1 because 1 unit of spare capacity is needed to reduce W^1_{eq}, and n_2 is also 1. Also, S^1_{eq} is less than its W^1_{eq}, therefore its W^1_{eq} is reduced to 150 (Figure 3-19(c)). Further reduction in this step is not done as n_1 is 2 and n_2 is 1.

(2) *reduction of the W_{eq} of spans in U_2:* This step is followed if U_1 consists of only S_1. The spans in U_2 force the spare capacity of spans in U_1 in this case. The W_{eq} of each span in U_2 is reduced until doing so reduces the total spare capacity in the chain.

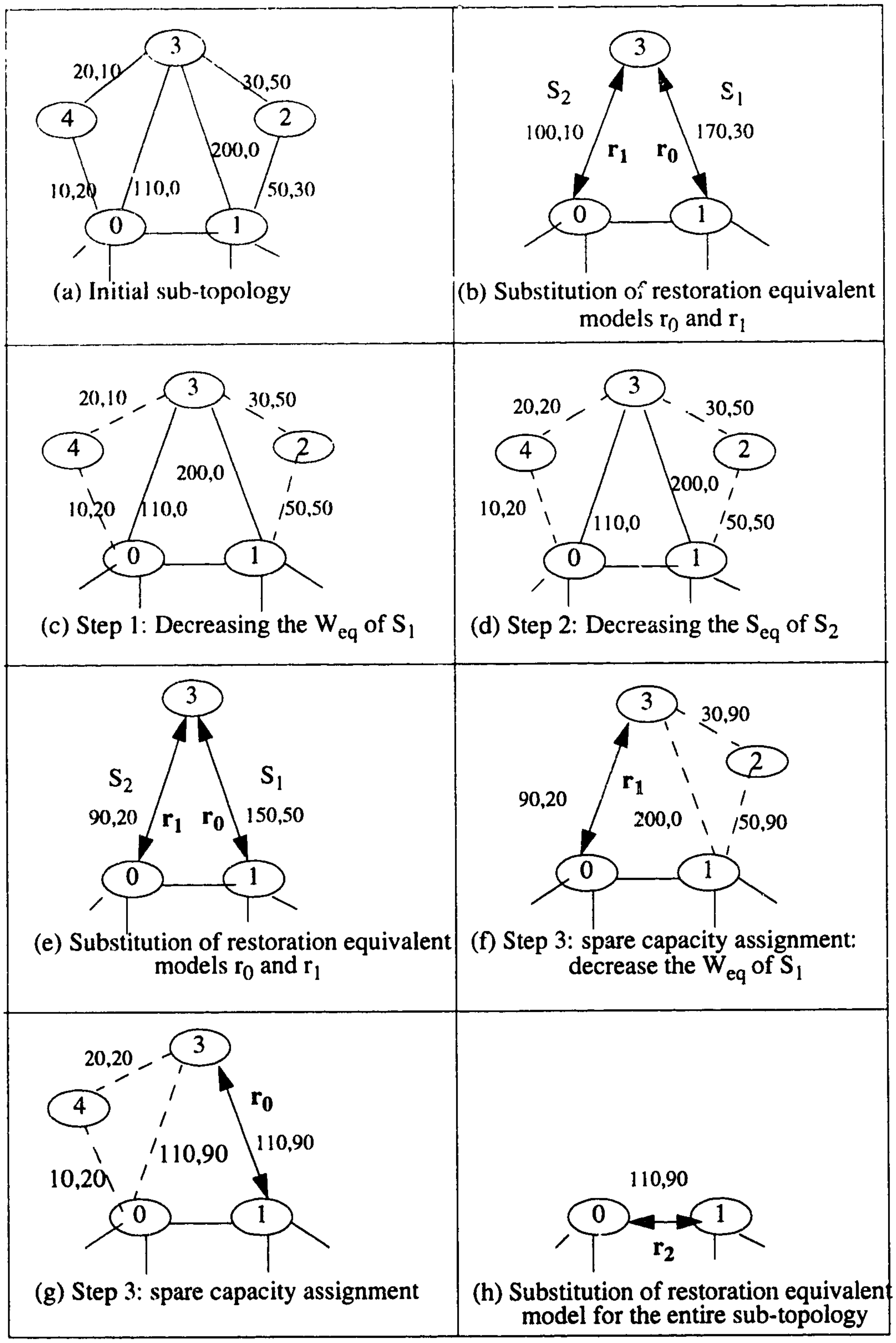
While reducing the W_{eq} of the spans in U_2, the W_{eq} of other spans in the chain may become equal to the W_{eq} of the spans in U_2. If any of these spans is real step (3) is followed, else these spans are included in U_2 and step (2) is repeated.

For example, in Figure 3-19(d), one spare is needed to reduce W^2_{eq} by 1, and one spare is saved in the chain, but S^1_{eq} of 50 is less than W^2_{eq} of 100. Therefore W^2_{eq} is reduced to 90 (Figure 3-19(d)). A further decrease in W^2_{eq} is not beneficial as 2 spares are needed to decrease W^2_{eq} and only one spare is needed to increase S^1_{eq}. Figure 3-19(e) shows the restoration equivalent model of the subtopologies reduced in Figure 3-19(d).

(3) *assignment of spare capacity*: In this step NRG1 and NRG2 differ. After steps 1 and 2, the total spare capacity of the chain has been minimized.

If any of the spans in U_1 is real then the W_{eq} of the chain cannot be reduced, and therefore spare capacity is directly assigned to all spans in the chain through equation (3-4). If none of the spans in U_1 is real and U_1 consists of more than one span, then in NRG1, spare capacity is directly assigned to all spans in the chain because a further decrease in the W_{eq} of U_1 also decreases the S_{eq} of the chain. In NRG2, the W_{eq} of U_1 is decreased only when doing so decreases the overall W_{eq} of the reduced span formed after reducing the chain, the span joining its end nodes (if present), and the previously reduced parallel chains (if present).

For the other cases, the number of spare capacity units, n_1, needed to reduce the W^1_{eq} by one, and the number of spare capacity units, n_2, saved in the chain by doing so are found. In NRG1, if the S^1_{eq} is less than W^1_{eq} and if n_1 is less than or equal to (n_2+1), then W^1_{eq} is reduced. This is done to maximize the S_{eq} and minimize the W_{eq} of the chain and is repeated until S^1_{eq} becomes equal to W^1_{eq}, or W^1_{eq} becomes equal to W^2_{eq}. In NRG2, W^1_{eq} is reduced until it becomes equal to W^2_{eq} if the cost of reducing it is less than or equal to the benefit obtained, $n_1 \leq n_2$. If $n_1 = n_2$ then W^1_{eq} is reduced only when doing so reduces the overall W_{eq} of the reduced span



(a) Initial sub-topology

(b) Substitution of restoration equivalent models r_0 and r_1

(c) Step 1: Decreasing the W_{eq} of S_1

(d) Step 2: Decreasing the S_{eq} of S_2

(e) Substitution of restoration equivalent models r_0 and r_1

(f) Step 3: spare capacity assignment: decrease the W_{eq} of S_1

(g) Step 3: spare capacity assignment

(h) Substitution of restoration equivalent model for the entire sub-topology

Figure 3-19: An example of the case (d) of SCP calculation step

formed after reducing the chain and span joining its end nodes. Once the reduction of W^{1}_{eq} is complete, spare capacity is assigned to all spans in the chain using equation (3-4).

For example, in Figure 3-19(f) both NRG1 and NRG2 reduce W^{1}_{eq} to 110 because doing so maximizes the S_{eq} and minimizes the W_{eq} of the chain. S^{1}_{eq} becomes 90. Further reduction in W^{1}_{eq} is not beneficial as n_1 is 2 and n_2 is 1. Therefore, spare capacity is assigned to S_1 and S_2 using equation (3-4), as shown in Figure 3-19(g). The final restoration equivalent model of chain consisting of S_1 and S_2 and the span joining its end nodes is shown in Figure 3-19(h).

3.2.5.1 Substitution of Restoration Equivalent Model

After the SCP calculation, an identified subtopology is replaced by its restoration equivalent model. The aim of this step is to represent reduced spans in a way that makes increasing their S_{eq} and decreasing their W_{eq} as simple as for real spans, i.e., of *O(1)*. If the complexity of increasing W_{eq} and decreasing S_{eq} of a reduced span is *O(1)* then the complexity of the substitution module is *O(N)*. To make the operations on a reduced span of complexity *O(1)*, the reduced spans can be represented as non-linear spans. The reduced spans can be represented as non-linear spans with *O(N)* complexity, therefore this does not increase the overall complexity of the heuristics. The non-linear representation contains information about the individual chains represented by the span as well as overall information about the reduced span.

In case of a TYPE1 reduced span, there is a span joining the end nodes. In a TYPE4 span more than one chain is present, and the span joining the end nodes of the chains may be present. This is taken care of in the overall representation of a reduced span, which includes:

(1) number of chains in the topology,

(2) end nodes of the chain,

(3) span number of the real span connecting the end nodes (if present),

(4) effective working equivalent of the chain(s), W_{eff}*(chains)*,

(5) spare equivalent, S_{eq},

(6) working equivalent, W_{eq},

(7) type of the reduced span, and

(8) effective working equivalent, $W_{0,eff}$, of the real span joining the end nodes.

Each chain of the subtopology is represented by:

(1) the number of spans in the chain and their span numbers,

(2) spare equivalent, S_{eq},

(3) working equivalent, W_{eq},

(4) type of the chain,

(5) effective W_{eq} of the chain, $W_{eff}(chain)$,

(6) relations for increasing the S_{eq} of the chain (*Seq_rel*). Each relation includes the following fields:

  (a) change in S_{eq},

  (b) change in W_{eq},

  (c) number of spare capacity units added for a unit increase in S_{eq} and the spans on which the spares are added,

  (d) upper and lower limits on the S_{eq} of the chain (*SeqL* and *SeqU*),

  (e) upper and lower limits on the W_{eq} of the chain (*WeqL* and *WeqU*),

  These relations include the variation in the chain's S_{eq} from its present value until the S_{eq} is infinite, INF. INF for a network is taken to be the maximum working capacity present on the spans of the original network.

(7) relations for decreasing the W_{eq} of the chain (*Weq_rel*). These fields are the same as for *Seq_rel*. These relations cover the decrease in the chain's W_{eq} from its present value until it becomes zero or equal to the W_{eq} of any real span in the chain.

The relations in *Seq_rel* represent an increase of S_{eq} through the addition of spare capacity only. The relations in *Weq_rel* represent the decrease of W_{eq} through spare capacity addition only. Spare capacity is not re-arranged in the chain as it makes span representation very complex and difficult to implement.

Consider the example subtopology shown in Figure 3-20, which consists of only one chain and the span joining the end nodes of the chain. A TYPE1 reduced span is formed, and the S_{eq} and the W_{eq} of the reduced span are calculated using equations (3.21) and (3.22) to be 90 and 10 respectively. The S_{eq}, W_{eq}, $W_{eff}(chain)$ and $W_{0,eff}$ of the chain are 10, 20, 20, and 90 respectively. Since the chain consists of all real spans, its W_{eq} cannot be reduced but its S_{eq} can be increased. The relation *Seq_rel* for the chain is shown in Table 3-1. The symbol '*X*' is used to represent a "don't care" situation.

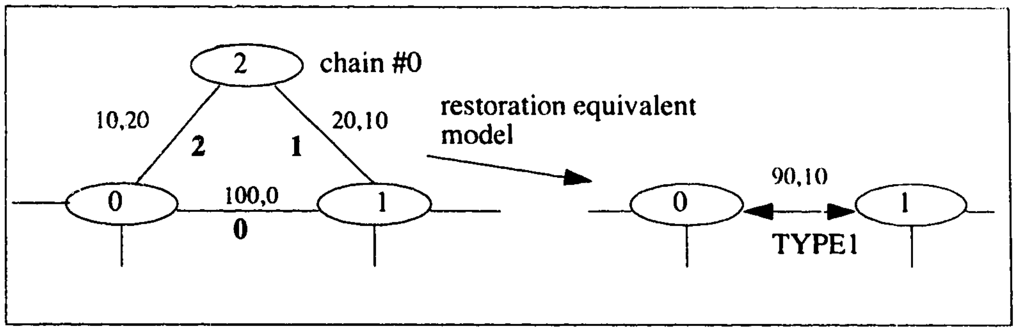


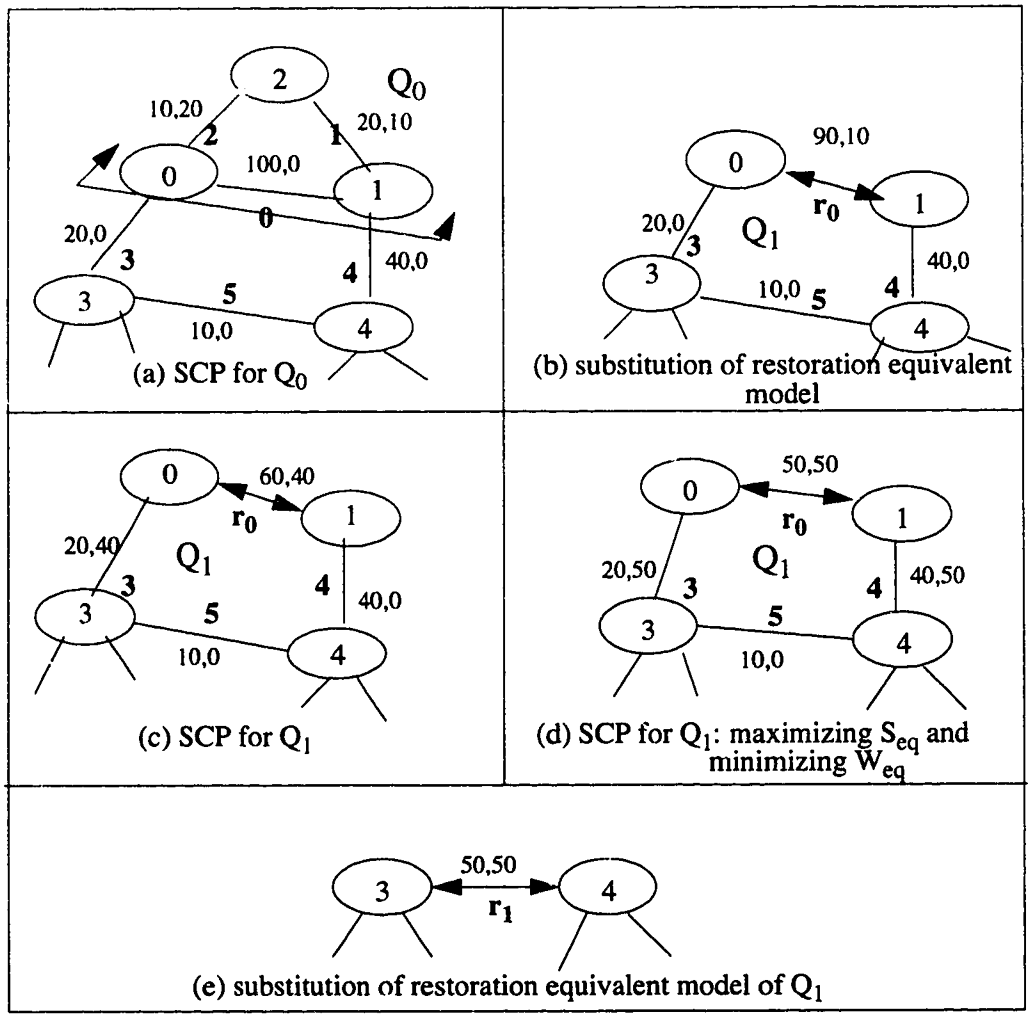
Figure 3-20: An example for non-linear span representation

Table 3-1: An example showing representation of Seq_rel for a chain

S_{eq} change	W_{eq} change	Number of spare capacity units added	WeqL	WeqU	Spans whose spare capacity is increased	SeqL	SeqU
1	0	1	X	X	1	10	20
1	0	2	X	X	1,2	20	INF

Now consider the subtopology in Figure 3-21. State Q_0 in Figure 3-21(a) involves reduction of the subtopology of Figure 3-20. After the reduction of the chain in state Q_0, a chain is formed consisting of spans 3, 4, and r_0 (Figure 3-21(b)). Span r_0 has the maximum W_{eq} in this chain, followed by span 4. Therefore, Q_1's SCP follows case (c) of the SCP calculation with S_1 as r_0 and S_2 as span 4. The first step is to make the spare equivalents of r_0 and span 3 equal to 40. While increasing the S_{eq} of r_0, the feasibility of reducing its W_{eq}

is also determined. Span r_0's W_{eq} can be reduced by increasing the S_{eq} of its chain, as its $W_{eff}(chain)$ is less than its $W_{0,eff}$. From the *Seq_rel* of span r_0, the S_{eq} of the chain can be increased to 20, in steps of one, by adding 1 spare on span 1, and thereafter by adding two units, one each on spans 1 and 2. If the W_{eq} of r_0 is reduced, then two spare capacity units are saved in the chain of Q_1. Therefore, while increasing the S_{eq} of r_0 to 40, its W_{eq} is reduced to 60 (Figure 3-21(c)). In Figure 3-21(d), the W_{eq} of span r_0 is decreased further to increase the S_{eq} and decrease the W_{eq} of the chain in Q_1. The S_{eq} of the chain of Q_0 becomes 50. Figure 3-21(e) shows the restoration equivalent model obtained after the reduction of the chain in state Q_1.



