Optimization Methods for Topological Design of Interconnected Ring Networks

by

Valery Brodsky

B. E. in Electrical Engineering, Stevens Institute of Technology
(1992)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1994

Author .....
Department of Electrical Engineering and Computer Science
January 19, 1994

Certified by .....
Anantaram Balakrishnan
Associate Professor of Management Science
Thesis Supervisor

Accepted by .....
Richard Larson
Co-Director, Operations Research Center

Optimization Methods for Topological Design of Interconnected Ring Networks

by

Valery Brodsky

Submitted to the Department of Electrical Engineering and Computer Science
on January 19, 1994, in partial fulfillment of the
requirements for the degree of
Master of Science in Operations Research

Abstract

Synchronous Optical Networks are fiber-optical telecommunications networks that provide a backbone for installation of multimedia services. This thesis presents a hierarchical view of the SONET network design problem. We analyze the technology and economics behind different survivable topologies for SONET networks. We create a planning model for survivable Interconnected Ring Network [IRD] and consider issues associated with expansion planning of the ring networks. We consider computational complexity of the [IRD] problem and develop and test fast heuristics for designing the ring network. The algorithm is tested on two types of demand structures. For a mesh hub-to-hub demand the heuristic solution is on the average within 5.30% from an optimal solution; for hub-to-central office star demand the heuristic is on the average within 1.90% of optimality.

Thesis Supervisor: Anantaram Balakrishnan
Title: Associate Professor of Management Science

Acknowledgments

I am thankful to my advisor professor A. Balakrishnan for his support, advice and tolerance of my English. I am grateful to Chiaolin Ming of Bellcore for her help and care. My thanks also go to my family who were always there for me. I acknowledge the help and support of my friends not necessarily in the order of importance: Sougata Datta, Yaron Zilberman, Barry Kostiner, Christian Voigtlaender, Jim Shor, all the ORC crowd, Rilke, Kafka and Nietzsche, my friends overseas, Barry Kostiner again, for his contribution to imlementation of GAMS model, and long dark winter nights.

Contents

1 Introduction **8**

2 Technical Background **11**

2.1 Synchronous Optical Network Standard 11

2.2 Survivability 12

2.3 Survey of Circuit Restoration Techniques 14

2.3.1 Methods of Protection 14

2.3.2 Self-Healing Rings 17

2.3.3 Path Rearrangement Using DCS 23

2.4 Economic Analysis of Network Expansion Based on Survivable Architectures 26

2.4.1 Objectives of SONET Network Planning 26

2.4.2 Economic Performance of Network Topologies 28

2.4.3 Ring Architecture Selection and Interconnection 32

3 Problem Description and Formulation **38**

3.1 SONET Problem Hierarchy 38

3.2 Interconnected Ring Network Problem Formulation 42

3.2.1 Model and Assumptions 42

3.2.2 Description of Costs and Constraints 48

3.3 Computational Complexity of [IRD] 50

3.4 Expansion Problem for Survivable Ring Networks 52

4 Initial Solution Heuristics **58**

4.1 [IN1]: Profitable Ring Selection Initial Heuristic 58
4.1.1 [PR]: Profitable Ring Selection Problem 58
4.1.2 Knapsack-Based Initial Solution Heuristic 61
4.2 [IN2]: Slack Packing Heuristic for the Initial Solution 67

5 Solution Improvement Heuristics **72**
5.1 [SWAP]: Swap Heuristic 72
5.1.1 Motivation 72
5.1.2 Swap Moves 73
5.1.3 Algorithm Description 74
5.1.4 Computational Complexity 77
5.2 [I/C]: Interconnection Heuristic 79
5.2.1 Motivation 79
5.2.2 Algorithm Description 80

6 Computational Results **83**
6.1 Data and Test Problems 83
6.2 Details of Implementation 85
6.2.1 A Lower Bound of the [IRD] 87
6.3 Computational Results for Small-Size Problems 88
6.4 Large Size Problems 96
6.5 Conclusion 100

A SONET Carrier Signal Rates **102**

B A Bound on the Maximum Number of Rings in an Optimal [IRD] Solution **103**

C Solution Example **106**

D C Code **113**

E Bibliography **176**

List of Figures

2-1 Two-Level Model of Telecommunications Network 15
2-2 Methods of Node Protection: Multihosting and Multihoming. 16
2-3 Self-Healing Ring 18
2-4 Add/Drop Multiplexor 19
2-5 4-fiber BSHR: Short Path Span Protection 22
2-6 DCS: Schematic Depiction 23
2-7 DCS: Block Diagram 24
2-8 Methods of Protection 29
2-9 Expected Loss of Traffic vs Cost 30
2-10 Worst Case Survivability vs Cost 30
2-11 BSHR to USHR Capacity Ratio 33
2-12 BSHR to USHR Cost Ratio 34
2-13 Ring Interconnection Using ADM 35
2-14 Ring Interconnection Using DCS 35
2-15 SONET Architecture Comparison 37

3-1 SONET Problem Hierarchy 41
3-2 ADM Cost 45
3-3 Facility Location Problem Transformation 51
3-4 Network Expansion Problem 53
3-5 Network Expansion Example 54

4-1 [IN1] Initial Solution Heuristic 66
4-2 [IN2] Initial Heuristic 2 71

5-1 Initial Solution: Greedy Is Not Always Optimal 73
5-2 Swap Moves 75
5-3 [SWAP]: Swap Heuristic for Local Improvement 78
5-4 [I/C]: Local Interconnection Improvement Heuristic 81

6-1 Program Structure 86
6-2 Heuristics Final Solution vs.Optimal Solution, Uniform Demand . . . 92
6-3 Heuristics Final Solution vs.Optimal Solution, Star Demand 94
6-4 Optimality Gap, 8-Node Uniform Demand 94
6-5 Optimality Gap, 2-Hub Star Demand 95

C-1 Sample Data 107
C-2 Heuristics Solution for the Sample Data 107

Chapter 1

Introduction

Due to recent developments in fiber-optical communications, transmission facilities are no longer a bottleneck of a telecommunications system. 4 Gbit/s transmission rate has been reached using a single mode optical fiber. Such high capacity allows incorporation of real-time applications in a single link. For example, digitization of a standard analog video signal requires 100 Mb/s, high-definition TV would take up to 1 Gb/s capacity, and a coded X-ray image uses from 50 to 100 Mb/s bandwidth. Thus it is possible to transmit any kind of data and information, from local calls to medical imaging, through a unified communication system. This concept is known as "integrated networking". In order to take a full advantage of new services, existing network structure needs to be updated to incorporate the fiber technology.

Local Bell Operating Companies (BOC) are gearing towards gradual conversion of the wire and cable facilities into fiber. Current traffic pattern in the telephone network does not necessarily make fiber a cheaper alternative to coaxial cables, but the introduction of broadband ISDN services in the future and competition from cable TV companies requires a backbone fiber-optical network. To understand the scale of changes facing BOC's, consider the following developments [NYT93].

In February 1993 US West announced plans to update the telephone network with fiber links and create a system for consumers to order movies on demand and talk on video phones. The cost of equipment is estimated to be $500 million. The company is trying to build quickly a high-capacity network, using the available facilities

and installing additional coaxial cable and fiber to satisfy the increased bandwidth requirements. US West plans to add 500,000 customers a year to the network. A copper line will still carry the voice communications for the last hundred feet into the customers' homes. In a separate venture, GTE will spend $ 240 million on 50 fiber optic networks in 12 states.

We can infer that the transition to the integrated services telecommunication network is capital intensive. Conversion to high-capacity networks will evolve gradually. The BOC and long-distance companies would prefer to make use of existing equipment and lines wherever possible. Planning will play a major role in the transition. Numerous ISDN applications for the high-capacity network are still under development, and network planning should allow fast and relatively inexpensive accommodation of the new emerging services. The changes in telecommunications technology require the emergence of a new standard, so that various switches and local networks can communicate to each other.

Methods of operations research can contribute to design of a cost-effective network expansion plan. In this work we concentrate on designing SONET networks. The objectives of this work are to present the recent developments in fiber optics technology, uncover the areas of applications of operations research to survivable SONET design and create a model and develop efficient solution methods for a selected problem.

The paper proceeds as follows. Chapter 2 provides a motivation of our work. It introduces the concept of survivability and develops understanding of survivability techniques. Chapter 3 introduces a hierarchical SONET design problem – a generic model for strategic planning, design and operations of SONET networks. We describe in detail a Interconnected Ring Network design problem [**IRD**] and formulate a mathematical model for the problem, and show that [**IRD**] is NP-hard. Chapter 4 considers initial solution heuristics. We introduce a Profitable Ring Selection problem [**PR**] and develop an initial solution heuristic based on Lagrangean Relaxation of [**PR**]. Another initial solution heuristic uses an insight about the application's context (telecommunications). Chapter 5 describes improvement methods for the initial solution. Chapter 6 discusses computational results and develops a lower bound based

on the context of the problem. The average gap between our algorithm's performance and a lower bound generated by GAMS optimization software is 3.46% for generated test problems. In the conclusion we discuss possible future research.

The thesis makes the following contributions. We develop a framework for a systematic approach to SONET Network design. The approach clarifies the technological details such that operations research methods can be applied to develop cost-effective solutions. We consider, in detail, the design of a network using self-healing rings. The problem emerged previously in a number of papers, for example [LAG93]. We develop and test fast and efficient heuristic algorithms. The algorithm consists of several subroutines for creation of initial solution and subsequent improvement. The subroutines can be combined in a modular fashion to achieve the best result. We consider models for an expansion problem. The problem, to our knowledge, has never been considered before for the ring network. We analyze computational complexity of the associated models.

Chapter 2

Technical Background

2.1 Synchronous Optical Network Standard

Synchronous Optical Network standard (SONET) was first introduced by Bell Communications Research Inc. (Bellcore) in 1985 as the means to facilitate development of optical telecommunications network. SONET was formally proposed to the international community in 1987 [BAL89]. The term "synchronous" was coined from the SONET basic signal: Synchronous Transport Signal 1 (STS1=51.84 Mbits/s). It is also referred to as OC1 (Optical Signal 1). All the higher rate signals are multiples of STS1 (See Appendix A for SONET signal rates).

SONET is designed to define optical signals, a synchronous frame structure and operations procedure for synchronous optical networks. It is clear from the related articles, that the signal part of SONET is widely used as guidelines for routing and optical network architecture design.

One of the major advantages of SONET is its capability to assemble signals of different rates in modular fashion. This allows the system to accommodate a wide range of different services, and to provide new services without changing the existing network [FLA90a]. SONET is a paradigm for the efficient development of high-capacity communication network, geared towards current as well as future applications.

2.2 Survivability

Survivability of a network is defined as survival of a fixed percentage of calls (information) in case of a failure. A *network* is a subset of central (local) offices with associated interoffice demands. *A failure* is any network failure such that some network elements cease to function normally. An example of the failure could be a single-link cut due to construction work, a switch malfunction, or a set of links and switching nodes affected by a natural disaster.

The last case is different from the previous two because more than one element of the network is affected by the failure. We refer to the failure of one element (link or node) as a single failure. Experience suggests that multiple failures occur mainly due to natural disasters. In case of survivability analysis it is assumed that probability of a multiple failure is negligible comparing with probability of a single failure. From now on we use the word failure to refer only to a single-link or single-node failure. We can approximate every single failure as being independent of other failures. Furthermore, we assume is that the repair time is much smaller than the interfailure time; thus, survivability analysis concentrates on coping with one single link or node failure at a time.

Survivability in case of failure is an issue because it takes unacceptably long period of time from a customer's point of view to restore the failed portion of the network. For that period the corresponding connection is lost.

A simple solution to designing survivable networks is to double all the existing links and switches. Thus if a certain element fails, the backup switch/link can be engaged. Such a solution incurs excessive cost. The approach can be improved using optimization methods and more flexible network structure, but the basic idea holds: to recover the demands in case of a failure, there should exist a backup switch or free link capacity in the network in order to reroute the demands.

A survivable network design involves additional cost due to control software, extra hardware, and personnel to service the system. Those costs have to be balanced by the savings due to survivability in order to make the implementation of the new

architecture feasible.

In order for a survivable network to be cost-justified, the cost of the failure per link/per node should be sufficiently high, and the failures should occur sufficiently often. Let us approximate as the measure of cost-effectiveness of protection as Failure Cost = [unit call cost] $\times$ [average number of units failed] $\times$ [number of failures per year]. Let us assume that if the Failure Cost is greater then certain threshold, it may be cost-efficient to install additional systems.

In terms of our model, the Failure Cost for the pre-SONET Plain Old Telephone Service (POTS) is low, thus survivable architectures for POTS are not justified. Let us investigate how the latest changes in technology have influenced the Failure Cost.

In some cases the increased capacity of fiber links leads to the loss of increased number of demands in case of a failure. For some important information the cost of a unit-call is high. Consider computerized trading. Computerized trading relies heavily on network services to complete the transaction. Whereas the volume of the traffic may not be high in this case, the importance of the information dictates a high cost per unit of the lost data.

Consider the frequency of the single link failures for 1987 ([GRO87]). The availability of fiber was 96.5% a year. This amounts to 300 hours a year downtime. Time to restore a physical link was between 6 and 12 hours in 1987. The cost of an outage is estimated to be \$75,000/min, or \$1.35 bln loss of service revenues for the year. Therefore, even discounting the node failures, single-link outages have high enough cost to justify protection.

The exact numbers on the cost of a minute of down time could be debated, but the conclusion is that even the down time of a 300 hours a year can cause a significant loss of income. Most probably, the availability figures have improved for the past few years. It is likely, however, that the cost of the outage did not decrease. Thus extrapolating, we can estimate that savings due to survivability could be in the order of hundreds of millions of dollars a year since the number of high capacity networks is steadily increasing.

Our analysis, however sketchy, suggests that the Failure Cost went up due to the

introduction of new technology and proliferation of new networking services. In the following sections of this chapter we cover the restoration methods and review the literature dealing with the economic analysis of survivability.

2.3 Survey of Circuit Restoration Techniques

2.3.1 Methods of Protection

Consider a hierarchical model of a communications circuit-switching network with customer's offices at the bottom of the hierarchy, and central offices, hubs, and gateways at successively higher levels. A node's position in the hierarchy is defined by its function, connectivity and the number of demands terminating and passing through the node. A customer node, for example, can be a building. An example of a hub can be a local switch, terminating all the telephone lines beginning with the same three-digit phone number. We can characterize the customer's office as having a small number of demands terminated and passed through. The higher level offices are capable of handling increasing number of demands and have increasingly more sophisticated hardware and software.

Traffic from the customer's offices to the corresponding central office is considered to be negligible for survivability to be applied at this level (recall our discussion of the Failure Cost from section 2.2). Thus we adopt here a two-level network model with the central nodes (CO's) at the bottom and hubs at the top level.

The major difference between the hubs and the central offices is the amount of traffic and connectivity. Consider the two level network model shown at Figure 2-1. The office to hub network has a star topology. The hub to central office demands are typically in the order of DS1 units, and do not have to use fiber optic links because the link would be underutilized. Signals lower than DS3 are sometimes referred to as **Virtual Tributary** signals (VT) since those signals have to be multiplexed at the hub level to the capacity of DS3/STS1, which is a basic carrier in SONET fiber-optic networks. All the hubs must be interconnected via high capacity links. Gateways

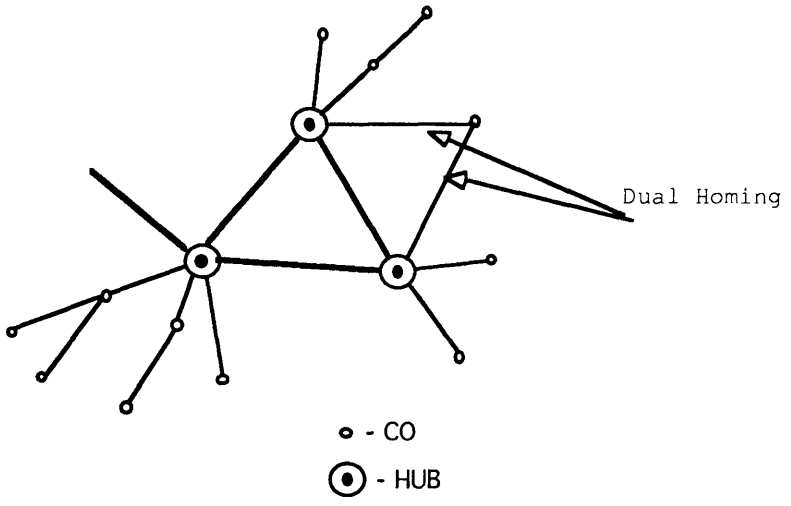


Figure 2-1: Two-Level Model of Telecommunications Network

(not shown on Figure 2-1) are treated as hubs that are not directly connected to any central office. The interhub demands are of DS3 order. The hubs are interconnected via fiber.

The hub-to-hub network is referred to as the transport network; the CO to hub connections form the access network. It is crucial that the hub receives all the corresponding VT signals in order to multiplex it into DS3 units, because the hub only monitors the switches at DS3 level failures and will not detect absence of one or more VT signals from the bundle. Thus we have two levels of signal protection: Virtual Tributary signal protection of the access network and DS3 protection of the transport network. Both protections are deployed at the physical level, to protect links (link protection) or nodes (node protection).

The third level of protection is logical protection: to identify and correct problems before they affect the service (see [FLA90a]). This includes maintenance and control. We restrict our discussion here to the physical link and node protection.

Even though two levels of network need to be protected separately, the methods of protection and demand restoration are the same.

Link protection deals with various path rerouting techniques. Two major approaches are used to incorporate link protection.

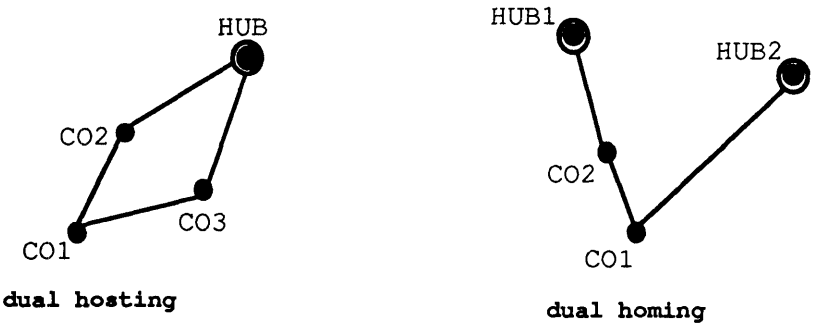


Figure 2-2: Methods of Node Protection: Multihosting and Multihoming.

1) Design a network with built-in protection capacity for each path and switch the traffic in case of failure. Examples of this approach are self-healing rings and 1:1 path protection. 1:1 protection provides backup path for every existing routing path in the network. For each demand pair $1:1$ protection uses a separate pair of fibers, so that the number of links for a network with k demands is $\frac{k(k+1)}{2}$. Similarly, 1:N protection provides one backup for N routes (see [WU88], [FLA90b]).

2) Do not reserve capacity for every circuit in case of a failure, but use intelligent switching elements, such as digital crossconnects, to find free capacity and switch the circuits on available links. This is called path rearrangement. This second approach is characterized by more efficient use of bandwidth, since different calls can use the same protection capacity (bandwidth reuse). Path rearrangement can be viewed as an improvement of 1:1 and 1:N strategies. This method does not require as much spare capacity as 1:1, and it provides higher than 1:N degree of survivability for protected circuits; but it also requires more expensive switching equipment (for references see [WU88], [FLA90b], [WU93]).

Figure 2-2 presents the methods of node protection. This particular figure reflects the access network node protection, but the method is general and can be applied to hubs. To protect a node, it can be either multihosted, or multihomed, or both. In case of multihosting a central office CO1 is connected to the hub through two other central offices CO2 and CO3 via different paths. If CO2 fails, the demands of CO1 that are connected to the hub via CO3, survive. The calls from CO1 passing through

CO2 are not restorable because of the technology. Dual homing is implemented by connecting a central office to two different hubs. If one of the hubs fails, all the demands are rerouted through another hub.

Note that the above methods of protection are technology independent and can be deployed for both wire and fiber links.

In the next section we describe network protection methods using rings.

2.3.2 Self-Healing Rings

Ring architecture is defined in the SONET standard. Understanding how a ring works is important for understanding the problems encountered by SONET planners. For more references on the rings see [DRA93].

A ring, as implied by the term, is an architecture that connects all the nodes via a closed loop. Protection of the ring demands is based on the principle of **self-healing**. Since the underlying topology of the ring is a loop, for every pair nodes on the ring the ring contains exactly two edge-disjoint paths connecting those two nodes. If one of the links fail, another path on the ring exists such that the two nodes are still connected. This is termed as self-healing principle. Note that self-healing is not identical to 1:1 protection (see section 2.2). The equipment required for 1:1 protection is different from the ring hardware. For a self-healing rings with n nodes, n links are required independently of the number of the demand pairs. Protection switching in case of $1:1$ method occurs using switches, similarly to rerouting of an ordinary call. The rings use novel devices called Add/Drop Multiplexors (ADM) to achieve self-healing. These devices perform automatic switching in case of a service deterioration.

There are two different types of ring configurations: Unidirectional Self-Healing Rings (USHR) and Bidirectional Self-Healing Rings (BSHR).

Consider the schematic depiction of a ring shown in Figure 2-3. Suppose the ring is USHR.

The ring connects nodes A,B,C and D. The nodes are connected by two (geographically diverse) fibers. For each fiber, all the traffic is unidirectional: for instance the traffic on the inner fiber is counterclockwise, and the traffic on the outer ring is

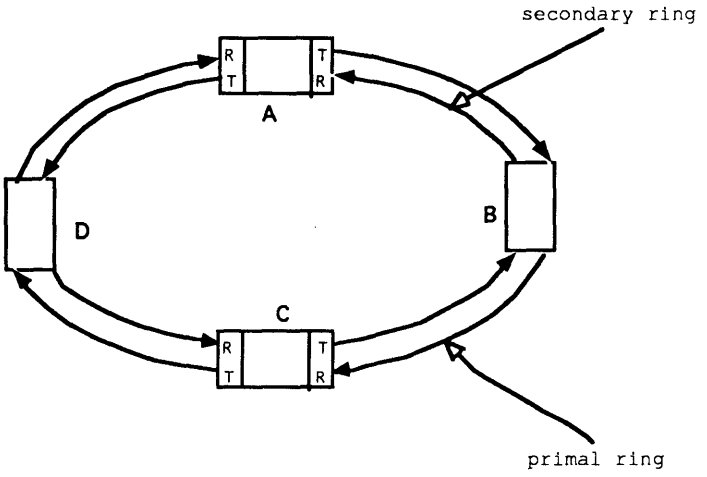


Figure 2-3: Self-Healing Ring

clockwise. Each direction is, confusingly, referred to as a ring. With all the links operational, the transmitter transmits on both rings, and all the receivers listen on the outer ring, called a primal ring.

Consider the traffic between the nodes A and C. The transmitter at A (TA) transmits towards D on the inner fiber and towards B on the outer fiber. TC transmits to B on the inner fiber and to D on the outer one. All the receivers are connected initially to the outer ring. RA receives all the messages from B, and RC receives from the direction of D. If any of the inner links fails, the service is not interrupted, the logical layer of the network detects the failure, and the failed link is restored immediately without any customer noticing the event. Suppose that the outer link (AB) has failed. RA will detect the deterioration of service and the ADM at node A will switch immediately to the inner ring. Thus, the service is restored in a very short time, crews will then be dispatched to repair the failed link. SHR architecture provides 100% survivability against single-link failures.

It is clear, that if a node fails, all the demands terminated at node are not recoverable. The self-healing mechanism in case of the node failure is similar to switching in case of a link failure. When a receiver detects a deterioration of the service it switches to another ring. Since for every demand pair there are two node disjoint paths, and since the node transmits in the both directions, all the demands that are

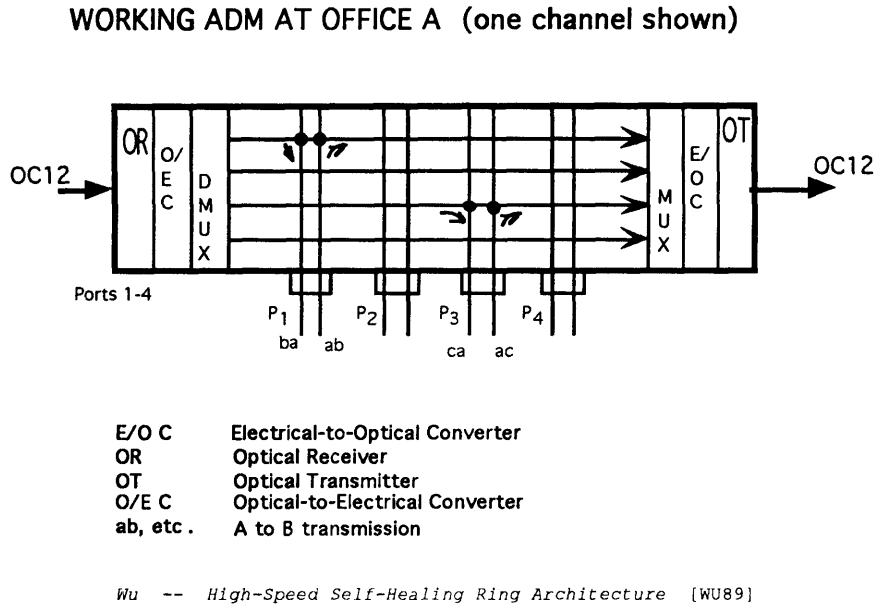


Figure 2-4: Add/Drop Multiplexor

not terminated at the failed node will survive.

Quick and reliable ring switching is made possible by special hardware called Add/Drop Multiplexor (ADM), installed at the nodes and terminating fibers for transmit and receive. Under normal conditions, each ADM contains two transmitters broadcasting into the primal and secondary rings and in case of a failure, the multiplexor switches to the alternate ring. An add/drop part of ADM is depicted at Figure 2-4.

Consider the ring shown in Figure 2-3. Suppose the demand pattern is as follows: demand d_1 is planned between the nodes A and B, and d_2 is planned between B and C. Then USHR routing scheme allocates capacity d_1 on the links $(AD), (DC)$ and (CB) for B to communicate to A, and capacity d_1 on the link (BA) for A to communicate to B. Capacity d_2 is allocated on (BC) for B to communicate to C and on the links $(CD), (DA)$ and (AB) for C to transmit to B. Thus each link of the ring is crossed by *all* the demands, because each node receives and transmits in opposite directions. Every link must have sufficient capacity for all the calls.

Since a node on the USHR ring transmits in both directions, every call on the ring

crosses *all* the links. The link capacity is used inefficiently. If all the calls are routed such that the number of calls crossing each link is minimized ("the shortest" path), the link capacities are utilized more efficiently. Four-fiber bidirectional line-switched ring (BSHR) permits more efficient utilization of capacity.

The idea is to allow bidirectional routing on the primary and secondary rings. Consider again the Figure 2-3. Let each ring contain two fibers. For BSHR, all the traffic will be routed on the "shortest" path for each demand. For our example, the traffic will be routed only on the links (AB) and (BC). The bandwidth allocated for calls is d_1 of each fiber on the link AB, and d_2 on the link BC. Consider the routing table below. As compared to the USHR ring, BSHR ring contains spare capacity on the links (CD) and (DA). This capacity can be used to install additional services.

Consider the operation of a 4-fiber BSHR. During the normal operation, there are two working fibers and two protection fibers. When a failure occurs, there exist two ways of traffic protection: short path span protection and long path ring protection. In the short path span protection, if one fiber of a link fails, two nodes adjacent to the failed link communicate and reroute the traffic on the protection fiber (see Figure 2-5). Long path ring protection occurs when both fibers of the span fail (catastrophic cable cut). The protection fiber along the alternative (long) path is used to replace the failed short span. This method utilizes the same approach as USHR switching, transfer the communication off the failed ring. The nodes adjacent to the failures control the ring switch.

A cheaper alternative to 4-fiber BSHR is 2-fiber BSHR. Each ring of the 2-BSHR consists of one fiber, similar to USHR, but the available bandwidth of each fiber is divided in two (one half for the working channel, another half for protection of another fiber). Thus, a logical "4-channel" is created. The operation of this ring is similar to 4-fiber BSHR, except that the short path protection is not available. 2-fiber BSHR uses less hardware than the corresponding 4-fiber ring, but the usable bandwidth is also decreased.

Table 2-1 Routing for the Example

Span	USHR	BSHR
AB	$2(d_1 + d_2)$	$2(d_1)$
BC	$2(d_1 + d_2)$	$2(d_2)$
CD	$2(d_1 + d_2)$	-
DA	$2(d_1 + d_2)$	-

Assume now that the bandwidth of all the links is $C = d_1 + d_2$ (i. e. the capacity is numerically equal to sum of the demands d_1 and d_2). Suppose we want to install a new service between the nodes A and B, requiring d_2 units. Since every link of the USHR ring has traffic $d_1 + d_2$, its capacity limit is reached and no new services can be installed on the ring using its current configuration. When no new services can be installed on a ring we say that **ring exhaust** has occurred.

Consider the BSHR ring. Link (AB) has d_2 units of unused bandwidth. Thus, additional traffic can be routed on the BSHR in its current configuration. Both cases show that using BSHR protection increases the ring's traffic-carrying capacity. In general, finding the "shortest" path routing on BSHR is a difficult problem, (refer to [SHU93] and [COS93]). Since BSHR uses twice more fiber than USHR and requires more equipment, BSHR rings are more expensive. Thus the tradeoff for the achieved efficiency is higher cost.

Each type of ring has its own advantages and disadvantages. The switching time on all of the rings is 50-100 milliseconds, with USHR being the fastest. This is an acceptable speed. The threshold in service restoration is 2 seconds, according to [WU93], because most existing circuit switching services will not see adverse impacts of an outage if the outage time is less then 2 seconds.

USHR is the cheapest, with 2-fiber BSHR being the next cheapest, and 4-BSHR being the most expensive. 4-BSHR presents the most flexibility in terms of using available capacity, with 2-BSHR being intermediate, and USHR having the least efficient utilization of bandwidth.

