HEURISTIC APPROACHES FOR TELECOMMUNICATIONS NETWORK MANAGEMENT, PLANNING AND EXPANSION

edited by

Robert Doverspike
Iraj Saniee

Springer Science+Business Media, LLC

HEURISTIC APPROACHES FOR TELECOMMUNICATIONS NETWORK MANAGEMENT, PLANNING AND EXPANSION

A Special Issue of the Journal of Heuristics

edited by

Robert Doverspike
and
Iraj Saniee

Reprinted from the
Journal of Heuristics
Volume 6:1 (2000)

SPRINGER SCIENCE+BUSINESS MEDIA, LLC

Journal of Heuristics

Volume 6, Number 1, April 2000

Special Issue: Heuristic Approaches for Telecommunications Network Management, Planning and Expansion

Guest Editors: Robert Doverspike and Iraj Saniee

ISBN 978-1-4419-4970-7 ISBN 978-1-4757-5392-9 (eBook)
DOI 10.1007/978-1-4757-5392-9

Library of Congress Cataloging-in-Publication Data

A C.I.P. Catalogue record for this book is available
from the Library of Congress.

Journal of Heuristics, 6, 5–7 (2000)
© 2000 Kluwer Academic Publishers

Preface

The telecommunications industry is experiencing a world-wide explosion of growth as few other industries ever have. However, as recently as a decade ago, the bulk of telecommunications services were delivered by the traditional telephone network, for which design and analysis principles had been under steady development for over three-quarters of a century. This environment was characterized by moderate and steady growth, with an accompanying slower development of new network equipment and standardization processes. In such a near-static environment, attention was given to optimization techniques to squeeze out better profits from existing and limited future investments. To this end, forecasts of network services were developed on a regular planning cycle and networks were optimized accordingly, layer by layer, for cost-effective placement of capacity and efficient utilization. In particular, optimization was based on fairly stable set of assumptions about the network architecture, equipment models, and forecast uncertainty.

Thanks to the explosive growth of data networks, the Web, E-commerce, and the Internet, traffic forecasts are often either non-existent or inherently unstable. Assumptions about models for equipment capabilities and network architectures are short-lived and risky. In addition, networking problems are more difficult because of the heterogeneity of services, varying bandwidth needs, architectural alternatives and differing qualities of service. Given that realistic optimization problems that can account for the complexities of such networks are fundamentally very complex to solve, optimization tools for the emerging networks based on traditional network planning cycles and network management approaches run the additional risk of being irrelevant, and possibly even giving misleading solutions by the time they are available for use. However, one should not conclude that efficient utilization of network resources is no longer necessary. To the contrary, avoiding congestion, careful routing and judicious capacity expansion, to name just a few activities, deserve even more attention than before. New services and networking capabilities are introduced daily and networking needs change much faster than it takes to develop tools by traditional standards. One way to cope in this environment is to produce tools that are model-flexible, robust and, as a result, simple; if necessary, they can be thrown away or redone rapidly as the underlying assumptions change.

Such requirements are not characteristic of traditional mathematical programming and stochastic analysis approaches for network design and management. Thus, it is important to develop approaches that use approximations and solution methods that take advantage of engineering insights into the underlying problems, or in short, *heuristic approaches*. As an example, it may be expedient to formulate an approach that gives "good" solutions to a fixed service forecast, but is robust to service forecast changes than an approach that optimal solutions to a fixed forecast, but gives poor solutions under a changed forecast. Also, experience has shown that carefully crafted heuristic optimization can yield "reasonable" solutions much more rapidly than the more traditional mathematical programming approaches; in fact, traditional methods often cannot solve complex, real network problems directly.

To address come of these concerns, this special edition is devoted to heuristic approaches for telecommunications network management, planning, and expansion. We solicited papers on these topics with the hope of illustrating the use of heuristic methods in telecommunications. The nine papers that passed the review process (out of eighteen papers received) cover a wide variety of problems of current interest. These are briefly highlighted below, where for contrast we indicate some of the relevant traditional operations research disciplines in parentheses:

ATM/Data Networks

- Performance comparisons (through simulation) of dynamic routing in ATM networks when link occupancy data is cached vs. obtained in real-time (analytical modeling)
- Design of switch placements and homing in new data networks under very uncertain service forecasts and backbone network assumptions (integer programming)
- Design of Virtual Paths in ATM networks (large-scale non-linear integer programming)

Data Network Case Study

- Use of heuristics to develop a business case to intelligently select the right type of switch port in a Frame Relay network (a variety of disciplines)