Q0
10,20
20,10
100,0
20,0
40,0
10,0
(a) SCP for Q0
90,10
r0
Q1
20,0
40,0
10,0
(b) substitution of restoration equivalent model
60,40
20,40
(c) SCP for Q1
50,50
20,50
40,50
(d) SCP for Q1: maximizing Seq and minimizing Weq
r1
(e) substitution of restoration equivalent model of Q1

Figure 3-21: An example showing the need for non-linear span representation

Span r_1 is TYPE1 and has W_{eq}, S_{eq}, $W_{eff}(chain)$, and $W_{0,eff}$ of 50, 50, 50, 0, and 0 respectively. The W_{eq} of chain of r_1 is dominated by span r_0, therefore it can be reduced to 40. The W_{eq} of r_0 is decreased by increasing the S_{eq} of its chain. The non-linear representation of span r_0 is used in forming the *Weq_rel* and *Seq_rel* for span r_1. For the *Seq_rel* calculation, all three spans in the chain have 50 spares, therefore the S_{eq} of the chain can be increased only by increasing the spare capacity on all three spans. Table 3-2(a) represents *Weq_rel* and Table 3-2(b) represents *Seq_rel* for span r_1.

Table 3-2(a): An example showing representation of Weq_rel for a chain

S_{eq} change	W_{eq} change	number of spare capacity units added	WeqL	WeqU	spans whose spare capacity is increased	SeqL	SeqU
0	-1	2	40	50	1,2	X	X
1	-1	4	40	50	1,2,3,4	50	60

Table 3-2(b): An example showing representation of Seq_rel for a chain

S_{eq} change	W_{eq} change	number of spare capacity units added	WeqL	WeqU	spans whose spare capacity is increased	SeqL	SeqU
1	0	3	X	X	0,3,4	50	INF

The complexity of creating a non-linear representation of a reduced span is *O(N)*, where *N* is the number of nodes in the original network, but the complexity of accessing this reduced span is *O(1)*. Therefore, in the SCP calculation step, decreasing the W_{eq} and increasing the S_{eq} of a reduced span are *O(1)*. This reduces the worst case complexity of the SCP calculation to *O(N)*.

3.2.6 Network Expansion

SCP of the reduced network may require an S_{eq} on a reduced span that is larger than the value obtained during network reduction. Network expansion corrects such cases by adding spares. After increasing the S_{eq} of all such reduced spans, network expansion generates a complete SCP solution for the original network.

The complexity of increasing the S_{eq} of a reduced span using the non-linear span representation is $O(1)$. In the worst case $O(S)$ reduced spans exist in a reduced network, and therefore the complexity of network expansion is $O(S)$.

3.2.7 Complexity

The network reduction algorithm involves several calls of the greedy algorithm inside an $O(N)$ loop. The greedy algorithm consists of three steps: identification, SCP calculation, and substitution of the restoration equivalent model. The complexity of each of these steps with the non-linear span representation is $O(N)$. Since none of the modules is inside any other module, the worst case complexity of the network reduction algorithm is $O(N^2)$.

The complexity of network expansion is $O(S)$, therefore the overall complexity of network reduction and expansion is $O(N^2)$.

3.2.8 Performance and Test Results

The performance parameters for network reduction algorithms are capacity efficiency, execution time, and network reduction gain. *Network reduction gain* is the percentage of the original nodes and spans removed from the network. Seven networks having sizes ranging from 7 to 20 nodes were chosen to compare NRG1, NRG2, and IP. Table 3-3 presents some parameters of these networks. Table 3-4 compares NRG1 with NRG2, while Table 3-5 compares both of these heuristics with IP.

Table 3-3: Test networks for network reduction

Network	Number of nodes	Number of spans	Average node degree	Number of working links
Net5	10	15	3	920
Net6	15	20	2.67	850
Net7	15	19	2.53	650
Net8	17	23	2.7	1250
Net9	17	23	2.7	950
Net10	20	32	3.2	1930
Net11	7	11	3.14	500

Table 3-4 Comparison between NRG1 and NRG2

Network	NRG1 number of spare links	NRG2 number of spare links	NRG1 execution time (Sec)	NRG2 execution time (Sec)
Net5	1140	1020	0.69	0.70
Net6	1170	1110	0.4	0.4
Net7	1110	1070	0.31	0.3
Net8	1790	1790	1.23	1.19
Net9	1300	1330	0.59	0.56
Net10	2040	2040	1.58	1.56
Net11	490	490	0.26	0.26

Table 3-5: Comparison of NRG1 and NRG2 with IP

Network	IP number of spare links	IP execution time (Sec)	NRG1 % excess redundancy w.r.t. IP	NRG2 % excess redundancy w.r.t. IP
Net5	950	13	20	7.3
Net6	1110	11	5.4	0
Net7	1070	9	3.7	0
Net8	1760	12	1.7	1.7
Net9	1300	12	0	2.3
Net10	1950	39	4.6	4.6
Net11	440	11	11.3	11.3

The results in Table 3-4 and 3-5 show that both heuristics are very fast and that network capacity for NRG2, in most cases, is within 10% of the optimum. NRG1 is up to 20% from the optimum. Notably, when add0_sub1 design tightening was used to tighten the SCP designs of Net5 to Net11, no capacity was removed.

Table 3-6 presents the network reduction gain for NRG1 and NRG2 operating on Net1, Net2, Net3, and Net4. The results show that for some networks reduction gain can be as high as 50%.

Table 3-6: Network reduction gain for Net1, Net2, Net3, and Net4

Network	Number of spans	Number of nodes	Number of spans in the reduced network	Number of nodes in the reduced network	% reduction in number of spans	% reduction in number of nodes
Net1	24	11	15	7	37.5	36.36
Net2	31	20	17	10	45.16	50
Net3	59	30	57	28	3.4	6.67
Net4	81	53	47	28	41.97	47.16

3.2.9 Conclusions

Two greedy heuristics, NRG1 and NRG2, were developed and implemented based on chain-wise reduction principles. Network reduction gain for these heuristics can be up to 50%. The time complexity of NRG1 and NRG2 using a non-linear span representation is $O(N^2)$. Execution time for NRG1 and NRG2 was less than 2 seconds for all seven test networks, which is 10 times faster than that of IP.

The SCP results of NRG2 were found to be within 10% of the optimum for six of the seven test networks, while those from NRG1 were more than 10% greater than the optimum in many cases. Execution time for both these approaches was nearly the same. Therefore, NRG2 is a better heuristic than NRG1.

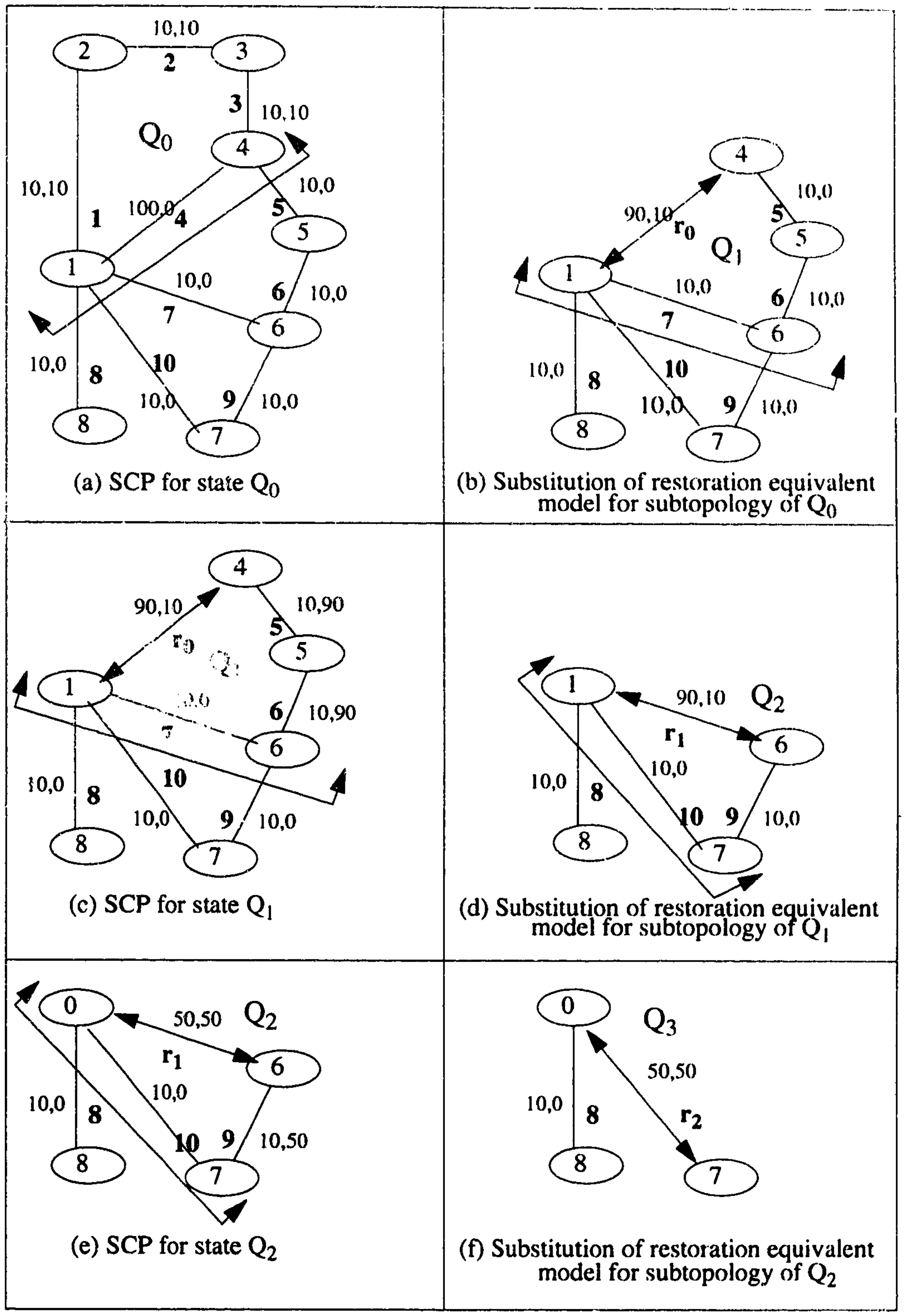
The add0_sub1 design tightener was not able to find excess capacity in any of the designs from either NRG1 or NRG2. This shows that both heuristics are efficient spare placers.

3.3 A Backtracking Method

The greedy chain-wise reduction does not guarantee an optimum SCP solution. This is because the SCP calculated at state Q_i can affect the SCP at some later state Q_k. The SCP at state Q_i is locally optimum but at state Q_k it is not. One way to overcome this problem is to backtrack to state Q_i and start all over again from there. Thus, to find an optimum solution we can consider a backtracking method. Here a subtopology is reduced until the total spare capacity of the subtopology is optimum. If this is not true at any state, the method backtracks to the earlier state which made the solution non-optimal and starts again from that state.

Consider the example shown in Figure 3-22. Q_0, Q_1, Q_2, and Q_3 are four states. The SCP at state Q_0 in Figure 3-22(a) is done using equation (3-4), which is followed by substitution of the restoration equivalent model, r_0 (Figure 3-22(b)). Equations (3.21) to (3.24) are used in the calculation of the restoration equivalent model at each state. Figure 3-22(c) shows the SCP for state Q_1 and Figure 3-22(d) shows the substitution of Q_1's restoration equivalent model, r_1. At state Q_2 (Figure 3-22(d)), the W_{eq} of span r_1 is maximum which forces the other spans in the chain to have high spare capacities. The W_{eq} of span r_1 is reduced to 50 as this also maximizes the S_{eq} of the chain consisting of r_1 and span 9. Figure 3-22(e) shows the SCP for state Q_2. Figure 3-22(f) presents Q_2's restoration equivalent model, r_2. In Figure 3-22(f), r_2 forces the spare capacity on span 8 to 50. The high W_{eq} of r_2 is due to r_1 which has a high W_{eq} inherited from r_0. In order to protect one working link on r_0, one spare link must be placed on spans 5, 6, 8, and either of 7 or 9. but only three units of spare capacity were needed at state Q_1 (Figure 3-22(b)) to reduce the W_{eq} of r_0 by 1. Therefore, the SCP at state Q_3 is not optimum due to the decision made at state Q_0. The algorithm undoes the previous SCP from state Q_0 onwards, backtracks to state Q_0, reduces the W_{eq} of r_0 to 10 (Figure 3-22(g)), and starts again from there.

In practice, it is very difficult to keep track of when to backtrack, and to determine the extent to which the W_{eq} of an earlier state must be reduced. Also, it is difficult to undo the SCP. Furthermore, non-linear span representations for backtracking must include all possible relations for increasing S_{eq} and decreasing W_{eq}. Many of these relations involve rearrangement (deletion and addition) of spare capacity in a chain. In some cases, these relations are interdependent. Therefore, it is not possible to guarantee an exact non-linear span representation for states Q_i with $O(N)$ complexity. Figure 3-23 presents an example showing the limitations of non-linear span representations. Table 3-7 shows all possible relations for decreasing the W_{eq} of r_1 of Figure 3-23(d). The first relation in Table 3-7 involves a rearrangement where all three relations are interdependent. If r_1's W_{eq} is decreased by one using the first relation then the S_{eq} of r_1's chain becomes 49. The third relation is no longer exact as its *SeqL* and *SeqU* have changed to 49 and 69 respectively. Hence the third relation must be re-derived. This increases the complexity of the non-linear span representation.