Due to its low cost, USHR seems to be the ring of choice for the telecommunication carriers in the short to medium-term future. Currently, there exist

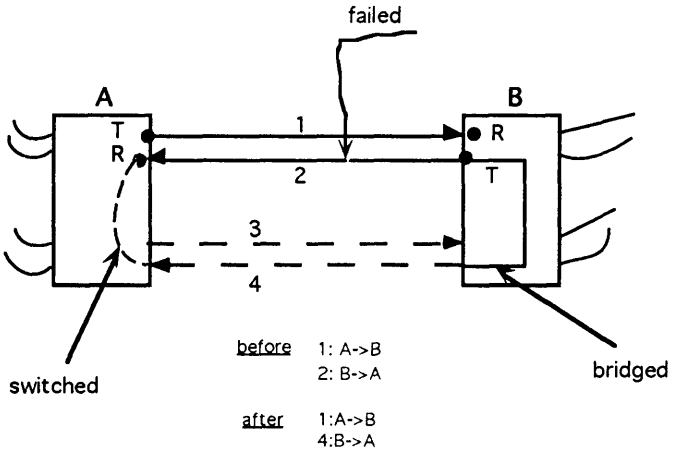


Figure 2-5: 4-fiber BSHR: Short Path Span Protection

45Mbit/s, 135Mbit/s, 405Mbit/s, 565Mbit/s, 1.2Gbit/s, and 2.2Gbit/s Self-Healing Rings (SHR); 2.5 Gb/s has been achieved. Potential threshold for USHR is 10 Gb/s. This is enough capacity for the existing applications. In the near future, traffic carrying capability may become more important and 4-fiber BSHR will play a more significant role in the network. 2-fiber BSHR should be designed in a such a way that it may be easily upgraded to a 4-fiber BSHR. Thus 2-BSHR could be used as an intermediate step on the way to building a 4-fiber BSHR.

The major problem with all the ring configurations is ring exhaust (defined in the beginning of this section). Ring capacity is limited by the fiber capacity and ADM processing capability. Once the capacity is exhausted, another ring must be added, or new ADM's will have to be installed. Ring exhaust becomes a problem when the ring has many nodes. For downtown inter-hub rings ring exhaust becomes an issue even with as few as 5 nodes on the ring [FLA90b]. Building a new ring may be more expensive than just adding a new path between the nodes to provide an alternate restoration path. To deal with exhaust, wave division multiplexing was suggested (see for example [CHL90]). This technology will have to overcome the limitations of optical signal processing.

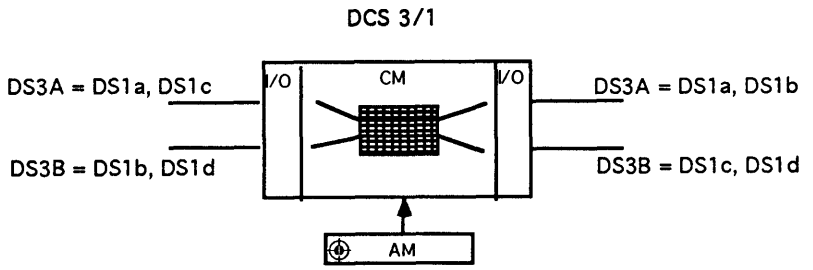


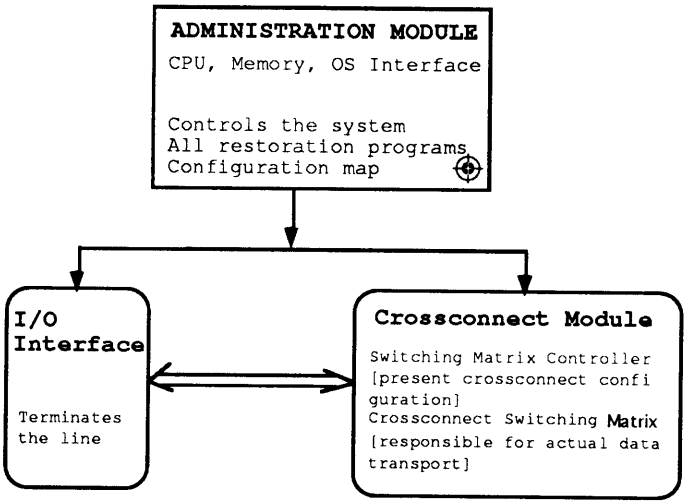
Figure 2-6: DCS: Schematic Depiction

2.3.3 Path Rearrangement Using DCS

Digital Crossconnect Systems were introduced commercially in 1981 to allow dynamic customer controlled reconfiguration of the customer's system. DCS is a computer that can multiplex, demultiplex and switch signals. A schematic diagram of DCS is shown in Figure 2-6. We will consider the operation of DCS on the example of DCS 3/1. The acronym stands for the fact that the Digital Crossconnect System has its input and output signals at DS3 level, and the crossconnection takes place at DS1 rate (see Appendix A for SONET signal rates).

In our example, we have two DS3 channels – DS3A and DS3B. The inputs are demultiplexed by the Input/Output Module (I/O) into DS1a, DS1b, DS1c, DS1d signals. The DS1 signals are switched by a Crossconnect Matrix (CM), controlled by an Administration Module (AM). Any switching configuration can be obtained by altering the information of AM. The DS1a and DS1b signals are switched to the output A, and the DS1c and DS1d signals are switched to B. The I/O module multiplexes the signal at DS3 rates. For a block-diagram of DCS see Figure 2-7.

In case of a link failure, the Digital Crossconnect System at the nodes initiates a search for a new path. Survivable network design using DCS has to assure that in case of a failure, there is enough spare capacity in the system to reroute the traffic off the failed link. For a mesh architecture, the time to restore service is as long as 5-15 minutes using today's centralized control of DCS's at the nodes. This restoration time is unsatisfactory for real-time applications. The time may be reduced using distributed control approach [WU93], and new generation Broadband DCS (B-DCS)



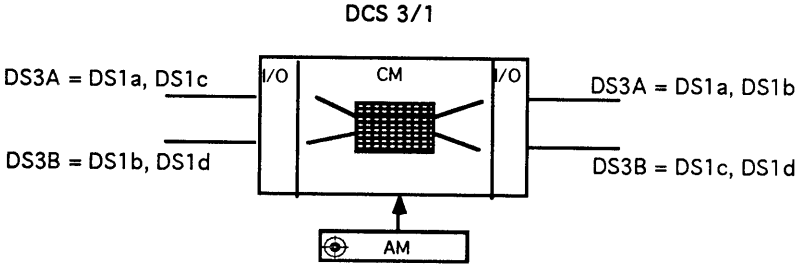


Figure 2-7: DCS: Block Diagram

devices designed specifically for dynamic path-rearrangement.

Distributed DCS restoration algorithm, shown in [WU93], is an example of a path rearrangement scheme.

Each DCS stores information that includes working and spare capacity associated with every link terminated at the system. When a failure is detected, one of the two ends of the affected path (Sender) broadcasts restoration messages to the adjacent nodes. The intermediate nodes (tandem nodes) update and rebroadcast restoration messages, until they reach the other end of the path (Chooser). When the message is received by the Chooser, it means that all the paths necessary for rerouting have been established. The Chooser sends an ACK message back to the Sender. After receiving the ACK signal, the Sender transmits a CONFIRM back to the Chooser via new routes. At this stage, all the tandem DCS's reconfigure their switch matrices. This procedure is repeated for each commodity crossing the failed link.

In the Bellcore study on restoration speed of DCS mesh networks (see [WU93]) Broadband SONET DCS were used to perform path restoration. The authors reported simulation results for a network with 15 nodes, hop count of 9 for every path, and 100% survivability for every link. The network was designed for 1682 working DS1 channels and 938 spare ones. The results show that even for the fast distributed algorithm, the restoration time is between 3.1 and 18.2 seconds, which exceeds the 2-second maximum permissible restoration time requirements.

Restoration time could be divided into two categories: time for computing a new map within AM, and the new map transfer (the physical path rearrangement). The simulation shows that it takes approximately 115 ms to reroute one DS1 demand, with 100 ms going towards the map transfer procedure (see the algorithm above). To decrease the cross-connect time, a parallel DCS is proposed. The study concludes that DCS self-healing schemes may not meet the 2-second restoration objective. It may be more effective to use DCS restoration together with other survivability methods in a so-called hybrid restoration approach. The authors propose that the rings be used for applications requiring fast restoration, and other applications be protected through DCS, thus overcoming ring exhaust and assuring fast restoration time for real-time

applications.

In this section two approaches to survivability were considered: self-healing rings and path rearrangement using Digital Crossconnect Systems. Both schemes, although technically feasible, have their drawbacks: rings are prone to ring exhaust when demand increases, and DCS's have unsatisfactory restoration time. We now assess the economic feasibility of restoration technologies in order to find which technology has lower cost.

2.4 Economic Analysis of Network Expansion Based on Survivable Architectures

2.4.1 Objectives of SONET Network Planning

In this section we consider SONET-based deployment of survivable architectures. The deployment process could be divided into three stages. At the initial stage, single SONET spans are deployed. During the intermediate stage, the fiber spans are connected into structures to maximize the economic and operational benefits of SONET. The final stage involves the evolution to all-SONET communications network. The initial and intermediate stages can be combined. In other words, from the very beginning, Plain Old Telephone Service (POTS) must be designed to be updated by coherent SONET structures, such as rings, rather than single fiber spans. Hence, from the planning point of view, the transformation to SONET network consists of two phases: installation of the appropriate SONET links and hardware, retaining the use of the existing cable links where possible, and, as the amount of services and customers increases, conversion to all-SONET network.

During the planning process, the planner is must consider the following factors:

1. The capital cost of deployment;

2. Impact of the new topology on the operations of the network, including maintenance and new services introduction;

3. Upgradability;

4. Survivability; and

5. Restoration time.

The **cost** of building a network is a major factor to consider in the decision process. **Operations cost** i.e. the ongoing cost of operating and maintaining the network, on the other hand, is found to be a secondary factor in selecting among survivable architectures [MAR93]. Marzec et al. provide two main reasons: SONET network is fiber-based, hence failures are less frequent (of course, the cost of each failure is higher). The most labor-intensive level, the DS1 level, requires far less maintenance events in SONET than in POTS.

Upgradability is defined as the network's ability to absorb new services and nodes at a minimum cost. Upgradability is very important for SONET networks. Many SONET applications are still under development. Upgradability varies depending on the architecture. For example, DCS networks can be upgraded with less cost than ring networks, since the rings cannot be flexibly expanded if the capacity has reached the limit. In the DCS case free capacity is allocated more efficiently, and if a new service is added, free capacity may already exist in the network, or protection for some less important applications may be dropped.

Survivability becomes an issue with the concentration of demands into a span. Survivability requires the protection of all the vital applications. The cost of survivability is reflected in terms of additional systems needed to protect the demands.

Restoration time calls for the shortest possible time of service interruption in case of a failure of an element. Depending on the application, tolerable restoration time could be from milliseconds, as for real-time applications, to seconds in case of data transfer.

The objective of SONET planning is to minimize the cost of expanding the network, while meeting the survivability, upgradability, and restoration time specifications.

2.4.2 Economic Performance of Network Topologies

This section analyzes the costs and performance of survivable topologies. The usual approach in the literature (see [MAR93], [WU88] and [FLA90b]) is as follows. For a given set of nodes and a list of demands, a network of certain topology is generated. Performance measures, such as cost, cost-to-survivability ratio and others are used to compare performance of different topologies for given test data. The researchers base their conclusions on the sample problems .

By examining a number of independent studies and comparing the results we can make some conclusions regarding the performance of the different architectures under consideration.

In the study by [MAR93] 15-node interoffice network was modeled. A research software program, Strategic Solutions, was used to generate a near-minimum cost network. The authors analyzed the following architectures (see Figure 2-8).

1:1 – every path is protected one-for-one, no diverse routing (two links occupy the same conduit).

1:1/DR — one-for-one protection with diverse routing.

USHR/SH – unidirectional self-healing ring with single homing.

USHR/DH – dual-homed ring; if one hub fails, the ring is not cut off and can communicate through another hub.

BSHR4/SH – four-fiber bidirectional self-healing ring with single homing.

BSHR4/DH —

DCS-mesh – DCS-based path rearrangement.

DCS/USHR/SH – a USHR single-homed architecture with DCS instead of ADM at the hub nodes.

DCS/BSHR4/DH —

The 2-fiber BSHR architecture is omitted, since it occupies the intermediate position between USHR and 4-BSHR.

We plotted the results of the experiments with scatter plots Figure 2-9 and 2-10. Two measures of performance are Expected Loss of Traffic (ELT) and Worst Case Survivability (WCS).

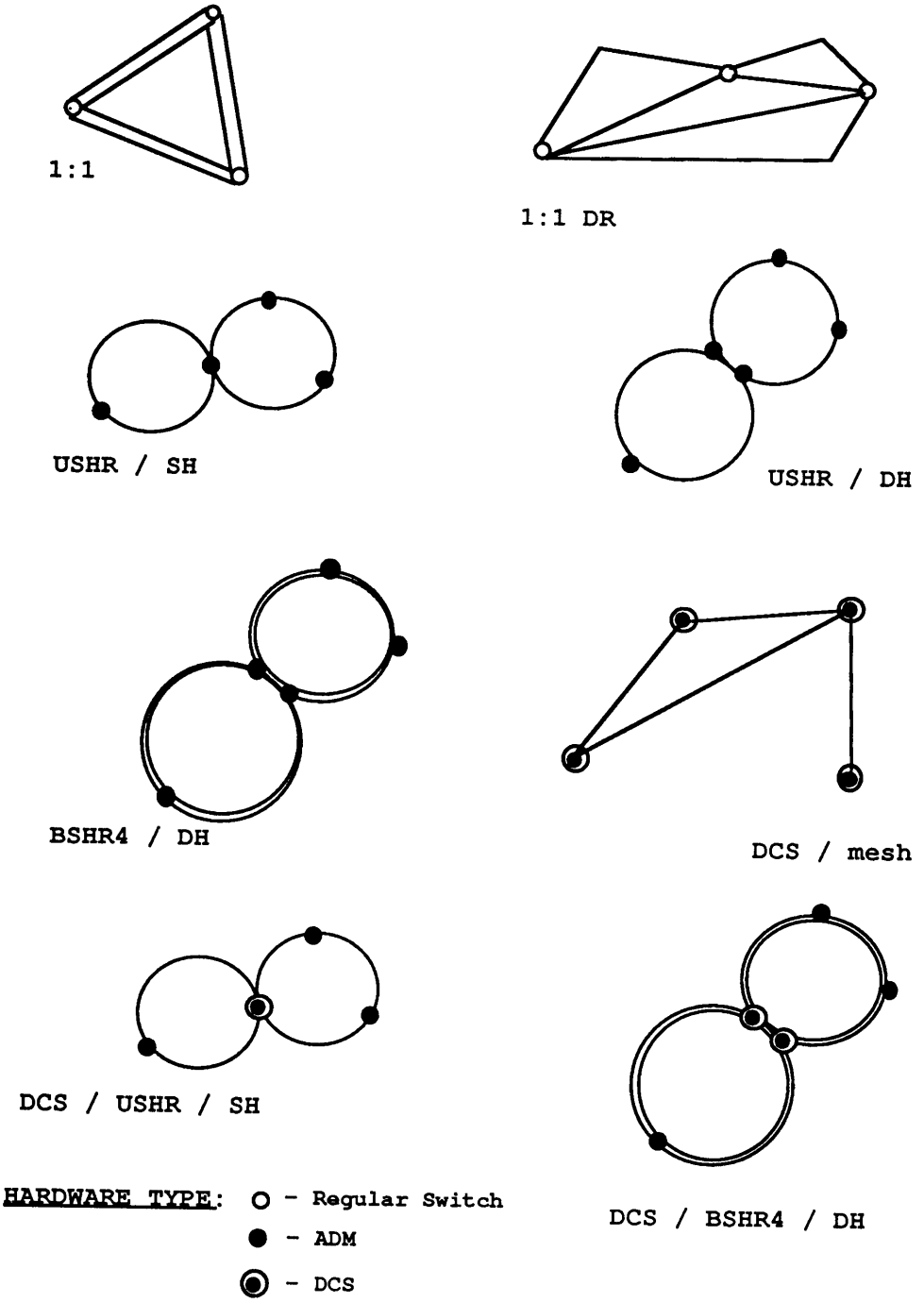


Figure 2-8: Methods of Protection

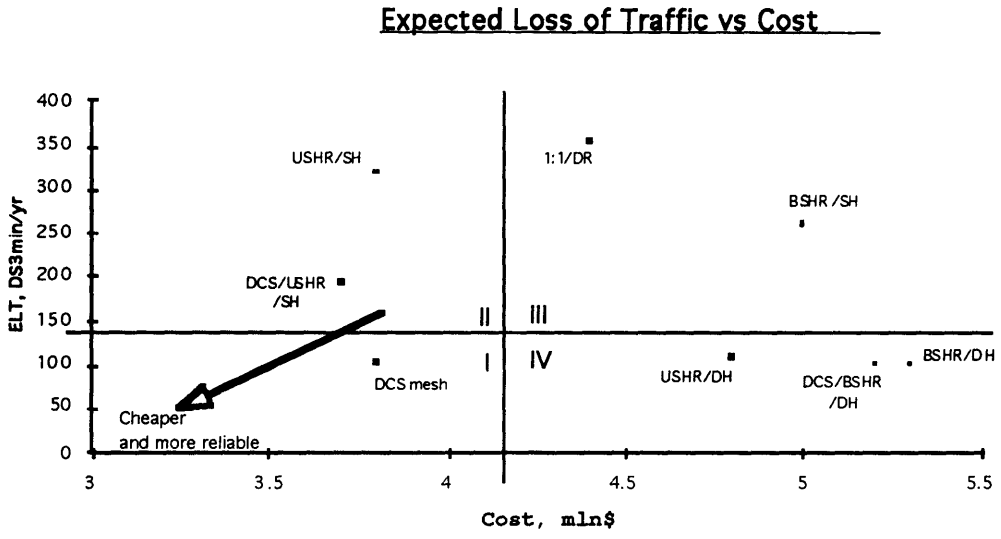


Figure 2-9: Expected Loss of Traffic vs Cost

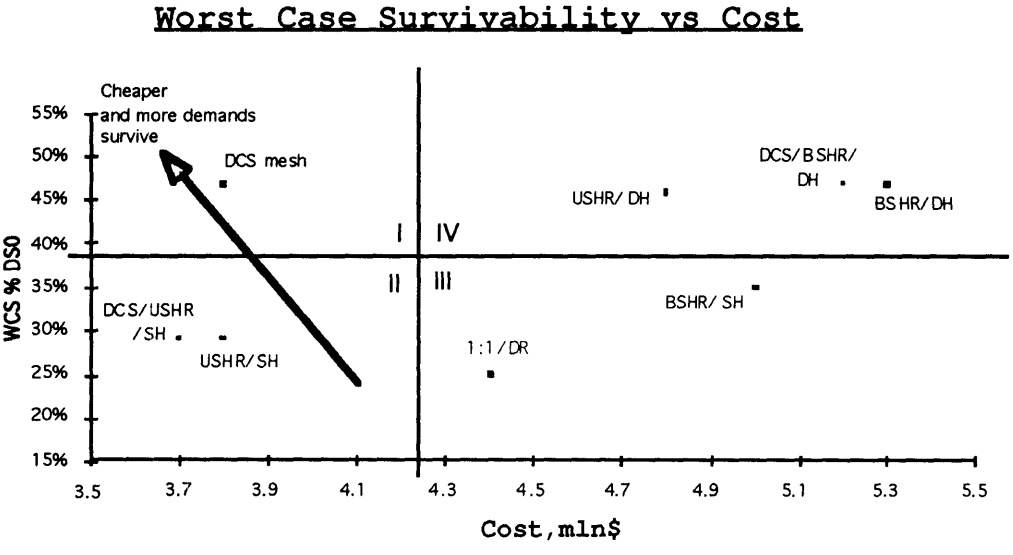


Figure 2-10: Worst Case Survivability vs Cost

Expected Loss of Traffic is the total amount of traffic that a network is expected to lose per year on the average due to failures. In this case a failure includes a breakdown of more than two links simultaneously. The ELT chart is divided into four quartiles. Quartile **I** is the cheapest and the most reliable. Quartile **II** is cheap, but less reliable. Quartile **III** is less reliable and more expensive, and quartile **IV** is reliable, but more expensive. DCS mesh architecture has the least cost and the least expected loss of traffic. Single-homed rings, while also inexpensive, may lose more traffic than the DCS architecture. The ELT figure for rings can be improved using dual homing, but the cost of the architecture goes up. USHR rings are cheaper than 4-fiber BSHR rings. An interesting observation is that placing DCS's in the hub nodes reduces the cost of the dual-homed ring architecture. This fact seems to be counter-intuitive since DCS are more expensive than ADMs. We discuss this phenomena in Section 2.4.3. Clearly, conventional 1:1 architecture would not be the architecture of choice for a network planner.

Worst Case Survivability is the percentage of circuits surviving the worst possible single-link or single-node failure. The WCS plot is depicted on Figure 2-10. Notice that there is an upper bound to the WCS figure, meaning that for any architecture and demand pattern $WCS < 100\%$. One of the "natural" upper bounds on WCS could be the maximum number of demands terminated at a single node. When this node is out of service, all the corresponding calls will be lost no matter how good the protection is.

As in the previous case, we divide the chart into four quartiles. Quartile **I** represents the most cost-efficient architecture with the highest proportion of demands surviving in case of a single failure. The second quartile has inexpensive architectures, but more demands are lost in case of the failure. The third quartile has more expensive architecture and poor WCS performance (these configurations are clearly not desirable). The fourth quartile has more expensive architectures, but good WCS performance. Here again the DCS mesh architecture is a clear winner. The inexpensive single homed rings do not guarantee the survival of more than 30% of demands in case of the failure. Dual homing improves the rings WCS performance considerably,

while increasing the cost. Traditional 1 : 1 architecture, again, does not perform well.

An important conclusion is that the SONET architectures can enhance survivability cost-effectively. Protecting against single-link failures provides the greatest improvement in ELT. As a result of the study, the DCS mesh architecture emerges as the most attractive option both for its low cost and good performance. The dual-homed ring architectures deliver adequate performance, but at a cost higher than the DCS. The link protection brings the most improvement to survivability. The node protection decreases the loss further, but it is secondary in the order of magnitude compared with the link protection. The latest developments show that the ring architecture is currently used more widely then the DCS path rearrangement in a SONET network. The main reason is the slow restoration time of the current DCS schemes as mentioned in Section 2.3.3. Digital Crossconnect Systems are used chiefly at the hub nodes.

2.4.3 Ring Architecture Selection and Interconnection

As discussed in the previous section, the ring architecture is currently the most effective SONET topology. While designing a ring interconnected topology two questions arise: how we select a suitable ring technology, and how are the separate rings interconnected.

We have previously described three most common types of ring architectures. We must choose between the lower cost USHR rings and more capacity efficient 4-BSHR ring (or , possibly, 2-BSHR upgradable ring).

The **type of ring architecture** that is most appropriate for a particular context depends on the demand pattern. Consider two possible expansion situations. A hub and some or all of the central offices from the cluster are assigned to the same ring. This kind of a ring is called an access ring since the central offices access the network via the hub. The routing of the new ring has a centralized pattern, with most of the traffic routed towards or from the hub. Suppose now that the nodes that grouped on the ring are different hubs. The demand in this case will be distributed almost uniformly between every pair of nodes.

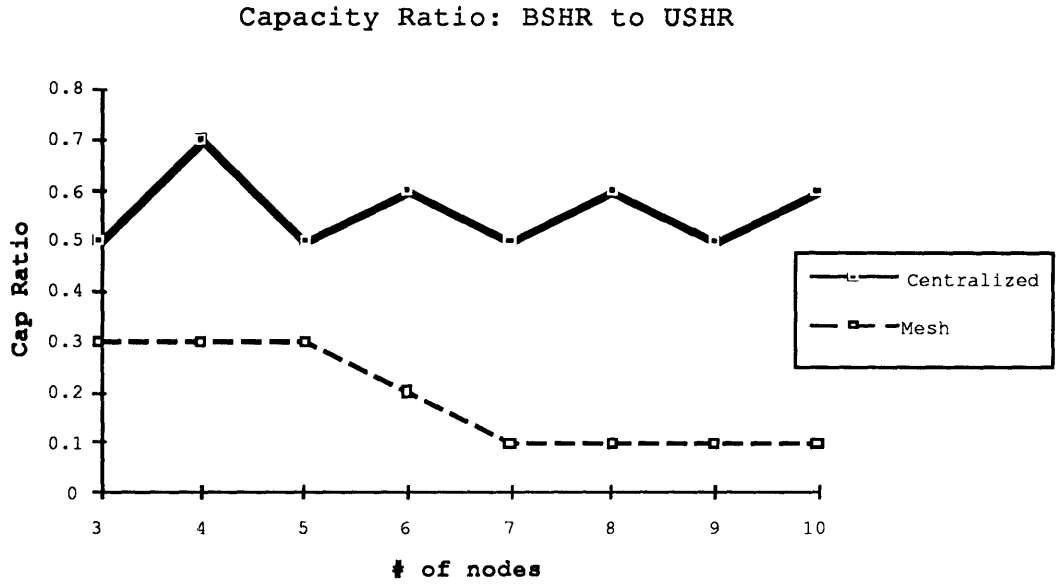


Figure 2-11: BSHR to USHR Capacity Ratio

A study was undertaken by [WU90] to identify the most efficient ring technologies for the above demand patterns. The number of nodes was varied from three to ten. The capacities and the costs of the architectures were measured, and the ratios were calculated.

The results suggest that for the centralized demand pattern, USHR architecture is cheaper than BSHR, even though the latter uses the bandwidth more efficiently. The study for the mesh demand shows that for the number of nodes greater or equal than five, BSHR topology is preferable. We present the results on Figures 2-11 and 2-12.

The data imply that the ring architecture depends on the nature of the demand pattern between the nodes grouped on a ring, and the network size.

For the centralized demand pattern, the USHR ring preferable. Because most of the demands are terminated at the same node (hub) the possibility of reusing the same bandwidth by different demand pairs is reduced, and so the advantage of BSHR rings over USHR rings in bandwidth reuse in reduced. Since BSHR is known to use roughly twice more equipment than similar USHR, the fact that the bandwidth can

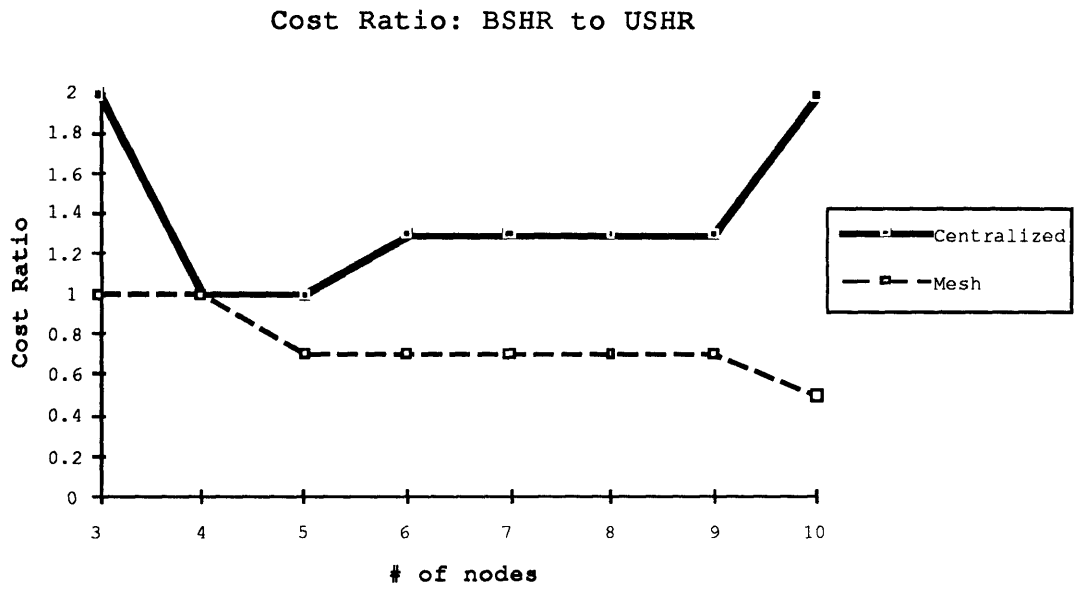


Figure 2-12: BSHR to USHR Cost Ratio

not be efficiently reused leads to the conclusion that the cost of BSHR will be at least as high as the cost of USHR in case of the centralized pattern.

For the mesh demand, there is a possibility to direct all the traffic on the short path between the nodes. Since cutting a link (a node) will probably not redirect *all* the traffic along the long path, the resulting capacity requirements for BSHR is lower than the equivalent capacity of USHR, therefore reducing the necessary capacity and thus the cost.

Consider the second issue of interconnecting the rings after choosing an appropriate technology. Because the ring has finite capacity, networks contain many rings. Those rings can be interconnected through single or dual homing. As noted previously, dual homing is the more attractive alternative because of the improved worst case performance of the network. Figure 2-13 depicts a typical ring interconnection using ADM's with dual homing. Because the ADM has limited switching capabilities, each ring requires an ADM at the hub node. As the number of rings in the interconnection grows, the complexity and cost of the system increases without additional benefits: since there is no flexibility in grooming the demands, the STS channels cor-

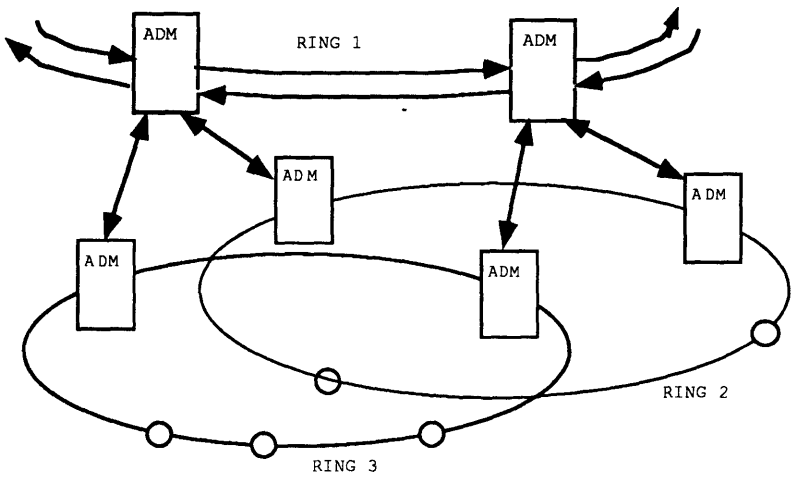
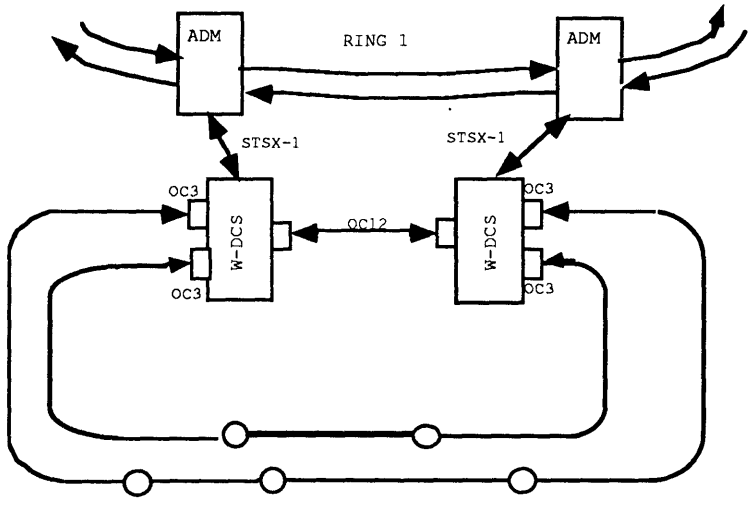


Figure 2-13: Ring Interconnection Using ADM



From [WIL93]

Figure 2-14: Ring Interconnection Using DCS

responding to connections of the ADM's are not utilized more efficiently. Figure 2-14 shows a better approach to ring interconnection. DCS's are used instead of ADM's at the hub. This method is especially useful when a number of access rings are interconnected with a transport ring at the same hub nodes. Because the total traffic of the access rings is usually smaller than the transport capacity, a limited number of DCS is enough to interconnect the traffic. The utilization of such a system will be very high and thus it is more cost effective than using a large number of ADM's. Note that the DCS systems in this configuration are used for grooming only, and not for switching, therefore, inadequate DCS switching time is not an issue.