Local Networks

- Design of distribution (feeder) access transmission networks (optimization)
- Design of LANs (capacitated multi-commodity flow)

Restoration in Transmission/Transport Networks

- Routing and slotting in SONET/SDH rings (combinatorial optimization)
- Design of mesh restoration in transmission networks (large-scale linear programming)

Circuit-Switched Networks

- How to recognize and avoid cycles in Common Channeling Signaling (CCS) networks (graph theory, combinatorics)

We recognize that there are many more areas in this growing field than this collection addresses. Furthermore, there is intricate interplay between use of specially crafted heuristics and formal mathematical methodologies with guaranteed performance that are touched on by some of the papers, but which would be a topic by itself. We hope however, that this collection brings to the attention of researchers and practitioners an array of techniques and case studies that meet the stringent "time to market" requirements of this industry and which deserve exposure to a wider audience.

Telecommunications will face a tremendous challenge in the coming years to be able to design, architect, and manage networks in such a rapidly evolving industry. Development and application of heuristic methods will be fundamental to our ability to meet this challenge.

Robert Doverspike, AT&T Labs Research
Iraj Saniee, Bell-Labs, Lucent Technologies
June 1st, 1999

Journal of Heuristics, 6, 9–20 (2000)
© 2000 Kluwer Academic Publishers

Telecommunications Network Case Study: Selecting a Data Network Architecture

ERIC ROSENBERG
AT&T Labs, Middletown, NJ 07748, USA
email: eric.rosenberg@att.com

Abstract

This paper documents a model that was pivotal in deciding which of two architectures should be selected for a frame relay data communications network. The choices are either to continue using the current architecture, or to make a large incremental investment in new equipment which reduces the number of high speed inter-office trunks required to interconnect the switches. The analysis requires optimizing the mix of two types of customer port cards to determine the maximum customer port capacity of a switch. Simple approximations are used to estimate the number of inter-office trunks and trunk cards required. Based in large part on the costs computed by this model, an executive level decision was made to move to the new architecture.

Key Words: data communications, network architecture, network design, frame relay, virtual circuits, economic analysis

1. Introduction

To design a data communications network, it is necessary to consider a variety of factors. One factor is the set of available technologies, with their complex set of rules on, e.g., capacity limits, queuing delays, and throughput. Another factor is the forecast of demand for the data communications service, since the network must be designed to provide sufficient capacity to last a year or two, at which point a migration to the next generation technology is probably required, to be competitive in price and features. A third is the network reliability, which depends on intrinsic equipment failure rates and the extent to which redundancy is used. In general, a great many quantities must be estimated and analyzed to make multi-million dollar decisions that set the direction of the network.

This paper describes how a major frame relay data network architecture decision was made. The purpose of the paper is take some of the mystery out of architecture selection, and show how it can be done using approximations that are based on extensive data. We begin by describing the network.

The data network is a collection of frame relay switches that are connected by high-speed transmission facilities, called trunks. Customers pay for customer ports at each city they want to interconnect. The allowable port speeds are multiples of the DS0 rate (64Kbps), up to the DS1 rate (1536Kbps, or 24 DS0s). Thus, for example, a customer might buy a port with speed 256Kbps (4 DS0s) and a port with speed 768Kbps (12 DS0s). Note that a customer port represents a certain amount of bandwidth dedicated to the customer, and not a physical entity such as a connector.

Customers also pay for permanent virtual circuits (PVCs). A PVC is a connection between two customer ports at a specified speed, e.g., 8Kbps. The designation "permanent" means that once a PVC is set up, by the network management system, between two customer ports, it remains available to the customer until it is disconnected. The designation "virtual" means that there is no physical wire implementing each PVC.

A PVC-based frame relay network offers a low cost alternative to a dedicated private line network by taking advantage of statistical multiplexing, which is based on the premise that not all PVCs will be used at the same time. The cost savings arises since less capacity is required than would be required if we designed for the worst case in which all PVCs were utilized simultaneously. The degree to which we exploit statistical multiplexing is governed by the "overbooking factor" β. We have $\beta > 1$, and the higher β is, the greater the savings in trunk costs that can be realized, but the more likely it is that performance is impaired.