(a) SCP for state Q_0

(b) Substitution of restoration equivalent model for subtopology of Q_0

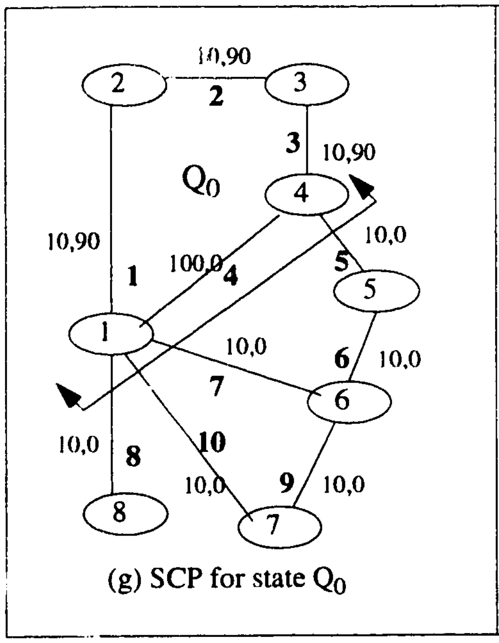
(c) SCP for state Q_1

(d) Substitution of restoration equivalent model for subtopology of Q_1

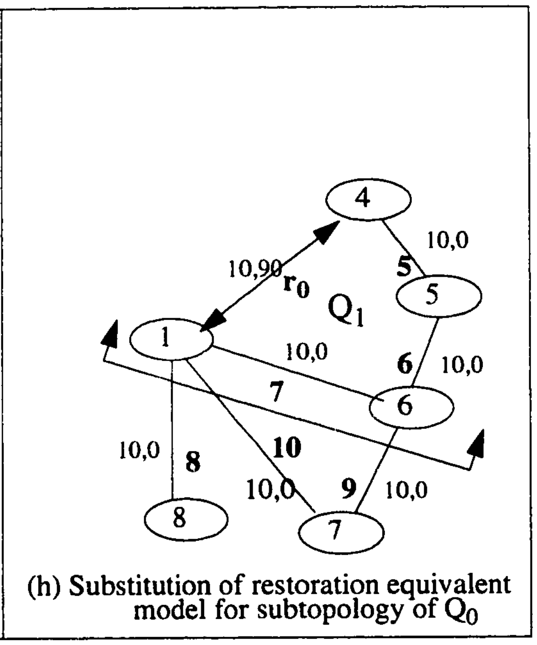
(e) SCP for state Q_2

(f) Substitution of restoration equivalent model for subtopology of Q_2

Figure 3-22: An example of backtracking for network reduction



(g) SCP for state Q_0



(h) Substitution of restoration equivalent model for subtopology of Q_0

Figure 3-22: An example of backtracking for network reduction (cont'd)

Table 3-7: An example showing a limitation of non-linear span representation for back-tracking

S_{eq} change	W_{eq} change	Number of spare capacity units added	WeqL	WeqU	Spans whose spare capacity is increased	Spans whose spare capacity is decreased	SeqL	SeqU
-1	-1	1	30	50	1,2,3	4,6	30	50
0	-1	3	30	50	1,2,3	-	X	X
1	-1	5	30	50	1,2,3,4,6	-	50	70

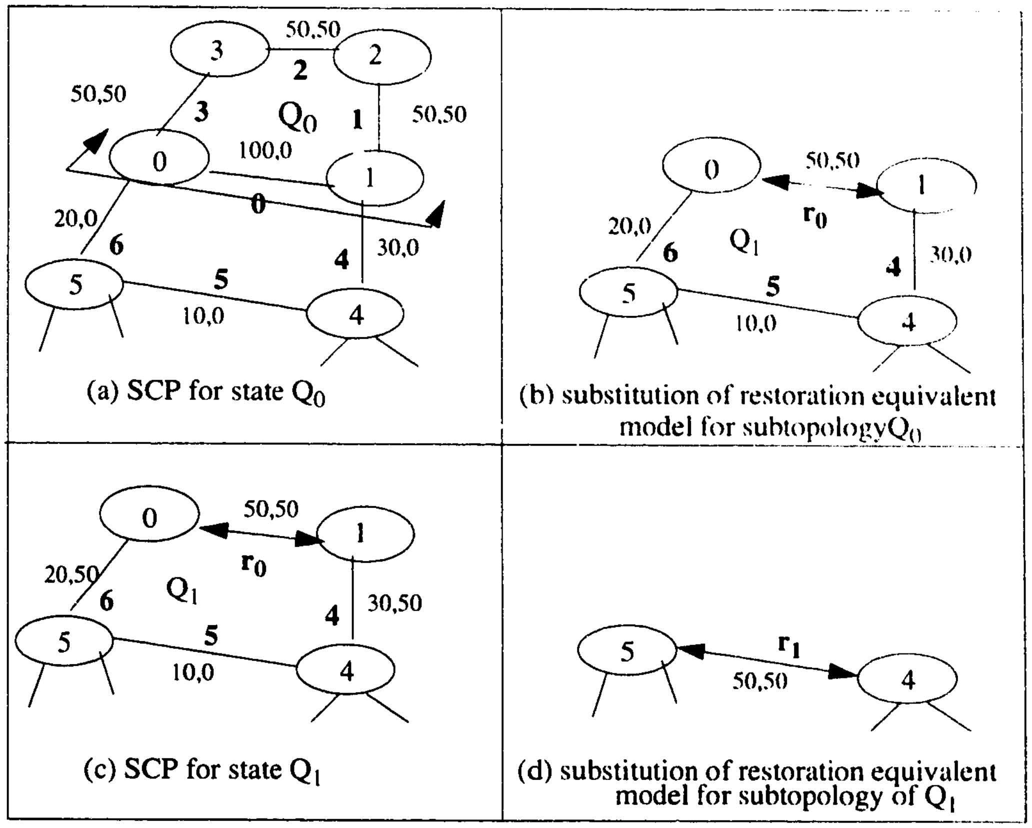


Figure 3-23: An example for showing limitations of non-linear span representation of complexity O(N) for backtracking

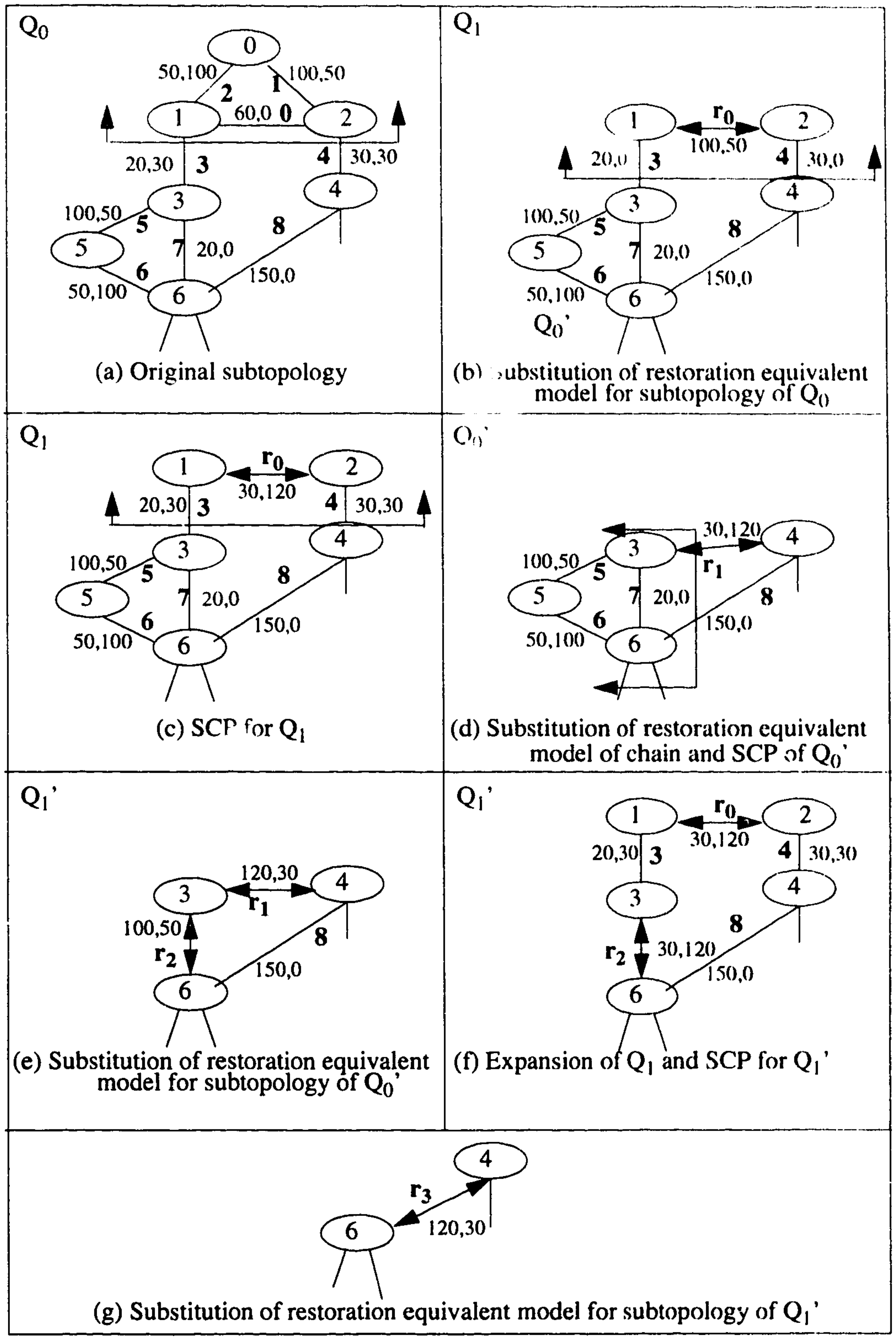
Restricted backtracking for network reduction performs only two successive chain-wise reductions. The non-linear span representation used up to Q_l is exact, and is of *O(N)* complexity as it involves no rearrangement. The reduced span obtained after reducing state Q_l of each subtopology is marked. TYPE1 and TYPE4 reduced spans are debarred from further reduction, while TYPE2 and TYPE3 spans are first expanded and then used if they are part of a chain.

Network reduction using restricted backtracking is nearly the same as that of the greedy method, except that backtracking sometimes involves rearrangement of spare capacity. Like the greedy method, the primary aim of restricted backtracking is to minimize total spare capacity and the secondary objective is either to maximize S_{eq} or minimize W_{eq}. Depending on the secondary objective, two heuristics are possible: NRRBT1 which maximizes S_{eq}, and NRRBT2 which minimizes W_{eq}.

The identification step for restricted backtracking identifies the longest chain present in states Q_0 and Q_1. If TYPE2 and TYPE3 spans are present in a chain, they are expanded to form a bigger chain. If a marked TYPE1 or TYPE4 span is identified in a chain then that chain is not treated further. The identification step also undoes the SCP from state Q_1 for all marked TYPE3 reduced spans. This is done to avoid the possibility of rearrangement (discussed later) during the SCP calculation.

Figure 3-24 presents an example showing rearrangement of spare capacity in a chain during its SCP, and Figure 3-25 shows how such a rearrangement possibility can be avoided. The purpose of rearrangement is to minimize total spare capacity, maximize S_{eq}, and minimize W_{eq} of the subtopology.

The SCP of the chain for each state uses equation (3-4). In Figure 3-24, spans r_0, r_1, r_2, and r_3 are reduced spans found using equations (3.21) to (3.24). These represent the restoration equivalent models for chains in Q_0, Q_1, Q_0' and Q_1' respectively. Figure 3-24(f) shows the SCP and Figure 3-25(g) shows the restoration equivalent model. In Figure 3-24(f), the W_{eq} and S_{eq} of the chain in Q_1' are 30. The overall W_{eq} of the chain and span 8 is 120. This can be decreased by increasing the S_{eq} of the chain. This is done by removing two spare links, one each from span 0 and 7 (r_0 and r_2), and putting them on spans 3 and 4. This step is shown in Figure 3-24(h). The final W_{eq} and S_{eq} values after rearrangement are both 75 (Figure 3-24(i)). Total spare capacity values for the greedy algorithm and restricted backtracking are the same for this example.



(a) Original subtopology

(b) Substitution of restoration equivalent model for subtopology of Q_0

(c) SCP for Q_1

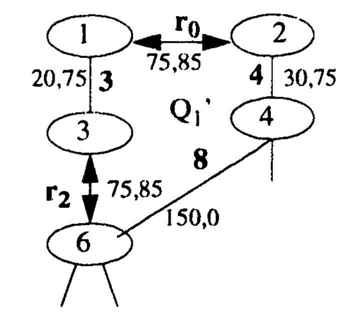
(d) Substitution of restoration equivalent model of chain and SCP of Q_0'

(e) Substitution of restoration equivalent model for subtopology of Q_0'

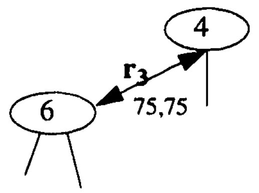
(f) Expansion of Q_1 and SCP for Q_1'

(g) Substitution of restoration equivalent model for subtopology of Q_1'

Figure 3-24: An example showing rearrangement of spare capacity



(h) rearrangement of spare capacity in chain of Q_1'

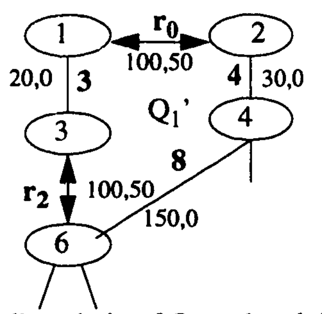


(i) substitution of restoration equivalent model for subtopology of Q_1'

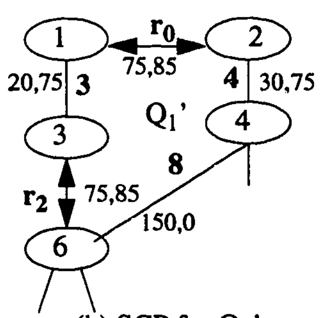
Figure 3-24: An example showing rearrangement of spare capacity (cont'd)

The possibility of rearrangement at state Q_l arises only when at least one TYPE3 span is a part of Q_l's chain, and a TYPE1 or TYPE4 reduced span is formed after reduction of Q_l. The rearrangement can be avoided during the SCP of the chain by undoing the SCP of the TYPE3 reduced spans' Q_l state while expanding TYPE3 chains. After this the SCP calculation for NRRBT1 and NRRBT2 is the same as that of their greedy counterparts.

For example, Figure 3-25(a) shows the result of undoing the SCP of Figure 3-25(c) while expanding span r_l of Figure 3-25(e). The SCP for Figure 3-25(a) uses case (d) of the SCP calculation, and Figure 3-25(b) shows the result.



(a) Expanding chain of Q_1 and undoing SCP done at state Q_1



(b) SCP for Q_1'

Figure 3-25: An easy method for avoiding rearrangement at state Q_1

Substitution of the restoration equivalent model is also nearly the same as for the greedy method. Since reduction is done just up to state Q_l, only a non-linear representation of state Q_0 is required. Apart from *Seq_rel* and *Weq_rel*, the non-linear span representation includes relations for rearrangement, *Rearrange_rel*. *Rearrange_rel* consists of operations on reduced spans in state Q_l. The operations are recorded in such a way that they can be used for undoing some or all of the previous SCP operations on a reduced span.

For example, in Figure 3-21(c) the S_{eq} of r_0 is increased from 10 to 40 while the W_{eq} is reduced from 90 to 60. *Rearrange_rel* for r_0's chain corresponding to this step is shown in Table 3-8. The SCP of Figure 3-21(c) can be undone by executing the relation which lies in the chain's S_{eq} range. For example the S_{eq} of the chain can be decreased to 39 by removing spare links from spans 1 and 2. *Rearrange_rel* is used during identification and network expansion.

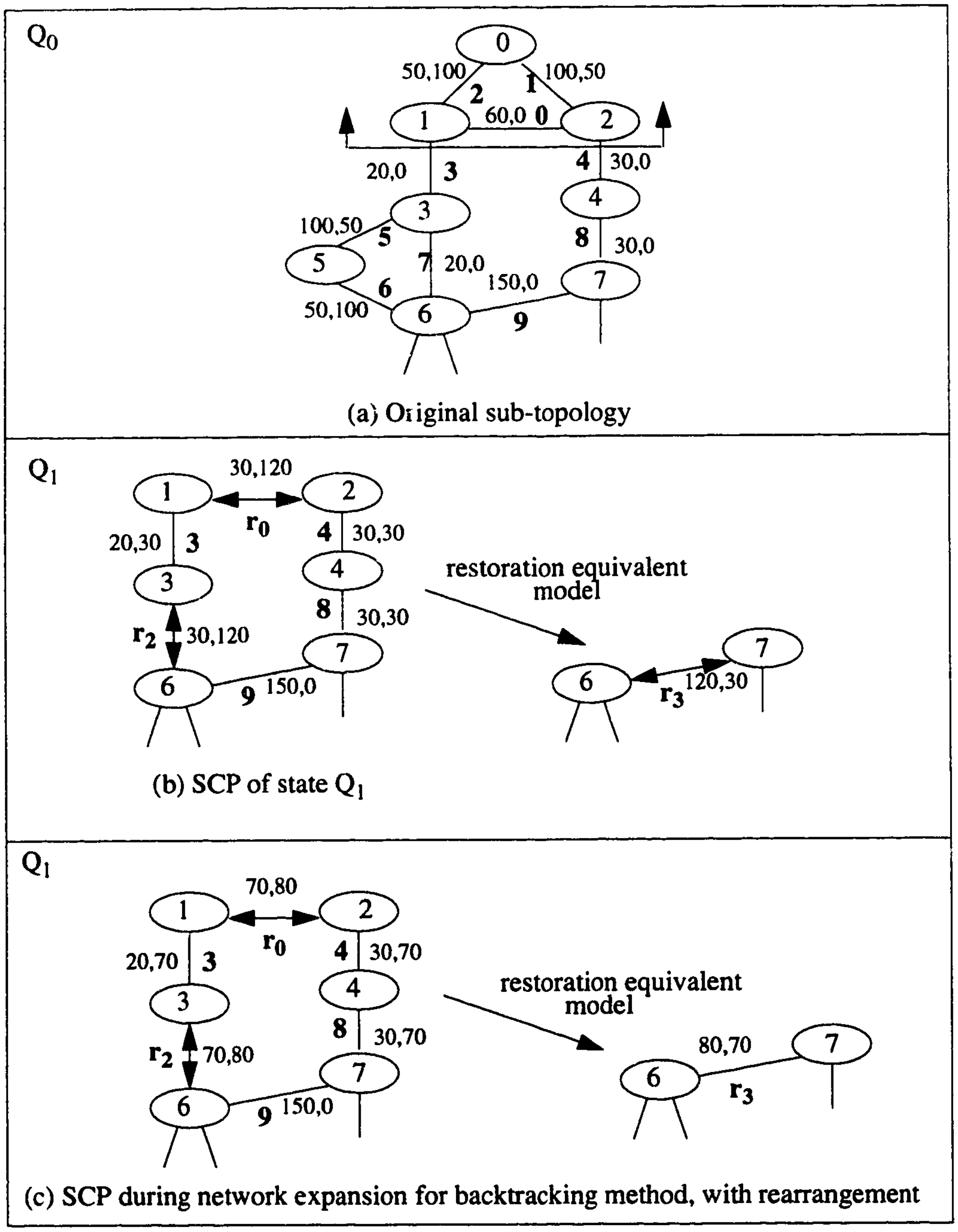
Table 3-8: An example showing representation of Rearrange_rel for a chain

S_{eq} change	W_{eq} change	Number of spare capacity units added	WeqL	WeqU	Spans whose spare capacity must be decreased	SeqL	SeqU
1	0	1	X	X	1	10	20
1	0	2	X	X	1,2	20	40

3.3.1 Network Expansion

The operation of increasing S_{eq} during expansion is the same as for the greedy method except that in some cases rearrangement is used to increase the S_{eq} and decrease the W_{eq} of a reduced span. For all unmarked reduced spans (those obtained after the reduction of Q_0), S_{eq} is increased. Marked spans are rearranged if this is feasible.