Summary of Ring and DCS Mesh Architectures

Rings and DCS mesh are two complimentary architectures. The main advantage of the ring is its fast restoration time. Upgradability is a problem because of the ring exhaust and therefore in the high traffic areas such rings can accommodate limited number of nodes. When we must add a new node, a new ring has to be created and interconnected with the old one. Therefore, the areas with high demand such as downtowns of big cities favor the DCS mesh architecture. DCS's are easily upgradable to accommodate new nodes. At downtown areas DCS utilization will be high. The main problem with DCS architecture is slow restoration time.

Figure 2-15 summarizes the advantages and shortcomings of two architectures.

	Path Rearrangement	Interconnected Rings
Total Cost	COMPARABLE for the nodes with DS3-size traffic More expensive for lower density traffic	Less expensive
Hardware **Hardware cost**	B-DCS more expensive	ADM cheaper
Survivability	flexible, up to 100%	100%
Restoration time	currently 5-15 min. ; distributed algorithm simulation: up to 20s.	50 - 100 ms.
Bandwidth use	efficient	USHR - not efficient; BSHR - better, but less efficient than DCS
Upgradability	easy	hard: ring exhaust

Figure 2-15: SONET Architecture Comparison

Chapter 3

Problem Description and Formulation

3.1 SONET Problem Hierarchy

Telecommunications networks require large investment. As [NYT93] shows, the scale of investment is in the range of hundreds of millions of dollars. It is important that decisions made by a network planner at every stage from design to operations support are cost-effective. Even as little as 1% in savings may justify significant work to optimize the network topology. Since the generic SONET network problem is complex, it is important to present a structured view of the issues.

In this section we develop a hierarchical view of the SONET network planning problem. Consider the planning hierarchy shown in Figure 3-1. The generic problem is divided into topology, routing, and operations support. Integrating all three subproblems creates an effective network expansion and management plan.

Consider the **topology selection** problem. For a given demand matrix and hardware costs, select a minimum cost survivable topology. Usually topology selection and link capacities allocation are determined at the same time. Survivable architectures currently under consideration are the network of rings and interconnected DCS's. If the ring architecture is selected, the composition of the separate rings must be determined. For each group of nodes, we must select an appropriate ring architecture, and

designate hub nodes. If we find the DCS architecture more preferable, the hardware for each node must be chosen, and the nodes have to be interconnected via fiber and existing coaxial cables in an efficient way.

Routing problems are considered for a given topology. Initial routing considers the routing of the calls to assure survivability requirements in the case of DCS, and efficient bandwidth use for the rings. Real-time rerouting involves the rerouting of demands in case of a link (node) failure. Routing is traditionally an engineering, rather than operations research problem. Routing approaches use such well known methods as the "hot-potato" method, or some more sophisticated techniques, such as dynamic bandwidth allocation. The problem of optimal routing is important. For instance, for BSHR rings the ring capacity depends on the routing, i. e. the capacity can be minimized by selecting appropriate routing schemes.

Real-time rerouting algorithms can be considered as an operations support activity. Generally an issue is an operations problem if the demands, the network structure, capacity allocation and initial routing are given. Another example of an operations issue is critical link analysis. For a given routing pattern, it is important to know which group of links leads to irretrievable loss of traffic. The critical link problem can be thought of as the network sensitivity analysis.

Consider the problem of **network expansion planning**. For a given topology, we consider the following problem: can we increase certain demands of the network without adding any additional capacity? Can we add new nodes to the network without building new rings (DCS paths)? If we have to build new ring , what is the most cost-effective network expansion? Models created for topology design can be used to answer some of the questions. For example, [DAL92] uses topology design model to perform the sensitivity analysis.

As our discussion suggests, all the subproblems are complementary and compatible. In Figure 3-1, the vertical direction indicates compatibility of subproblems, and horizontal layers suggest complementarity. Consider, for example, the design problems associated with ring architectures. Any subproblem, or a set of subproblems, can be solved separately. The lower the subproblem is in the hierarchy, the easier

it is to solve it to optimality. But local optimum obtained in such a way could be significantly off the optimal solution within the context of the overall SONET design problem. Therefore, the dilemma is that solving the Survivable SONET Network problem with one mathematical model is computationally intensive, whereas solving special subproblems, though locally optimal, may not produce a good solution for the network as a whole.

A lot of interesting work has been done in the areas of the SONET network design. [WU91] considers the topological design problem and attempts to select the best *mix* of SONET survivable architectures – both the rings and point-to-point connections/DCS. The objective is to produce a sound SONET network expansion strategy. A rule is formulated first with respect to the selection of nodes to be point-to-point connected. After an architecture is selected for all the given demand pairs, heuristics are applied to build the topology.

In [MON88] and [GRT92], the authors develop a cutting plane approach to design a communications network such that for every pair of nodes there exists a given number of link-disjoint or node-disjoint paths. The method can be applied to designing survivable networks using crossconnect systems.

[DAL92] develops a model for a survivable network design using DCS systems. The authors implement the model using the MULTISUN software. Valid inequalities for the problem are considered and the cutting plane approach applied. Dahl and Stor also apply the MULTISUN model to solve operational problems such as a flow feasibility problem.

Results of the above articles are implemented as software systems. We do not have at this time any information on actual application of the programs to design a real-life SONET communications network.

[MAG91] develops a model for survivable network design using rings. He suggests a methods of solution based on decomposition. Magee also considers optimal BSHR routing and develops valid inequalities for the number of ADM's used. [COS93] formulates the BSHR optimal capacity allocation problem as a mathematical model and uses a dual ascent approach to route the demands efficiently on a BSHR ring.

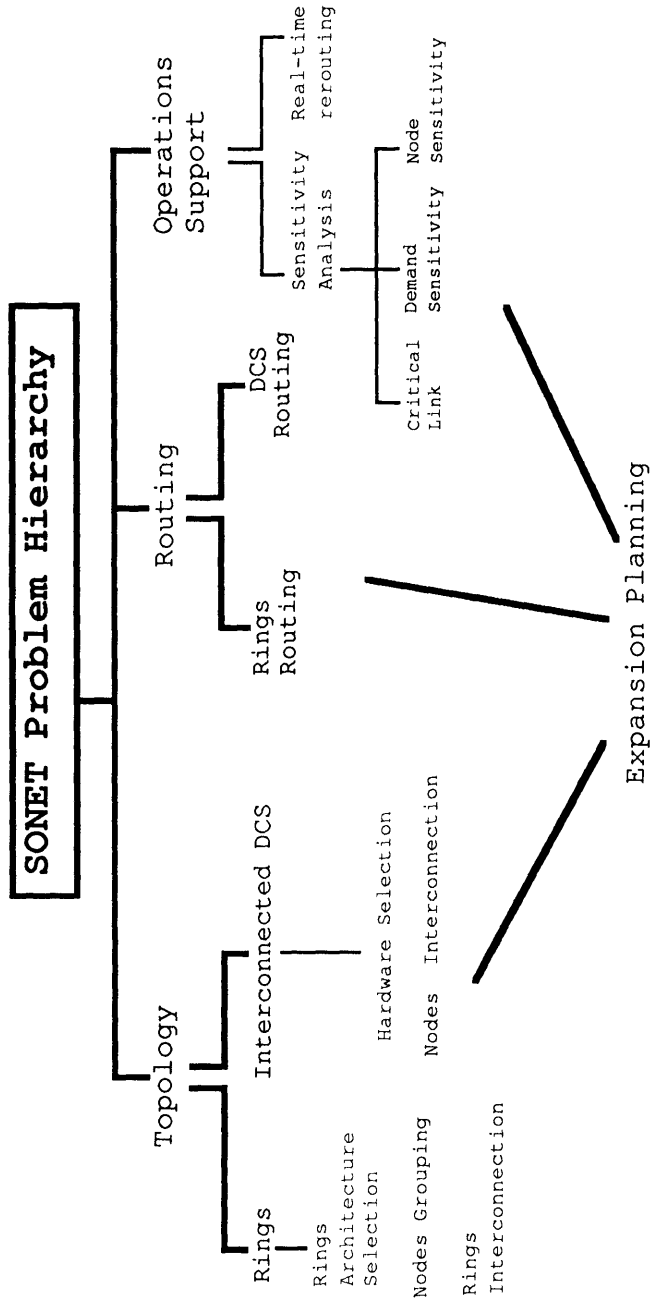


Figure 3-1: SONET Problem Hierarchy

[SHU93] discusses some properties of a feasible routing for a BSHR ring. Shulman shows that a feasible flow exists in the ring if and only if the sum of the demands across every cut is less or equal than the capacity of the cut. He uses this result to develop heuristics for optimal routing problem. [LAG93] considers a problem of designing a USHR ring interconnected network. The author relaxes integrality of the flow variables, formulates a mixed integer optimization problem, and uses tabu search to generate a good solution.

In the next section we consider one of the SONET topology design problems – the Interconnected Ring Network problem.

3.2 Interconnected Ring Network Problem Formulation

3.2.1 Model and Assumptions

The Interconnected Ring Network Design Problem (IRD) is stated as follows:

[IRD]: Given a set of nodes representing telecommunications offices, internode demands and available ADM capacities and costs, find a minimum cost grouping of the nodes into rings, such that every node belongs to at least one ring. Every interconnection must be dual-homed. In case of interring traffic an interconnection cost is incurred.

Assumption 1 *If two nodes belonging to two different rings require an establishment of a link, the communications are established via* **interconnection**.

For the model a **ring** is defined as a set, to which the nodes are assigned. In real life the rings are physical topologies. For a ring r define **intraring** traffic to be all the calls that are routed within the ring. Correspondingly, **interring traffic** for the ring r equals all the calls that are initiated on r and terminated on another ring (or initiated on another ring and terminated on r).

A generic Interconnected Ring Network Design model is described below.

Minimize

the cost of ADM systems at the nodes plus the cost of interconnected traffic subject to the following constraints:

assign: Each node is assigned to at least one ring.

capacity: For each ring, the node ADM system must have enough capacity to handle all the corresponding interring and intraring traffic on the ring.

balance: All the node-to-node demands are satisfied.

For the model we define two groups of variables: **ADM placement** variables $x_{ir} = 1$ if a node i belongs to a ring r, and 0 otherwise. The ADM placement variables can be considered as variables, assigning nodes to the sets (rings). A node may belong to more than one ring. This variable is defined for all possible node-ring pairs.

Demand routing variable y_{ijrs} equals the fraction of a demand d_{ij} that originates on ring r and terminates on ring s. In this work we consider circuit switching and thus origin and destination nodes are indistinguishable, i. e. $y_{ijrs} = y_{jisr}$. The variable y_{ijrs} is between 0 and 1. It is defined for all the non-zero demands and all possible node-ring assignments such that $j > i$. y_{ijrs} is nonzero if for the demand d_{ij} the node $i \in r$ and the node $j \in s$ (there is no need to define y_{ijrs} for all j: $y_{ijrs} = y_{jisr}$). Note that for $r = s$ the variable y_{ijrr} shows the fraction of demand d_{ij} confined within the ring r.

There are two categories of assumptions we make in this model: technical assumptions and modeling assumptions. Technical assumptions are those related to the technology used. The technical assumptions of the model are the symmetry of the nodal demands, circuit switching, bifurcated routing and dual homing of the rings. The modeling assumptions include deterministic demands, negligible fiber costs, USHR ring technology, no transhipment rings and the assumptions about ADM and interconnection cost structure. We explain these assumptions next.

Each demand is **deterministic**. We assume that the demands are supplied by a network planner. Demand matrix are usually consists of peak demands.

Each demand is **symmetric**: $d_{ij} = d_{ji}$ for every node i, j. In other words if the node i uses d_{ij} circuits to talk to j, the node j uses the same capacity to reply. Every circuit switching communications in duplex mode such as voice communications satisfy this assumption.

If two rings are interconnected, they have two nodes in common. **Dual homing** is shown to deliver the best performance results for the ring architecture (see chapter 2).

For the formulation of the problem, we relax bifurcated routing by allowing **non-integer splitting** of the demands. For bifurcated routing the demands can only be split into integer parts. The non-integer splitting assumption will underestimate the required capacity, since some of the fractional demands in the optimal solution will have to be rounded up. Our heuristics retain the bifurcated routing requirements, splitting the demands only in integral parts.

All the fiber links are available at negligible cost. We assume that fiber conduits are available at no cost. Variable fiber cost is incorporated in the interconnection cost. The assumption may hold for dense downtown networks, or any other networks for which the nodes are located close enough such that the cost of placing a fiber link is negligible compared to the cost of the hardware to be installed. The fiber costs are typically much smaller than the other expenditures ([SCOM93]).

Only USHR rings are used. This is the cheapest currently available ring technology. The bandwidth allocation of the USHR ring is independent of the routing, which greatly simplifies the ring capacity calculation. On the other hand, the capacity of BSHR depends on the routing algorithm, which makes the network design model intractable. The Interconnected Ring Network Design model [**IRD**] with USHR could later be used to approximate BSHR-ring network; a more accurate model would incorporate a BSHR ring capacity submodel in the problem formulation.

For a USHR ring, the required capacity of an ADM equals the sum of all the demands crossing the ring. It includes the intraring and the interring demands assigned to the ring. The necessary condition for the demand y_{ijrr} to be assigned to the ring r is that both variables x_{ir} and x_{jr} equal to 1. A necessary condition for a demand y_{ijrs}

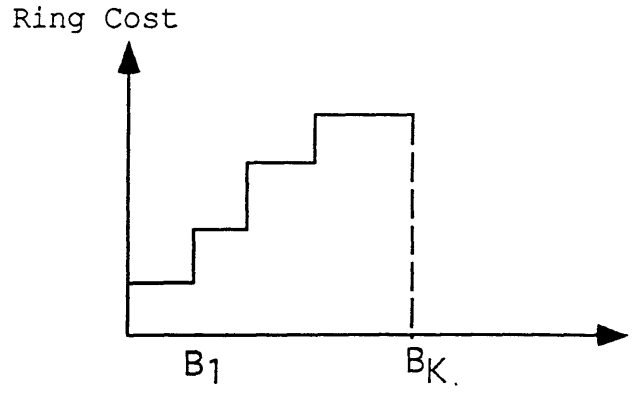


Figure 3-2: ADM Cost

to be interconnected is $x_{ir} = 1, x_{js} = 1$. Note that those conditions are *necessary*, but not sufficient. The balance constraints assure that the total number of calls routed on different rings for the pair (i, j) equals the demand d_{ij}. Thus, if c_r denotes the capacity of a ring r:

$$C_r = \sum_{\substack{(ij):j>i \\ i,j \in r}} y_{ijrr} d_{ij} + \sum_{\substack{j>i \\ r \neq s}} (y_{ijrs} + y_{jirs}) d_{ij} \tag{3.1}$$

where $y_{ijrs} \in [0, 1]$. The last sum of the equation consists of two terms y_{ijrs} and y_{jirs} because we sum only over $j > i$.

If some interring traffic is allocated to the ring pair r, s these two rings are **directly interconnected**. The consequence of this assumption is that there is no transhipment traffic in the network, i. e. for a given ring, all the calls either originate and terminate within the same ring, or they originate(terminate) in the ring and terminate(originate) in the adjacent one.

The costs require a special consideration. After carefully considering the underlying ring and ADM technology, we adopt the cost structure used by [LAG93]. We use a stepwise cost function for the fixed ADM cost (see Figure 3-2). The interconnection cost is variable, depending on the amount of the traffic passing through the interconnection, and the fiber costs are incorporated in the interconnection cost.

The cost of an ADM system is capacity-dependent. For instance, if the ADM is used in its standard configuration, its maximum capacity is B_0. To increase the

system's capacity, additional cards must be added and, perhaps, an enhanced version of control software must be installed. Therefore, the cost of the system of the capacity greater then B_0, but less than B_1, increases correspondingly. For every capacity between B_0 and B_1 the cost does not vary, and therefore it is taken to be a fixed cost for this interval. There is also a variable ADM cost, associated with a number of channels being added/dropped at the node. This cost varies only slightly within the range of the node demands (see [LAG93]), and therefore we ignore it.

An example of the costs involved in a typical ADM system is taken from [SCOM93]. The data refer to OC3 ADM, with the maximum capacity of 84 DS1 circuits.

Table 3-1 ADM Cost Example

Unit	Cost, $
Control Equipment	1,900
Synchronization-Timing	150
Terminal Cost (if used as terminal multiplexor)	1,125
Add/Drop Cost	900
Other Fixed Cost	1,500
4 DS1 card cost	1,500

All the costs given in the table are fixed costs. An ADM system capacity is assembled using DS1 cards. For example for an ADM system capable of handling 48 DS1 units and serving as a Terminal Multiplexor, 12 DS1 cards are required and the cost of $ 23,575 is incurred. We might remark that the costs are constantly changing due to market fluctuations and introduction of new technology.

Ring interconnection cost is modeled as a variable cost. Recall that the interconnection nodes have either multiple ADM's or a DCS system acting as a switch between the different rings. The variable cost models the ring passthrough capability, additional control software and hardware required by the interring traffic and variable costs of fiber. Since the fiber cost is negligible, we do not model the dual homing aspect explicitly. The capacity of the second interconnection equals the capacity of

the first one, and any node can be chosen as the second homing node. For a one-node ring the ADM system in the node must be duplicated. The cost of dual homing is twice the corresponding interconnection cost.

Our approach to modeling the problem is to emphasize grouping of *demands* d_{ij}, rather than the assignment of nodes to the rings. If demand d_{ij} is assigned ring r, then both nodes i and j belong to the ring. On the other hand, the fact that both nodes i and j are on the ring r, does not guarantee that the corresponding demand d_{ij} is on the same ring.

For each ring r assign a capacity B_r from the set of existing ADM sizes. For each ADM size B_i we create a sufficient number of rings to incorporate the existing demands. In the Appendix B we develop a better bound on the maximum number of rings in the optimal solution. If there are K available ADM sizes and R rings are available for each ADM size, we have KR constraints.

We formulate the **[IRD]** problem as follows.

[IRD]:

$$\text{Minimize } Z = \sum_{r}\sum_{i} c_r x_{ir} + \sum_{\substack{r,s \\ r\neq s}}\sum_{\substack{i,j \\ j>i}} a_{ij} y_{ijrs} d_{ij} \tag{3.2}$$

subject to

$$\sum_{r} x_{ir} \geq 1 \text{ for all nodes } i = 1 \ldots N \tag{3.3}$$

$$\sum_{i}\sum_{j>i} y_{ijrr} d_{ij} + \sum_{s,s\neq r}\sum_{j>i} y_{ijrs} d_{ij} + \sum_{s,s\neq r}\sum_{j>i} y_{ijsr} d_{ij} \leq B_r \text{ for } r = 1 \ldots KR \tag{3.4}$$

$$\sum_{r}\sum_{s} y_{ijrs} = 1 \text{ for all i,j such that } d_{ij} > 0 \text{ and } j > i \tag{3.5}$$

$$y_{ijrs} \leq x_{ir} \text{ for all } r, s \; j > i \tag{3.6}$$

$$y_{ijrs} \leq x_{is} \text{ for all } r, s \; j > i \tag{3.7}$$

$$x_{ir} \quad \text{are 0-1 variables} \tag{3.8}$$

$$y_{ijrs} \in [0,1] \text{ for all r, s, } j > i \tag{3.9}$$

This model is different from the one proposed in [MAG91] in two key aspects. Magee takes a more general approach for his full model, considering interconnection

capacities (different from ring capacities) and existence of transhipment rings (traffic passes through without being terminated or originated on the ring). The author designates a ring to be a special node in the network and an arc from the ring-node to an office-node signifies inclusion of the node on the ring. This approach gives a good understanding of the relation between the nodes and the rings, but yields a bulky and intractable model. The author proceeds to add simplifying assumptions to make the model more tractable. He considers two subproblems: a ring network design without interconnections and a single ADM size problem. He than proposes a method to solving the problem without interconnections by using decomposition approach. Our model uses a number of simplifying assumptions that make it a good approximation of reality and at the same time yield a more tractable model.

The ideas underlying the model described [LAG93] are similar to our model. The author considers additional constraints on the number of nodes on a ring. The major differences between our model and the model described in [LAG93] are in approaching the problem and the method of solution. We consider **[IRD]** as a subset of a more generic SONET network design problem. Thus our approach is not just to find a good solution to the Ring Interconnected Network, but to consider the whole set of operational and design problems arising in context with **[IRD]**. Laguna proposes to use Tabu Search as a heuristical method for solving **[IRD]**. We develop heuristics using various approaches that exploit the special structure of the model and the telecommunications context.

3.2.2 Description of Costs and Constraints

The fixed ADM cost c_r depends on the size of the corresponding ring. We assume that ADM costs exhibit economies of scale, i.e. if $B_i < kB_j$, then $c_i < kc_j$ and if $B_j < B_i$, $c_j < c_i$.

The variable cost of the interconnection is $a_{ij}/2$. This cost is a sum of ring interconnection cost that is the same for every ring interconnection, and a unit length fiber cost that depends on the nodes being interconnected. Thus the total variable cost depends in general on the nodes being interconnected. The unit fiber cost is

secondary in the order of magnitude to the ring interconnection cost, so as a matter of approximation for our heuristics we use a constant interconnection cost $a/2$. Our heuristics can also be applied in the case when the interconnection cost varies.

To reflect the dual homing, we double the interconnection cost. No additional constraints or costs are necessary, since by our previous assumption any ring can be interconnected with any other ring directly. Dual homing will not influence the ring capacity. For each homing node the corresponding ADM has to be able to handle all the ring traffic. Therefore, for our model the dual homing factor can be accounted by doubling the original interconnection cost, and the interconnection cost for the dual-homed rings is a_{ij}.

From our cost and ADM capacity assumptions, if it is profitable to have a fraction of a demand d_{ij} on a ring and if the ring capacity allows to incorporate the rest of the demand, then it is profitable to have *all* the demand d_{ij} on the ring. If there is not enough capacity, the traffic splitting will occur. See more on traffic splitting in Chapter 5, Section 5.1.

The model's constraints are of two kinds: generic constraints (3.3), (3.4) and (3.5), and support constraints (3.6) and (3.7).

The constraints (3.3) require each node to belong to at least one ring.

The capacity constraint (3.4) requires the sum of traffic to be less than ADM size. As mentioned in the Section 3.2.1, the number of the capacity constraints is KR.

Another approach is to have R capacity constraints, one for every ring, but set the right hand side to be equal the sum of all available capacities multiplied by capacity selection variable, such that no more than one ADM size is selected for every ring:

$$\sum_i \sum_{j>i} y_{ijrr} d_{ij} + \sum_{s \neq r} \sum_{j>i} y_{ijrs} d_{ij} + \sum_{s \neq r} \sum_{j>i} y_{ijsr} d_{ij} \leq \sum_k z_{rk} B_k \tag{3.10}$$

$$\sum_k z_{rk} \leq 1 \tag{3.11}$$

where z_{rk} is 1 if the ring r has a capacity B_k.

The cost c_r has to be changed to c_k and the variable x_{ir} to x_{irk}. This approach allows to reduce the number of constraints, while increasing the number of variables

(this is the approach used by [MAG91]).

The constraint (3.5) requires that for a given demand pair all the circuits are either allocated on the rings, or interconnected. Note that this constraint gives an exact estimation of USHR traffic. Suppose for some non-zero demand d_{ij} a node i belongs to a ring r, and a node j belongs to two rings s_1 and s_2. Firstly, for any feasible routing, $y_{ijrs_1} + y_{ijrs_2} = 1$, therefore the total traffic routed between the rings r, s_1, s_2 for the demand (ij) is d_{ij}. Both rings s_1 and s_2 can handle a portion of the interring traffic d_{ij}, but the corresponding capacity constraints will only account for the interring traffic assigned to this ring.

The support constraints are needed to establish the necessary logical relations between the variables. Constraints (3.6) and (3.7) force the variable y_{ijrs} to be 0 if one or both nodes do not belong to the corresponding rings.

The higher the ratio of the ADM cost to the interconnection cost, the more calls will be interconnected. Suppose the smallest ADM cost is B_0 and the interconnection cost is a. If $d_{ij} \leq \lfloor B_0/a \rfloor$, than it *may be* profitable to interconnect the demands.

In the next section we show that **[IRD]** is NP-hard.

3.3 Computational Complexity of [IRD]

Proposition 1 [IRD] *is NP-hard.*

We will show that a **capacitated facility location** problem is reducible to **[IRD]**, i. e., for any instance of the facility location problem, a corresponding instance of **[IRD]** can be constructed in polynomial time, such that solving the latter, solves the former as well.

The capacitated facility location problem can be stated as follows.

Given a set of M potential facility locations, a set of capacities B_i for each candidate facility i, a set of N customers and a set of D_j demands for a customer j; fixed facility placing costs c and variable cost h_{ij} of transporting products from a facility i to a customer j, choose the subset of locations at which to place facilities and assign the customers to those facilities to minimize the cost. (See [NEMH88])

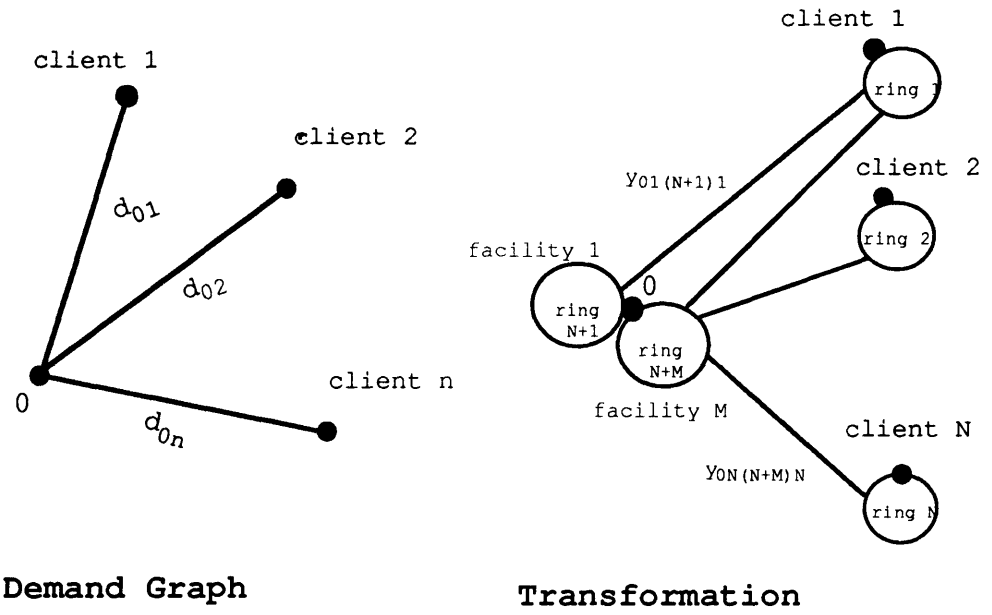


Figure 3-3: Facility Location Problem Transformation

Given the set of N customers and the set of M prospective locations consider the following transformation. Consider a graph with nodes $1 \ldots N$ representing the customers, with the node 1 corresponding to customer 1, node 2 corresponding to customer 2, and so on. Add an artificial node 0 and connect it to every customer with an arc. Let d_{0i} be the demand of customer for all i, $i = 1 \ldots N$. Let $x_{ii} = 1$ for $i = 1 \ldots N$, $x_{ij} = 0$ for $i = 1 \ldots N, j = 1 \ldots N, j \neq i$ and $x_{0j} = 0$ for $j = 1 \ldots N$. Thus each customer-node is assigned to a different one-node ring. Let the ADM fixed cost of each customer-node ring be 0 and the capacity $B_i = d_{0i}$, thus each of the rings can accommodate all the corresponding customer demands (calls).

Introduce rings $N+1 \ldots N+M$. A ring $N+j$ represents a facility $j, j = 1 \ldots M$. Let the capacity of every ring $N+j$ equal the capacity of the facility j. Let the cost c_{N+j} of an ADM system at the ring $N+j$ be the fixed cost of placing facility at the location j (we use facility j and location j interchangeably). Let $x_{iN+j} = 0$ for $i = 1 \ldots N, j = 1 \ldots M$ and $x_{0N+j} \in \{0, 1\}$ for $j = 1 \ldots M$.

Since each customer is assigned to a unique one-node ring and $d_{ij} > 0$ for all $i, j > N$, all the traffic in this ring network is interring traffic with the cost $a_{0jrj} = h_{rj}$

for $r = N+1 \ldots N+M, j = 1 \ldots N$. We have the following instance of **[IRD]**.