The tens of thousands of PVCs in the frame relay network constitute a set of switch-to-switch "demands," which must be routed over the trunking network. Since it is in general not economical to design a fully interconnected network in which each pair of switches is connected by one or more trunks, the average PVC will traverse more than one trunk. When a PVC is routed over a trunk, it consumes two trunk resources. The first is "PVC count": the number of PVCs using a trunk cannot exceed the maximum PVC count (no matter how little bandwidth the PVCs use). The second is bandwidth: the sum of the bandwidth of all the PVCs using the trunk, divided by the overbooking factor β, cannot exceed the trunk bandwidth. For example, if $\beta = 2$, we could route 16 PVCs, each with a bandwidth of 192Kbps, over a T1 trunk, which provides a usable bandwidth of 1536Kbps (DS1 rate).

Each trunk is connected to a connector on a "trunk card," which is a printed circuit board that plugs into a slot on the switch. The current plan (Plan A) uses, in each switch, n_{A} "type A" trunk cards, each providing p_{A} trunk connections. Each switch is deployed with the full complement of n_{A} cards, since typically all $n_{\mathrm{A}} p_{\mathrm{A}}$ trunk connections are required as the switch reaches maximum utilization, and deploying the full complement eliminates the labor expense and complication of installing additional cards later. The alternative plan (Plan B) uses, in each switch, n_{B} "type B" trunk cards, each providing p_{B} trunk connections. All trunks supported by type A and type B cards have the same bandwidth. However, there are two advantages of type B trunk cards. The first advantage is that, while $n_{\mathrm{B}} p_{\mathrm{B}} = n_{\mathrm{A}} p_{\mathrm{A}}$ (so that the total number of trunks supported by plans A and B are equal), we have $n_{\mathrm{B}} < n_{\mathrm{A}}$ and $p_{\mathrm{B}} > p_{\mathrm{A}}$, which means that there are more trunk connections per card slot with Plan B. This is important since card slots are precious (they can be used to connect more customers, implement new services, etc.)

The second advantage of a type B trunk card is that it supports more PVCs per trunk than a type A card. Let the maximum PVC count per trunk be $b_{\mathrm{pvc}}^{\mathrm{A}}$ for type A cards and $b_{\mathrm{pvc}}^{\mathrm{B}}$ for type B cards. We have $b_{\mathrm{pvc}}^{\mathrm{A}} < b_{\mathrm{pvc}}^{\mathrm{B}}$ due to technological improvements in the type B card. All trunks (on both type A and type B cards) have the same bandwidth capacity, which we denote by b_{bw}.

The maximum number of PVCs that can use a single trunk, using type X trunk cards (where $X = A$ or B), is thus given by

$$\min\left\{b_{\mathrm{pvc}}^{X}, \frac{\beta b_{\mathrm{bw}}}{\bar{w}}\right\},$$

where $\bar{w}$ is the average bandwidth of a PVC. Here we see the value of overbooking, since the trunk bandwidth is multiplied by the overbooking factor $\beta > 1$. Thus, for example, with DS3 trunks (45Mbps) and type A trunk cards, $b^{\mathrm{A}}_{\mathrm{pvc}} = 2500$, $b_{\mathrm{bw}} = 45\mathrm{Mbps}$, $\beta = 3$, and $\bar{w} = 50\mathrm{Kbps}$, the maximum number of PVCs per trunk is $\min\{2500, (3)(45)/(.050)\} = \min\{2500, 2700\}$, which means that the average trunk reaches its PVC count limit before its bandwidth limit. (These numbers are illustrative, and are not to be construed as the actual parameters.) Since trunks in the current frame relay network are PVC limited, rather than bandwidth limited, then the fact that $b^{\mathrm{A}}_{\mathrm{pvc}} < b^{\mathrm{B}}_{\mathrm{pvc}}$ means that using type B trunk cards will reduce the total number of trunks in the network.

This paper describes the methodology used to choose between continuing with the present data architecture (Plan A) using type A trunk cards, or making a substantial incremental investment in type B trunk cards, which will reduce the number of trunks required. Abandoning the current architecture also requires the expense of re-wiring the cables at each switch location.

2. Optimal shelf configuration

To compare the economics of Plans A and B, we estimate the number of inter-office trunks and trunk cards required to meet the demand through the end of 1998. (By the end of 1998 another major architectural decision will probably be required due to the emergence of new technologies.) Since trunk cards are installed in each switch, to compute the total trunk card cost for Plans A and B we must first determine the number of switches required, which in turn requires calculating the number of customer ports per switch. This leads to the problem of determining the optimal switch configuration, which we now address.