Consider the example presented in Figure 3-26. The subtopology of Figure 3-26(a) is the same as that of Figure 3-24 except that it has one less span. Figure 3-26(b) shows state Q_l which is obtained in the same way as Q_l' in Figure 3-24(f). After the reduction of state Q_l, a TYPE1 reduced span, r_3, is formed with a W_{eq} and S_{eq} of 120 and 30 respectively. Let the required value of S_{eq} for r_3 be 70.



(a) Original sub-topology

(b) SCP of state Q_l

(c) SCP during network expansion for backtracking method, with rearrangement

Figure 3-26: An example showing rearrangement during network expansion

The S_{eq} of r_3 can be increased if one unit of spare capacity is taken from spans r_0 and r_2 and one unit is placed on spans 3, 4, and 8 (Figure 3-26(c)). This operation increases the W_{eq} of spans r_0 and r_2 by 1 but the overall W_{eq} of r_3 decreases as $W_{0,eff}$ is greater than $W_{eff}(chain)$. W_{eq} and S_{eq} after SCP are 80 and 70. Therefore, the above rearrangement decreases the W_{eq} of r_3 while increasing its S_{eq}.

Rearrangement is possible only for TYPE1 and TYPE4 spans. It is done if the number of spares required is less than or equal to number of spares required to increase the S_{eq} of a reduced span. If rearrangement is not feasible, the S_{eq} of a reduced span is increased as in case of the greedy method.

3.4 Software Implementation

Both the greedy and restricted backtracking heuristics are coded using the "C" programming language. The code is written in a modular fashion and consists of three main modules for network reduction, namely identification of subtopologies, SCP calculation and substitution of the restoration equivalent model, and network expansion.

The executable version of the code for these heuristics and their interface with the max-latching heuristics is started by typing **nre** at the UNIX prompt. The command line arguments for this tool are:

nre { input network file with fixed w_i }

{ outfile network file to hold final s_i assignments }

{ network reduction heuristics: 0 for greedy and 1 for backtracking }

{ type of network reduction heuristics: 0 for maximizing S_{eq} and 1 for minimizing W_{eq} }

{ max-latching heuristics used }

{ hop limit for the max-latching heuristics }

{ output log file }

The output network file gives the final SCP design, and the output log file contains the information about the final network, such as the number of spare, working, and total links, total real distance, redundancy, and hop limit.

3.5 Summary

The objective of this chapter is to document capacity-efficient methods of $O(N^2)$ complexity for network reduction and expansion. The three methods considered were fixed topology identification and substitution, greedy, and restricted backtracking. All these methods involve three main steps: (1) identification of subtopologies, (2) SCP for an identified subtopology, and (3) reduction of the subtopology by substituting its restoration equivalent model.

Fixed topology identification and substitution identifies and reduces a predefined set of subtopologies in a network. The predefined set consists of subtopologies that are common in real transport networks and whose SCP can be done efficiently through solving simple cutset equations. The results were optimal only for subtopologies whose SCP is reducible to solving equations associated with a single cutset.

Fixed topology identification and substitution is not practical as it is very difficult to identify all chain-reducible subtopologies in the network as they occur in a large range of shapes and sizes. Also, efficient SCP for all such subtopologies cannot be done. Therefore, the fixed topology identification and substitution method is not generally useful for network reduction.

On the other hand, the greedy and backtracking reduction methods identify and reduce a chain-reducible subtopology in multiple steps. In each step, a chain and the span joining its end nodes are identified and reduced. While placing spares for the chain, the primary objective is to minimize the total spare capacity in the chain, and the secondary objectives are to maximize the S_{eq} and minimize the W_{eq} of the chain. In some cases it is not possible to simultaneously maximize S_{eq} and minimize W_{eq}. For these cases either S_{eq} is maximized or W_{eq} is minimized. In the greedy method the heuristic NRG1 maximizes the S_{eq} of the chain, while NRG2 minimizes the W_{eq} of the chain and span joining the end nodes of the chain. The restricted backtracking counterparts for NRG1 and NRG2 are NRRBT1 and NRRBT2 respectively. SCP and network expansion for restricted backtracking are the same as for the greedy method, except in cases that require rearrangement.

Results for NRG1 and NRG2 were observed for seven test networks ranging in size from 7 to 20 nodes. The results for NRG2 were within 10% of the optimum in nearly all cases while those from NRG1 were sometimes more than 10% greater than the optimum. This shows that NRG2 is more capacity efficient than NRG1.

The purpose of the backtracking method for chain-wise successive reduction is to achieve an optimal SCP. It was observed that backtracking could be implemented with $O(N^2)$ complexity for cases requiring only up to two successive chain-wise reductions. This is adequate for most real transport networks.

Network reduction gain, that is the reduction in number of nodes and spans, was observed to be over 35% for Net1, Net2, and Net4, for both the greedy and restricted backtracking methods. Therefore, these are two potential approaches for network reduction.

4 Max-latching Heuristics And Network Reduction

This chapter examines the combination of network reduction with max-latching heuristics to improve the execution time of spare capacity placement. All combinations of greedy and restricted backtracking with HeuristicA2 and HeuristicC1 have been considered. HeuristicA2 has been chosen as it has the lowest complexity of all the max-latching heuristics, while HeuristicC1 is the second most capacity-efficient heuristic. HeuristicC2, though the most capacity-efficient, is not considered as it is very complex. Net1, Net2, and Net4 have been chosen as test networks because they display reduction gains greater than 35%. All the combinations are compared with each other in capacity efficiency and execution time. The capacity efficiency for these combinations, with and without design tightening, is also compared with that for IP at both practical and large hop limits. We will number the combinations examined as follows:

(1) NRRBT1 and HeuristicA2

(2) NRRBT2 and HeuristicA2

(3) NRG1 and HeuristicA2

(4) NRG2 and HeuristicA2

(5) NRRBT1 and HeuristicC1

(6) NRRBT2 and HeuristicC1

(7) NRG1 and HeuristicC1

(8) NRG2 and HeuristicC1.

4.1 Experimental Results Without Design Tightening

Figure 4-1 shows the graph of redundancy vs. hop limit for combinations (1), (2), (4), (5), (6), and (8). For Net1, identical redundancies are obtained with either HeuristicA2 or HeuristicC1 and NRRBT1 and NRG1.

For Net2, all combinations involving HeuristicA2 have exactly the same redundancy values at all hop limits. A similar trend is also exhibited by HeuristicC1 combinations. For comparison, Figure 4-2 presents the graphs of redundancy vs. hop limit for HeuristicA2 and HeuristicC1 alone, and combinations (1) and (5).

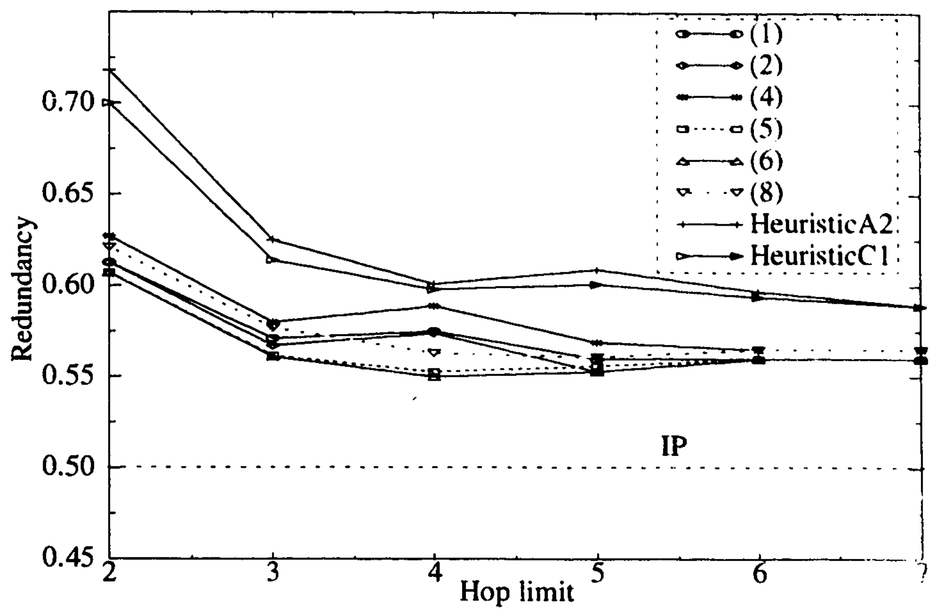


Figure 4-1: Redundancy vs. hop limit for Net1

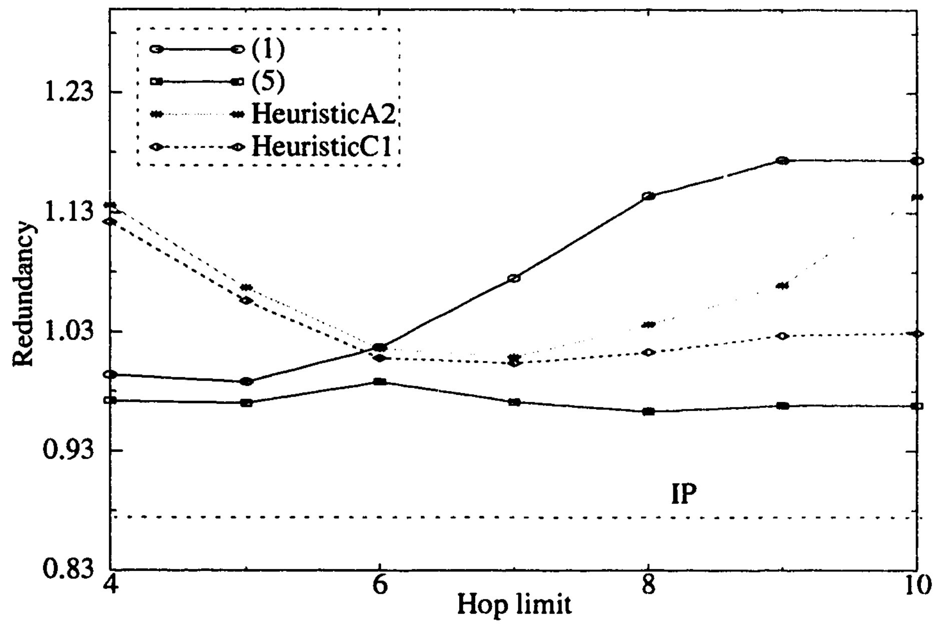


Figure 4-2: Redundancy vs. hop limit for Net2

For Net4, combinations (1) and (3), (2) and (4), (5) and (7), and (6) and (8) have exactly identical redundancy values at all hop limits. Figure 4-3 shows the graphs of redundancy vs. hop limit for (1), (2), (5), (6), HeuristicA2, and HeuristicC1.

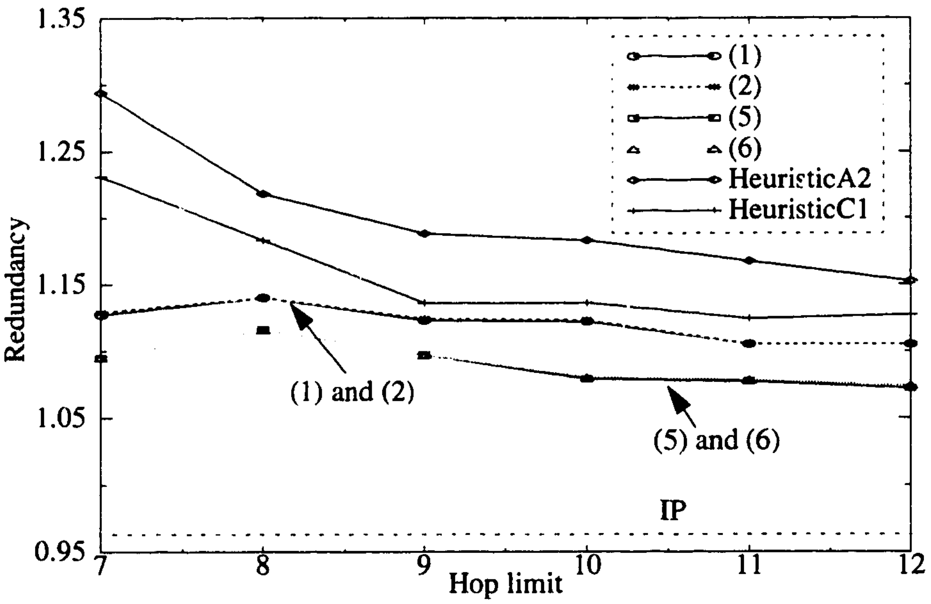


Figure 4-3: Redundancy vs. hop limit for Net4

Figures 4-1 to 4-3 show that all eight combined network reduction - max-latching heuristics exhibit similar characteristics. Also, the redundancy results for all combinations are more capacity efficient than HeuristicA2 and HeuristicC1 alone. This is because of efficient network reduction. Also, the redundancy for all combined heuristics saturated at lower hop limits than their pure max-latching counterparts.

Figures 4-1 to 4-3 show that combinations (2) and (6) which use NRRBT2 are the most capacity efficient, followed by NRRBT1 (combinations (1) and (5)). The combinations involving NRG1 and NRG2 are also as capacity efficient as their backtracking counterparts except for Net1. Furthermore, the combinations of HeuristicC1 are more capacity efficient than those of HeuristicA2.

Table 4-1 compares the redundancy for combinations involving NRRBT2 with those of IP for practical hop limits. Table 4-2 compares the redundancy for combinations using NRRBT2 with those from IP at large hop limits. The redundancy of NRRTBT2-HeuristicA2 is within 17% of IP using large hop limits, while NRRBT2-HeuristicC1, is within 13% of the optimum. Also, for practical hop limits the redundancy values of both these combinations are within 11% of the optimum.

Table 4-1: Comparison of NRRBT2-HeuristicA2 and NRRBT2-HeuristicC1 with IP at practical hop limits

Network	Hop limit	IP redun-dancy	NRRBT2-HeuristicA2		NRRBT2-HeuristicC1	
			redun-dancy	% excess redundancy w.r.t IP	redun-dancy	% excess redundancy w.r.t IP
Net1	6	0.4992	0.553	10.77	0.553	10.77
Net2	6	0.919	1.017	10.66	0.988	7.51
Net4	11	0.993	1.105	11.28	1.078	8.56

Table 4-2: Comparison of NRRBT2-HeuristicA2 and NRRBT2-Heuristic C1 with IP using large hop limits

Network	IP redundancy	NRRBT2-HeuristicA2		NRRBT2-HeuristicC1	
		redundancy	% excess redundancy w.r.t IP	redundancy	% excess redundancy w.r.t IP
Net1	0.4992	0.553	10.77	0.553	10.77
Net2	0.875	1.017	16.23	0.988	12.91
Net4	0.963	1.105	14.75	1.078	11.94

Table 4-3 presents the execution times for combinations (1), (3), (5), (7) and IP for the hop limits used in Table 4-1. The execution times for combinations (2), (4), (6), and (8) are the same as those of (1), (3), (5), and (7), respectively. The execution times of all four combinations and the individual heuristics are of the same order for Net1 and Net2. Also, all combinations ran more quickly than their pure max-latching counterparts for the large network, Net4. For HeuristicC1, the execution time for Net1 decreased by a factor of 6. NRRBT2-HeuristicC1 is nearly 50 times faster than IP.