[IRD$_i$]:

$$\text{Minimize } Z = \sum_{r=N+1}^{N+M} \sum_{i} c_r x_{0r} + \sum_{r=N+1}^{N+M} \sum_{j=1}^{N} a_{0jrj} y_{0jrj} d_{0j}$$

subject to

$$\sum_{r=N+j}^{N+M} x_{0r} \geq 1 \tag{3.12}$$

$$\sum_{j=1}^{N} y_{0jrj} d_{0j} \leq B_r \text{ for all rings } r = N+1 \ldots N+M \tag{3.13}$$

$$\sum_{r} \sum_{j} y_{0jrj} = 1 \text{ for all } d_{0j},\ j = 1 \ldots N,\ r = N+1 \ldots N+M \tag{3.14}$$

$$y_{0jrj} \leq x_{0r} \text{ for all } r = N+1 \ldots N+M \tag{3.15}$$

$$x_{0r} \quad \text{are 0-1 variables for } r = N+1 \ldots N+m \tag{3.16}$$

$$y_{0jr0} \in [0,1] \text{ for all r,j} \tag{3.17}$$

For a feasible solution of **[IRD$_i$]**, y_{0jrj} indicates a fraction of demand for a customer j that is satisfied from a location $r - N$ $r = N+1 \ldots N+M$, and if $x_{0N+j} = 1$ then a facility is to be built in j.

Since the capacitated facility location problem is known to be NP-hard, the **[IRD]** problem is also NP-hard *Q. E. D.*

In this next section we describe an expansions problem for the Interconnected Ring network.

3.4 Expansion Problem for Survivable Ring Networks

As we mentioned in Section 3.1, the problems arising during the operation of a network can be often modeled by modifying network design models. In this section we consider

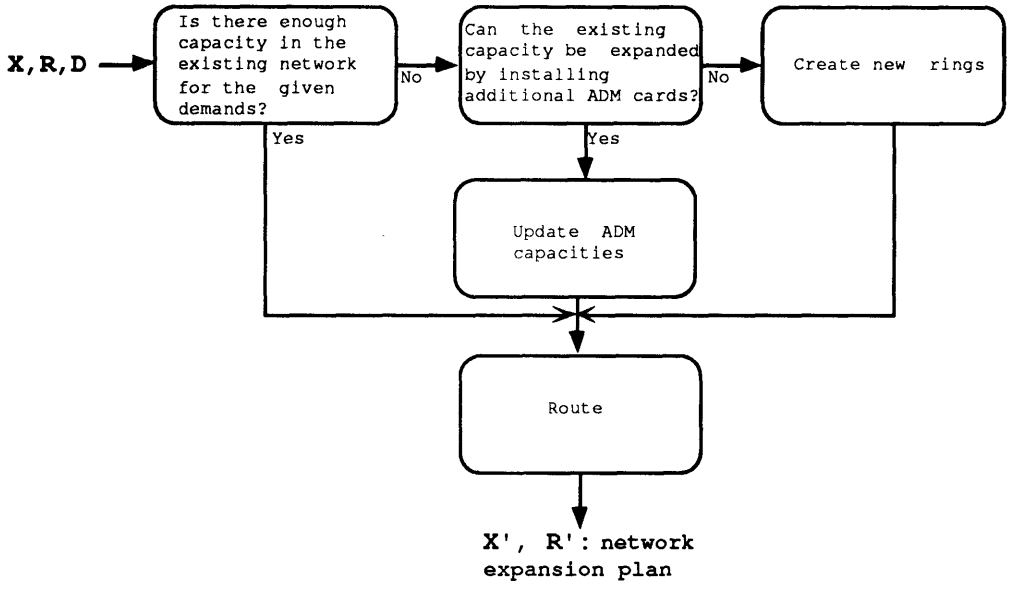


Figure 3-4: Network Expansion Problem

network expansion planning, for which the model [**IRD**] can be modified and applied.

Suppose we have an existing ring-interconnected SONET network. This network is built for a given demand matrix. After some time, new services have to be added. Thus, the demands between the nodes will change (probably, most of the demands will increase). The network administrator faces the following question: can we accommodate new services using the existing network and installed ADM capacities and interconnections? If not, can we support all the services without adding any new rings or new interconnections but only increasing the ADM capacities? If this ADM expansion strategy is feasible where do we install the new ADM systems? If the existing network cannot support all the services even after increasing the ADM capacities, which rings are exhausted? In other words, the problem posed here is to create a comprehensive network expansion plan for the case when the new services are added. The demand expansion problem is presented in Figure 3-4.

We are given $R = 1 \ldots r$ – the set of existing rings, the matrix of node-ring assignments $X = \{x_{ir} | i \in V, r \in R\}$ and the matrix of the demands $\mathcal{D} = \{d_{ij} | i \in V, j \in V\}$. $\mathcal{D}$ includes the new services.

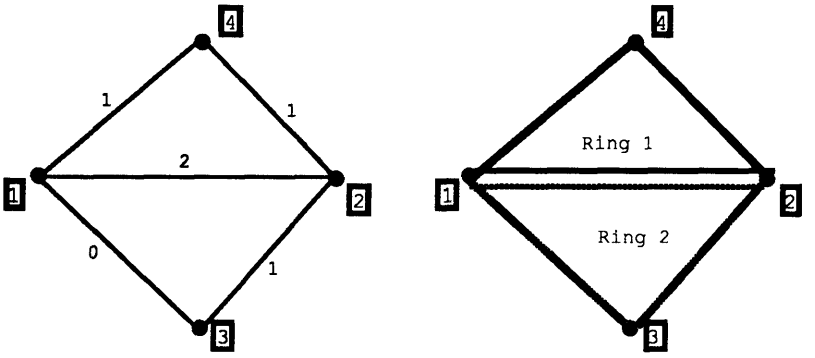


Figure 3-5: Network Expansion Example

At first sight, the problem of "fitting" new services may seem to be easy. A naive approach can be to freeze the initial routing, and try to fit new services sequentially. This greedy approach does not always work, since if a service can be fitted into more than one ring, the decision must be made based on the knowledge of routes of all the available demands, including the set of new ones. If the new calls cannot be fitted in any ring, there is a possibility that some of the *old* calls could be rerouted to accommodate the new service.

Consider an example on Figure 3-5. There are two rings: ring 1 includes the nodes 1, 2, 4, and ring 2 contains nodes 1, 2 and 3. The demands are as shown, and the capacity of each ring is 3 units. An initial routing assigns 3 units to the ring 2. Suppose, the new service consists of the demand $d_{13} = 1$. This demand must be routed on the ring 2, but the ring is full. Note, that if we reroute 1 unit of d_{12} on the ring 1, the ring 2 can accommodate the new demand.

The example shows that when there is more than one feasible routings, we may have to reroute old demands in order to accommodate the new services. Moreover, there may be a situation when the new service cannot be included on any ring, but it is possible to interconnect some two rings to install the service. This will add an interconnection cost, but the objective of the planner will have been achieved. In a big network there may exist a large number of different feasible routing and interconnections; thus, checking whether a ring has reached exhaust for a particular demand matrix is not a trivial problem.

Locating free capacity in the network, administering ADM capacities and interconnections and finding the exhausted rings are related problems. Thus it is possible to model the expansion problem with a single mathematical model.

We will use the ideas of the previous section to formulate a ring network expansion model **[EXP1]** . For a given set of rings and feasible routing, the model considers adding new demands at a minimum cost. If there is enough capacity in the network to incorporate the new demands, the model creates new feasible routing and the optimal cost is 0. If the optimal objective value is non-zero, then depending on the values of the corresponding variables (see below) either new rings must be built or new interconnections must be added or some of the old interconnections expanded.

Consider the **[IRD]** model. Suppose that X matrix is given. All the x_{ir} variables are constants. The assignment constraints (3.3) are satisfied automatically.

Consider an ADM system. In order to increase capacity of existing ADM, additional cards have to be installed. All the cards have a fixed cost, and a fixed capacity (see the ADM cost example section 3.2). Let δ be the capacity of the card. Then, if existing ADM capacity for a ring r is B_r, the total ADM capacity for ring r after the expansion is $B_r + \alpha_r\delta$, where α_r is defined to be integer from 0 to $K - 1 - r$, K is the number of the sizes available. We define an extra ADM size $B_K = M_B$ and a corresponding binary variable β_r for each ring r, where M_B is a very big number, such that B_K can accommodate all the demands of the network. If the ring uses the system B_K, it would signal the corresponding ring's exhaust.

The variables g_{rs} for all $r \neq s$ indicate extra interconnection capacity to be added. G_{rs} equals existing interconnection capacity for the rings r, s.

Routing variables y are defined as in **[IRD]**.If $x_{ir} = 1$ and $x_{js} = 1$, y_{ijrs} variable can be between 0 and 1, and if one or more x variables are zero, y_{ijrs} has to be zero. For **[EXP1]** model the routing variables include the old demands and the suggested expansion services.

Taking this discussion into consideration, we have the following model.

[EXP1]:

$$\text{Minimize } Z_1 = \sum_r C_r \alpha_r + \sum_r M_r \beta_r + a \sum_{r \neq s} g_{rs} \tag{3.18}$$

subject to

$$\sum_i \sum_{j>i} y_{ijrr} d_{ij} + \sum_{s \neq r} \sum_{j>i} y_{ijrs} d_{ij} + \sum_{s \neq r} \sum_{j>i} y_{ijsr} d_{ij} \leq B_r + \alpha_r \delta + \beta_r B_{K+1} \tag{3.19}$$

for all rings r

$$\sum_{j>i} (y_{ijrs} d_{ij} + y_{jirs} d_{ji}) \leq G_{rs} + g_{rs} \text{ for all } r \neq s \tag{3.20}$$

$$\sum_r \sum_s y_{ijrs} = 1 \text{ for all } d_{ij} > 0 \text{ and } j > i \tag{3.21}$$

$$y_{ijrs} \in [0, 1]$$

$$i_{rs} \geq 0$$

$$\beta_r, \alpha_r \quad \text{defined as above}$$

The cost C_r for each ring r is defined as C_r = number of nodes on the ring r $\times$ the cost of an expansion card, assuming all the expansion card have the same size and cost.

Let us see how this model helps a network planner to develop a cost-effective expansion plan. Suppose, there is enough capacity in the network to accommodate the new demands. Than, $Z_1 = 0$, and the model gives a new routing.

If there is not enough capacity in the network to accommodate the new demands, but the capacity of rings or interconnections can be expanded to accommodate the new services. M_r is set sufficiently large, so that $\beta_r = 0$. The α and g variables will provide the network expansion plan.

If there is not enough capacity, the pseudo ADM size B_{K+1} will be used. The corresponding rings would have reached the ring exhaust, and the new rings will have to be created. In order to build the new rings with minimum cost, we use the **[IRD]** model with zero costs for the already existing rings.

Suppose, a node or a group of nodes has to be added to the network. This problem can be treated similarly to **[EXP1]**. The assignment and forcing constraints have to

be added to the **[EXP1]** to represent the new nodes.

In general, the model **[EXP1]** can be used to develop comprehensive plans for addition of new nodes and new services simultaneously.

Chapter 4

Initial Solution Heuristics

In Chapter 3 Section 3.3 we showed that [**IRD**] is NP-complete. Thus solving [**IRD**] to optimality is difficult and computationally intensive. We design fast heuristics that produce a solution on the average within 10% or less from a lower bound. We develop a two-stage approach: first we use two initial solution heuristics, then we apply series of improvement methods to generate a near-optimal solution.

The following can be noted about the initial solution heuristic: it is required to be fast; the solution generated has to be reasonable, but further improvements will be made.

In this chapter develop initial solution methods for [**IRD**].

4.1 [IN1]: Profitable Ring Selection Initial Heuristic

In this section we describe a Profitable Ring Selection Problem [**PR**] and develop an initial solution heuristic based on a special structure of the model.

4.1.1 [PR]: Profitable Ring Selection Problem

The constraints (3.4) in [**IRD**] are knapsack constraints. A knapsack problem is a well-known problem and can be solved efficiently (see for example [AMO93]). In this

section we develop an approach to utilize the special structure of (3.4).By considering the objective function of [**IRD**] in more detail we develop an approach to generate an initial solution using a subproblem with the knapsack capacity constraint.

Consider the [**IRD**] objective function.

$$Z = \min\Big(\sum_{r}\sum_{i} c_r x_{ir} + a\sum_{\substack{r,s \\ r\neq s}}\sum_{\substack{i,j \\ j>i}} y_{ijrs} d_{ij}\Big) \tag{4.1}$$

We assume here that the interconnection cost a is the same for every interconnection. Let $y_{ij} = \sum_{\substack{r,s \\ r\neq s}} y_{ijrs}$. The variable y_{ij} is a fraction of the demand d_{ij} that is interconnected. Substituting y_{ij} into the objective function, we have:

$$Z = \min\Big(\sum_{r}\sum_{i} c_r x_{ir} + a\sum_{\substack{i,j \\ j>i}} y_{ij} d_{ij}\Big) \tag{4.2}$$

Since interconnecting demands incurs a cost proportional to the demand, we think of placing a demand inside a ring as being a profitable activity, as opposed to interconnecting. In practice the bigger the demand, the more the savings due to placing it inside the ring. For a small demand, interconnecting may be more profitable, but for the initial solution heuristic we assume that all demands have to be placed within rings.

Let $w_{ij} = 1 - y_{ij} = 1 - \sum_{\substack{r,s \\ r\neq s}} y_{ijrs}$ be an intraring fraction of d_{ij}. Substituting w_{ij} into (4.2) we have:

$$Z = \min\Big(\sum_{r}\sum_{i} c_r x_{ir} + a\sum_{\substack{i,j \\ j>i}} d_{ij} - a\sum_{\substack{i,j \\ j>i}} w_{ij} d_{ij}\Big) =$$

$$= D + \min\Big(\sum_{r}\sum_{i} c_r x_{ir} - a\sum_{\substack{i,j \\ j>i}} w_{ij} d_{ij}\Big) \tag{4.3}$$

where $D = a\sum_{\substack{i,j \\ j>i}} d_{ij}$. Multiplying the objective function by -1 and disregarding the constant D:

$$Z' = \max\left(a \sum_{\substack{i,j \\ j>i}} w_{ij} d_{ij} - \sum_r \sum_i c_r x_{ir}\right) \tag{4.4}$$

For each ring r let us define the ring profit P_r as

$$P_r \equiv a \sum_{\substack{i,j \in r \\ j>i}} y_{ijrr} d_{ij} - \sum_i c_r x_{ir} \tag{4.5}$$

The quantity P_r indicates savings due to placing a group of demands on the same ring versus interconnecting the demands.

Since $w_{ij} = \sum_r y_{ijrr}$, $Z' = \max \sum_r P_r$ and the original objective function $Z = D - Z'$. So minimizing the cost of building ring interconnected network is equivalent to maximizing the sum of the ring profits such that the constraints (3.3) - (3.7) are satisfied.

There is an ambiguity involving w_{ij} since the variable does not indicate to which rings the circuits belong to. By selecting only the most profitable ring, we resolve this matter.

For P_r defined as in (4.5) consider a problem of Profitable Ring Selection **[PR]** .

[PR]:

Given a set of nodes, internode demands and available ADM capacities and costs, find a maximum-profit ring.

We propose an initial solution heuristic based on **[PR]**. For the given data we select the most profitable ring. The procedure is repeated iteratively until all the demands are grouped. Since **[PR]** has a knapsack structure we use Lagrangean relaxation technique, relaxed **[PR]** is a knapsack problem that can be solved efficiently. The generated rings will group the demands according to the savings, so we expect the approach provide us with a good initial solution. Thus a motivation for using **[PR]** problem for initial solution heuristic is the relationship between **[PR]** and **[IRD]** and the special structure of the Profitable Ring Selection problem.

Consider the following formulation of **[PR]**. **[PR]:**

$$\text{Maximize } P_r = \sum_i \sum_j a_{ij} w_{ij} d_{ij} - c_r \sum_i x_i$$

subject to

$$\begin{aligned}
w_{ij} &\leq x_i \text{ for all i, } j > i && (4.6)\\
w_{ij} &\leq x_j \text{ for all i, } j > i && (4.7)\\
\sum_{j>i} w_{ij} d_{ij} &\leq B_i \text{ for all ADM sizes i} && (4.8)\\
x_i &\in \{0,1\} \text{ for all i,j} && (4.9)\\
0 &\leq w_{ij} \leq 1 \text{ for all i,j} && (4.10)
\end{aligned}$$

$x_i = 1$ if node i is placed on the ring. Note that if the optimal objective function $P_r^* > 0$ and if the ring has enough capacity all the non-zero w_{ij} are set to 1. If there is no such set of demands that $P_r > 0$ then $P_r^* = 0$ and no demands are grouped on the ring.

We remark that in general, a solution to [**PR**] will not necessarily be a part of an optimal solution to [**IRD**].

4.1.2 Knapsack-Based Initial Solution Heuristic

In Section 4.1.1 we mentioned that [**PR**] problem has an embedded knapsack structure. In order to utilize this fact we make the following simplifications to [**PR**] model. We assume that only one ADM size is given. This assumption leads to a single knapsack constraint in [**PR**]. We use the smallest available size B_0. There are two reasons for using the smallest size ADM: knapsack problem is pseudo polynomial, its running time is proportional to the right hand side of the capacity constraint; secondly by using the smallest ADM size we create more rings and therefore more opportunities for subsequent improvement. We assume the demand routing variables w_{ij} to be binary rather than fractional. As noted previously if it is profitable to include a part of a demand, and there is enough capacity on the ring, all the demand will be included.

If there is not enough capacity, but it is still profitable to group a fraction of the demand, the fraction of the demand will be included according to the **[PR]** formulation in Section 4.1.1. The remaining fraction must be included on another ring (we do not consider interconnections at this stage). If a demand is included on more than one ring, then we may incur extra costs due to additional ADM systems, therefore in the initial solution we want to avoid bifurcated routing.

The remaining issue is what happens if the optimal objective function of **[PR]** is 0, in other words it is not profitable to group any of the demands on the rings. This may occur if the ADM systems are very expensive, or the variable interconnection cost is cheap. For the initial solution we reduce the ADM cost to zero, and continue the iterations until all the demands are grouped.

Based on our simplifications we can now rewrite the **[PR]** model.

[PR]:

$$\text{Maximize } Z = \sum_i \sum_j a_{ij} w_{ij} d_{ij} - c_0 \sum_i x_i$$

subject to

$$\begin{aligned}
w_{ij} &\leq x_i \text{ for all i, } j > i && (4.11)\\
w_{ij} &\leq x_j \text{ for all i, } j > i && (4.12)\\
\sum_{j>i} w_{ij} d_{ij} &\leq B_0 && (4.13)\\
x_i, w_{ij} &\in \{0,1\} \text{ for all i,j} && (4.14)
\end{aligned}$$

Computational Complexity of the Profitable Ring Selection Problem

In this section we discuss computational complexity of **[PR]**.

Proposition 2 **[PR]** *is NP-complete.*

We will show that a knapsack problem is reducible to the Profitable Ring Selection problem **[PR]**.

Consider the following knapsack problem:

[KP]:

$$V = Max \sum_i c_i z_i$$

subject to

$$\sum_i y_i z_i \leq B$$

$$z_i \in \{0, 1\}$$

A choice has to be made among the list $\{z_1, z_2, \ldots, z_n\}$ elements. Each element z_i takes up y_i amount of space and brings in c_i units of profit. The following transformation from **[KP]** to **[PR]** is proposed.

Add an element z_0 to the list, such that $c_0 > 0$, $y_0 = 0$. Consider a knapsack problem **[KP']** defined on a list of elements $\{z_0, z_1, z_2, \ldots, z_n\}$ with the weights $\{y_1 \ldots y_n\}$ and profits $\{c_1 \ldots c_n\}$ equal to the ones defined in **[KP]** , and c_0 and y_0 defined as above.

Let V' be the value of the optimal solution for **[KP']**. Since $y_0 = 0$, and $c_0 > 0$, it is easy to see that **[KP']** and **[KP]** are equivalent and $V = V' - c_0$. Therefore, if we denote by $A \xrightarrow{P} B$ the fact that A can be transformed in polynomial time into B, and every instance of A can be answered, using B, showing $KP \xrightarrow{P} PR$ is equivalent to showing $KP' \xrightarrow{P} PR$.

To prove the proposition 2, we will show the polynomial-time transformation of **[KP']** into an instance of **[PR]** . Consider the following transformation. Let

$$d_{0j} = y_j$$

$$d_{ij} = 0 \text{ for all } i \neq 0$$

$$x_0 = 1$$

$$w_{0j} = z_j, i = 0, \ldots, n$$

$$w_{ij} = 0 \text{ for } i \neq 0$$

$$a_{0j} = c_j / d_{0j}$$

$$a_{ij} = 0 \text{ for all } i \neq 0$$

and the ring cost is 0.

Consider the following instance of [**PR**] problem:

$$Max \sum_j \frac{c_j}{d_{0j}} w_{0j} d_{0j}$$

subject to:

$$w_{0j} \leq x_0 = 1 \text{ for all j}$$

$$w_{0j} \leq x_j \text{ for all j}$$

$$\sum_j w_{0i} D_j \leq B_0$$

$$x_j, w_0 j \in \{0, 1\}$$

If $w_{0j} = 0, z_j = 0$, and if $w_{0j} = 1, z_j = 1$. So the optimal solution $\mathcal{W}$ of the instance of [**PR**] defines the optimal solution of [**KP'**], or, equivalently, [**KP**]. Therefore the knapsack problem can be transformed into [**PR**] in polynomial time and so the proposition 1 is proved. *Q. E. D.*

Lagrangean Relaxation for [PR]

Since [**PR**] is NP-complete, we propose to solve [**PR**] by relaxing the precedence constraints (4.11) and (4.12). Let μ_{ij} be a multiplier for the constraints (4.11) and π_{ij} be a multiplier for (4.12). Relax on (4.11) and (4.12). After simplification, the Lagrangean objective function for the [**PR**] becomes

[$\mathbf{PR}_r$]:

$$L(\mu, \pi) = Max\Big(\sum_i \sum_{j>i} w_{ij}(ad_{ij} - \mu_{ij} - \pi_{ij}) + \sum_i x_i (\sum_{j>i} (\mu_{ij} + \pi_{ij}) - c_r) \Big)$$

subject to

$$\sum_{j>i} w_{ij} d_{ij} \leq B \tag{4.15}$$

$$x_i, w_{ij} \in \{0,1\} \text{ for all i,j} \tag{4.16}$$

and the dual costs μ and π are non-negative. To simplify our notation we consider the cost of interconnection a_{ij} to be the same for all pairs of demands (i,j) and $a_{ij} = a$.

The Lagrangean subproblem is a knapsack problem with additional terms x_i in the objective function. Since the node variable x is unconstrained, it is set depending on a sign of its cost in the objective function. If the cost $\sum_{j>i}(\mu_{ij}+\pi_{ij})-c_r)$ is nonnegative, $x_i = 1$; if $\sum_{j>i}(\mu_{ij} + \pi_{ij}) - c_r) < 0$, $x_i = 0$. The knapsack problem is solved using dynamic algorithm described in [AMO93, pages 71-72]. We use subgradient optimization method to generate the dual prices. For the k-th iteration the dual prices are generated according to

$$\mu_{ij}^k = \max(0, \mu_{ij}^{k-1} + \theta^k (y_{ij}^{k-1} - x_i^{k-1}) \tag{4.17}$$

$$\pi_{ij}^k = \max(0, \pi_{ij}^{k-1} + \theta^k (y_{ij}^{k-1} - x_i^{k-1}) \tag{4.18}$$

where y_{ij}^{k-1}, x_i^{k-1} is the optimal solution of the iteration $k-1$, and θ^k is the iteration step. $\theta^1 = 1$ for the first iteration. It is more important to obtain a fast and a good feasible solution rather than to solve the problem to optimality. So the current best feasible solution is stored during the run and a fixed number of iterations is performed. The iteration step is changed according to $\theta^{k+1} = \theta^k/2$ if during the last 5 iterations no improvement of the objective function has taken place, and $\theta^{k+1} = \theta^k$ otherwise. We stop after the fixed number of iterations, or if an optimal value for **[PR]** was reached. For Lagrangean solution to be feasible for the original problem the precedence constraints (4.11) and (4.12) have to hold. For a feasible solution, we check optimality using complementary slackness conditions:

$$\mu_{ij}(w_{ij} - x_i) = 0 \tag{4.19}$$

$$\pi_{ij}(w_{ij} - x_i) = 0 \tag{4.20}$$

```
INPUT     Demand Matrix  { d_ij }
          ADM Size B_0   ADM Cost c_0  /* we use only 1 ADM size
                                         to generate initial solution */

OUTPUT    Ring-Demands Assignment List {R_i }

BEGIN [IN1]

     i =0;
     Do STEP^i While (not all the demands grouped)

          For unassigned demand pairs SOLVE [PR_i]
               Initialize the objective value L()=1000000;
               Do While (iteration k ≤200 or optimal)
                    θ^1=1;   μ^1=0;  π^1=0;
                    SOLVE KNAPSACK L(μ^k, π^k);
                    Find min_j( L(μ^j, π^j) );
                    If L() did not improve in the last 5 iterations
                       θ^(k+1) = θ^k/2;
          END [PR_i]
          If (L*() == 0)    SOLVE [PR_i] with the ADM cost =0;
          Assign demands to ring i;   i = i+1;
     END STEP^i

END [IN1]
```

Figure 4-1: [IN1] Initial Solution Heuristic

If the iterations limit is reached, but the optimal solution is not found, the best feasible solution is retained as the most profitable ring for the current step. The best Lagrangian solution can be retained to generate an upper bound of the optimal value of **[PR]**.

Algorithm Description

In this section we describe the initial solution algorithm **[IN1]** based on the relaxation of the Profitable Ring Selection problem.

Figure 4-1 describes the **[IN1]** heuristic. The input data are a demand matrix and a size and a cost of the smallest ADM system. As explained above, to form an

initial solution we use only one ADM size. The procedure is performed until all the demands are assigned to rings. For a given step i a ring R_i is created. **[PR]** is run for 200 iterations or until the optimality is reached. The best solution is used for the ring R_i. If the optimal objective value is 0, the ADM cost is set to 0 and the procedure is repeated for the current ring.

Computational Complexity of [IN1]

Consider first the Profitable Ring Selection heuristic **[IN1]**. Recall, that the algorithm forms the rings by selecting the most profitable ring from the uncovered demands. The most profitable ring is selected by solving a Lagrangean relaxation which is a knapsack problem. In the worst case, one demand pair at a time is grouped on a ring. Thus, the method requires $\lceil$ the number of demand pairs $\times$ the number of iterations of the Lagrangean relaxation $\times$ the number of operations to solve the knapsack problem $\rceil$ operations. Let B_0 denote the capacity of the knapsack and let N denote the number of the nodes. The knapsack problem is solved in $O(BN)$ iterations (see [AMO93]). Thus the worst-case complexity of **[IN1]** is $O(BN^2)$.

4.2 [IN2]: Slack Packing Heuristic for the Initial Solution

In this section we consider an alternative initial solution heuristic based on inherent properties of **[IRD]** as a telecommunications network design problem. Here, as in section 4.1.2, we consider a single ADM size B_0.

For a node i define a **nodal demand** D_i as the sum of all calls terminating at that node, i. e. $D_i = \sum_j d_{ij}$. The underlying idea of the heuristic **[IN2]** is to allocate all the nodal demands sequentially to rings. This method is expected to be faster than **[IN1]** since minimum amount of processing will be done compared to solving Lagrangean relaxation in **[IN1]**.

Allocation of the current demand pair depends on the demands (nodes) that have

been grouped before and the rings that have been created. The issue is to find a rule for the nodal demand selection for the given nodes and demand pair selection for the given nodal demand such that we generate a good initial solution.

The order of selection of nodes and demand pairs to group on rings influences the final cost of the solution. Consider the following example. Suppose an ADM has a capacity of 10 and the following demand matrix is given: $d_{12} = 3, d_{13} = 4, d_{23} = 2, d_{24} = 6$. Suppose the node 2 is selected first. Within the set of the nodal demands for the node 2 we select the demands in the following sequence: d_{24}, d_{23}, d_{12}. Two rings will be created, ring 1 with the nodes $2, 3, 4$ and demands d_{24}, d_{23} and ring 2 with the nodes $1, 2$ and demand d_{12}. Demand d_{13} is then added to the ring 2. Thus we have a solution consisting of two rings: the ring 1 with the nodes $2, 3, 4$ and the ring 2 with the nodes $1, 2, 3$. The cost of the solution is 60.

Consider another order of the node grouping: first we group the node 1 and all the associated demands, than the node 3 and all the node 3 demands that are not assigned, then the node 2 and then 4. It is easy to check that two rings are created, ring 1 with the nodes $1, 2, 3$ and ring 2 with the nodes $2, 4$. The cost of this solution is 50, and so due to the different node selection order in the second case we produced a better solution.

For a given set of demands routed on a ring R we define **free capacity** F_R of the ring as the additional number of circuits that can be allocated on the ring until the ring reaches exhaust. For a given node-to-ring allocation and a given demand d_{ij} being assigned, free capacity F_R shows whether it is possible to assign the demand to the ring.

Each node-to-ring assignments allocates nodal demands for a given node to existing rings or creates new rings. Thus each node-to-ring assignment changes the free capacity of the existing rings and adds the free capacity of the newly created rings. In order to find the best assignment for d_{ij} we have to know the free capacity of all the existing rings and the existing node-ring assignment (which previously assigned nodes belong to what ring). In order to have an efficient node selection rule it is important to assess free capacity that is required for a given set of nodal demands. For a nodal

demand D_i we measure the free capacity using **nodal demand slack** S_i. Define the quantity $S_i = \lceil D_i/B_0 \rceil B_0 - D_i$. The slack is different from free capacity since the free capacity is calculated for the already created rings and allocated demands, while the slack is calculated for a node independent of the ring assignment.

For the node selection rule we considered several strategies such as the largest nodal demand first, the smallest demand first, the smallest slack first and the largest nodal slack first. We have chosen the minimum slack first strategy to implement as our initial heuristic. The key observation is that the savings due to allocation of demand pairs to existing rings depend on the *number* of demand pairs allocated to existing rings, rather than the demand size being allocated.

Maximum demand first strategy attempts to assign first the nodes, requiring the largest number of ADM's. As a result of this strategy a large number of small nodal demands will have to be assigned to different rings and use separate ADM's; utilization of the ADM systems will be low, so the solution may expensive. Assigning smallest nodal demands first confronts this problem: for small size demands utilization will be higher, since in the beginning demands for the node may be grouped on the same ring. But a problem may arise because it is not guaranteed that enough free capacity will exist in the end to place bigger demands, and so a new ring will have to be created for every new nodal demand.

Our experience suggests that selecting nodal demands based on their slacks produces a better solution than selecting the demands based on their magnitudes.