Customer port cards (cards which are connected, typically by a local exchange network, to the actual customer), plug into equipment "shelves" on the switch. There are two types of customer port cards, "channelized" and "unchannelized," which plug into the S slots in a shelf. Both types have K physical ports (i.e., K connectors), each of which connects to a T1. "Unchannelized" cards provide one customer port, at up to the DS1 rate (1536Kbps) on each physical port. "Channelized cards" provide up to 24 customer ports on each of the K physical ports, with the restriction that the sum of the customer port speeds on a physical port cannot exceed the DS1 rate. Thus, for example, we could assign a 1024Kbps (16 DS0s) customer port to a physical port on either a channelized card or an unchannelized card; for the channelized card we could assign a 1024Kbps customer port and a 512Kbps (8 DS0s) customer port to the same physical port, or 24 customer ports of speed 64Kbps to the same physical port. (The capability of assigning the 24 DS0s in a T1 to different customer ports is provided by a DCS-1/0 digital cross connect device.) Thus each channelized card can provide the functionality of an unchannelized card. However, the channelized card is more expensive than the unchannelized card.

In this section we address two questions: (i) given a set of customer ports, determine if a given customer port should be assigned to a channelized or unchannelized card, and (ii) determine the optimal number of channelized and unchannelized cards on a shelf. Combining the answers to these two questions yields δ, the optimal number of customer ports per shelf.

2.1. *Port assignment: channelized vs. unchannelized*

Suppose we expect the customer port provisioning process to eventually fill up each of the K T1 lines on each channelized card to a capacity of 24-b DS0 channels (i.e., a "breakage" of b DS0s per T1). For example, if we assign only a 1024Kbps customer port and a 256Kbps customer port to a single physical port of a channelized card, then we have consumed $(1024 + 256)/64 = 20$ DS0s, so the breakage is 4.

Let C_{c} be the cost of a channelized card, and let C_{u} be the cost of an unchannelized card. Then the average cost per DS0 on the channelized card is $C_{\mathrm{c}}/K(24 - b)$. Suppose we use an unchannelized card whenever the port speed is at least 24α DS0s, for some $\alpha < 1$. Then the cost per DS0 on the unchannelized card is no more than $C_{\mathrm{u}}/(K\alpha 24)$. For the average cost per DS0 on the unchannelized card to be no more than the average cost per DS0 on the channelized card, we require

$$\frac{C_{\mathrm{u}}}{K\alpha 24} \le \frac{C_{\mathrm{c}}}{K(24 - b)}$$

which yields

$$\alpha \ge \bar{\alpha} = \frac{(24 - b)C_{\mathrm{u}}}{24C_{\mathrm{c}}}.$$

Thus, for example, if the channelized card is 1.5 as expensive as the unchannelized card, and the average breakage is 2, then we should use an unchannelized card whenever the port speed is at least (0.61)(24) DS0s, or 939Kbps.

Such a port assignment rule ("ports with bandwidth exceeding $24\bar{\alpha}$ DS0s should be assigned to an unchannelized card") is used within a port assignment process as follows. Shelves of port cards are installed with a mix of channelized and unchannelized cards (as determined in the next section). Port assignments are made on a real-time basis. When a request is received to provision a port on a switch, we use the port assignment rule: if it should go on an unchannelized card, we do so (if possible). If it belongs on a channelized card, we try to assign it to a channelized card already used (i.e., with some ports already assigned to it); if this cannot be done we try to assign the port to a new channelized card on the shelf. Note that we may be unable to assign a port to a channelized card even if there is spare capacity on the card; for example, if only 128Kbps of capacity are available then we cannot assign a 192Kbps port. If the port assignment cannot be made then a new shelf must be installed or exceptions made to the port assignment rule.

2.2. *The optimal number of channelized cards on a shelf*

We now return to the calculation of δ, the optimal number of customer ports per shelf, when there is a mix of channelized and unchannelized cards. Let y be the number of channelized cards on a shelf. Since there are S cards on a shelf and K physical ports per card, we have $S - y$ unchannelized cards per shelf for a total of $K(S - y)$ physical ports on the unchannelized cards.

Let θ be the average speed of all the customer ports in the network, and let θ_c be the average speed of all those customer ports that go on channelized cards. We have $\theta_c < \theta$, since the higher speed ports go on the unchannelized card. To compute θ_c, we use data on the distribution of customer port speeds in the network. Let I index the set of different customer port speeds available, and for $i \in I$, let s_i be the port speed and f_i be the fraction of all customer ports with that speed.