Table 4-3: Execution times for some combinations, IP, HeuristicA2, and HeuristicC1

Network	IP		(1) (sec)	(3) (sec)	(5) (sec)	(7) (sec)	A2 (sec)	C1 (sec)
	constraint set generation (sec)	IP run (sec)						
Net1	1.6	10.05	0.42	0.49	0.7	0.61	0.5	2.10
Net2	0.8	1	0.20	1.12	0.69	1.22	0.21	1.18
Net4	6.83	226	0.92	1.11	4.8	5.1	2.92	33.8

The combinations based on NRG1 and NRG2 took a little longer than those based on NRRBT1 and NRRBT2, as they use non-linear span representations for all successive chain-wise reductions. On the other hand, combinations using NRRBT1 and NRRBT2 are the fastest as these involve non-linear span representations only for the first reduction step.

4.2 Experimental Results With Design Tightening

Table 4-4 presents the redundancy results at practical hop limits for all combinations, HeuristicA2, and HeuristicC1 after design tightening. The redundancy for all eight combinations were within 2% of each other at all hop limits for Net1 and Net3. For Net2, this trend was observed only for practical hop limits. The results show that all combinations are equally good in case of Net2 and Net4, while combinations using NRRBT2 are the best for Net1. Furthermore, the best results among all combinations are within 1% of those for the individual heuristics.

Table 4-4: Redundancy of combined network reduction - max-latching heuristics at practical hop limits after design tightening

Network	Hop limit	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	A2	C1
Net1	6	.544	.534	.544	.543	.544	.534	.544	.543	.532	.53
Net2	6	.931	.931	.931	.931	.929	.929	.929	.929	.93	.922
Net4	11	1.01	1.01	1.01	1.01	.997	.997	.997	.997	1.014	1.004

Table 4-5 compares the best results from Table 4-4 with IP results at both practical and large hop limits. The practical hop limits for IP are the same as those used in Table 4-4. The combinations chosen are NRRBT2-HeuristicA2 and NRRBT2-HeuristicC1. All the results of Table 4-5 are within 7% of the IP results. Table 4-6 presents the execution times for tightening the SCP designs for combination (1), HeuristicA2, HeuristicC1, and IP. The execution times for combination (1), HeuristicA2 and HeuristicC1 are roughly the same, and typically about 25% of those of IP.

Table 4-5: Comparison of NRRBT2-HeursiticA2 and NRRBT2-HeuristicC1 (after DT) with IP results

Network	IP redundancy		NRRBT2-HeuristicA2		NRRBT2-HeuristicC1	
	at practical hop limits	at large hop limits	% excess redundancy w.r.t IP (practical hop limits)	% excess redundancy w.r.t IP (large hop limits)	% excess redundancy w.r.t IP (practical hop limits)	% excess redundancy w.r.t IP (large hop limits)
Net1	0.4992	0.4992	6.97	6.97	6.97	6.97
Net2	0.919	0.875	1.3	6.4	1.09	6.17
Net4	0.993	0.963	1.71	4.9	0.4	3.53

Table 4-6: Execution times for combination (1), IP, HeuristicA2, and HeuristicC1

Network	Hop limit	IP		With design tightening		
		constraint set generation (sec)	IP run (sec)	(1) (sec)	Heuristic A2 (sec)	Heuristic C1 (sec)
Net1	6	1.6	10.05	2.38	2.23	2.18
Net2	6	0.8	1	4.31	4.09	4.13
Net4	11	6.83	226	49.71	43.47	45.67

4.3 Conclusions

All eight combinations of the greedy and restricted backtracking network reduction algorithms with HeuristicA2, and HeuristicC1 were considered. All combinations were found to be more capacity efficient than max-latching alone for Net1, Net2, and Net4. The redundancy for all combinations was within 17% of the IP results at large hop limits and within 12% of IP results at practical hop limits. Furthermore, these combinations were twice as fast as HeuristicA2 and over six times faster than HeuristicC1 for the large network, Net4. For small networks, the execution times for all combinations and the individual heuristics were of the same order. Among all eight combinations, the combinations of NRRBT2 were the fastest and the most capacity efficient. The redundancy results after design tightening for all combinations, HeuristicA2, HeuristicC1 were within 3% of each other. All the redundancy results were within 7% of the optimum.

5 Case Study of Network Topology Optimization

This chapter presents an example of the application of the fast SCP heuristics to identify the best new span additions in a network. The problem addressed is: *Given a set of new span candidates, is their relative ranking the same from the max-latching and combined network reduction - max-latching heuristics as from IP?* If so, we can quickly rank many alternatives using fast SCP, and then invest the time required to precisely quantify the benefits of the best candidates by using IP.

The approach followed in this chapter is to add a new span, route the working demand over the geographically shortest routes for all demand pairs in the modified network, and perform SCP for the modified network.

Working demand is spread equally over all shortest routes if more than one shortest route exists for a demand pair. This method was implemented previously [30] with $O(D \cdot N \cdot \log(N))$ complexity using Dijkstra's shortest-path algorithm, where D is the number of demand pairs. Although for designing a fully restorable mesh network, shortest path routing does not always result in strictly minimum total capacity, it is found to be very close to optimal routing. The minimum total capacity has actually been observed to occur with route lengths of 1.0 to 1.2 times longer than shortest path routing when jointly optimizing working and spare capacity in a mesh-restorable network design [27].

The test case considered here is the set of six new span suggestions for Net4 shown in Figure 5-1 by dotted lines. HeuristicA2, HeuristicC1, NRRBT2-HeuristicA2, NRRBT2-HeuristicC1, HeuristicA2 with DT, and NRRBT2-HeuristicA2 with DT have been considered for the SCP. The beneficial impact of each new span candidate is ranked by the reduction in number of links (working + spare), and total real distance. The ranking of all six spans is compared to that of IP. The execution times for all the heuristics and IP have also been compared.

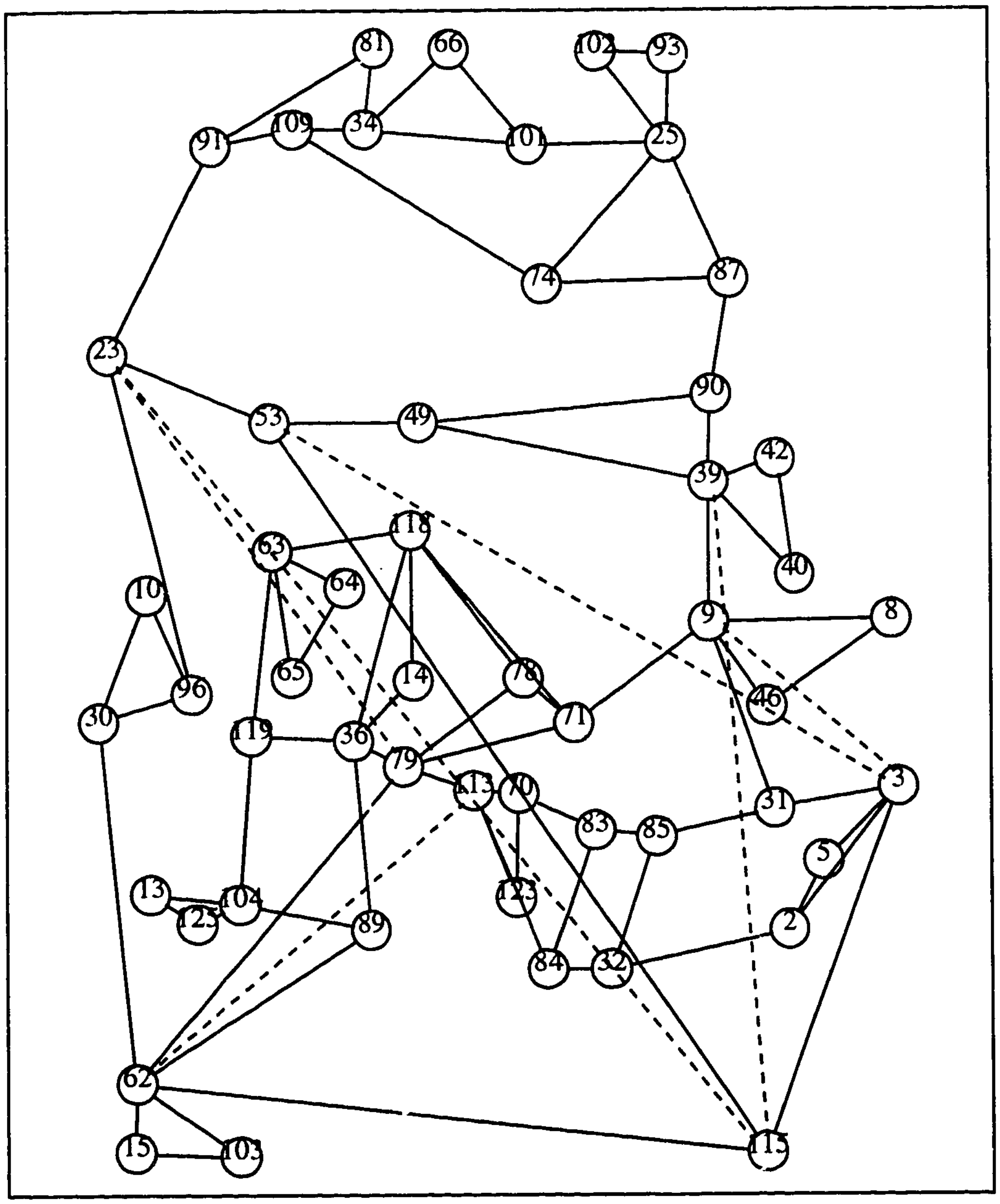


Figure 5-1: Six new span additions for Net4

5.1 Experimental Results

Table 5-1 presents the number of links (working + spare) and total real distance for IP. The objective function for IP-logical is to minimize the total spare capacity, while for IP-real the total real distance is minimized. The hop limit used for IP is 11.

Table 5-1: IP results for six new span additions

Spans	IP-logical		IP-real	
	no. of links	total real distance	no. of links	total real distance
23-79	3766	1108730	3864	1092056
39-115	3980	1153098	4077	1135027
62-113	4047	1165133	4113	1146326
3-9	4047	1172707	4135	1155794
23-115	4050	1189921	4161	1160864
3-53	4054	1180622	4161	1160864

The ranking for IP-logical shows that spans 62-113, 3-9, 23-115, and 3-53 are very close to each other in number of links, but designs with spans 23-79 and 39-115 use over 66 links less. Therefore, spans 23-79 and 39-115 are the most beneficial. This is also true for IP-real.

Tables 5-2(a) and 5-2(b) present the number of links and total real distance for the heuristics. The hop limit used for all cases is 11.

Table 5-2(a): Heuristic results for six new span additions

Spans	HeuristicA2		NRRBT2-HeuristicA2		HeuristicA2 (DT)	
	no. of links	total real distance	no. of links	total real distance	no. of links	total real distance
23-79	4046	1169970	4024	1169095	3801	1093882
39-115	4286	1238049	4221	1186000	4079	1142567
62-113	4371	1256512	4232	1202933	4084	1151723
3-9	4337	1262055	4304	1224405	4120	1156549
23-115	4415	1287196	4322	1258329	4137	1169927
3-53	4377	1269739	4314	1246414	4106	1161384

Table 5-2(b): Heuristic results for six new span additions

Spans	NRRBT2-HeuristicA2 (DT)		HeuristicC1		NRRBT2-HeuristicC1	
	no. of links	total real distance	no. of links	total real distance	no. of links	total real distance
23-79	3846	1098583	4009	1162202	3934	1140214
39-115	4081	1140947	4239	1229015	4182	1183376
62-113	4082	1154012	4273	1237669	4212	1190367
3-9	4140	1166661	4307	1242262	4257	1204010
23-115	4141	1175752	4330	1263238	4256	1228138
3-53	4142	1170031	4348	1267345	4251	1217590

Table 5-3 ranks the spans for the heuristics and IP-logical on the basis of number of links. Table 5-4 ranks the spans in terms of total real distance for the heuristics and IP-real.

Table 5-3: Ranking of spans on the basis of the number of links

Spans	A2	NRRBT2 - A2	A2(DT)	NRRBT2 - A2(DT)	C1	NRRBT2 - C1	IP-logical
23-79	1	1	1	1	1	1	1
39-115	2	2	2	2	2	2	2
62-113	3	3	3	3	3	3	3
3-9	3	4	4	4	4	6	3
23-115	6	6	6	5	5	5	5
3-53	3	5	5	6	6	4	6

Table 5-4: Ranking of spans on the basis of real distances

Spans	A2	NRRBT2 - A2	A2(DT)	NRRBT2 - A2(DT)	C1	NRRBT2 - C1	IP-real
23-79	1	1	1	1	1	1	1
39-115	2	2	2	2	2	2	2
62-113	3	3	3	3	3	3	3
3-9	4	4	4	4	4	4	4
3-53	5	5	5	5	6	5	5
23-115	6	6	6	6	5	6	5

The rankings in Tables 5-3 and 5-4 show that all heuristics follow the IP-logical rankings for the top three spans, and follow IP-real for the top four spans. Most of the heuristics have the same span rankings as that of IP-logical and IP-real for all six spans.

Table 5-5 presents the execution times for the heuristics, while Table 5-6 does so for IP-logical and IP-real.

Table 5-5: Execution time for the heuristics

Span	HeuristicA2 (sec)	NRRBT2-HeuristicA2 (sec)	HeuristicA2 (DT) (sec)	HeuristicC1 (sec)	NRRBT2-HeuristicC1 (sec)
23-79	4.60	1.71	102.23	50.49	9.55
39-115	4.29	1.73	112.08	43.19	9.00
62-113	4.33	1.6	113.84	45.63	9.17
3-9	4.29	1.62	99.61	46.26	9.48
23-115	3.75	1.46	97.18	39.31	8.76
3-53	3.97	1.55	100.21	42.47	9.29
Total	25.23	9.67	625.15	267.35	52.25

Table 5-6: Execution time for IP

Spans	IP		
	constraint set generation (sec)	IP-logical CPLEX^{TM} run (sec)	IP-real CPLEX^{TM} run (sec)
23-79	9.387	3841	4774
39-115	7.871	94	2715
62-113	8.767	263	139
3-9	8.515	4549	3529
23-115	7.84	84	2866
3-53	7.93	78	2550
Total	50.31	8909	16573

Tables 5-5 and 5-6 show that HeuristicA2, NRRBT2-HeuristicA2, and NRRBT2-HeuristicC1 took less thar ıe minute to rank all six spans while IP-logical and IP-reaı took nearly 2.5 and 4.6 hours respectively.

5.2 Conclusions

Six new span additions were tested for Net4 and were ranked on the basis of total number of links (working + spare) and total real distance for IP, HeuristicA2, HeuristicC1, NRRBT2-HeuristicA2, NRRBT2-HeuristicC1, HeuristicA2 with DT, and NRRBT2-HeuristicA2 with DT. IP with objective functions of minimizing spare capacity (IP-logical) and minimizing total real distance (IP-real) was used as a benchmark. The rankings for the heuristics were compared to those from IP. Rankings from the heuristics based on the number of links agreed with the top 3 from IP-logical, and with the top 4 IP-real rankings. The rankings for most of the heuristics agreed with those of IP-logical and IP-real for all six spans.

The least capacity efficient method, HeuristicA2 generated a design with 16% more redundancy than IP. Nevertheless, its rankings agreed with those of IP-logical and

IP-real for all six spans. This shows that the max-latching heuristics can accurately rank span additions even if their SCP results are not precise.

The benefit of using these heuristics is that they execute over 10 times faster than IP-logical and 25 times faster than IP-real. The fastest heuristic took less than 10 seconds for ranking all spans while IP-logical and IP-real took nearly 2.5 and 4.6 hours respectively.

6 Summary

This thesis explored some heuristics for fast and efficient spare capacity placement (SCP) for mesh-restorable networks. One of the main motivations for developing these heuristics is to identify the most beneficial potential span additions in a network. Available SCP approaches are too co.nplex to be used for this purpose. The identified spans can be analysed in detail using an exact IP cost formulation of the network design problem. We also think a very fast SCP heuristic will have general uses in on-line operational and planning systems.

Chapter 2 discussed max-latching heuristics for fast SCP. These heuristics are based on the simple idea of spreading the unrestorable working capacity of each span onto its pre-defined restoration routes, and then latching the maximum spare capacity values on each span. The routeset for each span consists of distinct, loop-free, topologically feasible restoration routes with length restricted by a specified hop limit, *H*. The performance of the max-latching heuristics was found to depend on the order in which spans are spared, hop limit, and working capacity distribution. The general variation of redundancy, execution time and memory usage as a function of hop limit was developed. Heuristics whose SCP results depend on the order in which spans are spared were discussed in detail.