If the maximum slack first selection strategy is used, the result will be abundance of free capacity in the first steps of the algorithm. But in the end, when the demands requiring more free capacity are grouped, there may be none left.

If the nodes with the smallest slack are grouped first this will produce less free capacity in the beginning but in the long run more rings with free capacities will exist, and bigger proportion of nodes with smaller demands could be allocated to the existing rings.

The allocation of a demand pair d_{ij} for a given node depends on the ring's free capacity and on the nodes assigned to the ring. Since the objective is to find free

capacity on the existing ring containing one or both nodes i, j and the bigger the demand d_{ij}, the more capacity it requires, a reasonable rule is to group the largest demand pair first.

The initial solution heuristic [**IN2**] is depicted on Figure 4-2. We consider one ADM size for the initial solution.

The input data are the demand matrix and the ADM size and cost. No interconnection cost is required because we do not consider interconnections at this stage. For each iteration of the algorithm a node I with minimal slack is selected. For all the demand pairs for the node I, group the demands beginning with the largest pair d_{Ij}. If there is a ring R_k such that both nodes I and j belong to that ring, and there is enough capacity on this ring to include d_{Ij}, assign the demand d_{Ij} to that ring. Otherwise if there is a ring such that the node I *or* the node j belong to the ring and there is enough capacity, include the demand d_{Ij} on that ring and add the other node to the ring. Otherwise create a new ring for d_{Ij}. After all the demands for the given node are grouped, mark the node. For all the unmarked nodes subtract the grouped demands and repeat the procedure. The algorithm stops when all the nodes are marked.

Computational Complexity of [IN2]

Consider the Slack Packing Initial Solution heuristic [**IN2**]. For each node, a slack is calculated, and for the node with minimum slack, the nodal demands are allocated to rings. This procedure first looks through available rings containing the node, and if no ring contains the node, or there is not enough capacity, a new ring is created. Thus if the maximum number of rings is R, the worst case performance of [**IN2**] is $O(NR)$. This heuristic can be made faster by allocating every node to a new ring without looking for common nodes in the already created rings. Since no bifurcated routing is allowed at this stage, the maximum number of rings equals the maximum number of demand pairs, so $R = N(N+1)/2$.

```
INPUT      Demand Matrix  {d_ij}
           ADM Size B_0   ADM Cost c_0
/* We use only 1 ADM size to generate initial solution */

OUTPUT     Ring-Demands Assignment List {R_i }


BEGIN [IN2]

     For all Nodes {i}
          Find Min Slack  S_I = min_i( ⌈D_i/B_0⌉ B_0 - D_i );

          For all demand pairs d_Ij>0
          Find d_Ij = max_j( d_Ij )
          (1) If ∃ring R_k: I∈R_k and j∈R_k and |R_k|≥d_Ij
                    R_k ← d_Ij;
          (2) If not (1) ∃ring R_k:I∈R_k or j∈R_k and |R_k|≥d_Ij
                    R_k ← d_Ij;
          If not (1) and (2)
                    Create new ring R_k+1 ← d_Ij;
          Mark the node;

          END "For all demand pairs.."
          For all unmarked nodes delete grouped demand pairs;

     END "For all nodes..."

END [IN2]



* F_Rk - free capacity of ring k.
```

Figure 4-2: [IN2] Initial Heuristic 2

Chapter 5

Solution Improvement Heuristics

5.1 [SWAP]: Swap Heuristic

5.1.1 Motivation

The heuristics described in the previous chapter provide initial feasible solutions. As our computational results suggest, the initial solution is created in the order of seconds. However the gap between the objective function of the initial solution and the optimal value is as large as 20%. The size of the gap suggests that we must apply improvement methods to the initial solution in order to decrease the optimality gap.

For example consider demand graph on figure 5-1. On this figure an arc represents a demand pair, and vertices represent the nodes. The weight of the arc is the value of d_{ij}.

We are given an ADM of size 15, the ADM cost of 2, and interconnection cost of 1. The most profitable ring is the ring consisting of the demands w_{14}, w_{13} and w_{34}. The initial solution heuristic **[IN1]** creates the following rings:

$$R_1 : \quad w_{14}, w_{13}, w_{34}$$

$$R_2 : \quad w_{15}, w_{45}$$

$$R_3 : \quad w_{14}, w_{34}$$

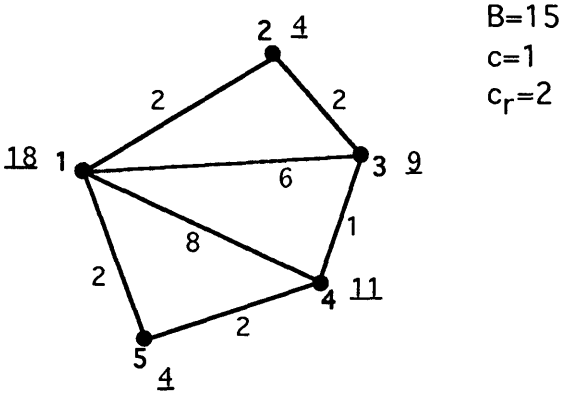


Figure 5-1: Initial Solution: Greedy Is Not Always Optimal

The cost of this solution is $3 \times 2 + 3 \times 2 + 3 \times 2 = 18$. Choosing ring R_1 at the first step of **[IN1]** results in a solution that is not optimal. Consider another solution:

$$\begin{aligned} R_1: &\quad w_{14}, w_{45}, w_{15} \\ R_2: &\quad w_{12}, w_{13}, w_{23} \\ \text{interconnect:} &\quad w_{34} \end{aligned}$$

The cost of the last solution is $3 \times 2 + 3 \times 2 + 1 \times 1 = 13$, and so initial solution created in the first grouping has a lot of room for improvement.

We propose two local improvement methods: reallocation of demands or parts of the demands from one ring to another, and ring interconnecting heuristic. In this section we describe the local swap heuristic, and the next section is devoted to interconnections.

5.1.2 Swap Moves

Define $D_{iR} = \sum_j (y_{ijRR} d_{ij})$ as the **ring-nodal demand** for a node i and a ring R. In this section we define demand swap moves. In the next section we describe how to select the demand pair and the destination ring.

In our notation D_{iR} is the nodal demand to be reallocated, we refer to R as the *source ring*. R_1 is the ring where the nodal demand is moved. We refer to R_1 to as

the *destination* ring and d_{kl} is a demand belonging to R_1 that must be swapped with D_{iR}. Define a **move** as any reallocation involving a nodal demand D_{IR}, destination ring R_1 and demand pair to be moved d_{kl} (d_{kl} may be 0).

Recall that F_R is a free capacity of the ring R, $F_R = B_K - \sum_{i,j>i}(y_{ijRR}d_{ij} + y_{jiRR}d_{ij})$. B_K is the maximum available ADM size. F_R more calls can be allocated on the ring R until it reaches exhaust. The local moves are depicted on Figure 5-2. The swaps are described below.

ADD If there is enough capacity on the ring R_1 to accommodate D_{iR} demands for a node i on the ring R.

SW If there is not enough capacity on R_1 to accommodate D_{iR} units, but there is a demand $d_{kl} \in R_1$ such that $F_{R_1} + d_{kl}$ is enough to accommodate D_{iR}, and $F_R + D_{iR}$ is enough to incorporate d_{kl}, in other words if it is possible to perform a swap between the rings R and R_1 using the demands D_{iR} and d_{kl}.

SW_PARTIAL If for R, R_1, D_{iR} and d_{kl} defined as above, SW can not be performed because the source ring can not accommodate d_{kl} units, but a portion y_{klRR} of d_{kl} can be included on F_R and released free capacity on the ring R_1, $F_R + y_{klR_1R_1}d_{kl}$ is enough to accommodate D_{iR}. Note that in this case the demand d_{kl} is split between the rings R and R_1 (bifurcated routing).

5.1.3 Algorithm Description

The idea of Demand Swapping heuristic [**SWAP**] is to achieve a decrease in cost by reallocating some of the nodal demands to the other rings. The savings are achieved through decrease of the number of ADM systems in the initial solution.

In order to select which nodal demands to move first, we introduce **inefficiency**, a measure of "excessive" ADM systems at the node.

We previously viewed [**IRD**] as a way of routing demands on the rings. Another view of the Interconnected Ring Network could be to consider a problem of optimal ADM sizing and placement. For each node we have to install enough systems to

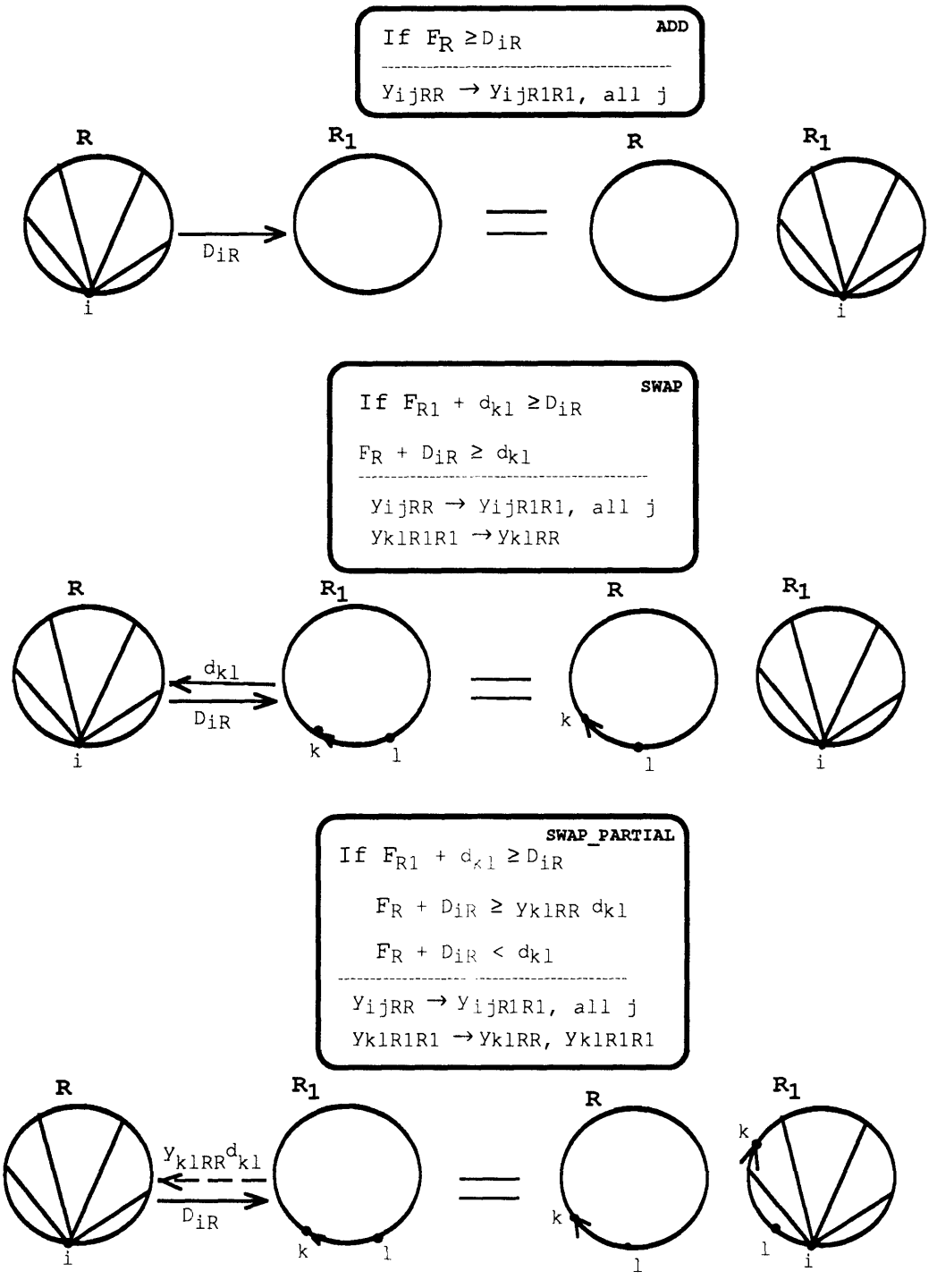


Figure 5-2: Swap Moves

handle all the traffic in, out, and through the node with the minimum cost. From the point of view of the node, the minimum number of ADM systems required is such that there is enough capacity to terminate the nodal demands. For the initial solution for a single size B_0, the minimum number of ADM systems required in the node with total nodal demand D_i is $\lceil D_i/B_0 \rceil$. Suppose that an initial solution installs N_i ADM's at the node, and $N_i \geq \lceil D_i/B_0 \rceil$. The nodes inefficiency $\mathcal{I}_i = N_i - \lceil D_i/B_0 \rceil$ is defined as the number of extra ADM systems over and above necessary to terminate the nodal demands. The greater $\mathcal{I}_i$ is, the more ADM systems at the nodes are underutilized.

We remark here that due to the ring architecture, for each ring node a fraction of the ADM capacity must be devoted to handling the passing demands. Therefore, even in the optimal solution there may be nodes with $\mathcal{I}_i > 0$. But as a rule of thumb, the greater the inefficiency, the easier it is to delete the extra ADM's from the rings and thus to achieve additional savings. We use the slack measure in the context of the swap routine to select the nodes for which some ADM systems may be deleted first.

In this section we describe the swap heuristic **[SWAP]**. All the swap moves are described in the section 5.1.2.

As a result of moving of all the demands associated with i off the ring R, the node i is deleted from the ring. Thus savings can be made equal to the cost of the ADM. At the same time, other nodes may be added or deleted from the source or the destination ring.The ring's capacity may increase or decrease, so the corresponding ADM costs may change. For each move we calculate the move's profit MP. In general, MP can be expressed as the difference in cost between the old and the new solution. If $MP > 0$, then the move is profitable. Since any move is local, i.e. the structure of no more than two rings at a time change, each move's profit can be expressed as the difference in costs of the source and destination rings before and after the move:

$$MP = n_R \times c_R + n_{R1} \times c_{R1} - (n_R - nd_R + na_R) \times c'_R - (n_R 1 - nd_R 1 + na_R 1) \times c'_R 1 \quad (5.1)$$

In our notation n signifies the number of nodes in the original solution, c is the

corresponding ADM cost, nd is the number of nodes deleted from the source (destination) ring and na is the number of nodes added; c' is the ADM cost on the ring if the move is performed. Note that for **[SWAP]** we allow the ADM capacities to change, and so the solution after **[SWAP]** may have rings of different sizes.

We start local improvement with the most inefficient node I. We attempt to move this node's demand from the given ring to every other ring and compute each candidate move profit. For each destination ring the move with the largest profit is chosen from all the demand pairs d_{kl} on the destination ring. If the best move's profit is negative, the move is not performed. If $MP > 0$ for the given D_{IR} a move with the best profit is performed. If $MP = 0$ the move is also performed as an "opportunity" move.

Figure 5-3 describes the algorithm. For every ring containing the node I and designated as a source, all the other rings are destinations. For every destination the best move is found. The maximum of all the move profits for all the destination rings is taken. If the move is non-negative, the move is performed, i.e. we update the node-to-ring allocation list. The procedure is repeated for all the rings containing the node I. Then the node I is marked, the most inefficient node out of all the remaining nodes is taken and the swap procedure is repeated until all the nodes are marked.

Our computational experiments suggest that adding a few empty rings to the initial solution improves the final cost. The reason is that empty rings can be used to complete the moves with $MP = 0$(in other words to relocate some group of demands without a profit). We think of this moves as an "opportunity" moves.

5.1.4 Computational Complexity

Consider the swap heuristic. For every node, starting with the most inefficient one, and for every ring, containing the node, a search for a profitable swap is performed. Thus the complexity is $O(NR^2)$.

```
INPUT      ADM Sizes { B_i }
           ADM Costs { c_i }
           Ring-Demands Assignment List {R_i }

OUTPUT     Ring-Demands After Swap Assignment List {R_i* }

BEGIN [SWAP]

     For all nodes i
     Find the most inefficient node: I= ArgMax_i (N_i ⌈D_i/B_0⌉ );

        For all source rings R
           For all destinations rings R_i

              If F_Ri ≥ D_IR Find MP for ADD;

              Else For all demand pairs on R_i
                                   such that there is enough capacity
                    Find MP;
              END Else

              Find MP_i= Max(MP for all demands for R_i);
           END "For all destinations..."

        MP= Max_i (MP_i);

        If MP ≥ 0 Perform the move;
     END "For all source..."

     Mark the node;
     END "For all nodes..."

END [SWAP]
```

Figure 5-3: [SWAP]: Swap Heuristic for Local Improvement

5.2 [I/C]: Interconnection Heuristic

5.2.1 Motivation

In the chapter 3 we stated the major assumptions related to ring interconnection. We assumed the rings to be dual homed. The unit-call interconnection cost is a_{ij}. The interconnection cost is variable.

In this section we consider a heuristic improving a solution by adding cost-saving interconnections.We consider the interconnection cost $a_{ij} = a$ to be independent of the nodes, a_{ij}. The Interconnection heuristic can also be applied to variable interconnection cost a_{ij}. The input to the Interconnection Heuristic **[I/C]** is the initial solution that has been improved by the Swap Heuristic.

First we analyze savings due to interconnection of rings. Consider two rings R and R_1, with a demand d_{ij} being an interring demand, $i \in R, j \in R_1$. Compare this situation with a solution such that d_{ij} is confined within a ring R (or R_1). In the case of interconnection and ADM, capacity of d_{ij} has to be allocated in the both rings R and R_1. If the demand is confined within the ring R, the capacity d_{ij} has to be allocated only within the ring R and the capacity d_{ij} in the ring R_1 is free. Thus interconnection requires more capacity then confining the same demand within a ring. If a demand d_{ij} is to be placed in the ring, than the nodes i and j have to be installed on that ring. If d_{ij} is to be routed from R to R_1, than i must be placed on the ring R, and j one on the ring R_1. It is not clear whether it could be profitable to interconnect demands.

Consider a solution obtained from the **[SWAP]** heuristic. Let $D_{iR} = \sum_j y_{ijRR} d_{ij}$ be the total nodal demand of the node i on ring R. If the node i is assigned to some other ring R_1 and the ring has enough free capacity to accommodate D_{iR} more units, then all the calls D_{iR} can be interconnected to the node i on the other ring, and the node i can be deleted from the ring R. Thus c_R dollars will be saved and interconnection cost aD_{iR} must be paid. If the difference $c_r - aD_{iR} > 0$, than the cumulative savings can be realized by interconnecting the ring-nodal demands D_{iR}. The reason for the savings is that some of the facilities on the rings may be underutilized, and by

interconnecting demands through those facilities, some of the existing ADM systems may be removed.

Interconnecting demands to R_1 may require increasing this ring's capacity (if a bigger ADM size is available). If the ring R contains n_R nodes and the ring R_1 contains n_{R_1} nodes, the interconnection profit IC can be computed according to

$$IC = c_R + n_{R_1} \times c_{R_1} - n_{R_1} \times c'_{R_1} - a \times D_{iR} \tag{5.2}$$

where the prime superscript indicates the interconnected solution. n_{R_1} is the number of nodes in the ring R_1, the ring to which given calls will be interconnected from R. To obtain an upper bound on the value of demand that can be profitably interconnected assume that the capacity of R_1 has not increased. Then if $D_{iR} \leq c_R/a$ it may be profitable to interconnect the node i demands on the ring R. Note, the smaller the ring-nodal demand is, the more savings result from its interconnection.

5.2.2 Algorithm Description

We propose the local interconnection improvement algorithm **[I/C]** depicted at Figure 5-4. From the previous discussion we have to consider only the ring-nodal demands less than c_{max}/a as possible candidates for interconnection.

In our heuristic we consider interconnecting one group of ring-nodal demands D_{iR} at a time. This simplifies the implementation.

The input data for the routine is the node-ring assignment list after **[SWAP]**, ADM sizes and costs, and the interconnection cost. For all the nodes I and all the rings R the minimal ring-nodal demand D_{IR} is found. If free capacity D_{IR} is located in one of the rings R_k other than R such that $I \in R_k$, the interconnection profit is calculated according to (5.2). If the best interconnection profit is positive, the interconnection is performed on the nodes-ring assignment list: the node I is deleted from the ring R, and the corresponding demands D_{IR} are interconnected. If the interconnection profit is negative, the node is marked. The procedure terminates when all the demands less than c_{rmax}/a are interconnected or marked.

```
INPUT      ADM Sizes {B_i}
           ADM Costs {c_i}
           Interconnection Cost a
           After Swap Ring-Demands Assignment List {R_i* }

OUTPUT     Demands-Ring Final Assignment List {R_i** }

BEGIN [I/C]

     For all unmarked nodes i, rings r such that D_ir ≤ c_rmax/a

          Find D_IR = Min_i,r(D_ir);
          For all rings R_k ≠ R

               For all rings R_k such that F_Rk ≥ D_IR and I∈R_k
                    Find IC_k = c_R - a D_IR - [n_Rk (c_Rk-c'_Rk)];
          END "For all rings R_k ≠ R "

          Find IC = Max_k(IC_k);
          If IC >0   delete I from R;
                     interconnect D_IR to R_k;
          Else Mark D_IR;
     END "For all unmarked nodes..."

END [I/C]
```

Figure 5-4: [I/C]: Local Interconnection Improvement Heuristic

The interconnection heuristic [**I/C**] conducts a similar search and thus has the same complexity as the local swap heuristic [**SWAP**].

Chapter 6

Computational Results

6.1 Data and Test Problems

To test the performance of our heuristics we conducted a series of numerical experiments.

We were unable to obtain actual data on costs, demands and ADM systems. Therefore, we designed the data in such a way that they will be representative of the problem. As a benchmark, we used the data mentioned in [LAG93].

The **sizes** of ADM hardware vary depending on the applications. It is important that the ratio of available ADM sizes and given demands is such that there is an ADM size that can incorporate the biggest demand pair in the system, for otherwise the problem is infeasible. We use two ADM's with respective capacities of 48 and 64 DS1 units.

Two different **demand patterns** are generated. One is a "uniform" interhub demand: the number of the hubs is fixed, and there exists a demand between two nodes with probability $p = 0.5$. The demands are drawn from a uniform distribution $U(0, 24)$ and rounded up. Another pattern is a "star" demand: there exists a fixed number of hub nodes, for each hub node a random number of CO's is generated from $U(0, 8)$ (this number is rounded up). The inter hub demand is uniform, drawn from $U(0, 48)$. The demand between a hub and Central Office from the same cluster is uniform $U(0, 8)$. For two CO's belonging to the different clusters, demands from

$U(0,4)$ are generated, thus signifying the intercluster traffic on the CO level. 10 problems are generated for each type of demands. See the Appendix C for an example of a test case. Our approach allows us to capture the effects of heavy interhub traffic, hierarchical network structure and the existence of demands that could be cost-effectively interconnected.

An example of ADM **cost** is given in Chapter 3. According to [SCOM93] those costs are subjected to virtually daily change. Therefore instead of trying to simulate the exact cost we decided to use costs that would capture the logical relationship between interconnecting and confining calls to rings.

Experience suggests, as noted in [FLA90b], that it is profitable to group nodes with high nodal demand on the rings, whereas the smaller demand nodes are interconnected. The unprofitable demands usually consist of a small number of DS1 calls. In Chapter 5 we showed that the decision whether the demands should be interconnected or placed on a ring for a certain node is based on the ratio of ADM cost to the interconnection cost. We know that it is *always* profitable to place the node on a ring if the total demands between the node and the ring is greater or equal than $\lceil c_r/a \rceil$, where c_r is the cost of ADM on the ring r and a is the interconnection cost, assuming the capacity of the ring r permits the inclusion of the node (if there is not enough capacity, then the interconnection is not possible either).

We use the cost data from Laguna: the ADM cost of 114 for the 48-unit system and the interconnection cost 15. For all the demands less than 8, the interconnection may be profitable. For the system with capacity of 64 units we use a cost of 150. This number captures the economy of scale. The cost of a unit-card is $\frac{150-114}{64-48} = 2.25$. This is 1.9% of the base ADM cost. The base cost of the ADM system of the example chapter 3 is \$23,575, the unit-card cost is $1600/4 = 400$. Thus the unit-card cost for the example is 1.6% of the base price, and we can conclude that our ADM price preserves the economy of scale effect accurately.

Our test data reflect the cost structure and the nature of the data in actual telecommunications networks. The ADM size allows to include some of the nodal demands simultaneously, but not all the demands. Thus there is a need for an algo-

rithm to pack the demands. The ADM costs capture the economy of scale effect, and the ADM cost to interconnection cost ratio represents the real-life situation when it may not be profitable to interconnect the smaller demands. The demand structure captures network hierarchical architecture, the mesh topology for the hubs, and the star topology for hub-CO clusters.

6.2 Details of Implementation

The heuristics are implemented in C language on a Sun SPARC station at the Operations Research Center. The code is included in the Appendix D. The C language was chosen for its speed and flexibility as compared with mathematical packages like Mathematica and Maple. The heuristics implementation reflects our modular approach of Interconnected Ring Network design problem. Different subroutines can be combined to achieve the best results. Since the running time of each heuristic is negligible this can be done in an efficient real-time manner. The program structure is depicted at Figure 6-1.

A switch on the figure indicates that any of the connected modules can be used to generate a solution. The Data Generation layer is used to generate a test problem. The Initial Solution layer creates a starting feasible solution. The Swap and the Interconnection layers improve the solution. The *.dat* entities are the files created by the corresponding modules.

The data related to demand grouping are kept in an array of structures *node*. For each demand, this structure contains the type of allocation (inside a ring or interconnect) and the ring it is grouped on. For the same pair d_{ij}, if a part of the calls is routed differently (bifurcated routing) a new record is created.

To confirm performance of the algorithm we use GAMS optimization package on the SUN station to solve small size problems of 8 nodes to optimality. The GAMS program is included in Appendix C. Our computational experience suggests that even for a small size problems the GAMS solution time can be significant.

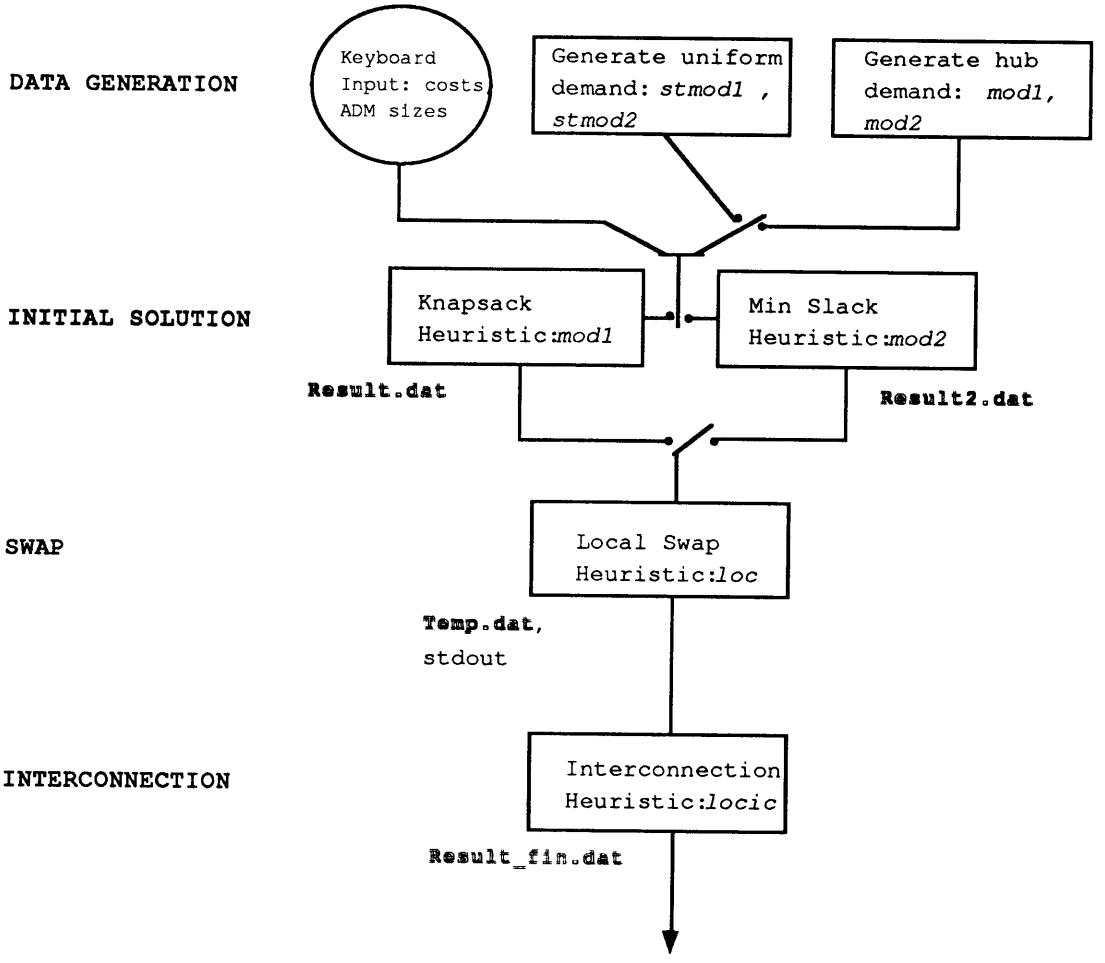


Figure 6-1: Program Structure

6.2.1 A Lower Bound of the [IRD]

To assess the goodness of a heuristic solution it is important to know a lower bound on the optimal solution. Small-size problems can be solved to and the heuristic solution for those problems can be compared to the optimal solution. Appendix C contains a sample GAMS output for an eight-node problem. The package uses branch and bound procedure to solve mixed integer models. By default the procedure stops whenever the system has performed 10,000 iterations, or the optimality has been reached, or a gap between the integer solution and the best relaxation generated by the branch and bound is within 0.1. By setting the gap to an arbitrarily small value, for example 0.0001, and increasing the number of iterations, we can solve small size (8 nodes) problems to optimality, or almost to optimality. With bigger size problems there are a number of limitations: the memory limit may be reached, or the allocated iteration resource exhausted. Even if this is not the case, it takes a great amount of time to produce an acceptable integer solution on GAMS. Even one of our 8-node examples required more than 1 million iterations and still was not able to meet the optimality requirements.

The problem's LP relaxation can be used to generate the bound. Another approach is to solve [**IRD**] using GAMS mixed integer model. By setting parameter "optcr" to some relatively large fraction, such as 0.3, we indicate that the solution can stop when the gap between the best integer solution and the best relaxation is 30%. We can use the best relaxation value for a lower bound.

Yet another way of generating a lower bound is to use the node inefficiency approach. If a nodal demand for a node i is D_{iR}, the minimum-cost number of ADM systems in the node can be found by solving the following LP.