Since, as we have shown above, only port speeds of $1536\bar{\alpha}$ and higher go on the unchannelized card, we have

$$\theta_c = \frac{\sum_{i \in I_{\bar{\alpha}}} s_i f_i}{f_c},$$

where

$$I_{\bar{\alpha}} = \{i \in I : s_i < 1536\bar{\alpha}\}$$

and

$$f_c = \sum_{i \in I_{\bar{\alpha}}} f_i$$

is the fraction of all ports going on channelized cards.

With a breakage of b DS0s on each of the K T1 lines on a channelized card, for the y channelized cards we get $Ky(1536 - 64b)/\theta_c$ customer ports.

Having computed f_c and θ_c, the next problem is to determine the optimal value of y, the number of channelized cards on a shelf. The possible choices for y are 0 to S. We will optimize y by the brute force method of computing, for each y, the cost of the number of shelves required for N ports, and picking the y yielding the minimum cost. Since the quantities of interest are linear in N, it suffices to take $N = 1$ in evaluating all the expressions below; we leave N in the formulas to explicitly show the functional dependency.

When $y = 0$, we only use unchannelized cards, and we require $N/(SK)$ shelves, at a cost of $C_u N/K$, since there are S cards per shelf.

When $y = S$, we use only channelized cards, and each shelf provides $SK(1536 - 64b)/\theta$ customer ports. Note that we use θ as the average speed, since when there are no unchannelized cards we make no distinction between ports with speeds less than $1536\bar{\alpha}$ Kbps (the breakpoint between assigning to an channelized or unchannelized card) and ports with speeds exceeding $1536\bar{\alpha}$. With N ports, we need

$$\frac{N\theta}{SK(1536 - 64b)}$$

shelves, at cost

$$\frac{C_c N\theta}{K(1536 - 64b)}.$$

Finally, we need to examine the values $y = 1, 2, \ldots, S - 1$. Let $A_c(y)$ be the number of shelves we need to accommodate the required number of channelized cards. Then

$$A_c(y) = \frac{Nf_c\theta_c}{Ky(1536 - 64b)}.$$

Let $A_u(y)$ be the number of shelves we need to accommodate the required number of unchannelized cards. Then

$$A_u(y) = \frac{N(1 - f_c)}{K(S - y)}.$$

Thus, for $y = 1, 2, \ldots, S - 1$, the required number $A(y)$ of shelves is

$$A(y) = \max\{A_c(y), A_u(y)\}.$$

The cost of a shelf with y channelized cards and $S - y$ unchannelized cards is $yC_c + (S - y)C_u$. It follows that we must solve the problem

$$\min_{0<y<S}\{A(y)(yC_c + (S - y)C_u)\}.$$

Having computed the cost for $y = 0, 1, \ldots, S$, we can determine the optimal $y^\star$. From $y^\star$ we can compute δ, the optimal number of customer ports per shelf. If $y^\star = S$ then $\delta = SK(1536 - 64b)/\theta$. Otherwise, if $y^\star < S$, the $y^\star$ channelized cards on a shelf provide $Ky^\star(1536 - 64b)/\theta_c$ customer ports, and the $S - y^\star$ unchannelized cards provide $K(S - y^\star)$ customer ports. Thus δ is given by

$$\delta = \begin{cases} Ky^\star(1536 - 64b)/\theta_c + K(S - y^\star) & \text{if } y^\star < S \\ SK(1536 - 64b)/\theta & \text{if } y^\star = S \end{cases}.$$

3. Estimating network components

In this section we estimate the number of inter-office trunks, trunk cards, and switches required for Plans A and B.

3.1. Estimating inter-office trunks

Frame relay switches are located in buildings, called "offices," with special capabilities such as connectivity to the local exchange carrier, battery backups, and alarms to detect malfunctions. Larger cities, such as New York and Los Angeles, have more than one office. Each office houses one or more switches. Connections between switches within an office (called intra-office trunks) are relatively easy to implement and have a negligible cost. On the other hand, high speed connections (e.g., at the DS3 rate or higher) between offices (called inter-office trunks) are expensive. Let c_{T} be the average cost per month of an inter-office trunk.

In the current frame relay network being analyzed, trunks are PVC limited, rather than bandwidth limited. Even with type B trunk cards, which support more PVCs than type A cards, trunks will still be PVC limited (this is due to the overbooking factor, which dramatically increases the effective bandwidth capacity on trunks). This means that, when using network average measures to estimate trunk requirements with either type A or B cards, it suffices to consider only PVC counts, and ignore bandwidth.