In practice, *S*! span order sequences and working capacity distributions are possible, where *S* is the number of spans. Therefore, it is not feasible to test all sequences to find the best one. Hence three classes of heuristics were considered. Class A heur ics (HeuristicA1, HeuristicA2, and HeuristicA3) place spares for spans in decreasing order of their contribution towards network restorability. The Class B heuristic (HeuristicB1) place spares for spans sequenced by increasing working capacity. Class C heuristics (HeuristicC1 and HeuristicC2) tried several as opposed to only one sequence. HeuristicC1 tried *S* sequences with the first span distinct in each sequence, while HeuristicC2 tried $S * (S-1)$ sequences with the first two spans distinct in each sequence. Four real transport networks were chosen as test cases. For all heuristics the minimum or near-minimum redundancy was observed within four hops of the minimum possible hop limit value. Furthermore, the graphs of redundancy vs. hop limit for all the heuristics showed that class C heuristics were the most capacity-efficient and least working capacity distribution dependent. HeuristicC2 was the most capacity-efficient followed by HeuristicC1.

HeuristicA2 and HeuristicB1 have complexities of $O(S \cdot H \cdot {d_{avg}}^{H})$, where S, H, and d_{avg}, are number of spans, hop limit, and average node degree, respectively. These are the least complex heuristics. HeuristicC1 and HeuristicC2 have complexities $O(S^2 \cdot H \cdot {d_{avg}}^{H})$ and $O(S^3 \cdot H \cdot {d_{avg}}^{H})$, respectively. HeuristicC2 is too complex to be considered for fast SCP.

The redundancies for HeuristicA2 and HeuristicB1 were within 30% of IP for practical hop limits, while those of HeuristicC1 and HeuristicC2 were within 25%. A design tightening procedure, add0_sub1, involving simple elimination of surplus spare capacity from the SCP designs while maintaining 100% restorability was used to improve the capacity efficiency of the SCP designs. Designs produced using HeuristicA2, HeuristicC1, and HeuristicC2 were thereby tightened. The results at practical hop limits for all three heuristics were within 2% of each other, and within 10% of IP. The complexity of the add0_sub1 design tightener is $O(S^2 \cdot \log (w_{max}) \cdot N \cdot \log (N))$, where N is the number of nodes and w_{max} is the maximum working capacity in the network.

Chapter 3 presented network reduction approaches of $O(N^2)$ complexity to improve the execution time and capacity efficiency of the max-latching heuristics. Three methods were considered: fixed topology identification and substitution, greedy, and backtracking. These methods aim at reducing subtopologies which are common in real networks and whose efficient SCP can be done simply. It has been observed that chain-reducible subtopologies occur in a vast range of shapes and sizes in real transport networks. The SCP of chain-reducible subtopologies can be done efficiently.

Fixed topology identification and substitution identifies and reduces a pre-defined set of subtopologies in a single step. Its implementation is subtopology-specific and, therefore, it is difficult to identify and reduce a large number of subtopologies. On the other hand, the greedy and backtracking methods reduce subtopologies in multiple steps. In each step a chain and the span joining its end nodes are identified and reduced. All subtopologies consisting of chains can be reduced through such multiple reduction steps. While doing the SCP for the identified chain, the primary objective is to minimize the total spare capacity in the chain, and the secondary objectives are to maximize the spare capacity equivalent (S_{eq}) and minimize the working capacity equivalent (W_{eq}) of the chain. In some cases it is not possible to simultaneously maximize S_{eq} and minimize W_{eq}. For these cases either S_{eq} is maximized or W_{eq} is minimized. The greedy heuristic NRG1 maximizes S_{eq} while NRG2 minimizes W_{eq}. The backtracking counterparts for NRG1 and NRG2 are NRRBT1 and NRRBT2 respectively.

The backtracking method generates an optimal SCP for a subtopology, and can be implemented with $O(N^2)$ complexity for subtopologies requiring one or two successive chain-wise reductions. This is adequate for real transport networks. The percentage reduction in number of spans or nodes through NRG1, NRG2, NRRBT1, and NRRBT2 for three real transport networks was observed to be over 35%.

Chapter 4 presented the execution time and capacity efficiency results for HeuristicA2, HeuristicC1, and their combinations with NRG1, NRG2, NRRBT1, and NRRBT2. The results showed that the combined network reduction - max-latching heuristics were faster and more capacity-efficient than their individual max-latching counterparts. Among all these heuristics, combinations using NRRBT2 were the fastest and the most capacity-efficient. The results for these heuristics were also compared after design tightening and were within 3% of each other. The redundancy of all the combined heuristics at practical hop limits before tightening was within 12% of IP. After tightening, the results were all within 7% of IP.

Chapter 5 presented an example of network topology optimization. Six new span additions were tested as candidates for addition to a large existing network. Two IP tableaus with the objective functions of minimizing total spare capacity (IP-logical) and minimizing total real distance (IP-real), were used as benchmarks. The heuristics considered were HeuristicA2, HeuristicC1, NRRBT2-HeuristicA2, NRRBT2-HeuristicC1, HeuristicA2 with design tightening, and NRRBT2-HeuristicA2 with design tightening. The rankings of spans on the basis of number of links (working + spare) for these heuristics agreed with the top 3 IP-logical rankings, while rankings based on real distances agreed with the top 4 IP-real rankings. The rankings of most of the heuristics agreed with those of IP-logical and IP-real for all six spans. This shows that the max-latching heuristics can be used to identify the most beneficial spans in a network even if their redundancy is much greater than from IP.

The execution times for all heuristics were 14 times less than IP-logical and 26 times less than IP-real. Among all the heuristics HeuristicA2 was the least efficient. The redundancy for it was 16% of IP for the original network, but its rankings agreed with IP-logical and IP-real rankings for all six spans.

In closing, this thesis has contributed a number of new insights, techniques, and options for fast SCP and it has specifically characterized variations of max-latching and combined network reduction - max-latching heuristics for fast SCP. If required, the capacity efficiency of these heuristics can be improved using the add0_sub1 design tightener.

6.1 Future Work

This thesis identified the two simple concepts of max-latching and network reduction for fast SCP in mesh-restorable networks. The work done in implementing these concepts is a good starting point for future research in fast SCP heuristic design.

Many fast and efficient SCP heuristics can be designed using the max-latching and network reduction concepts. SCP was observed to be dependent on the order in which spans were spared, the working capacity distribution of the network, network topology, average node degree, and hop limit. A detailed analysis supported by experimental results is needed to identify the dependence of the best span sequence on working capacity distribution, network topology, average node degree, and hop limit. With this analysis, it may be possible to predict the cases in which a given max-latching heuristic will perform best.

Bibliography

1. McDonald, J. C., " Public network integrity - avoiding a crisis in trust," *IEEE Journal on Selected Areas in Communications*, Vol. 12, No. 1, pp. 5-12, January 1994.

2. Grover, W. D., *Selfhealing networks - A distributed algorithm for k-shortest link disjoint paths in a multi-graph with applications in realtime network restoration*, Ph.D. Dissertation, Department of Electrical Engineering, University of Alberta, Fall 1989.

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

4. Nellist, J. G., "Fiber optic system availability," *Proc. FiberSat Conf.*, Vancouver, BC, Canada, pp. 367-372, 1986.

5. Saul, D. F., "Constructing Telecom Canada's coast to coast fiber optic network," *Proc. IEEE Globecom'86*, pp. 1684-1688, 1986.

6. Grover, W. D., "Influence of selfhealing & scavenging technology on network availability planning," *Telecom Canada Report CR 89-16-04*, February, 1990.

7. Krten, O. J., " Hypothetical reference digital path for dynamic network architecture - availability considerations," *Telecom Canada/BNR Report OCTL87-0002 P 87-02B*, February 1988.

8. Venables, B. D., *Algorithms for the spare capacity design of mesh restorable networks*, M.Sc. Thesis, Department of Electrical Engineering, University of Alberta, Fall 1992.

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

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

11. Yang, C. H. and Hasegawa, S., "FITNESS: Failure immunization technology for network service survivability," *Proc. IEEE Globecom'88*, pp. 1549-1554, 1988.

12. Grover, W. D., Venables, B. D., Sandham, J. H. and Milne, A. F., " Performance studies of a selfhealing network protocol in Telecom Canada long haul networks", *Proc. IEEE Globecom'90*, pp. 452-458, 1990.

13. Sakauchi, H., Nashimura, Y. and Hasegawa S., "A self-healing network with an economical spare-channel assignment," *Proc. IEEE Globecom'90*, pp 438-443, 1990.

14. Grover, W. D., Bilodeau, T. D., and Venables, B. D., " Near optimal spare capacity planning in a mesh-restorable network," *Proc. IEEE Globecom'91*, pp. 2007-2012, 1991.

15. Sakauchi, H., Okanoue, Y., Hasegawa, S., "Spare-channel design schemes for self-healing networks," *IEICE Trans. Communications*, Vol. E75-B, No. 7, pp. 624-633, July 1992.

16. Grover, W. D., "Case studies of restorable ring, mesh and mesh-arc hybrid networks," *Proc. IEEE Globecom'92*, pp. 633-638, December 1992.

17. MacGregor, M. H., *The Self-traffic Engineering Network*, PhD thesis, Department of Computer Science, University of Alberta, Fall 1991.

18. MacGregor, M. H., Grover, W. D., and Maydell, U. M., "The self-traffic engineering network," *Canadian J. Electrical and Computer Engineering*, Vol. 18, No. 2, pp. 47-57, 1993.

19. Gibbons, A., *Algorithmic Graph Theory*, Cambridge, MA, Cambridge University Press, 1985.

20. Dijkstra, E. W, "A note on two problems in connection with graphs," *Numerische Math.* Vol. 1, pp. 269-271, 1959.

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

21. Garey, M.R., and Johnson, D. S., *Computers and Intractability- A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., 1979

22. Khachiyan, L. G., "A polynomial algorithm in linear programming," *Soviet Mathematica Doklady*, 20, pp. 191-194, 1979.

23. Venables, B. D., Grover, W. D., and MacGregor, M. H., "Two strategies for spare capacity placement in mesh restorable networks," *Proc. IEEE ICC'93*, pp. 267-271, May 1993.

24. Herzberg, Meir, and Bye, S. J., " An optimal spare-capacity assignment model for survivable networks with hop limits," *IEEE Globecom'94*, pp. 1-6, 1994.

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

26. Dunn, D. A., Grover, W. D., and MacGregor, M. H., "Development and use of a random network synthesis tool with controlled connectivity statistics," *TRLabs WP-90-10*, August 1990.

27. Grover, W. D., Bilodeau, T., and Venables, B. D., "Near optimal synthesis of a mesh

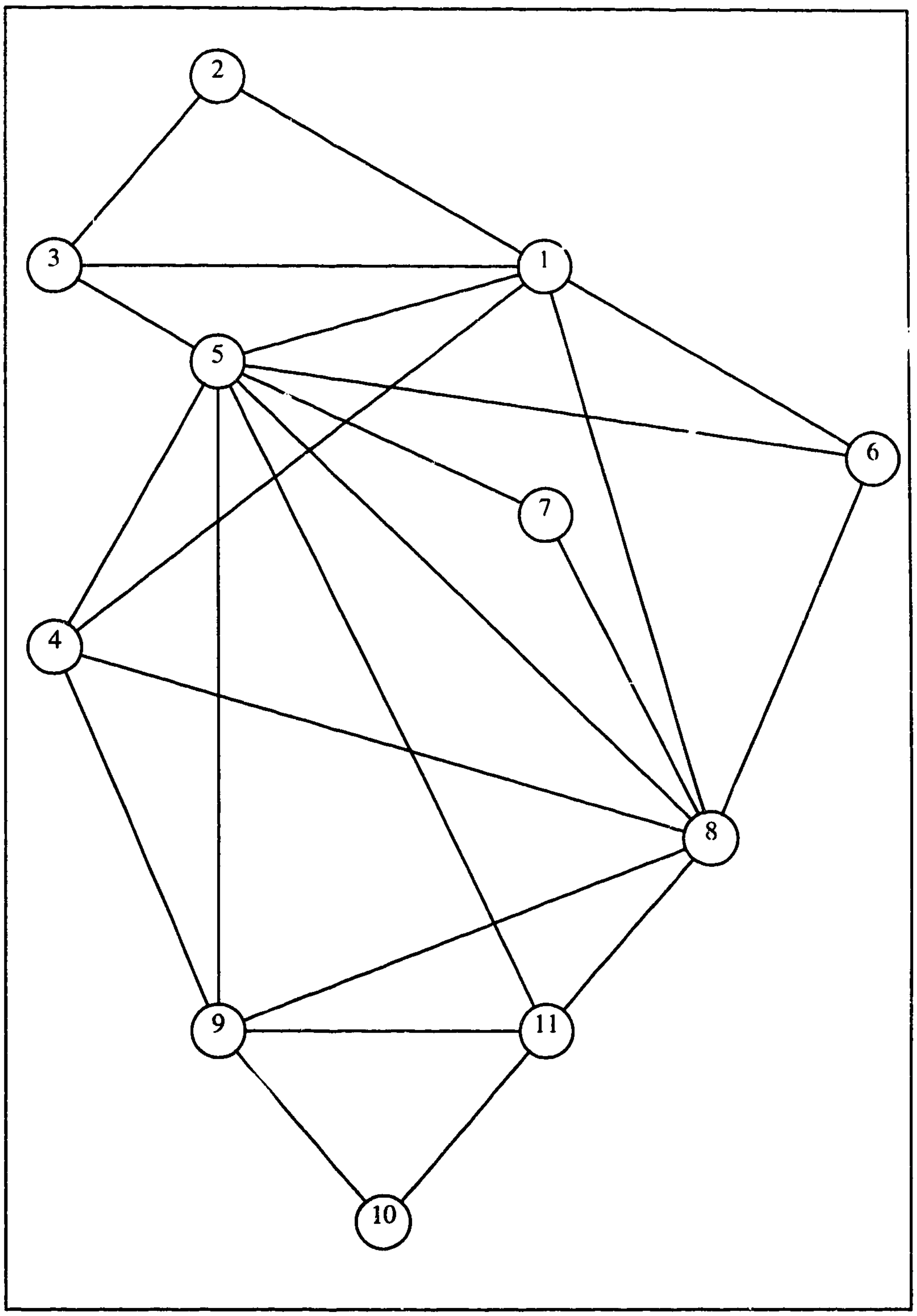
restorable network", Telecom Canada Report CR-90-16-01, 1991.

28. Iraschko, R. R., *Path Restorable Networks*, PhD Thesis dissertation in preparation for Fall 1996 defense, Department of Electrical Engineering, University of Alberta.