[NADM]

$$\text{Minimize } V_i = \sum_k c_k x_k \tag{6.1}$$

subject to

$$\sum_k x_k B_k \geq D_i \tag{6.2}$$

$$x_k \quad \text{integer} \tag{6.3}$$

x_k is an integer variable equal to a number of size k ADM's installed at the node i, c_k is the cost of the ADM size k, B_k is the capacity of the ADM k, and $D_i = \sum_j d_{ij}$ is the total nodal demand of the node i. Constraint (6.2) is a multiple knapsack constraint. **[NADM]** can be solved quickly by modifying the algorithm used for a knapsack problem with 0-1 variables [AMO93, page 127].

The optimal solution to **[NADM]** gives the minimum number of ADM systems for the node i in order to terminate the nodal demands. By summing the optimal costs V_i over all the nodes i a lower bound for **[IRD]** can be generated.

6.3 Computational Results for Small-Size Problems

In this section we present computational results for small size problems. We used 8-node randomly generated networks for a mesh demand topology and 2-hub star networks for hierarchical demand. The GAMS optimization software was used to solve the problems to optimality and obtain a performance measure. The sizes of the GAMS models range from 921 constraints and 481 variables to 14781 constraints and 7498 non-zero variables. 20 problems were generated, 10 for each topology, out of which 19 were solved within 1% of optimality by GAMS. One problem was not completed after three days and is considered an outlier.

The GAMS solution time varied from 3 minutes to 1 day. It is difficult to compare solution time for different problems, since the tasks were running in UNIX environment, at different priorities and with other tasks running at the same time. The solution time for the algorithm, using the Profitable Ring Heuristic **[IN1]** (Model 1) was 2 minutes 10 seconds for a batch of 10 problems. The program, using the Slack

Packing heuristic **[IN2]** produced a final solution within 30 seconds for the batch. It is clear that our algorithm is orders of magnitude faster than the corresponding GAMS model, and it is possible to use it for real-time solutions.

The tables below represent the results of the runs. IN1 corresponds to the Profitable Ring Selection initial heuristic **[IN1]**, IN2 uses the Slack Packing heuristic **[IN2]** to generate the solution.

Table 6-1 GAMS Solution for 8-node Uniform Demand

Problem #	Constraints, Variables	LP Relaxation	[NADM] Bound	GAMS Lower Bound	GAMS Integer Solution: Gap≤.01
1	921, 481	912	984	1467.36	1482
2	3621, 1857	912	1710	1603.7	3078
3	1425, 741	912	912	1443.8	1458
4	2045, 1057	912	948	1595	1611
5	924, 481	1052.9	1176	1740.7	1758
6	2045, 1057	990.5	948	1316.2	1329
7	923, 481	1045.5	1134	1797	1815
8	1428, 741	912	1056	1666.4	1824
9	918, 481	912	912	1241.7	1254
10	918, 481	928.9	948	1173.3	1185

[NADM] bound is a lower bound generated using the method of Section 6.2.1. If the value in the column "GAMS Lower Bound" equals the corresponding value in the column "GAMS Integer Solution", the problem was solved by GAMS to optimality.

Table 6-2 [IRD] Solution for 8-node Uniform Demand

Problem #	IN1 Initial Solution	IN1 After Swap	IN1 Final Solution	IN2 Initial Solution	IN2 After Swap	IN2 Final Solution
1	1848	1482	1482	1548	1482	1482
2	3222	2736	2727	3306	2850	2850
3	1812	1710	1632	1992	1482	1458
4	2154	1884	1815	2220	1710	1710
5	2310	1932	1932	2100	1818	1818
6	2052	1548	1443	1734	1596	1428
7	1938	1824	1824	1950	1854	1854
8	2508	2052	1935	2250	2070	2070
9	1470	1254	1254	1548	1254	1254
10	1620	1368	1368	1398	1254	1254

Table 6-3 GAMS Solution for 2-hub Star Demand

Problem #	Constraints, Variables	LP Relaxation	[NADM] Bound	GAMS Lower Bound	GAMS Integer Solution Gap$\leq$.01
1	2528, 1301	1482	1710	1699.2	1710
2	14781, 7498	1596	1824	1865.5	1884
3	3555, 1828	1026	1062	1242	1254
4	2776, 1431	1254	1176	1357	1368
5	5420, 2773	1140	1212	1254	1368
6	5420, 2773	1050	798	1050	1050
7	5421, 2773	1254	1368	1468.5	1482
8	1491, 778	684	756	906	912
9	14781, 7498	1596	1746	1849	1854
10	7072, 3609	1254	1404	1467.6	1482

Table 6-4 [IRD] Solution for 2-hub Star Demand

Problem #	IN1 Initial Solution	IN1 After Swap	IN1 Final Solution	IN2 Initial Solution	IN2 After Swap	IN2 Final Solution
1	1998	1824	1824	1710	1710	1710
2	2268	1938	1938	1938	1938	1938
3	1734	1254	1254	1482	1254	1254
4	1500	1500	1500	1368	1368	1368
5	1848	1482	1482	1482	1368	1368
6	1050	1050	1050	1140	1050	1050
7	1806	1596	1596	1596	1482	1482
8	1056	912	912	1026	912	912
9	2268	1938	1938	2442	1938	1869
10	1812	1482	1482	1596	1596	1596

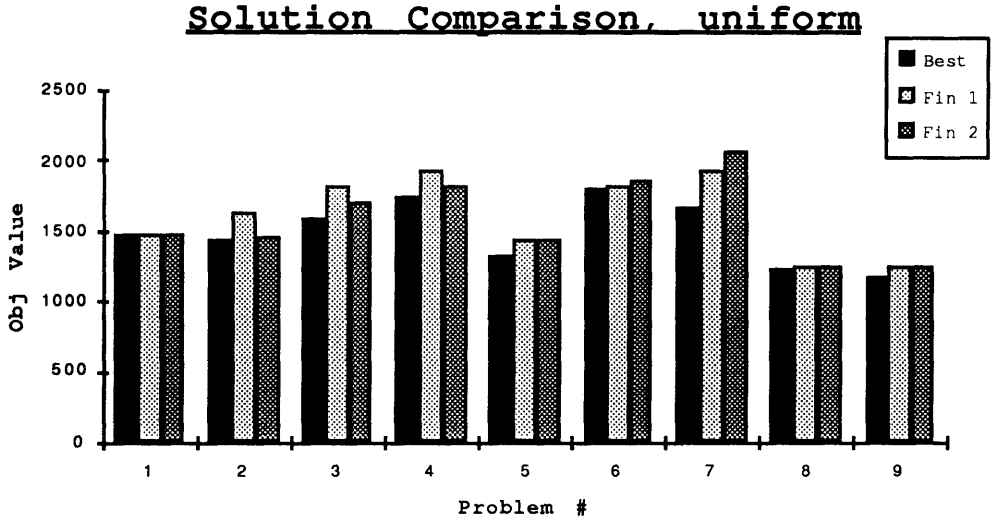


Figure 6-2: Heuristics Final Solution vs.Optimal Solution, Uniform Demand

19 test cases do not allow to reach statistically significant conclusions on the performance of the algorithm, but the test experiments provide a basis to judge the performance of the algorithm. Observe from the Tables 6-1 and 6-3 that relaxation generated using our approach from Section 6.2.1 is consistently better than the LP relaxation of the problem.

The results shown on Figure 6-2 and 6-3 suggest that initial solution does influence the final result, but there is no conclusive evidence that any of the initial heuristics preferable. Thus it is suggested that the algorithm is run with the all the available initial solutions and the best result is taken.

Figures 6-4 and 6-5 suggest that for the combined runs, i. e. running the algorithm for both initial solutions and selecting the best of the two, the results are within the 10 % optimality gap. The following table summarizes performance of the algorithm.

Table 6-5. Performance Data

	Gap LP-MIP Average	Gap MIP-Heur Average	Gap Improvement Average	Gap MIP-Heur 1 Best	Gap Mip-Heur 1 Worst
8unif_IN1	-	8.15%	21.73%	.99%	16.12%
8unif_IN2	-	6.38%	17.30%	.98%	24.22%
8unif best of 2	50.03%	5.30%	-	-	
2hub_IN1	-	2.68%	8.59%	0	18.18%
2hub_IN2	-	4.90%	15.78%	0	9.09%
2hub best of 2	13.66%	1.90%	-	-	
Total for 40 runs	-	5.30%	15.26%	-	

Table 6-5 summarizes the results for 8 nodes uniform and 2 hubs star architecture for two different kinds of initial solution heuristics. The average gap LP-MIP is the gap between the LP relaxation and the Mixed Integer Solution of the GAMS model. The average gap MIP-Heur is the gap between the lower bound generated by GAMS and the final solution produced by the heuristics. We consider it acceptable if the average gap is within the 10% of the optimal solution. This criterion is not satisfied in mesh problems with the Profitable Ring initial heuristic. However, this does not mean that the **[IN1]** heuristic is not justified. Firstly, it performs better with the star demand, and secondly, if the best of two solutions is used, we can achieve even more improvement. Then the gap for the mesh problems goes to 5.30%, down from 6.38%,

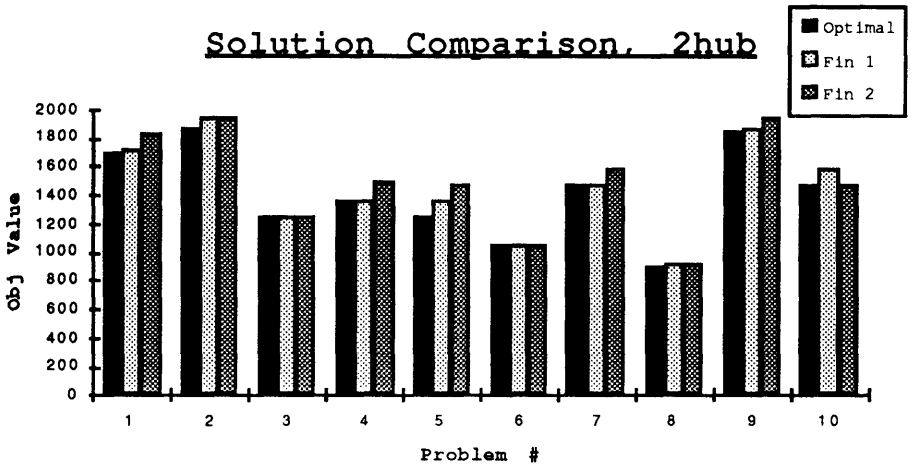


Figure 6-3: Heuristics Final Solution vs.Optimal Solution, Star Demand

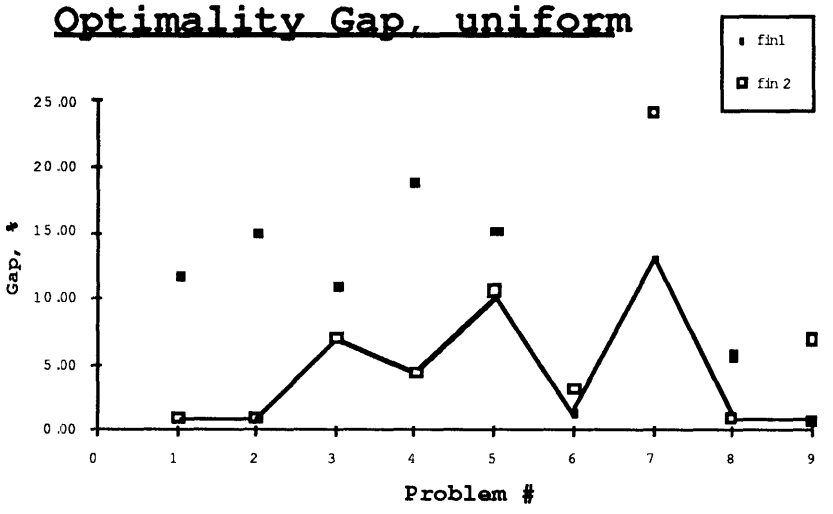


Figure 6-4: Optimality Gap, 8-Node Uniform Demand

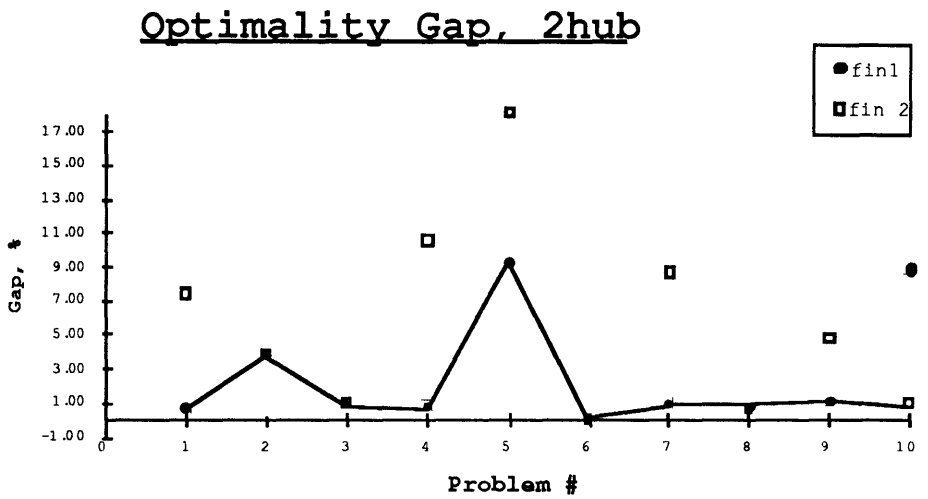


Figure 6-5: Optimality Gap, 2-Hub Star Demand

and the gap for the star demand is 1.90%, down from 2.68%. The gap calculated between the best of two solutions produced by IN1 and IN2 and the lower bound for **all** the runs is 5.30%. The Gap Improvement column shows that the **[SWAP]** and **[I/C]** heuristics contribute significantly to the improvement of the solution. The average improvement for all the problems and methods is 15.26%. In the best case there was an optimal solution produced by our algorithm, in the worst case the gap was 24.22%.

The table of results gives an insight into performance of the heuristic. When two initial solutions were used, and the best result was chosen, the average gap between the solution and the optimum of the **[IRD]** was 5.30%. When the demands have a more centralized structure, the algorithm performs better. The local improvement methods improve the solution of the heuristic significantly. For certain outliers, the optimality gap can be significant 24.22

The results of the tests suggest that our algorithm delivers an efficient and fast solution to the Interconnected Ring Network design problem.

6.4 Large Size Problems

We tested the performance of our algorithm on randomly generated problems containing 15 and 30 nodes for the uniform demands , and 4 and 6 hubs for the star demand. GAMS on SUN station was not able to solve the problems, since the files created in the process of the runs required too much memory. We used a lower bound described in Section 6.2.1. Table 7-6 summarizes running time of the heuristics. For all the test sets, the running time is given for a **batch** of 10 problems. Note that the time given is the running time, not the CPU time. For some batches it is possible that other tasks were running on the same machine simultaneously, so that the running time given in the table is the **maximum** running time for the given batch.

Table 6-6 Large Problems Running Time

Problem Type And Model Used	Initial Solution, 10 Problems Running Time	Local Improvement, 10 Problems Running Time
15node, IN1	30 min [1]	15 s
15node, IN2	8 s	22 s
30node, IN1	10 min [2]	5 min 17 s
30node, IN2	12 s	4 min 20 s
4hub, IN1	4 min	8 s
4hub, IN2	5 s	7 s
6hub, IN1	6 min 15 s	18s
6hub, IN2	14 s	30 s

Table 6-7 contains the results for 15 node uniform demand problems.

Table 6-7 15node Uniform Demand Problems

Prob #	IN1 Init	IN1 Final	Improv %	IN2 Init	IN2 Final	Improv %	Gap, [NADM]
1	8016	7263	10.37	10134	8181	23.9	3966
2	7866	6531	20.44	8628	6963	23.4	3444
3	6876	6612	3.99	8100	6906	17.3	3336
4	7126	6462	10.28	7950	6205	27.4	3294
5	6992	5727	22.09	7992	6006	33.07	3330
6	9502	8241	15.30	9870	8041	22.7	3702
7	7350	6891	6.66	8106	7071	14.6	3450
8	7728	7248	6.62	9228	7728	19.4	3780
9	7812	7635	2.32	9606	6313	39.5	3708
10	6618	6015	10.02	7998	5937	34.7	3216

For the 15node problems average improvement of the objective value due to the local improvement methods was 10.81% for IN1 and 25.60% for the IN2.

Table 6-8 Result for 30node Uniform Demand Problems

Prob #	IN1 Init	IN1 Final	Improv %	IN2 Init	IN2 Final	Improv %	Gap, [NADM]
1	37,020	27,907	32.7	36,048	30,936	16.52	13,362
2	37,860	30,351	24.7	37,878	31,575	19.96	14,136
3	38,070	27,484	38.5	37,800	33,219	13.79	13,782
4	38,6108	30,030	28.6	39,246	32,346	21.33	14,538
5	35,250	26,047	35.5	35,598	30,444	16.93	13,080
6	34,512	26,067	32.4	35,850	27,987	28.10	12,252
7	35,226	27,886	26.3	37,700	32,409	16.33	12,870
8	38,970	27,780	40.3	39,462	36,087	9.35	13,548
9	38,070	28,230	34.9	38,076	30,243	25.90	13,626
10	34,860	27,400	27.2	35,634	30,729	15.96	12,606

For the 30node problems average improvement of the objective value due to the local improvement methods was 32.10% for IN1 and 18.42% for the IN2.

Table 6-9 Result for 4hub Star Demand Problems

Prob #	IN1 Init	IN1 Final	Improv %	IN2 Init	IN2 Final	Improv %	Gap, [NADM]
1	4648	4215	10.27	5598	5016	11.60	3960
2	5138	4362	17.79	5592	4497	24.35	3444
3	5864	5028	16.63	6384	4836	32.01	3336
4	3078	2970	3.64	2964	2862	3.56	2294
5	4263	3672	16.09	4434	2799	58.41	2330
6	4800	4338	10.65	5706	3936	44.97	3702
7	3904	3189	22.42	3900	3117	25.12	2450
8	3802	3426	10.97	3876	2250	72.27	1780
9	3762	3297	14.10	3660	2712	34.96	2508
10	5130	5130	0.00	5718	5244	9.04	3216

For the 4hub problems average improvement of the objective value due to the local improvement methods was 12.26% for IN1 and 31.63% for the IN2.

Table 6-10 Result for 6hub Star Demand Problems

Prob #	IN1 Init	IN1 Final	Improv %	IN2 Init	IN2 Final	Improv %	Gap, [NADM]
1	6840	6840	0.00	8898	6021	47.78	5210
2	6862	6519	5.26	8148	7413	9.92	5598
3	7248	6381	13.59	7674	6864	11.80	5292
4	9260	8028	15.35	9162	8586	6.71	6132
5	8094	7260	11.49	9138	7914	15.47	6594
6	9014	8232	9.50	10056	7950	26.49	6936
7	9544	7011	36.13	7188	6090	18.03	5142
8	9280	8136	14.06	8874	6966	27.39	6372
9	9960	8835	12.73	9576	8661	10.56	6594
10	9704	8325	16.56	9174	7938	15.57	6132

For the 6hub problems average improvement of the objective value due to the local improvement methods was 13.47% for IN1 and 18.97% for the IN2.

We observed the following properties of the final heuristics solution. For the uniform demand:

- For each ring there exist a virtual hub, i. e. a node that has communicates with almost every other node on the ring;
- Small nodal demands requiring 1 ADM system ($D_i \leq B_0$) tend to be placed on one ring only;
- Small demand pairs are interconnected if there is enough capacity.

For the star demand:

- The hubs are placed on the same ring (rings);

- Central Offices belonging to the same cluster are placed on the same rings with the corresponding hubs;
- There is virtually no interconnection in the network.

6.5 Conclusion

The thesis has presented the ideas behind the recent developments in telecommunications, both from technical and from planning/economic perspective. We described developments such as optical networks, multimedia, SONET standard, SONET topologies and survivablity. A hierarchical approach to a SONET network design and management is introduced.

We approached one of the practical problems that have arisen recently in telecommunications field. Even though there are other methods that have been proposed to design a ring network, for example [LAG93], in this work we treat the Interconnected Ring Network Design problem in more systematic way, in context of SONET-based network design.

We considered a number of models for problems associated with **[IRD]** and made inferences about computational complexities. An Interconnected Ring Network Design problem is introduced with the fiber costs. Model **[EXP1]** is created for network expansion with new services or new nodes. A problem of selecting the most profitable ring in the network **[PR]** is considered, as well as the efficient ADM to node assignment problem **[NADM]**.

We develop a fast and efficient algorithm for **[IRD]**. The algorithm is based on a number of heuristics. Whereas heuristics are not guaranteed to deliver the optimal solution, the test problems show that the average optimality gap for our algorithm is 5.20% for the problems with uniform demands and 1.90% for the problems with star demands. Moreover, the algorithm is cheap, since it does not require heavy computational resources and can be implemented on a personal computer; it is also fast, so that the problem can be solved in real time.

There is significant work ahead in the telecommunications field for operations re-

searchers. Virtually every day a new practical problem arises. Among the new issues are digital superhighways, requiring cost efficient network expansion plans. Optimal routing on the BSHR ring remains an interesting problem. Lightwave routing and routing using parallel DCS systems are among the newest areas of interest. To continue our work, efficient algorithms have to be created for the operational problems for the ring networks. Stochasticity of the demands may provide for more efficient capacity allocation. Network sensitivity to more than one link failure may be considered.

In general, while some of the problems provide no new theoretical insight, fast and efficient algorithms will prove to be very cost-effective for the telecommunications industry and will enhance the value of approach of operations research in this field.

Appendix A

SONET Carrier Signal Rates

CCITT Signal	Level, Mbit/s	SONET Signal	Level, Mbit/s
DS1	1.544	OC1	51.84
DS2	6.312	OC3	155.52
DS3	44.736	OC9	466.56
		OC12	622.08
		OC18	933.12
		OC24	1244.16
		OC36	1866.24
		OC48	2488.32

Appendix B

A Bound on the Maximum Number of Rings in an Optimal [IRD] Solution

Consider the formulation of the Interconnected Ring Network problem given in Chapter 3. The number of constraints (3.4) depends on the proposed number of rings. Theoretically, the number of rings can be as big as $\sum_i D_i$, where D_i is the total nodal demand of the node i. Thus even for a small size problem with 8 nodes, average demand of 4 between a pair of nodes, and a 25% of all the nodes communicating, the number of rings can be as big as $(\frac{8\times7}{2\times4}) \times 4 = 28$. If one is to solve the problem to optimality using an available optimization package, the maximum number of rings in the formulation increases greatly the running time. For example, in one of our 8-node problems, increasing the number of rings of each size from 4 to 7 increased the running time from 1 hour to 8 hours. This is not surprising, since the straightforward solution approach uses branch and bound technique, the speed of which exponentially depends on the number of variables. Since increasing the number of rings increases the number of variables correspondingly, the running time increases.

For practical purposes it is important to have a better upper bound on the number of rings used in the optimal solution. In this section we show that given a set of nodal demands $\{D_i\}$ and a set of ADM sizes $\{B_i\}$, there exists an optimal solution to **[IRD]**

that uses no more than $\lceil 2 \times \sum_i D_i/B_i \rceil$ rings of size i.

We say that two non-empty rings r_1 and r_2 are merged in a ring r, if the resulting ring contains the union of all the demands belonging to the original rings.

Lemma 1 *Suppose for some feasible solution of* **[IRD]** *there exist two rings* r_1 *and* r_2 *of size* i *such that the sum of all the traffic crossing both rings is less than* B_i. *Than the two ring can be merged, and the cost of the resulting solution is less or equal than the cost of the initial solution.*

Consider the capacity constraints corresponding to the two rings:

$$\sum_i \sum_{j>i} y_{ijr_1r_1} d_{ij} + \sum_{s,s\neq r_1} \sum_{j>i} y_{ijr_1s} d_{ij} + \sum_{s,s\neq r_1} \sum_{j>i} y_{ijsr_1} d_{ij} \leq B_{r_1} \tag{B.1}$$

$$\sum_i \sum_{j>i} y_{ijr_2r_2} d_{ij} + \sum_{s,s\neq r_2} \sum_{j>i} y_{ijr_2s} d_{ij} + \sum_{s,s\neq r_2} \sum_{j>i} y_{ijsr_2} d_{ij} \leq B_{r_2} \tag{B.2}$$

Since $B_{r_1} + B_{r_2} \leq B_i$, we can add B.1 to B.2 and the inequality sign is not violated. Let $y_{ijrr} = y_{ijr_1r_1} + y_{ijr_2r_2} + y_{ijr_1r_2} + y_{ijr_2r_1}$, $y_{ijrs} = y_{ijr_1s} + y_{ijr_2s}$ and $y_{ijsr} = y_{ijsr_1} + y_{ijsr_2}$ for all $s \neq r_1, r_2$. Set $x_{ir} = \max(x_{ir_1}, x_{ir_2})$. Substitute the new constraint and new variables into **[IRD]** constraint set. The new ring is feasible, since at least one x_{ir} is non-zero. The new objective function is changed by

$$(\sum_i c_r x_r - \sum_i c_{r_1} x_{r_1} - \sum_i c_{r_2} x_{r_2}) - a(y_{ijr_1r_2} + y_{ijr_2r_1}) \tag{B.3}$$

Since the nodes of r are the union of r_1 and r_2, the expression (B.3) is less or equal than zero, and so the cost of the solution with the two rings merged is no greater than the cost of the initial solution.

We will now prove the proposition. Let $C_r = \sum_i \sum_{j>i} y_{ijrr} d_{ij} + \sum_{s,s\neq r} \sum_{j>i} y_{ijrs} d_{ij} + \sum_{s,s\neq r} \sum_{j>i} y_{ijsr} d_{ij}$ - the traffic on a ring r ($C_r \leq B_r$). Note, that for any feasible solution $\sum_r C_r \leq \sum_j D_j$, where D_j is the total nodal demand for the node j. This is true, because the number of demands in the network is $\frac{1}{2} \sum_j D_j$. In the worst case, when all the demands are interconnected, each demand is accounted at two rings. Thus $\sum_r C_r \leq 2 \times \frac{1}{2} \sum_j D_j = \sum_j D_j$.

Suppose the optimal solution has N_i rings of size B_i, and

$$N_i > 2\lceil \sum_i D_i/B_i \rceil \tag{B.4}$$

If there are two rings i, j such that the sum of their capacities $C_i + C_j \leq B_i$, these rings can be merged according to lemma 1 such that the new cost function is less or equal than the old one.

Suppose that there are no two rings i, j such that $C_i + C_j \leq B_i$. In other words, for all the ring pairs i, j, $C_i + C_j > B_i$. We add up all the inequalities. Each ring C_i is present in $N - 1$ inequalities. The total number of inequalities is $\binom{N}{2} = \frac{N(N-1)}{2}$. Thus we have

$$(N-1)\sum_i C_i > \frac{N(N-1)}{2} B_i \tag{B.5}$$

Dividing both parts by $(N-1)$ (from B.4, $N > 1$):

$$\sum_i C_i > \frac{N}{2} B_i > 2 \times \frac{\sum_i D_i}{B_i} \frac{B_i}{2} = \sum_i D_i \tag{B.6}$$

This is a contradiction because as noted before, $\sum_i C_i \leq \sum_j D_j$. Thus there exist two rings such that $C_i + C_j \leq B_i$, and therefore the merge is possible. We can continue the merging operation as long as B.4 holds, for then we can always show that there exist two rings that we can merge.

Q. E. D.

Appendix C

Solution Example

In the section we show an example of data and solution to the Interconnected Ring Network Design Problem. Figure C-1 illustrates the data. Arcs on the graph indicate existence of demand for nodes represented by the corresponding vertices, and the weights of the arcs equal to the corresponding nodal demands. Figure C-2 shows the solution created by the heuristics. We chose the best out of two final solutions. GAMS listing and the listing created by the heuristics is provided.