Let ϕ be the average PVC utilization ($\phi < 1$) in the current network (using type A trunk cards), so that the average trunk has $\phi b_{\mathrm{pvc}}^{\mathrm{A}}$ PVCs. The utilization ϕ is significantly less than the value 1, since we must reserve sufficient PVC capacity in the trunking network so that, if a single trunk fails, there will be enough PVC capacity among the other trunks to successfully re-route traffic off the failed trunk.

We assume that, under plan B (with only type B trunk cards), ϕ is also the average PVC utilization for the network, so that the average trunk will have $\phi b_{\mathrm{pvc}}^{\mathrm{B}}$ PVCs. Thus, since $b_{\mathrm{pvc}}^{\mathrm{B}} > b_{\mathrm{pvc}}^{\mathrm{A}}$ and trunks are PVC limited, using only type B cards will reduce the total number of trunks required.

Since trunks are PVC limited, we can estimate the total number T_{io}^{X} of inter-office trunks in the network, using type X (where $X = A$ or B) trunk cards:

$$T_{\mathrm{io}}^{X} = \left\lceil \frac{Vh_{\mathrm{io}}}{\phi b_{\mathrm{pvc}}^{X}} \right\rceil \tag{1}$$

where V is the total number of PVCs in the network, h_{io} is the average inter-office hop count (the average number of inter-office trunks traversed by a PVC), and $\lceil x \rceil$ is the smallest integer greater than or equal to x. The parameter h_{io} can be computed from the routes for all the PVCs in the network by summing the number of inter-office hops over all PVCs and dividing by the number of PVCs in the network. This formula can be understood as follows: multiplying V by h_{io} yields the total number of inter-office hops for all the PVCs in the network. Dividing Vh_{io} by $\phi b_{\mathrm{pvc}}^{X}$ (the average number of PVCs per trunk) yields the number of trunks.

Note that the estimate T_{io}^{X} says nothing about where the inter-office trunks are in the network; it only predicts the total number, using network averages. Dividing by the average PVC count $\phi b_{\mathrm{pvc}}^{X}$ is crucial; dividing by the maximum PVC count b_{pvc}^{X} would greatly underestimate the required number of trunks. The formula works quite well: the number of trunks predicted by the formula is within 5% of the actual number of trunks in March, 1997; for year end 1997 the number of trunks predicted by the formula was also within 5% of the number of trunks calculated by the production network design tool. However, this tool takes many hours to run for a network of the size we expect by 1998, and is not appropriate to use for strategic planning.

3.2. *Estimating trunk cards*

Each inter-office trunk or intra-office trunk (between switches in an office) terminates, at each end, on a connector on a trunk card. To ensure that we install a sufficient number of trunk cards in an office, we must estimate the total number of trunk terminations in each office.

The first step is to estimate the total number of inter-office and intra-office trunks. Let h_{tot} be the average number of inter-office and intra-office trunks traversed by a PVC. Then $h_{\text{tot}} > h_{\text{io}}$, since h_{io} does not count the intra-office hops. By reasoning as in (1) above, the total number T^X_{tot} of inter-office and intra-office trunks in the network with type X trunk cards is given by

$$T^X_{\text{tot}} = \left\lceil \frac{V h_{\text{tot}}}{\phi b^X_{\text{pvc}}} \right\rceil. \tag{2}$$

Since each trunk has two ends, the total number of trunk terminations is $2T^X_{\text{tot}}$.

Next we estimate the number of trunk terminations in each office. This estimate will be used to verify that there are sufficient switches in each office at year end 1998 to provide the required trunk terminations, and otherwise is not used in the final cost calculations.

To estimate the trunk terminations at each office, let g_i be the fraction of customer ports at office i. We observe that the two endpoints of all the PVCs account for nearly half the total trunk terminations, and the number of PVC endpoints at office i is proportional to g_i. Moreover, since larger offices tend to be used more than smaller offices as intermediate (via) switches, the number of trunk terminations at office i due to PVCs taking more than one hop is roughly proportional to g_i. Therefore, the total number of trunk terminations at office i can be estimated by

$$\left\lceil \frac{2g_i V h_{\text{tot}}}{\phi b^X_{\text{pvc}}} \right\rceil. \tag{3}$$

Since trunk card type X provides p_X trunk connections, the total number of trunk cards of type X required in office i is

$$\left\lceil \frac{2g_i V h_{\text{tot}}}{\phi b^X_{\text{pvc}} p_X} \right\rceil. \tag{4}$$