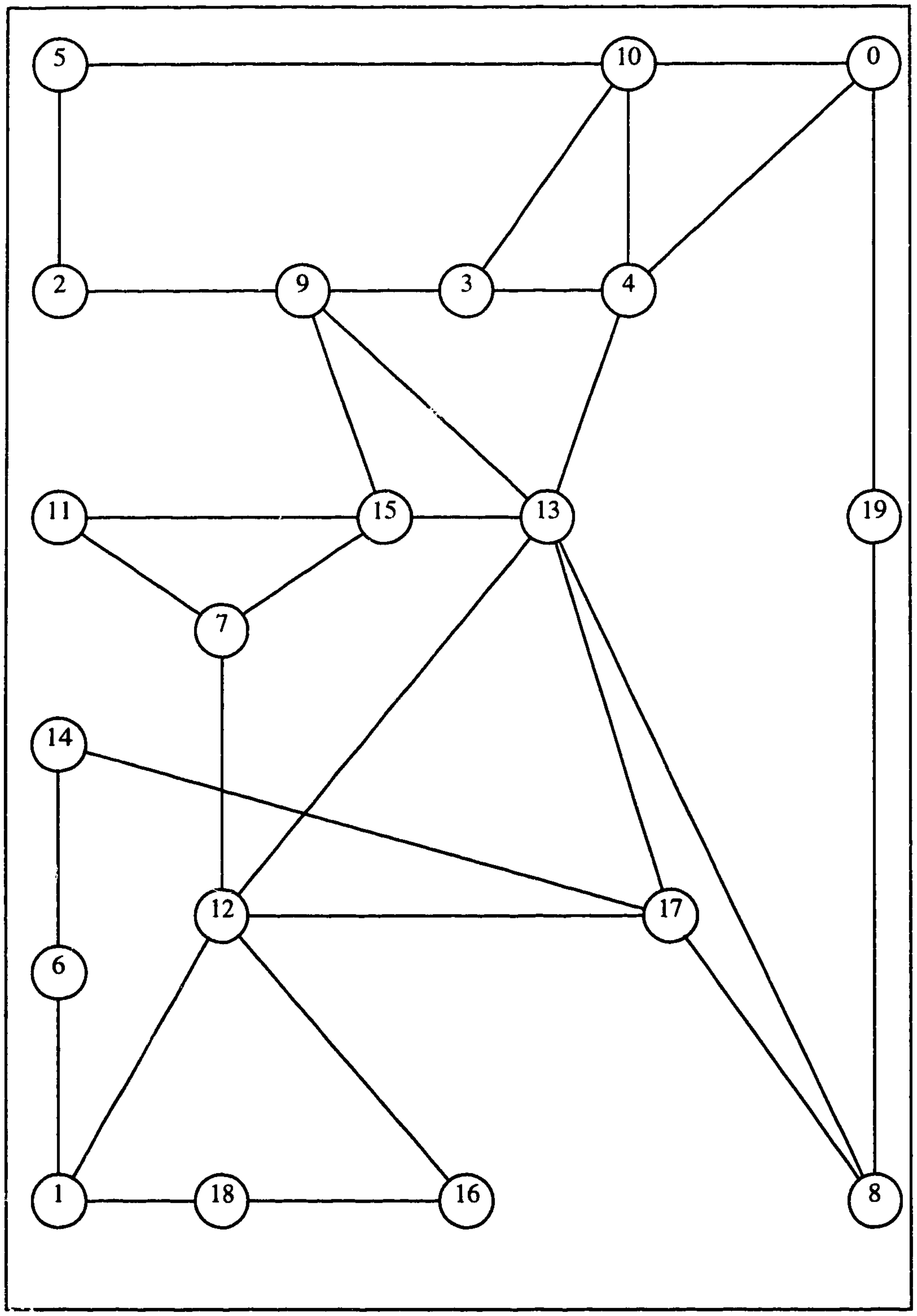
29. MacGregor, M. H., and Grover, W. D., "Optimized *k*-shortest paths algorithm for facility restoration," *Software-Practice and Experience*, V01. 24(9), pp. 823-834, 1994.

30. Slevinsky, J. B., Grover, W. D., and MacGregor, M. H., "An algorithm for survivable network design employing multiple self-healing rings," *IEEE Globecom'93*, pp. 1568 - 1573, 1993.