The following is the example of a sample solution, solved both by GAMS and by the heuristic **[IN 2]** was used for the initial solution.

```
*********************************************
**  GAMS Solution :1329       **
**  Best integer possible: 1316       **
**      **
*********************************************

Problem 6 -- 8 nodes  Density - "uniform"
---    2045 rows, 1057 columns, and 5977 non-zeroes.


Started     11.20 12-18
```

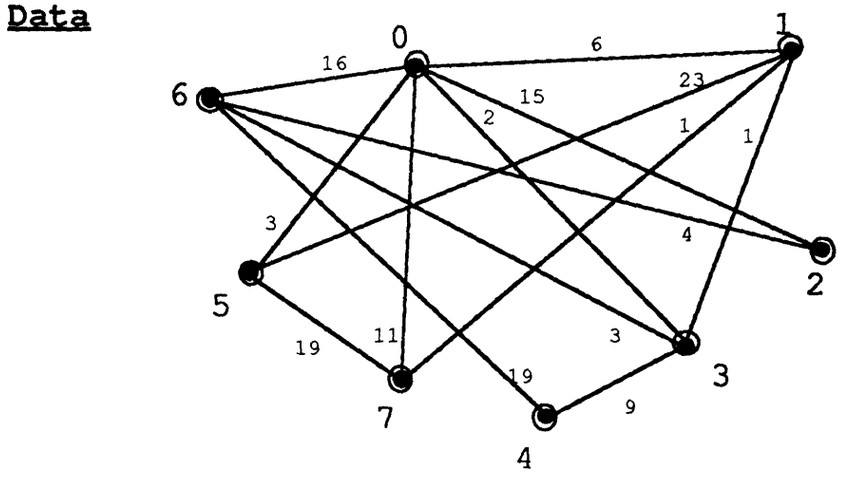


Figure C-1: Sample Data

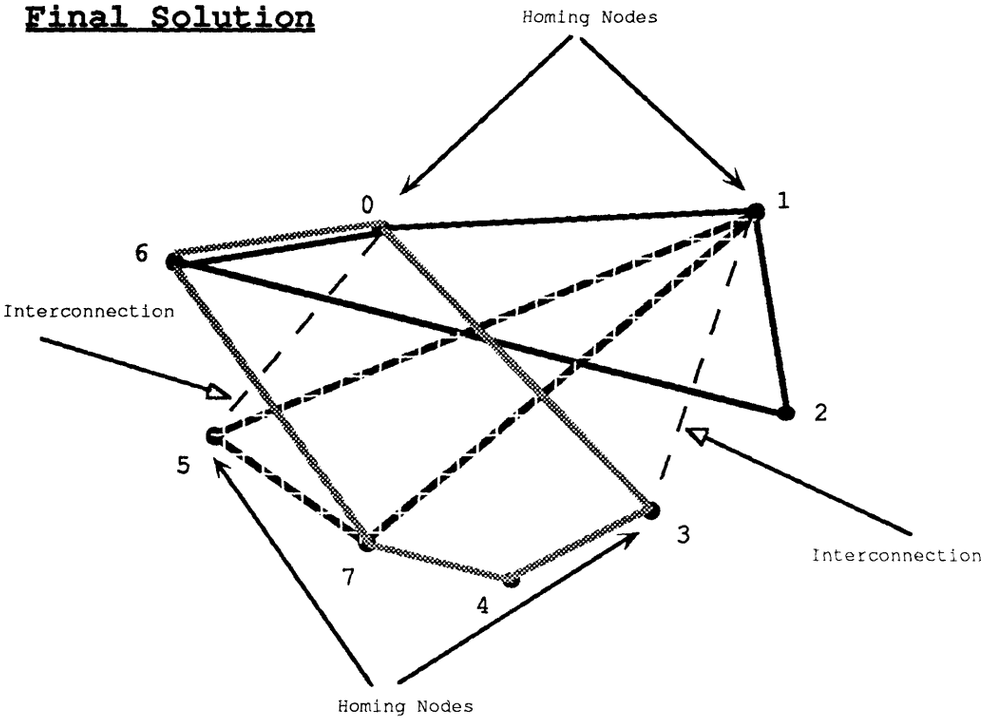


Figure C-2: Heuristics Solution for the Sample Data

```
Finished    11.27 12-18
Reniced

B1=48 B2=64
a1=114 a2=150
i/c=15

rings 0-2:48; 3-4:64

Solution satisfies tolerances.

LP Relaxation :          990.545455    (326 iterations)
MIP Solution  :         1329.000000    (6951 iterations, 327 nodes)
Final LP      :         1329.000000    (0 iterations)

Best integer solution possible :        1316.158723
Relative gap                   :         0.00975663

VAR Y



0.1.4.4      .        1.000     1.000      .
0.2.0.0      .        1.000
0.3.0.1      .        1.000
0.5.4.4      .        1.000     1.000      .
0.6.0.0      .        1.000     1.000
0.7.4.4      .        1.000
1.3.4.1      .        1.000
1.5.4.4      .        1.000
1.7.4.4      .        1.000
```

```
2.6.0.0       .        1.000
3.4.1.1       .        1.000
3.6.1.1       .        1.000
4.6.1.1       .        1.000
5.7.4.4       .        1.000
```

```
**********************************************
**  Heuristic Solution to[IRD]      **
**  Mod 2 Used to Create an Initial Solution **
**  Obj: 1428  Optimality Gap: 8.51%      **
**      **
**********************************************

INITIAL SOLUTION

PROBLEM 6
0.1 6
0.2 15
0.3 2
0.5 3
```

```
0.6 16
0.7 11
1.3 1
1.5 23
1.7 1
2.6 4
3.4 9
3.6 3
4.6 19
5.7 19
Cost:1734.000000


AFTER SWAP SOLUTION


EXITING: all set problem 6
Y[-0][-1]=-6 on ring -0
Y[-0][-2]=-15 on ring -0
Y[-0][-3]=-2 on ring -0
Y[-0][-5]=-3 on ring -1
Y[-0][-6]=-16 on ring -0
Y[-0][-7]=-11 on ring -2
Y[-1][-3]=-1 on ring -0
Y[-1][-5]=-23 on ring -1
Y[-1][-7]=-1 on ring -1
Y[-2][-6]=-4 on ring -0
Y[-3][-4]=-9 on ring -2
Y[-3][-6]=-3 on ring -2
Y[-4][-6]=-19 on ring -2
```

```
Y[-5][-7]=-19 on ring -1
5 nodes on ring 0 and bw=44
4 nodes on ring 1 and bw=46
5 nodes on ring 2 and bw=42
0 nodes on ring 3 and bw=0
0 nodes on ring 4 and bw=0
 cost=1596.000000


FINAL SOLUTION

Y[0][1]=6 on ring 0
Y[0][2]=15 on ring 0
Y[0][3]=2 on ring 2
Y[-1000][5]=3 on ring 1
Interconnected to 0
Y[0][6]=16 on ring 0
Y[0][7]=11 on ring 2
Y[1][-1000]=1 on ring 0
Interconnected to 2
Y[1][5]=23 on ring 1
Y[1][7]=1 on ring 1
Y[2][6]=4 on ring 0
Y[3][4]=9 on ring 2
Y[3][6]=3 on ring 2
Y[4][6]=19 on ring 2
Y[5][7]=19 on ring 1
Y[0][-1000]=3 on ring 0
Interconnected to 1
Y[-1000][3]=1 on ring 2
```

```
Interconnected to 0
4 nodes on ring 0 and bw=47
3 nodes on ring 1 and bw=46
5 nodes on ring 2 and bw=45
0 nodes on ring 3 and bw=0
0 nodes on ring 4 and bw=0
 cost after i/c=1428.000000
```

Appendix D

C Code

This appendix contains C code. The program *mod1.c* creates an intial solution using **[IN1]**. The program *mod2.c* creates an initial solution using **[IN2]**. The program *locbup.c* uses the heuristics **[SWAP]** and **I/C] to improve the solution.**

Mod1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>

#define RAND_MAX  (int)0x7fffffff
#define LAGR_ITERATIONS 1

main()

{
/*********************************************************
Variables: N - the dimension of the demand matrix
d - demand matrix
D - total node demand
```

```
z - optimal solutions for the knapsack dynamic programming
c - costs for Lagrangean
a - ADM cost
x - answers to knapsack  X - current best
y - 1 if an arc i,j is on the ring Y - the current best
mu - dual prices
intc - interconnection cost
L - current best obj
Un - an array of uncovered nodes (they are unprofitable)

The data are stored in a file result.dat
*******************************************************/
struct dlist{
  struct dlist *next;
  int i;
  int j;
  int n;
  int d;
  int r;
  int del_flag;  /* set to one if the demand belongs to a ring */
} dex;

struct dlist *head, *current, *new;

int tot_rings, tot_nodes, bw, rsize;
float cost, rcost;
int N_of_prbl=10, n_of_nodes, n_bands, B[100], band, Curr_band;
int d[100][100], y[250][250], x[1000], Y[250][250], X[1000];
int N=30, n_demands;
```

```
int Dprobl[1000], D[1000], iter, D_var;
int total_nodes, tot_demand;
float tot_cost, ring_profit;
float a[100], intc, L, L_curr, step, cc;
int i, ii, j, k, l, n, min, i0,i1, sum, lagr_count;
int flag, profit_flag, unc_flag;
float p_talk;
float mu[250][250], pi[250][250], c[1000], z[1000][1000], Z;
FILE *fp, *bar;
char record[BUFSIZ], buffer[BUFSIZ];

/* Data entry */
printf("Enter the number of bands\n");
scanf("%d",&n_bands);
printf("Enter cap-ty B= ");
for(i=0; i<=n_bands-1; i++)   scanf("%d",&B[i]);
printf("\n");
/*for(i=0; i<=n_bands-1; i++)   printf("%d ",B[i]);*/

printf("Enter the ADM cost a = \n");
for(i=0; i<=n_bands-1; i++)   scanf("%f",&a[i]);
printf("\n");
/*for(i=0; i<=n_bands-1; i++)   printf("%f ",a[i]);*/
printf("\n");
printf("Enter the i/c cost intc = ");
scanf("%f", &intc);
printf("\n");

total_nodes = 0;
ring_profit = 0;
```

```
/* Open a file to store results & gams data */
if((fp=fopen("result.dat","w")) == NULL)
{
printf("Can't open result.dat\n");
exit(1);
}
if((bar=fopen("barry.dat","w")) == NULL)
{
printf("Can't open barry.dat\n");
exit(1);
}

/*GENRATE PROBLEMS N_of_prbl TIMES, SOLVE EACH TIME AFTER GENERATE */
for(ii=0; ii<=N_of_prbl-1; ii++)
{
  sprintf(record,"PROBLEM %d\n\n",ii);
  if(fputs(record,fp)==EOF)
  {
    printf("Error on writing to a file\n");
    exit(1);
  }


/* Generate dem. matrix. This method generates av.density demand mtrx.
 Later, make it a subr, and let it generate also a low-dens and a high-dens.
Possible sizes: 20, 50, 100 */

for ( i=0; i<=N-1; i++)
{
```

```
for(j=0; j<=i; j++)
{
k = rand();
if (k<=RAND_MAX/2)
{

d[i][j] = 1+ (int)(1.0*B[0]/2*rand() / (RAND_MAX + 1.0));
d[j][i] = d[i][j];
  }
else
{
d[i][j] = 0;
d[j][i] = 0;
}
}
} /* End of generating dem mtrx */


/* Zero diagonal elements */
for (i=0; i<=N-1; i++)
d[i][i] = 0;


/* On-screen print & store to barry, also store non-zero demands to dlist */
head =  (struct dlist *) malloc(sizeof(dex));
current =head;
n =0;

printf("The Demand Matrix \n");
for(i=0; i<=N-1; i++)
```

```
{
for(j=0; j<=N-1; j++)
{
printf(" %d", d[i][j]);
if((d[i][j]!=0) && (j>i))
{
current->i =i;
current->j =j;
current->n =n++;
current->d =d[i][j];
current->del_flag =0;
current->next =(struct dlist *) malloc(sizeof(dex));
current->r =-1;
current =current->next;
sprintf(record,"%d.%d  %d\n",i,j,d[i][j]);
if(fputs(record,bar)==EOF)
{
printf("Error on writing to a file barry144\n");
exit(1);
}
}
          }
printf("\n");

}
current->next =NULL; /* the last record contains only NULL*/
n--; /* tot # of non-zero demands */

printf("\n");
```

```
/* Calculating the tot demand */
tot_demand = 0; /* need it to find the final cost */
for(current=head; current->next != NULL; current =current->next)
  tot_demand += current->d;
printf("\n ");

/* ORGANIZE A CYCLE TO CREATE NEW RINGS */
profit_flag = 0; /* ring_flag=1 if no more profitable rings */
unc_flag = 1; /* zero if there is a node uncovered */
tot_rings =-1;

for(; !(profit_flag*unc_flag); )
{
L = -100000000; /* current best obj */
tot_rings++;

for(band=0; band<=n_bands-1; band++) /* RUN FOR EVERY BAND */
{

/* Start Lagrangean relaxation */

lagr_count =0;  Z =10000000;   step =1.0;  Curr_band =-1;

n_demands = n;
/* initialize mu's and pi's: mu=0 */
for ( i=0; i<=n_demands; i++)
{
/*printf("OK\n");*/
for(j=0; j<=n_demands; j++)
{
```

```
mu[i][j] = 0;
  pi[i][j] = 0;
y[i][j] =0;
}
}

/* !!!! Remember to set mu[i][i] to 0 !!!! */

flag = 0;
k = 0; /* Number of iterations */

printf("CREATING A RING...\n");

for(;k<=LAGR_ITERATIONS && !flag; )
{

/* Generating costs AND D[I] */
for(current=head; current->next != NULL; current =current->next)
{
if(current->del_flag)
{
D[current->n] =B[band]+1; c[current->n] =0;
}
else
{
D[current->n] =current->d;
   c[current->n] = intc*D[current->n] - mu[current->i][current->j] -
pi[current->i][current->j];
  if(c[current->n]<0)  D[current->n] = B[band]+1;
}
```

```
}

/* if c[i] is negative, make D[i] big, such that element will never fit into
    the sack */

/* Doing a knapsack */

/* Initialize z */
/* First, fill in with -1 */
for(i=0; i<=n; i++)
{
for(j=0; j<=B[band]-1; j++) z[i][j]= -1;
}

/* fill in the 1st row - 0 if nothing fits */
for(j=0; j<D[0]-1; j++)  z[0][j] = 0;
for( ; j<=B[band]-1; j++)  z[0][j] = c[0];

/* And the rest of z */
for(i=1; i<=n; i++)
{
for(j=0; j<D[i]-1; j++)  z[i][j]=z[i-1][j];
if(c[i]>z[i-1][j] && D[i]<=B[band])  z[i][j] = c[i];
if(c[i]<=z[i-1][j] && D[i]<=B[band])  z[i][j] = z[i-1][j];
j++;
for( ; j<=B[band]-1; j++)
{
if(z[i][j]==-1)
{
if(z[i-1][j] > z[i-1][j-D[i]] + c[i])
```

```
z[i][j] = z[i-1][j];


else

  z[i][j] = z[i-1][j-D[i]] + c[i];


}
}

} /* End knapsack dynamic */

/* CHANGE THE STEP SIZE IF HAS HAD 5 CONSECUTIVE DECREASES OF LAGR OBJ */
if(Z>z[n][B[band]-1])
{
Z =z[n][B[band]-1];
lagr_count++;
}
else  lagr_count =0;

/*printf(" VALUE = %f \t",z[N-1][B[band]-1]);*/

/* Trace back to find the optimal solution */
j = B[band]-1;
for(i=n; i>=0; i--)
{
if(z[i][j]==z[i-1][j])  x[i] = 0; /* x-of the knps*/
else
```

```
{
x[i] = 1;
j -=D[i];
if(j==-1)  j=0;
}
}

/* Current y's, x is just a label*/
for(current=head; current->next != NULL; current =current->next)
{
   if(x[current->n]==1)  y[current->i][current->j] =1;
   else  y[current->i][current->j] =0;
}

/* X-DEPENDING ON THE CORRESPONDING COST */
for(i=0; i<=N-1; i++)
{
cc =0;
for(j=i+1; j<=n; j++)  cc +=mu[i][j];
for(j=0; j<i; j++)  cc +=pi[j][i];
cc -=a[band];
if(cc>=0)  x[i] =1;
else  x[i] =0;
}

/* check the slackness and feasibility */
i1 =1;
for(i=0; i<=n; i++)
{
for(j=i+1; j<=n; j++)
```

```
{
if(mu[i][j]*(y[i][j]-x[i])==0 && y[i][j]<=x[i] && pi[i][j]*(y[i][j]-
x[j])==0 && y[i][j]<=x[j])  i0 =1;
else  i0 =0;
i1 *=i0;
}

}

k++;

/* new mu's */
if(lagr_count>=3)  step =step/2.0;
if(!i1)
{
for(i=0; i<=N-1; i++)
{
for(j=i+1; j<=N-1; j++)
{
mu[i][j] = mu[i][j]+ 1.0*step* (y[i][j] -x[i]);
if(mu[i][j]<=0)  mu[i][j] = 0;
pi[i][j] = pi[i][j]+ 1.0*step* (y[i][j] -x[j]);
if(pi[i][j]<=0)  pi[i][j] = 0;

}
}

}
else  flag = 1;  /* flag=0 if not optimal (can'y leave the cycle) i1=0
if no opt-slckn or feas for a current eq-n
```

```
flag=1 iff all i0 are ones */

/* Generate a current best */
L_curr = 0;
for(current=head; current->next != NULL; current =current->next)
{
if(y[current->i][current->j]==1)
{
L_curr += 1.0 * intc * current->d;
x[current->i] = x[current->j] = 1;
}
}
for(i=0; i<=n; i++)  L_curr -= 1.0*x[i]*a[band];

if(L_curr>L)
{
Curr_band =band;
L = L_curr;
iter = k;
for(i=0; i<=n; i++)  X[i] = x[i];
   for(i=0; i<=n; i++)
{
for(j=i+1; j<=n; j++)
Y[i][j] = y[i][j];
}
}



}  /* End of solving Lagrangean */
```

```
/*if(L_curr==0) intc=0;*/

} /* END OF DOING BANDS */

/* DELETE THE G-ed DEMANDS */
for(current=head; current->next != NULL; current =current->next)
{
if(Y[current->i][current->j]==1)
{
   current->del_flag =1; current->r =tot_rings;
}
}

/* Print the answer */
printf("\n\nOBJ is %f\n",L);
printf("Found on Iteration %d",iter);
printf("\n\n");

ring_profit += L; /* ????? */


sum =0;
for(current=head; current->next != NULL; current =current->next)
if( !current->del_flag) sum +=current->d;

if(sum<=B[n_bands-1])
{
tot_rings++;
for(current=head; current->next != NULL; current =current->next)
{
```

```
if( !current->del_flag)
{
current->r =tot_rings;  current->del_flag =1;
}
}
}

if(L>0)  profit_flag =0;
else
{
profit_flag = 1;
/* SEE IF ALL THE NODES WERE GROUPED */
unc_flag =1;
for(current=head; current->next != NULL; current =current->next)
if((int)current->del_flag!=1) unc_flag =0;

/*delete the node if it's been grouped: make it infeasible for the
knapsack constraint */
}



} /* end of ring creating */

tot_rings++;


/*CALCULATE THE COST */
tot_nodes = cost =0;
for(i=0; i<=tot_rings-1; i++)
```

```
{
bw =0;
for(j=0; j<=N-1; j++)  x[j] =0;
for(current=head; current->next!= NULL; current =current->next)
{
if(current->r==i)
{
x[current->i] = x[current->j] =1;
bw +=current->d;
}
}

for(rsize=0; bw>B[rsize]; rsize++) ;
sum =0;
for(j=0; j<=N-1; j++)  sum +=x[j];
rcost =a[rsize]*sum;
cost +=rcost;
printf("%d nodes on ring %d and bw=%d\n",sum,i,bw);
tot_nodes +=sum;
}
printf(": cost=%f\n",cost);
sprintf(record,"Cost:%f\n",cost);
if(fputs(record,fp)==EOF)
{
printf("Error on writing to a file\n");
exit(1);
}
```

```
/* PRINT IT TO RESULT */
for(current=head; current->next!= NULL; current =current->next)
{
sprintf(record,"Y[-%d][-%d]=-%d on ring -%d\n", current->i, current->j,
 current->d, current->r);
/*sprintf(record,"%d.%d %d \n", current->i, current->j, current->d);*/


if(fputs(record,fp)==EOF)
{
printf("Error on writing to a file\n");
exit(1);
}
}



/*FREE THE MEMORY FOR A NEW DEMAND LIST OF THE NEXT ii PROBLEM  */
for(current=head; current->next != NULL; current =new)
{
new =current->next;
free(current);
}

} /*END FOR ii PROBLEM */


fclose(fp);
fclose(bar);
```

```
} /*END OF MAIN */
```

Mod2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>

#define RAND_MAX  (int)0x7fffffff
#define MAX_RINGS 200
#define N 15  /* # of nodes */

main()

{
/*********************************************************
Model2 - max slack heuristic

Variables: N - the dimension of the demand matrix
d - demand matrix
D - total node demand
a - ADM cost
x - answers to knapsack  X - current best
y - 1 if an arc i,j is on the ring Y - the current best
intc - interconnection cost
```

```
The data are stored in a file result2.dat
*********************************************************/
struct dlist{
  struct dlist *next;
  int i; /* d[i][j] */
  int j;
  int r; /*ring that d[i][j] belongs to; r=1..; set initially to 0 */
  int d;
  int del_flag;  /* set to one if the demand belongs to a ring */
} dex;

struct dlist *head, *current, *new;

int N_of_prbl=10, n_of_nodes, n_of_bands, B[100], band;
int d[100][100], I, J, slack;
int r_cap[MAX_RINGS], rI[MAX_RINGS], rJ[MAX_RINGS], rcom[2*MAX_RINGS],
tot_rings, ring, bw;
int D[100], ceil;
int tot_nodes, tot_demand, rsize;
int x[N];
float rcost,cost;
float a[100], intc;
double dd;
int i, ii, j, k, l, n, max, i0,i1, sum;
int unc_flag, flag, cflag, r_flag;
int rflag; /* 0-no suitable ring, 1-1node ring, 2-2node ring */
float p_talk;
FILE *fp;
char record[BUFSIZ], buffer[BUFSIZ];
```

```
/* Data entry */
printf("Enter the number of bands\n");
scanf("%d",&n_of_bands);
printf("Enter cap-ty B= ");
for(i=0; i<=n_of_bands-1; i++)   scanf("%d",&B[i]);
printf("\n");

printf("Enter the ADM cost a = \n");
for(i=0; i<=n_of_bands-1; i++)   scanf("%f",&a[i]);
printf("ADM cost %f\n",a[0]);
printf("\n");
/*for(i=0; i<=n_bands-1; i++)   printf("%f ",a[i]);*/
printf("\n");
printf("Enter the i/c cost intc = ");
scanf("%f", &intc);
printf("\n");

/* Open a file to store results  */
if((fp=fopen("result2.dat","w")) == NULL)
{
printf("Can't open result2.dat\n");
exit(1);
}

/* Store INIT DATA */
sprintf(record,"bands-%d ",n_of_bands);
strcat(record,"cap's");
for(i=0; i<=n_of_bands-1; i++)
{
sprintf(buffer,"-%d ",B[i]);
```

```
strcat(record,buffer);
}
strcat(record," ADM_costs");
for(i=0; i<=n_of_bands-1; i++)
{
sprintf(buffer,"-%f ",a[i]);
strcat(record,buffer);
}
strcat(record," i/c_cost-");
sprintf(buffer,"%f \n",intc);
strcat(record,buffer);

if(fputs(record,fp)==EOF)
{
    printf("Error on writing to a file\n");
    exit(1);
}


/*GENRATE PROBLEMS N_of_prbl TIMES, SOLVE EACH TIME AFTER GENERATE */
for(ii=0; ii<=N_of_prbl-1; ii++)
{
  sprintf(record,"PROBLEM %d N-%d\n",ii, (int)N);
  if(fputs(record,fp)==EOF)
  {
    printf("Error on writing to a file\n");
    exit(1);
  }
```

```
/* Generate dem. matrix. This method generates av.density demand mtrx.
Later, make it a subr, and let it generate also a low-dens and a high-dens.
Possible sizes: 20, 50, 100 */

for ( i=0; i<=N-1; i++)
{
for(j=0; j<=i; j++)
{
k = rand();
if (k<=RAND_MAX/2)
{

d[i][j] = 1+ (int)(1.0*B[0]/2*rand() / (RAND_MAX + 1.0));
d[j][i] = d[i][j];
  }
else
{
d[i][j] = 0;
d[j][i] = 0;
}
}
} /* End of generating dem mtrx */


/* Zero diagonal elements */
for (i=0; i<=N-1; i++)
d[i][i] = 0;


/* On-screen print & store to barry, also store non-zero demands to dlist */
```

```
head =  (struct dlist *) malloc(sizeof(dex));
current =head;
n =0;

printf("The Demand Matrix \n");
for(i=0; i<=N-1; i++)
{
for(j=0; j<=N-1; j++)
{
printf(" %d", d[i][j]);
if((d[i][j]!=0) && (j>i))
{
current->i =i;
current->j =j;
n++;
current->d =d[i][j];
current->del_flag =0;
current->r =-1;
current->next =(struct dlist *) malloc(sizeof(dex));
current =current->next;
}
          }
printf("\n");

}
current->next =NULL; /* the last record contains only NULL*/
printf("Tot # of non-0 dem's %d, Max # of rings %d\n",n,MAX_RINGS);
printf("Running...\n");
 /*n - tot # of non-zero demands */
```

```
/* CALCULATE NODE DEM'S D[i] */
for(i=0; i<=N-1; i++)
{
D[i]= 0;
for(j=0; j<=N-1; j++)
D[i] = D[i] + d[i][j];
}

for(i=0; i<=N-1; i++)
  printf("D[%d]=%d \n ",i,D[i]);

/* INIT ALL RING CAP'S TO 0 */
for(i=0; i<=MAX_RINGS-1; i++)  r_cap[i] =B[n_of_bands-1];
tot_rings =0;

/* ORGANIZE A CYCLE TO CREATE NEW RINGS */
unc_flag = 0; /* zero if there is a node uncovered */

for(; !unc_flag; )
{

/* CALCULATE MIN(MAX SLACK) */
slack = 10000;
printf("Hey\n");
for(i=0; i<=N-1; i++)
{
for(ceil=0; D[i]>ceil*B[n_of_bands-1]; ceil++)   ;
if( ceil*B[n_of_bands-1] - D[i] <= slack && D[i]!=0)
{
slack = ceil* B[n_of_bands-1]  - D[i];
```

```
I =i;
}
} /* if all the demands for i are grouped, the ceil...-... is B[max], and
if there is an uncov node, its slack would be less - do not have explicitly to d

/* FIT CORRESPONDING DEMANDS IN THE RINGS */
flag =0; /*set to 1 if all the d[I][j] are grouped */
for( ; !flag; )
{
J = max = 0;
for(current=head; current->next!= NULL; current =current->next)
{
/* find the max d[I][J] */
if(current->d>max && !current->del_flag && (current->i==I || current->j==I)
 )
{
max =current->d;
if(current->i==I)  J =current->j;
else  J =current->i;
}
}

if (max==0) break; /* exit the cycle for this node if all the I-demands
 are grouped */

/* Is there a ring. containing I,J? */
i =0;
for(current=head; current->next!= NULL; current =current->next)
{
if( (current->i==I || current->j==I) && current->r!=-1)
```

```
{
rI[i] =current->r; /* list of all rings that I belongs to */
i++;
}
}
rI[i] =-1;
j =0;
for(current=head; current->next!= NULL; current =current->next)
{
if( (current->i==J || current->j==J) && current->r!=-1)
{
rJ[j] =current->r;
j++;
}
}
rJ[j] =-1;

/* SET r_flag, r_com contains the rings with the nodes I or J or both  */

r_flag =0;
k =0;
for(i=0; rI[i]!=-1; i++)
{
for(j=0; rJ[j]!=-1; j++)
{
if(rI[i]==rJ[j] && r_cap[rI[i]]>=d[I][J] )
{
rcom[k] =rI[i]; k++; r_flag =2;
}
}
```

```
}
rcom[k] =-1;
if(r_flag!=2)
{
for(i=0; rI[i]!=-1; i++)
{
if(r_cap[rI[i]]>=d[I][J])
{
rcom[k] =rI[i]; k++; r_flag =1;
}
}
for(i=0; rJ[i]!=-1; i++)
{
if(r_cap[rJ[i]]>=d[I][J])
{
rcom[k] =rJ[i]; k++; r_flag =1;
}
}
}
rcom[k] =-1;

/* FIND d[I][J] */
for(current=head; current->next!= NULL; current =current->next)
if( (current->i==I && current->j==J) || (current->j==I && current->i==J) )
  break;

/* PLACE THE DEMAND ON THE 'BEST' RING */
switch(r_flag)
{
case 0:
```

```
current->r =tot_rings;
r_cap[tot_rings++] = B[n_of_bands-1] - d[I][J];
current->del_flag =1;   break;

case 1: case 2:
max =0;
for(i=0; rcom[i]!=-1; i++)
{
if(r_cap[rcom[i]] - d[I][J] >= max)
{
ring = rcom[i];
max = r_cap[rcom[i]] -d[I][J];
}
}
current->r = ring;
current->del_flag =1;
r_cap[ring] -= d[I][J];  break;
}
} /* end for D[i] */

/* GET NEW D[i] */
cflag =0;
for(i=0; i<=N-1; i++) D[i] =0;
for(current=head; current->next!= NULL; current =current->next)
{
if(!current->del_flag)
{
  D[current->i] += d[current->i][current->j];
  D[current->j] += d[current->i][current->j];
  cflag =1;
```

```
}
}
if(!cflag)   unc_flag =1; /* end demand placing - all placed */
} /* end placing cycle */

/* PRINT IT TO RESULTS.DAT */

/*strcat(record,"\nProblem \n");*/
for(current=head; current->next!= NULL; current =current->next)
{
sprintf(record,"Y[-%d][-%d]=-%d on ring -%d\n", current->i, current->j,
 current->d, current->r);
/*sprintf(record,"%d.%d %d \n", current->i, current->j, current->d);*/


if(fputs(record,fp)==EOF)
{
printf("Error on writing to a file\n");
exit(1);
}
}

/*CALCULATE THE COST */
tot_nodes = cost =0;
for(i=0; i<=tot_rings-1; i++)
{
bw =0;
for(j=0; j<=N-1; j++)  x[j] =0;
for(current=head; current->next!= NULL; current =current->next)
{
```

```
if(current->r==i)
{
x[current->i] = x[current->j] =1;
bw +=current->d;
}
}

for(rsize=0; bw>B[rsize]; rsize++) ;
sum =0;
for(j=0; j<=N-1; j++)  sum +=x[j];
rcost =a[rsize]*sum;
cost +=rcost;
printf("%d nodes on ring %d and bw=%d\n",sum,i,bw);
tot_nodes +=sum;
}
/* Tot Demand */
sum =0;
for(i=0; i<=N-1; i++)
{
for(j=0; j<=N-1; j++)
sum +=d[i][j];
}

printf("  cost=%f\n",cost);
sprintf(record,"Cost:%f 2D:%d\n",cost,sum);
if(fputs(record,fp)==EOF)
{
printf("Error on writing to a file\n");
exit(1);
}
```

```
/*FREE THE MEMORY FOR A NEW DEMAND LIST OF THE NEXT ii PROBLEM  */
for(current=head; current->next != NULL; current =new)
{
new =current->next;
free(current);
}

}/* END ii PROBLEM */

fclose(fp);

printf("OVER!!\n");

} /*END OF MAIN */
```

Locbup.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <string.h>

#define RAND_MAX  (int)0x7fffffff
#define MAX_RINGS 200
#define N_of_Problems 10
#define N_of_Iterations 1
```

```
#define MAX_SIZE 4096
#define NONE 0
#define SIMPLE_ADD 1
#define SWAP_DEMAND 2
#define SWAP_PARTIAL 3
#define Nd 100


struct dlist{
  struct dlist *next;
  int i; /* d[i][j] */
  int j;
  int r; /*ring that d[i][j] belongs to; r=1..; set initially to 0 */
  int d;
  int ic; /* =r if the demand is interconnected to r, init=-1 */
  int to_node;
} dex;
struct dlist *head, *current, *new;

struct move_list {
int i;
int j;
int d;
int r;
int move_flag;
}  *move;

int n_of_bands, B[100], a[100], ic;
```

```
char *atoai(char *ptr1, int *array, int n);

int capacity(int ring);

void copy_ring(int *node_demand, int node, int ring_to_copy, struct dlist
 *Ring[]);

void set_move(int I1, int J1, int d, int ring, int flag);

struct dlist *check_merge(struct dlist *, int , int);

void delete_node(struct dlist *dem_ptr);

void rswap(struct dlist *ring[], int I, int I1, int J1);

void add_node(int i, int j, int r, int d, int i1);

float obj(int nrings, int N);


main()
{

/*****************************************************************
Local Search

*****************************************************************/

struct nlist{
int flag; /* 0 if the node has not been in a swap */
```

```
int r;
int d;
} nex;
struct nlist *node[Nd];

struct dlist *R_ring[100], *R1_ring[100], *addr;
struct dlist *Move[100];

int r[MAX_RINGS];