3.3. *Estimating switches and shelves*

Let d_i be the number of switches required in office i. To calculate d_i, let γ be the number of shelves that can be accommodated on a switch (this is a known constant). We assume that each switch in office i has a full complement of shelves. Let P be the total number of customer ports in the network. Since there are $g_i P$ customer ports at office i, and $\gamma\delta$ customer ports per switch, it follows that

$$d_i = \left\lceil \frac{g_i P}{\delta\gamma} \right\rceil. \tag{5}$$

Since a switch provides n_X slots for card type X, and card type X provides p_X trunk connections, then these d_i switches provide $d_i n_X p_X$ trunk connections. If this is less than the number required (from (4) above), then additional switches must be installed simply to provide the required trunk connections. For both Plans A and B, with a year end 1998 planning horizon, no additional switches to supply trunk connections were required beyond the switches necessary to support the required number of ports. The estimates (4) of trunk terminations have no further use in the remaining economic analysis.

4. Economic comparisons of plans A and B

The economic analysis compares the costs of Plans A and B in 1998. The costs include switch and shelf costs, trunk costs, and labor costs. Since the number of switches and shelves is the same for both plans, we ignore these costs. Let c_X be the cost of a type X trunk card, for $X = A$ or B. The time value of money is not significant, since we are only considering one year, and will be ignored.

4.1. Plan A equipment cost

Since there is a lead time of several months for delivery and installation of switches and shelves, sufficient switches and shelves have already been ordered to provide enough customer port capacity to last through the end of 1997. Each of these switches has a full complement of n_A trunk cards (the type B cards will not be available until the beginning of 1998). If we continue using Plan A in 1998, we will have to equip each of the new switches ordered in 1998 with n_A type A trunk cards.

Let P^{97} be the forecast for the total number of ports in the network at the end of 1997, and let P^{98} be the forecast for the end of 1998. From (5), the number of new switches required in office i at the end of 1998 is

$$\left\lceil \frac{g_i P^{98}}{\delta\gamma} \right\rceil - \left\lceil \frac{g_i P^{97}}{\delta\gamma} \right\rceil.$$

Since each of these switches requires n_A type A trunk cards, the equipment cost of plan A in 1998 is

$$C_\mathrm{E}^\mathrm{A}(i) = \left(\left\lceil \frac{g_i P^{98}}{\delta\gamma} \right\rceil - \left\lceil \frac{g_i P^{97}}{\delta\gamma} \right\rceil \right) n_\mathrm{A} c_\mathrm{A}.$$

4.2. *Plan B equipment cost*

To implement plan B we must equip each switch in the network with n_B type B trunk cards. The number of switches required in office i at the end of 1998 is

$$\left\lceil \frac{g_i P^{98}}{\delta\gamma} \right\rceil.$$

Since each of these switches requires n_B type B trunk cards, the equipment cost of plan B in 1998 is

$$C_E^B(i) = \left\lceil \frac{g_i P^{98}}{\delta\gamma} \right\rceil n_B c_B.$$

4.3. *Plan A inter-office trunk cost*

Let V^{97} be the forecast for the total number of PVCs in the network at the end of 1997, and let V^{98} be the forecast for the end of 1998. Recalling (1), define

$$T_{io}^X(V) = \left\lceil \frac{V h_{io}}{\phi b_{pvc}^X} \right\rceil.$$

In Plan A we start 1998 with $T_{io}^A(V^{97})$ inter-office trunks and end 1998 with $T_{io}^A(V^{98})$ inter-office trunks. Assuming linear growth in 1998, the average number of trunks in 1998 is the average of these two quantities. Since c_T is the monthly trunk cost, the cost in 1998 for inter-office trunks under plan A is

$$C_T^A = (12)(1/2)\left(T_{io}^A(V^{97}) + T_{io}^A(V^{98})\right) c_T.$$

4.4. *Plan B inter-office trunk cost*

For Plan B, we use type B trunk cards, available at the start of 1998. However, we cannot, at the start of 1998, migrate all type A trunks (trunks connecting to a type A trunk card) to type B trunks (trunks connecting to a type B trunk card), since prudence dictates that we begin with limited deployment and wait a few months until we are sure that the cards are operating correctly. Accordingly, we assume that, beginning in 1998, all new trunks required will be type B trunks, and that after M months all the existing type A trunks will be transitioned to type B. Clearly the longer M is, the less savings we will realize in 1998 due to the increased efficiency of type B trunks. We assume that the transition from type A to type B trunks occurs instantaneously at the end of month M. In reality, the transition may take several weeks.

In Plan B we start 1998 with

$$T_1 = T_{io}^A(V^{97})$$

inter-office trunks, which exist for M months.