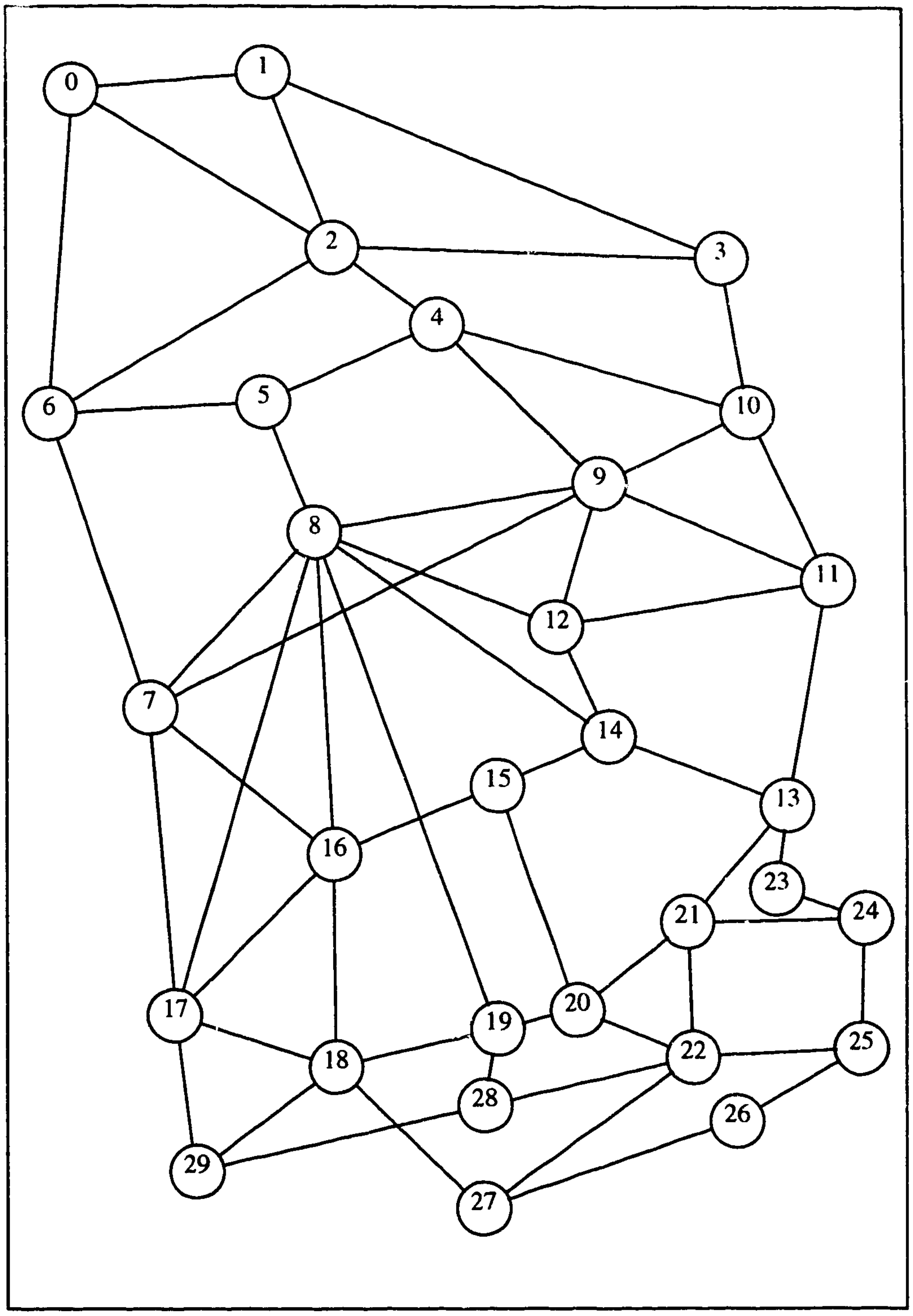
APPENDIX A: TEST NETWORKS



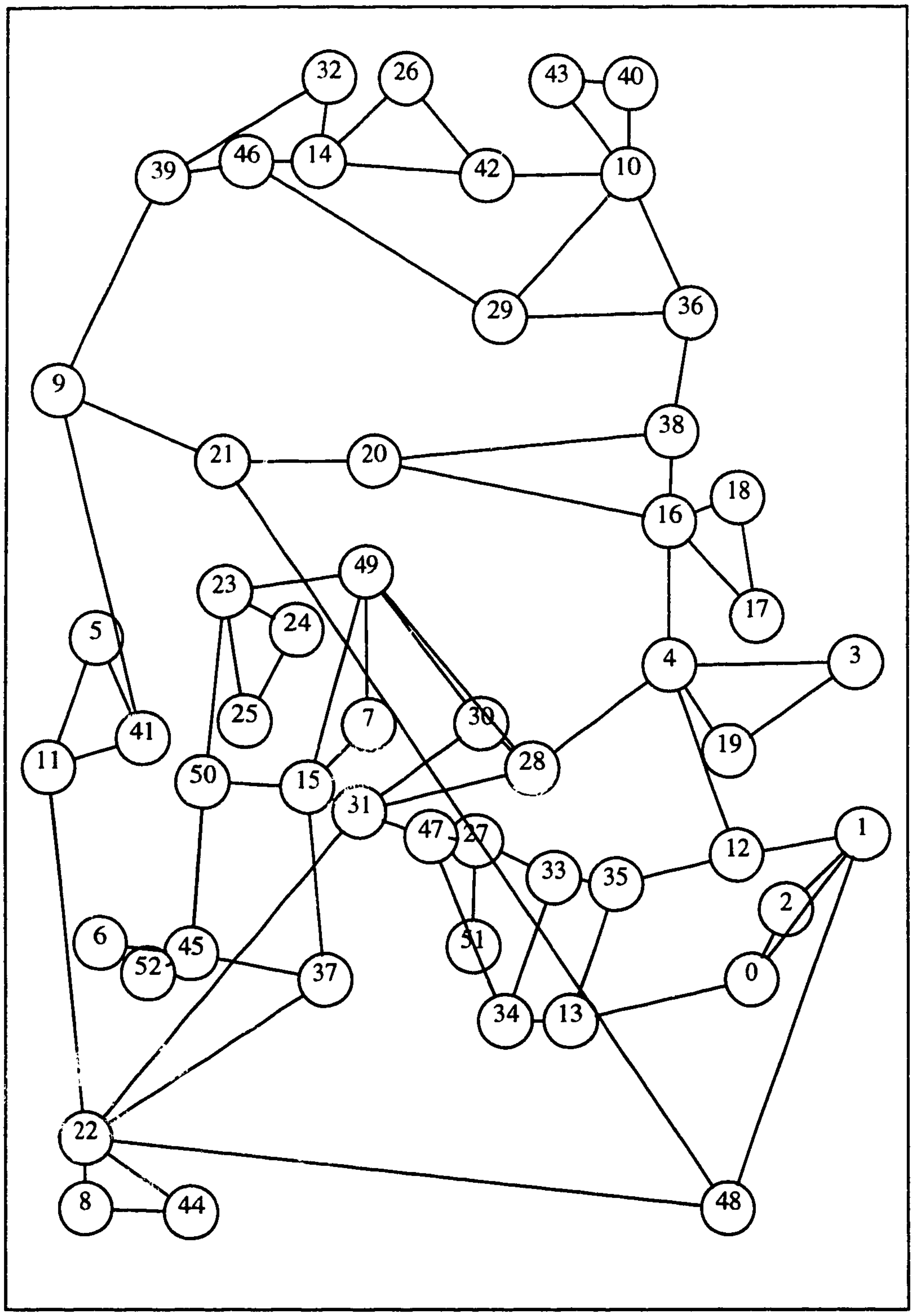
A-1: Net1



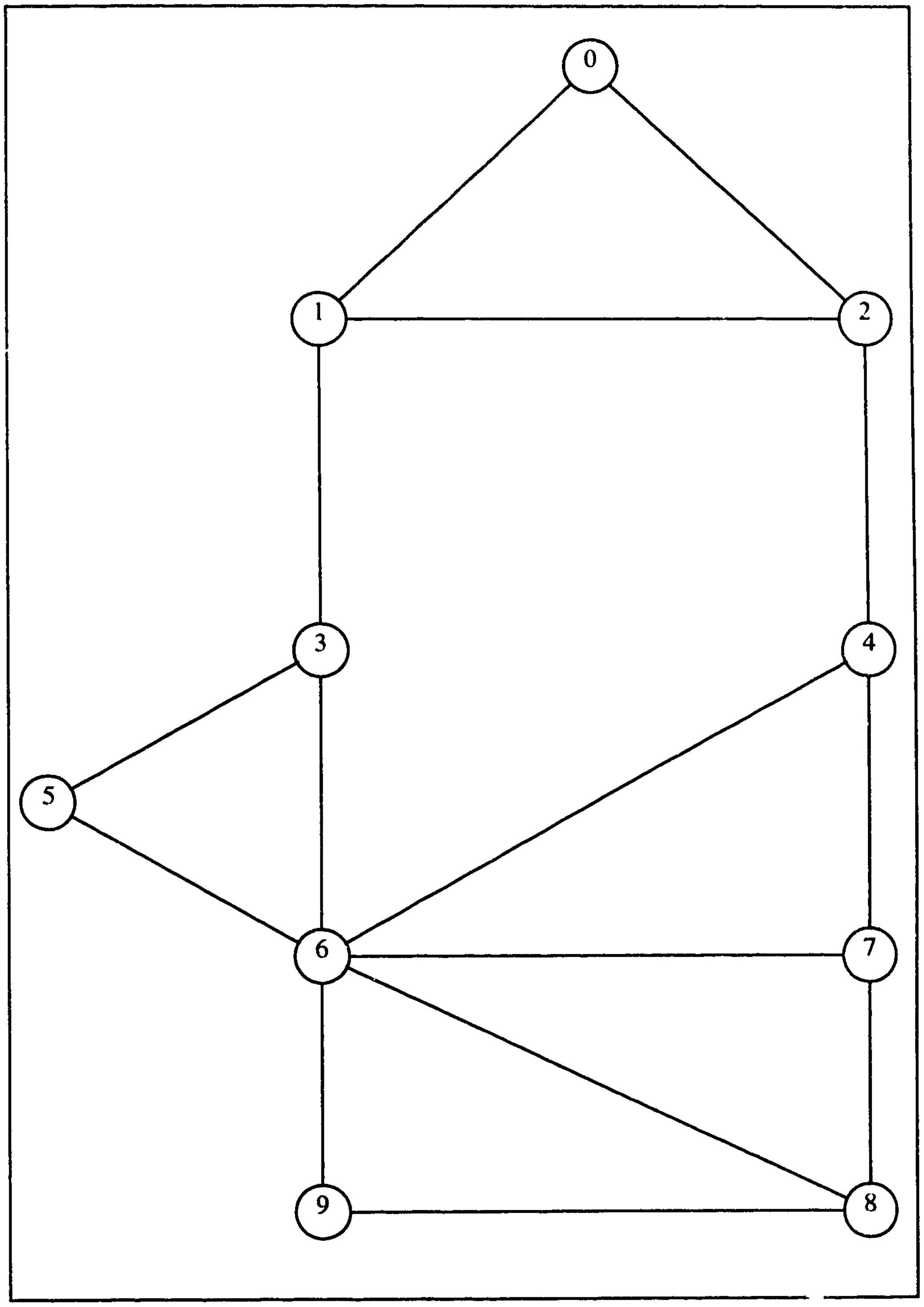
A-2: Net2


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

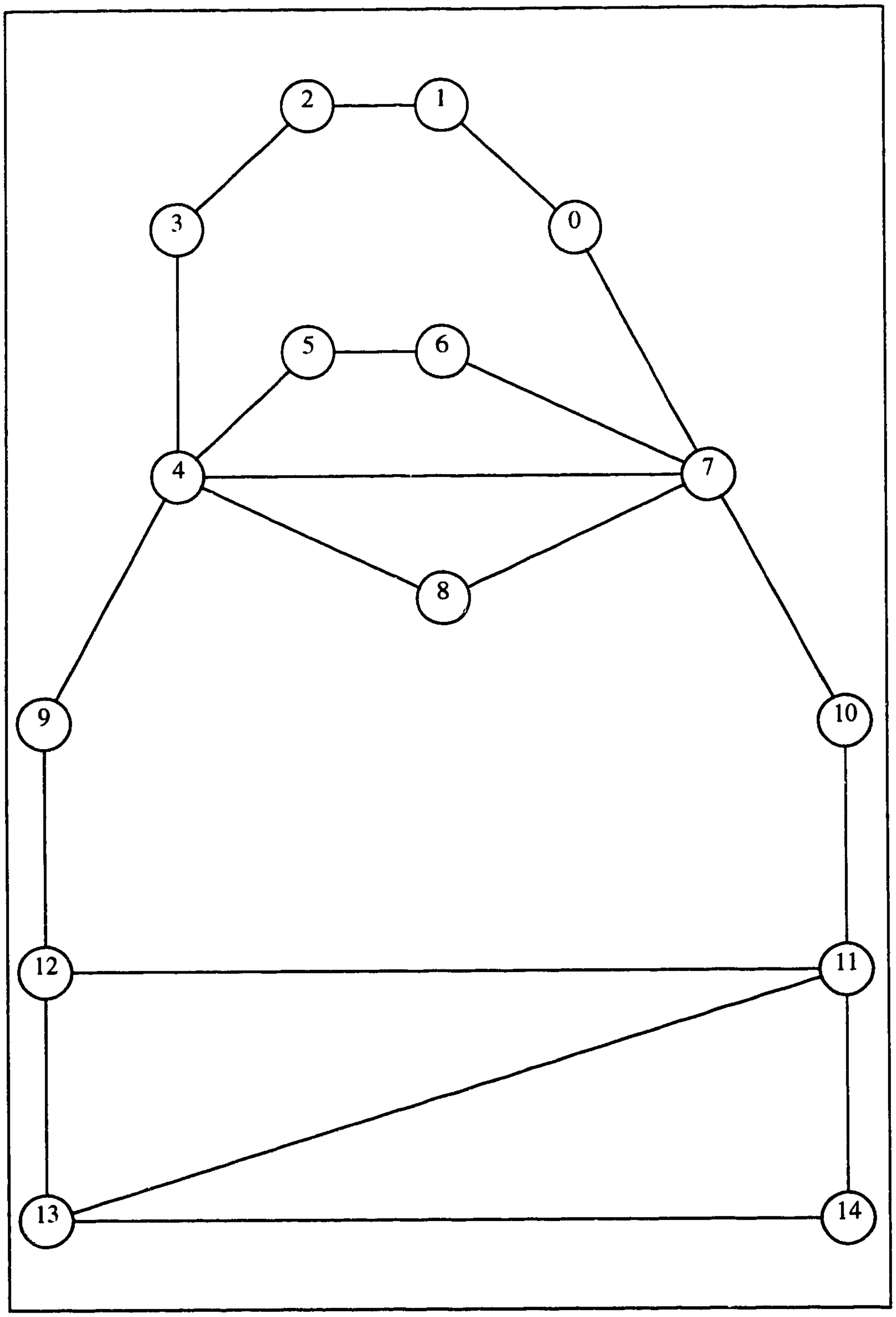
A-3: Net3


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

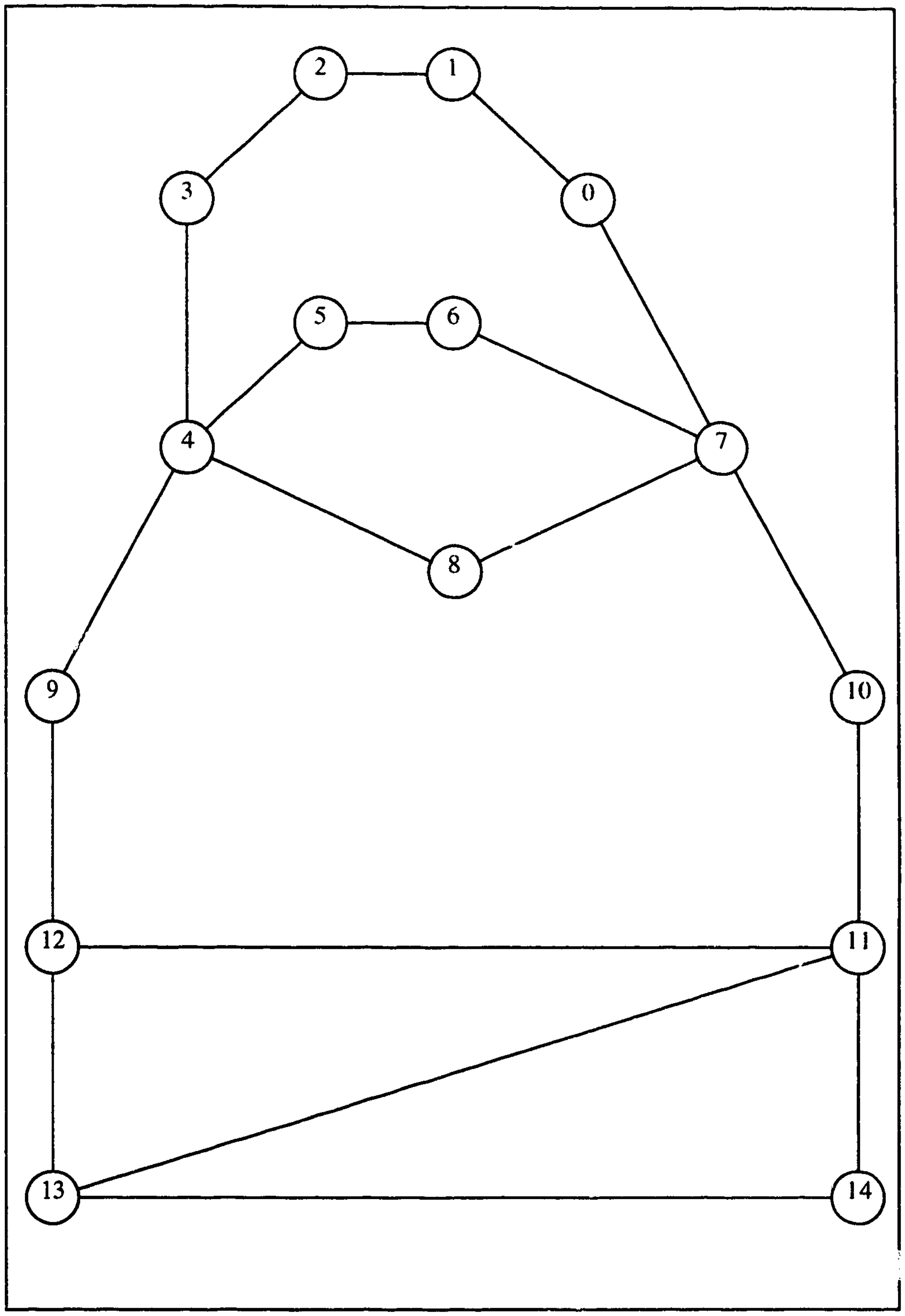
A-4: Net4



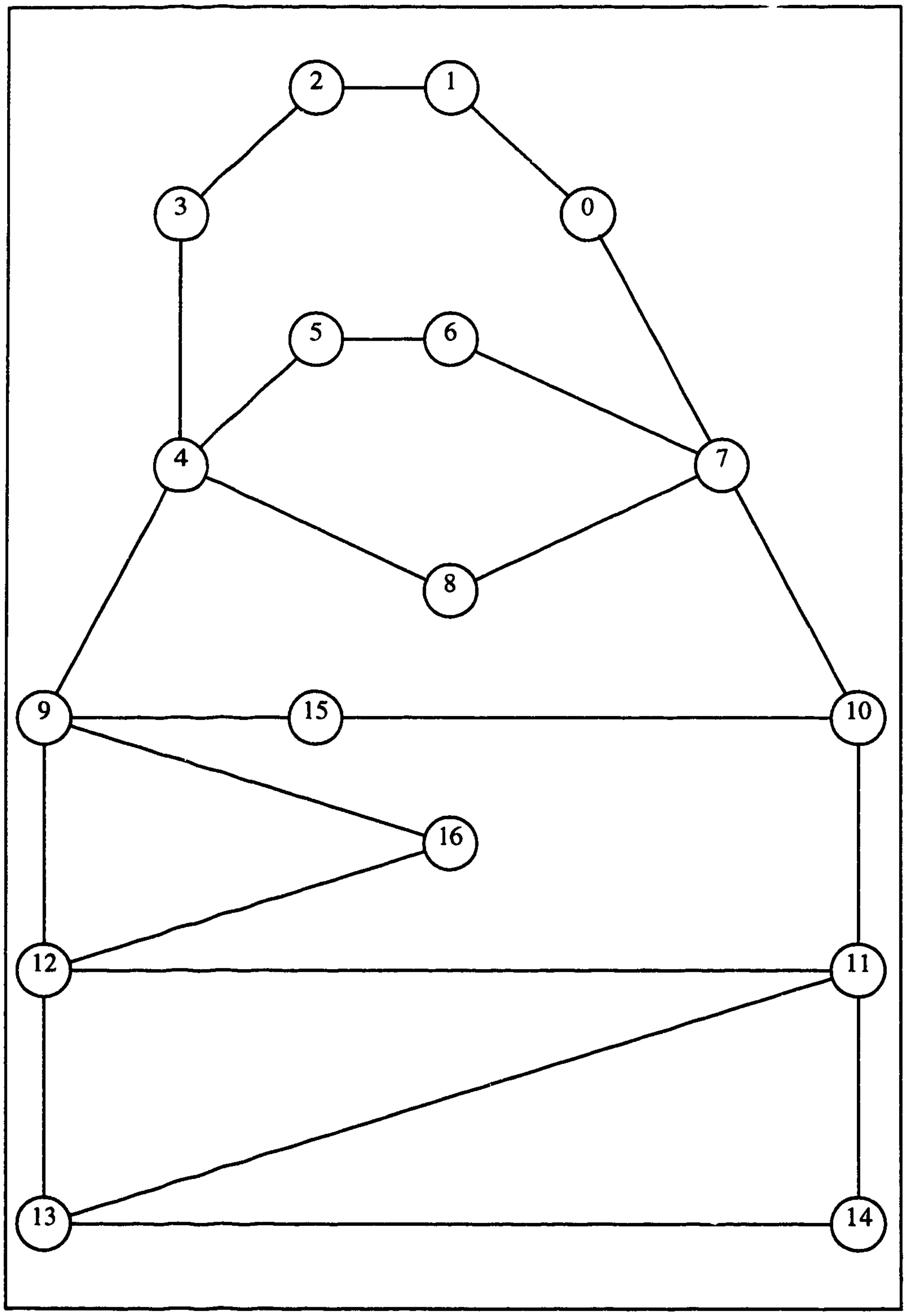
A-5: Net5



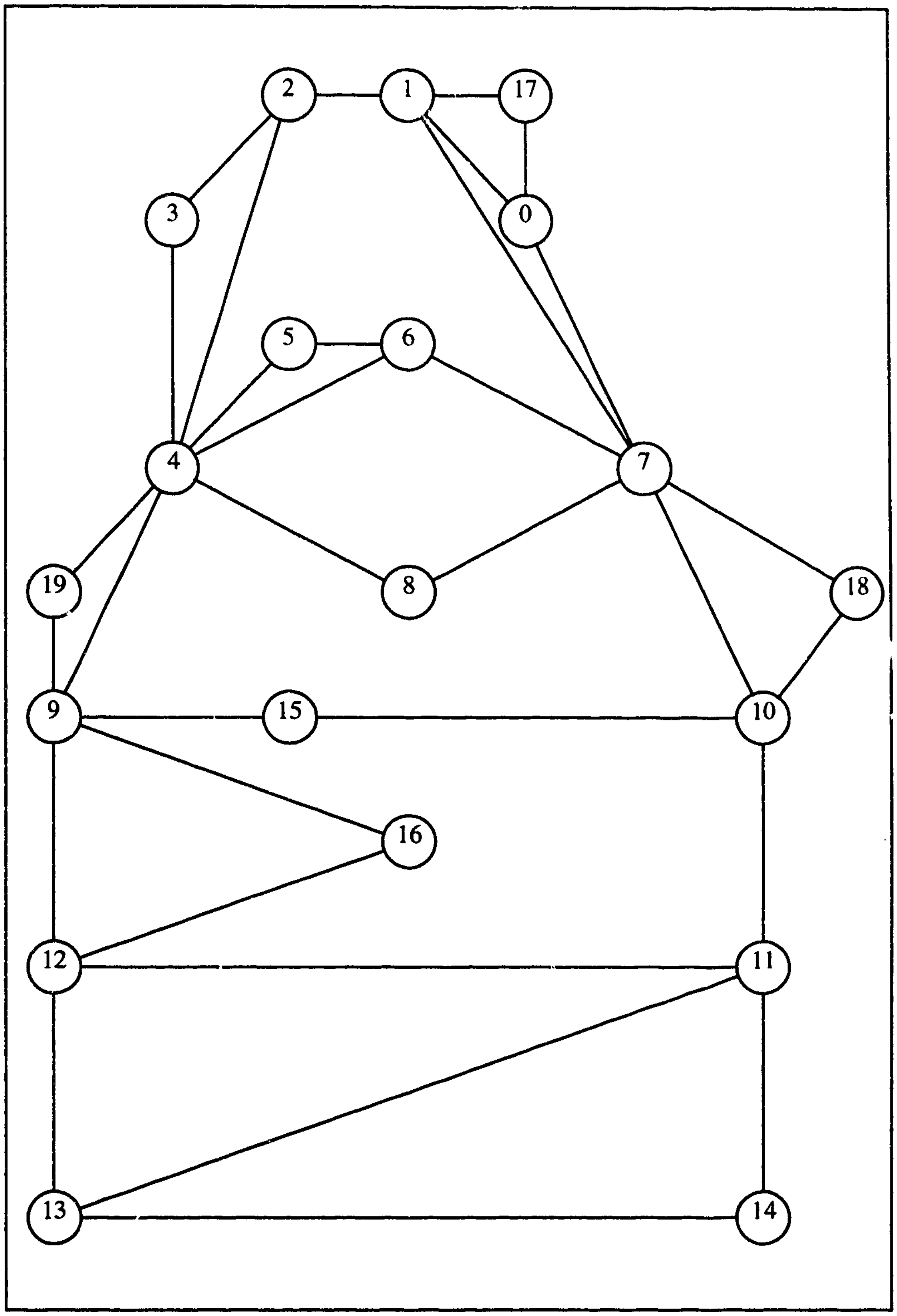
A-6: Net6



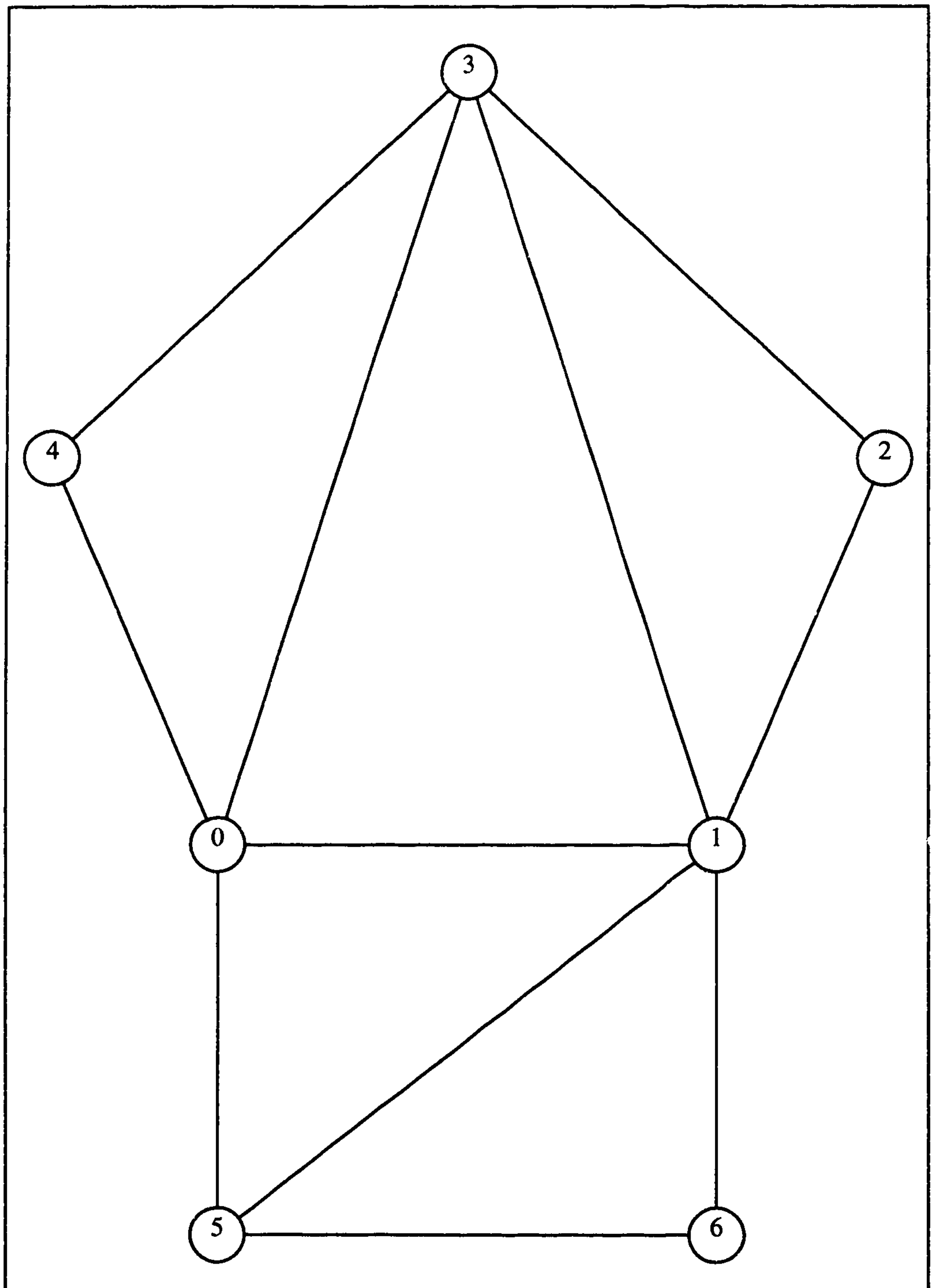
A-7: Net7



A-8: Net8 and Net9



A-9: Net10



A-10: Net11