FILE *fp;
char record[BUFSIZ];
char *buf_ptr;

int nrings, min_nsystems;

int i, j, j1, ii, sum, dummy, k, kk, k1, k2, flag, J1, i3, i4, i5, i6;

int swap_flag; /*1 if all the nodes had swap attempts */
int I, Dir, R_accept, R_move, Cr, Cr1, Cr_new, Cr1_new, C[MAX_RINGS],
 Br[MAX_RINGS], ar[MAX_RINGS], R;
float inef;

int x[Nd], xadd[Nd], x1add[Nd], y[Nd], rdel, r1del, radd, r1add, r1dj , r1di;
int I1, D1, rai, raj, xrel, xrl[Nd], xrr1add[Nd], x1rec[Nd];
int k3, k4, c3, c4, c5, c6, it;
int move_profit, cur_move, move_flag;

int rnodes, r1nodes, raft_cost, rbef_cost, r1aft_cost, r1bef_cost,
 rnd[Nd], r1nd[Nd];
```

```
int tot_nodes, tot_rings, bw, rsize, N, cap_flag;
float cost, rcost, ic_cost;

int *D, *D_end, *D_addr; /* pointer to an array of D[node][ring],
 allocate dynamically */
int icflag, min;

/* READ THE DATA FROM THE result2.dat of result.dat */

if((fp=fopen("result2.dat","ra")) == NULL)
{
printf("Can't open result2.dat\n");
exit(1);
}

/*FIRST, THE COSTS */

fgets(record, BUFSIZ, fp);
if( ferror(fp) )
{
printf("Error reading result2.dat\n");
exit(1);
}

buf_ptr =record;
buf_ptr =atoai(buf_ptr,&n_of_bands,1);
printf("bands %d\n",n_of_bands);

buf_ptr =atoai(buf_ptr,B,n_of_bands);
```

```
for(i=0; i<=n_of_bands-1; i++) printf("B= %d ",B[i]);
printf("\n");
buf_ptr =atoai(buf_ptr,a,n_of_bands);
for(i=0; i<=n_of_bands-1; i++) printf("a= %d ",a[i]);
buf_ptr =atoai(buf_ptr,&ic,1);
printf("ic= %d \n",ic);

printf("\n");

/* FOR EACH PROBLEM ii */

for(ii=0; ii<=N_of_Problems-1; ii++)
{
fgets(record, BUFSIZ, fp); /* read N -- the number of nodes */
if( ferror(fp) )
{
printf("Error reading result2.dat\n");
exit(1);
}
buf_ptr =atoai(record,&N,1);

head =  (struct dlist *) malloc(sizeof(dex));
current =head;

/*READ THE RING RECORDS  */
for(; !feof(fp); )
{

fgets(record, BUFSIZ, fp);
if( ferror(fp) )
```

```
{
printf("Error reading result2.dat\n");
exit(1);
}

buf_ptr = record;
if(strstr(record,"Cost") != NULL)  break; /* stop reading if the end
of problem*/
buf_ptr =atoai(buf_ptr,&(current->i),1);
buf_ptr =atoai(buf_ptr,&(current->j),1);
buf_ptr =atoai(buf_ptr,&(current->d),1);
buf_ptr =atoai(buf_ptr,&(current->r),1);
current->ic =  current->to_node =-1;

current->next =(struct dlist *) malloc(sizeof(dex));
current =current->next;

}
current->next =NULL; /* the last record contains only NULL*/

for(i=0; i<=N-1; i++) node[i] = (struct nlist *)malloc(sizeof(nex));

/* ALLOCATE FOR move */
move = (struct move_list *)malloc(sizeof(struct move_list));

/* FOR EACH ITERATION */

for(it=0; it<=N_of_Iterations-1; it++)
{
```

```
/* FIND THE NUMBER OF RINGS */
nrings =0;
for(current=head; current->next!= NULL; current =current->next)
if(current->r>=nrings)  nrings = current->r;
nrings++; nrings +=6;  /* extra ring */

/* FIND NODES WITH 0 INEFFICIENCY AND SET THE FLAGS */

/* IF THERE IS A NODE NOT SET, DO THE SWAPS */

swap_flag =0;

/* INITIALIZE node struct */
for(i=0; i<=N-1; i++)
{
for(j=0; j<=nrings-1; j++)  r[j] =0;
node[i]->d = 0; node[i]->flag =0;
for(current=head; current->next!= NULL; current =current->next)
if(current->i==i || current->j==i)
{
r[current->r] =1;
node[i]->d +=current->d;
}
node[i]->r =0;
for(j=0; j<=nrings-1; j++)  node[i]->r +=r[j];
}

I = 10000;
for( ; !swap_flag && I!=-1; )
{
```

```
for(i=0; i<=N-1; i++)
{
for(j=0; j<=nrings-1; j++)  r[j] =0;
for(current=head; current->next!= NULL; current =current->next)
{
if(current->i==i || current->j==i)
  r[current->r] =1;
}
node[i]->r =0;
for(j=0; j<=nrings-1; j++)  node[i]->r +=r[j];
}


/* find the most inefficient node */
inef =0; I =-1;
for(i=0; i<=N-1; i++)
{
for(min_nsystems=0; node[i]->d > min_nsystems * B[n_of_bands-1];
 min_nsystems++)    ;
if(1.0*node[i]->r/min_nsystems-1> inef && !node[i]->flag)
{
I=i; inef=node[i]->r/min_nsystems-1;
}
}

/* if all the nodes swapped, break out */
if (I==-1)
{
 printf("\nEXITING: all set problem %d\n" ,ii);
```

```
 break; /* break the swap cycle */
}

/* DO THE SWAPS FOR I FOR ALL THE RINGS */

for(j=0; j<=nrings-1; j++)
{

rdel = r1del =-1;
radd = r1add = 1;

/* FIND Cr AND Dir FOR A GIVEN RING */
copy_ring(&Dir, I, j, R_ring);
move_profit =-1;
move_flag = NONE;

if(Dir>0)
{
/* SEE WHICH NODES TO DELETE FROM j */
  for(i=0; i<=N-1; i++)  x[i]=rnd[i] =0;

  for(k=0; R_ring[k] != (struct dlist *) NULL; k++)
    rnd[R_ring[k]->i] = rnd[R_ring[k]->j] =1;
  rnodes =0;
  for(k=0; k<=N-1; k++) rnodes +=rnd[k];

  x[I] =1;
  for(k=0; R_ring[k] != (struct dlist *) NULL; k++)
  {
J1 =-1;
```

```
if(R_ring[k]->i==I)  J1 =R_ring[k]->j;
else  if(R_ring[k]->j==I)  J1 =R_ring[k]->i;
flag =0;

if(J1!=-1)
{
for(kk=0; R_ring[kk] != (struct dlist *) NULL; kk++)
{
  if( (R_ring[kk]->i!=I && R_ring[kk]->j==J1) || (R_ring[kk]->j!=I
 && R_ring[kk]->i==J1) )
  {
     flag =1; break;

   }
}

if(!flag && J1!=-1)  x[J1] =1;
}
     }




  rdel =0;
  for(i=0; i<=N-1; i++) rdel +=x[i];

  Cr = capacity(j);

  R_accept = B[n_of_bands-1]- Cr + Dir;
```

```
/*LOOK THROUGH THE RINGS FOR THE BEST MOVE */
  for(j1=0; j1<=nrings-1; j1++)
  {

if(j1!=j)
{

/*COPY RING j1 INTO R1_ring*/
copy_ring(&dummy, -1, j1, R1_ring);


for(k4=0; R1_ring[k4] != (struct dlist *) NULL; k4++)
     r1nd[R1_ring[k4]->i] = r1nd[R1_ring[k4]->j] =1;
  r1nodes =0;
          for(k4=0; k4<=N-1; k4++) r1nodes += r1nd[k4];

Cr1 = capacity(j1);
for(c3=0; Cr>B[c3]; c3++) ;
for(c4=0; Cr1>B[c4]; c4++) ;/* for a */

/* a3 -cost of rold; a4 - cost of r1old */
R_move =Dir - B[n_of_bands-1] + Cr1;



/* IS THERE ENOUGH CAP ON R1 TO MOVE Dir ? */
if(Cr1+Dir<=B[n_of_bands-1])
{
r1del = radd= 0;
r1add =0;
```

```
for(i=0; i<=N-1; i++)  y[i] =1;
for(i=0; i<=N-1; i++)
{

  for(i3=0; i3<=N-1; i3++)
     xrl[i3]  =0;
  for(k3=0; R_ring[k3] != (struct dlist *) NULL; k3++ )
 if(R_ring[k3]->i==I || R_ring[k3]->j==I)  xrl[R_ring[k3]->i] =
xrl[R_ring[k3]->j] =1;

  for(k1=0; R1_ring[k1] != (struct dlist *) NULL; k1++ )
   if( (R1_ring[k1]->i==i && xrl[i]==1) || ( R1_ring[k1]->j==i &&
xrl[i]==1 ) )  y[i] =0;
}
for(i=0; i<=N-1; i++)
  if(xrl[i]*y[i]==1) r1add +=1;

/* calculate the ring capty, and use the corresponding cost */
for(c5=0; Cr-Dir>B[c5]; c5++);
for(c6=0; Cr1+Dir>B[c6]; c6++) ;
/* calculate the move's profit */
cur_move = rnodes*a[c3] + r1nodes * a[c4] - (rnodes-rdel+radd)*a[c5] -
(r1nodes-r1del+r1add) * a[c6];

/*SET A NEW MOVE */
if(cur_move>move_profit)
{
  move_profit = cur_move;
  move_flag = SIMPLE_ADD;
  I1  =-1;
```

```
    J1 =-1;
  D1 = -1;

  set_move(I1,J1,D1,j1, SIMPLE_ADD);

        }


}

else
{
/* COMPARE DEMANDS ON j1 WITH Dir, FIND MOVES */
  for(k1=0; R1_ring[k1] != (struct dlist *) NULL; k1++ )
  {
    if(R1_ring[k1]->d>=R_move)
    {
/*which nodes to add to r1*/
/*[nodes to delete from j]: x[N]
[demands to delete from j]:R_ring[k]->i==I or R_ring[k]->==I
[demands to delete from j1]: R1_ring[k1]->i R1_ring[k1]->j */
r1di =r1dj =1;
I1 =R1_ring[k1]->i;
  J1 =R1_ring[k1]->j;
D1 =R1_ring[k1]->d;

for(k3=0; R1_ring[k3] != (struct dlist *) NULL; k3++ )
{
  if( (R1_ring[k3]->i==I1 && R1_ring[k3]->j!=J1) || (R1_ring[k3]->j==I1
 && R1_ring[k3]->i!=J1) )  r1di =0;
```

```
  if( (R1_ring[k3]->i==J1 && R1_ring[k3]->j!=I1) || (R1_ring[k3]->j==J1
 && R1_ring[k3]->i!=I1) )  r1dj =0;
}
r1del =r1di+r1dj;
/* r1di-1 if i delete from r1, r1dj; */

/* find radd */ rai = raj =1;
for(k3=0; R_ring[k3] != (struct dlist *) NULL; k3++ )
{
  if( (R_ring[k3]->i==I1 && x[R_ring[k3]->i]!=1) || (R_ring[k3]->j==I1
 && x[R_ring[k3]->j]!=1) ) rai =0;
  if( (R_ring[k3]->i==J1 && x[R_ring[k3]->i]!=1) || (R_ring[k3]->j==J1
 && x[R_ring[k3]->j]!=1) ) raj =0;
}
radd = rai+raj;

/* find r1add */

/* create array of nodes from r necessary to relocate Dir */

for(i3=0; i3<=N-1; i3++) xrl[i3] =0;
for(k3=0; R_ring[k3] != (struct dlist *) NULL; k3++ )
if(R_ring[k3]->i==I || R_ring[k3]->j==I)  xrl[R_ring[k3]->i] =
xrl[R_ring[k3]->j] =1;

/* find all the nodes on r1 */
for(i=0; i<=N-1; i++) x1rec[i] =0;
for(k3=0; R1_ring[k3] != (struct dlist *) NULL; k3++ )
 x1rec[R1_ring[k3]->i] =x1rec[R1_ring[k3]->j] =1;
```

```
/* find which nodes from r to add to r1 */
for(i=0; i<=N-1; i++) xrr1add[i]=0;
for(i=0; i<=N-1; i++)
{
  if(xrl[i]==1)
  {
    if( x1rec[i]!=1 || (x1rec[i]==1 && i==I1 && r1di==1) ||
(x1rec[i]==1 && i==J1 && r1dj==1) ) xrr1add[i]=1; /* 1 if have to add node i to
r1 */
  }
}

/* calculate r1add */
r1add =0;
for(i=0; i<=N-1; i++) r1add+= xrr1add[i];

    } /* end evaluating move for a single demand of r1*/

/* Move and Move profit */
/* calculate the ring capty, and use the corresponding cost */

if(R1_ring[k1]->d>=R_move)
{
  if(D1<=R_accept)
  {
    Cr_new = Cr -Dir + R1_ring[k1]->d;
    Cr1_new = Cr1 - R1_ring[k1]->d +Dir;
  }
  else
  {
```

```
    Cr_new = Cr -Dir + R_accept;
    Cr1_new = Cr1-R_accept +Dir;
  }                 /* calculate the move's profit */
  for(c5=0; Cr_new>B[c5]; c5++);
  for(c6=0; Cr1_new>B[c6]; c6++) ;
  cur_move = rnodes*a[c3] + r1nodes * a[c4] - (rnodes-rdel+radd)*a[c5]
 - (r1nodes-r1del+r1add) * a[c6];
}

else  cur_move = -2;

if(cur_move>move_profit)
{
  move_profit = cur_move;
  if(D1<=R_accept)
  {
    move_flag = SWAP_DEMAND;
    set_move(I1,J1,D1,j1, SWAP_DEMAND);
    }
  else
  {
    move_flag= SWAP_PARTIAL;
    set_move(I1,J1,D1,j1, SWAP_PARTIAL);

  }
}

  } /* for all the demands on r1 */
}
} /* end if i!=j */
```

```
} /* end comparing j and j1 */
   } /*end if Dir>0 */

/*MAKE THE MOVE */
   switch(move_flag) {
    case NONE:  break;  /* no profitable move was found between Dir
 and the rest of the rings */

case SIMPLE_ADD:
  rswap(R_ring,I,-1,-1);
break;

case SWAP_DEMAND:

copy_ring(&dummy, -1, move->r, R1_ring);
rswap(R_ring,I,I1,J1);

for(k1=0; R1_ring[k1] != (struct dlist *) NULL; k1++ )
{
if( (R1_ring[k1]->i==move->i && R1_ring[k1]->j==move->j) ||
(R1_ring[k1]->j==move->i && R1_ring[k1]->i==move->j) )  break;
}
move->r =j;
addr =check_merge(R1_ring[k1],-1,-1);
if(addr!=NULL)
{
for(current=head; current->next!= NULL; current = current->next)
if(current==addr) break;
```

```
current->d +=R1_ring[k1]->d;
delete_node(R1_ring[k1]);
}
else
{
for(current=head; current->next!= NULL; current = current->next)
if(current==R1_ring[k1]) break;
current->r = move->r;
}

break;

case SWAP_PARTIAL:
copy_ring(&dummy, -1, move->r, R1_ring);
rswap(R_ring,I,move->i,move->j);

for(k1=0; R1_ring[k1] != (struct dlist *) NULL; k1++ )
{
if( (R1_ring[k1]->i==move->i && R1_ring[k1]->j==move->j) ||
 (R1_ring[k1]->j==move->i && R1_ring[k1]->i==move->j) )   break;
}
move->r =j;
addr =check_merge(R1_ring[k1],-1,-1);
if(addr!=NULL)
{
for(current=head; current->next!= NULL; current = current->next)
 if(current==addr) break;
current->d +=R_accept;
}
else add_node(move->i,move->j,j,R_accept,-1);
```

```
for(current=head; current->next!= NULL; current = current->next)
 if(current==R1_ring[k1]) break;
current->d -=R_accept;

break;

   }
/*COPY TO current THE SWAP */

} /*end comparing j and other rings */ /*!!! REset the flags on current!!! */


/*set the swap flag */
node[I]->flag =1;
for(i=0; i<=N-1; i++)
swap_flag *= node[i]->flag;

/*swap_flag =1; */ /* delete this when run for real */


} /* end swap for I */

/* PRINT THE RESULTS FOR A SWAP ON SCREEN*/
for(current=head; current->next!= NULL; current = current->next)
   printf("Y[-%d][-%d]=-%d on ring -%d\n", current->i, current->j,
 current->d, current->r);

/*CALCULATE THE COST */
printf(" cost=%f\n", obj(nrings,N));
```

```
/* D O   I N T E R C O N N E C T I O N S */

D = (int *) malloc(sizeof(int)*nrings*N+1);
D_end = D+ nrings*N;
for( D_addr=D; D_addr<=D_end-1; D_addr++) *D_addr = 0;


/* START INTERCONNECTION */

icflag =0;
for( ; !icflag; )
{
/* ALLOCATE MEMORY FOR D[] - an array of rind-nodal demands */


/* calculate */
for( D_addr=D; D_addr<=D_end-1; D_addr++)
if(*D_addr!=-1)  *D_addr = 0;


for(current=head; current->next!= NULL; current = current->next)
{
if(current->i>=0 && *(D + current->r*N + current->i)>=0)
 *(D + current->r*N + current->i) += current->d;
if(current->j>=0 && *(D + current->r*N + current->j)>=0)
 *(D + current->r*N + current->j) += current->d;
}
```

```
/* find min D[i][j] */
min = 0; I1 =J1 =-1;
/*find the ring capacities*/
for(i=0; i<=MAX_RINGS-1; i++)  C[i] =Br[i] =ar[i] =0;
for (i=0; i<=nrings-1; i++)
C[i] =capacity(i);
for(i=0; i<=nrings-1; i++)
{
for(j=0; C[i]>B[j]; j++) ;
Br[i] =B[j];
ar[i] =a[j];
}

for(i=0; i<=nrings-1; i++)
{
for(j=0; j<=N-1; j++)
{
if(ar[i]- (*(D+i*N+j)) *ic> min && *(D+i*N+j)>0 )
{
min =ar[i]- *(D+i*N+j)*ic;
I1 =i; J1 =j; /* demand *(D+i*N+j) on ring I1, delete node J1 */
/* copy D[I1][J1] in R_ring */
k=0;
      for(current=head; current->next!= NULL;
 current = current->next)    if(current->r==I1 && (current->i==J1 ||
 current->j
 == J1) )    R_ring[k++] = current;
R_ring[k] = (struct dlist *) NULL;
```

```
}
}
}

/* if the min demand to interconnect is too big(unprofitable) exit */
if( min==0 ) icflag =1;

/* else interconnect */
else
{
cap_flag =0;

for(current=head; current->next!= NULL; current =current->next)
{
if( (current->i==J1 || current->j==J1) && current->r!=I1 &&
 C[current->r]+*(D+I1*N+J1)<=Br[current->r] )
{
  /* interconnect: tell the other ring "-1000"
in the node number means interconnected; set D[][] to-1 */

/* for a demand d[J1][i] to be interconnected, if
the destination ring has both nodes J1 and i, just swap*/

    R =current->r; cap_flag =1;

for(k=0; R_ring[k] != (struct dlist *) NULL; k++ )
        {  swap_flag =-1;
  if(R_ring[k]->i==J1) {
   if(*(D+R*N+R_ring[k]->j) >0)  swap_flag =SIMPLE_ADD; }
  else  {if(*(D+R*N+R_ring[k]->i) >0) swap_flag
```

```
 =SIMPLE_ADD;}

  if (swap_flag==SIMPLE_ADD) {
    for(new=head; new->next!= NULL; new
 =new->next)  { if(( new->j==R_ring[k]->j && new->i==R_ring[k]->i)
 || ( new->j==R_ring[k]->i && new->i==R_ring[k]->j) && new->r==I1)
  { new->r =R; break; }

}
} /* end if */
   else
 {  add_node(J1, -1000, R, R_ring[k]->d, I1 );
        for(new=head; new->next!= NULL; new =new->next)
  {
if(new->r==I1 && (new->i==J1 ) )
{
   new->to_node =new->i;
   new->i =-1000;
   new->ic =R;
}
if(new->r==I1 && new->j==J1)
{
   new->to_node =new->j;
   new->j =-1000;  new->ic =R;
}
  }
 } /* end else */

} /* end k */
```

```
    break;
}
}

*(D+I1*N+J1) =-1;

} /* end else interconnect */

/* see if less than the threshold for the ring */

} /* end of icflag*/

/* PRINT THE I/C RESULTS ON SCREEN*/
for(current=head; current->next!= NULL; current = current->next)
{
   printf("Y[%d][%d]=%d on ring %d\n", current->i, current->j,
 current->d, current->r);
   if(current->i<0 || current->j<0)
 printf("Interconnected to node %d on ring %d\n",current->to_node,current->ic);
}

/*CALCULATE THE COST */

printf(" cost after i/c=%f\n", obj(nrings,N));

/* CLEAN UP */
```

```
} /* end iteration */


/*FREE THE MEMORY FOR A NEW DEMAND LIST OF THE NEXT ii PROBLEM  */

for(current=head; current->next != NULL; current =new)
{
new =current->next;
free(current);
}

free(D);


printf ("\n\n");
} /* end "for each problem" */

/* PRINT!! */

fclose(fp);

printf("OVER!!!\n");

}  /* END MAIN*/


/*CONVERT ARRAY TO ARRAY OF INTEGERS; IF N=1, CONVERT TO AN INTEGER*/
char *atoai(char *ptr1, int *array, int n)
```

```
{
int i;
char *ptr2;

for(i=0; i<=n-1; i++)
{
ptr2 = strstr(ptr1, "-");
ptr2++;
*(array+i) = atoi(ptr2);
ptr1 =ptr2;
}

return ptr2;

} /* END OF atoai */


/* FIND A RING CAPACITY */
int capacity(int ring)

{
int Cr;

Cr =0;
for(current=head; current->next!= NULL; current = current->next)
{
if(current->r==ring)
Cr += current->d;
}
return Cr;
```

```
} /*end capacity */


/*COPY RING INTO A BUFFER */

void copy_ring(int *node_demand, int node, int ring_to_copy,
 struct dlist *Ring[])

{
int i;

i=0;
*node_demand =0;
for(current=head; current->next!= NULL; current = current->next)
/* FIND THE CAP OF j AND COPY RING j INTO R_ring */
{
if(current->r==ring_to_copy)
{
Ring[i] = current;
i++;
if(current->i==node || current->j==node)  *node_demand += current->d;
}
}
Ring[i] = (struct dlist *)NULL;

} /* end copy_ring*/


void set_move(int I1, int J1, int d, int ring, int flag)
```

```
{

move->i = I1;
move->j =J1;
move->d = d;
move->r =ring;
move->move_flag =flag;

} /*end set_move */


/* checks if the demand pair d[ij] already exists on the ring r */

struct dlist *check_merge(struct dlist *move_node, int I1, int J1)

{

for(current=head; current->next!= NULL; current = current->next)
{
if( (current->i==move_node->i && current->j==move_node->j
&& current->r==move->r) || (current->i==move_node->j && current->j==move_node->i
&& current->r==move->r) )
{
if( (current->i!=I1 && current->j!=J1) ||  (current->j!=I1
 && current->i!=J1) ) return current;
}
return (struct dlist *)NULL;

}
```

```
} /* end check_merge */


/* deletes the demand from the dlist (in case of merge) */
void delete_node(struct dlist *dem_ptr)
{

struct dlist *previous;

previous = head;
for(current=head; current->next!= NULL; current = current->next)
{
if(current==dem_ptr)  break;
previous = current;
}

current = previous;
current->next = dem_ptr->next;
free(dem_ptr);  /*!!! double check */

} /* end of delete_node */


/*MAKE THE SWAP */
void rswap(struct dlist *ring[], int I, int I1, int J1)
{

int k;
struct dlist *addr;
```

```
for(k=0; ring[k] != (struct dlist *) NULL; k++ )
{
if(ring[k]->i==I || ring[k]->j==I)
{
addr =check_merge(ring[k],I1, J1);
/* i can merge demands for a move, merge */
if(addr!=NULL)
{
for(current=head; current->next!= NULL; current = current->next)
  if(current==addr) break;
current->d +=ring[k]->d;
delete_node(ring[k]);
}
else
{
for(current=head; current->next!= NULL; current = current->next)
 if(current==ring[k]) break;
current->r = move->r;
}
}
}

} /*end rswap */

void add_node(int i, int j, int r, int d, int i1)
{

struct dnode *last;
```

```
for(current=head; current->next!= NULL; current = current->next) ;

current->i =i;
current->j =j;
current->r =r;
current->d =d;
current->ic =i1;

current->next =(struct dlist *)malloc(sizeof(dex));
current = current->next;
current->next =NULL;

} /*end add_node */



/*CALCULATE THE COST */

float obj(int nrings, int N)

{

float cost, ic_cost, rcost;
int i, bw, j;
int x[Nd], rsize, sum;

cost =ic_cost =0;
for(i=0; i<=nrings-1; i++)
{
bw =  rcost =0;
```

```
for(j=0; j<=N-1; j++)  x[j] =0;
for(current=head; current->next!= NULL; current =current->next)
{
if(current->r==i)
{
if(current->i>=0)  x[current->i] =1;
else  ic_cost +=ic*current->d;
if(current->j>=0)  x[current->j] =1;
else   ic_cost +=ic*current->d;
bw +=current->d;
}
}

for(rsize=0; bw>B[rsize]; rsize++) ;
sum =0;
for(j=0; j<=N-1; j++)  sum +=x[j];
rcost =a[rsize]*sum;
cost +=rcost ;
printf("%d nodes on ring %d and bw=%d\n",sum,i,bw);
}
return cost+ic_cost/2;

} /* end obj */
```

Appendix E

Bibliography

Bibliography

[BAL89] Ballart R. Ching Y-C. ,"SONET: Now It's the Standard Optical Network", *IEEE Communications Magazine,* March 1989.

[CHL90] Chlamtac, Franta, "LightNet and Optimal Lightpath Problem", *Proceedings of the IEEE*, Vol. 78, No. 1, January 1990

[DRA93] Drake V. J. "A Review of Four Major SONET/SDH Rings", *IEEE Global Conf.on Commun.*, 1993.

[FLA89] Flanagan T. "Principles and Technologies for Planning Survivability", *IEEE Global Conf. on Commun.*, 1989.

[FLA90a] Flanagan T. "Planning a SONET Network", *IEEE Global Conf. on Commun.*, 1990.

[FLA90b] Flanagan T. "Fiber Network Survivability", *IEEE Communications Magazine,* June 1990.

[GRO87] Grover W. D. "A Fast Distributed Restoration Technique for Networking Using Digital Crossconnect Machines", *IEEE Global Conf.on Commun.*, 1987.

[HAS87] Hasegawa S. et al. "Dynamic reconfiguration of Digital Cross-Connect Systems with Network Control and Management", *IEEE Global Conf.on Commun.*, 1987.

[MAR93] Marzec R. P. et al. "Selection of a SONET Network Architecture – Survivability Can Be Enhanced Cost-Effectively", *IEEE Int. Conf. on Commun.* , 1993.

[TOB90] Tobagi F. A. ,"Fast Packet Switch Architectures For Broadband Integrated Services Digital Networks", *Proceedings of IEEE,* Vol.78, January 1990.

[WIL93] Williams J. M., Johnson J. A., "The Role of the Wideband Digital Cross-connect System in Survivable Ring Networks", *IEEE Int. Conf. on Commun.* , 1993.

[WU88] Wu T-H. Kollar D. J. Cardwell R. H. "Survivable Network Architectures for Broad-Band Fiber Optic Networks: Model and Performance Comparison", *Journal of Lightwave Technology*, Vol. 6, No. 11, November 1988.

[WU89] Wu T-H. Kollar D. J. Cardwell R. H. "High-Speed Self-Healing Ring Architecture for Future Interoffice Networks", *IEEE Global Conf.on Commun.*, 1989.

[WU90] Wu T-H. Lau R. C. "A Class of Self-Healing Ring Architectures for SONET Network Applications", *IEEE Global Conf.on Commun.*, 1990.

[WU93] Wu T-H. et al. , "A Service Restoration Time Study for Distributed Control SONET Digital Cross-Connect System Self-Healing Networks", *IEEE Int. Conf. on Commun.* , 1993.

[NYT93] *3 Suppliers Are Selected for Big US West Project"*, The New York Times, June 16, 1993.

[AMO93] Ahuja R. K. Magnanti T. L. Orlin J. B. "Network Flows", Prentice-Hall, Englewood Cliffs, 1993.

[COS93] Cosares S. Sanjee I. "An Optimization Problem Related to Balancing Loads for Survivability in Telecommunications Networks", to be published.

[DAL92] Dahl G. Stor M. "MULTISUN - Mathematical Model and Algorithms", Technical Report R 46/92 (translated from Danish), 1992.

[GOL] Goldberg A. V. "Finding a Maximum Density Subgraph", University of California at Los Angeles.

[GRT92] Grotschell M. Monma C. L. Stoer M. "Computational Results with a Cutting Plane Algorithm for Designing Communications Networks with Low-Connectivity Constraints", *Operations Research*, Vol. 40, No 2, March-April 1992.

[LAG93] Laguna M. "Optimal Design of SONET Rings for Interoffice Telecommunication", to be published.

[MAG91] Magee T. "SONET Planning Model and Observations", Technical Memorandum, University of Colorado at Boulder, October 1991.

[MON88] Monma C. L. Shallcross D. F. "Methods for Designing Communications Networks with Certain Two-Connected Survivability Constraints", *Operations Research*, Vol. 37, No 4, July-August 1989

[SHU93] Shulman A. et al. "Multicommodity Flows in Ring Networks", GTE Labs Inc. to be published.

[SCOM93] Shulman A. personal communication.

[WU91] Wu T-H. Cardwell R. Boyden M. "A Multi-Period Design Model for Survivable Network Architecture Selection for SONET Interoffice Networks", *IEEE Transactions on Reliability*, Vol. 40, No 4, October 1991.

[NEMH88] Nemhauser G. L. Wolsey L. A. "Integer and Combinatorial Optimization", John Wiley and Sons, 1988.