Assuming linear growth in 1998, the number of type B trunks for growth required at the end of M months is

$$T_2 = T_{\rm io}^{\rm B}((M/12)(V^{98} - V^{97})).$$

The average number of type B trunks for 1998 growth during the first M months is $(1/2)T_2$.

After M months, all the type A trunks are transitioned to type B. The type B trunks now contain the PVCs that were moved off the type A trunks as well as PVCs added in 1998. The number of inter-office trunks after M months is therefore

$$T_3 = t_{\rm io}^{\rm B}((M/12)(V^{98} - V^{97}) + V^{97})$$

and the number at the end of 1998 is

$$T_4 = t_{\rm io}^{\rm B}(V^{98}).$$

The average of T_3 and T_4 is the average number of trunks for the last $12 - M$ months of 1998.

It follows that the 1998 cost of inter-office trunks under Plan B is

$$C_{\rm T}^{\rm B} = \left(MT_1 + M\frac{T_2}{2} + (12 - M)\frac{(T_3 + T_4)}{2}\right)c_{\rm T}.$$

4.5. Labor costs

Plan B requires a labor expense not required under Plan A, namely the expense of converting from type A to type B trunks at the end of M months of 1998. We estimate that this requires L full time people for one month. Thus the labor cost for plan B is $C_{\rm L}^{\rm B} = LW/12$, where W is the loaded salary of one person.

4.6. The bottom line

The 1998 costs $C^{\rm A}$ and $C^{\rm B}$ of plans and A and B are obtained by summing trunk, labor, and equipment costs:

$$C^{\rm A} = C_{\rm T}^{\rm A} + \sum_i C_{\rm E}^{\rm A}(i)$$

$$C^{\rm B} = C_{\rm T}^{\rm B} + C_{\rm L}^{\rm B} + \sum_i C_{\rm E}^{\rm B}(i).$$

5. Conclusion

Crunching the numbers showed that the 1998 cost of Plan B was several million dollars less than that of Plan A. Based upon these financials, and the fact that Plan B requires fewer switch slots for trunk cards than Plan A, an executive decision was made to adopt Plan B.

This decision having been made, there are many tasks required to implement Plan B. These include revising the production network design tools to handle type B trunk cards, generating a new network design for the end of 1998 so that the high capacity facilities can be ordered and installed, and developing operations plans so the transition to type B trunks does not disrupt customer service.

Acknowledgments

I wish to thank Ron Levine, John Medamana, Scott Poucher, and an anonymous referee for their valuable suggestions, and Michael Shapiro for a careful reading of the paper.

Journal of Heuristics, 6, 21–38 (2000)
© 2000 Kluwer Academic Publishers

ATM Routing Algorithms for Multimedia Traffic in Private ATM Networks

R. IZMAILOV
NEC USA, Inc., C&C Research Laboratories, 4 Independence Way, Princeton, NJ 08540, USA

A. IWATA
C&C Media Research Laboratories, NEC Corporation, 4-1-1 Miyazaki Miyamae-ku, Kawasaki, Japan 216

B. SENGUPTA
NEC USA, Inc., C&C Research Laboratories, 4 Independence Way, Princeton, NJ 08540, USA

Abstract

The goal of this paper is to recommend a good Private Network-to-Network Interface (PNNI) routing algorithm for private ATM networks. A good routing algorithm has to work well with multimedia traffic with several quality of service (QoS) requirements (such as cell loss ratio, cell delay and its variation etc.) in different networks under various traffic conditions. The multiplicity of QoS requirements makes the routing problem NP-complete, so our approach to the problem is based on large scale simulations involving several empirical algorithms (compliant with the PNNI routing specification) which have been tested for different network topologies and traffic scenarios. Based on analysis of tradeoffs involving performance metrics (such as blocking rate, complexity, load distribution) we recommend a consistently good routing algorithm for single domain ATM networks.

Key Words: ATM networks, dynamic routing, PNNI, simulations, QoS, multimedia

1. Introduction

The problem of routing in Asynchronous Transfer Mode (ATM) networks poses a significant challenge for developers, since some of the key issues were not standardized by the ATM Forum, and were left at the discretion of individual manufacturers. In particular, the Private Network-to-Network Interface (PNNI) (PNNI 1.0, 1996) routing specification was designed to allow local choice in the route selection algorithm. Therefore, network administrators and developers of private ATM networks have to implement routing algorithms compliant with the PNNI formal specification. One of the main problems in the development of such algorithms is the multiplicity of quality of service (QoS) requirements of ATM calls (Guillen, Kia, and Sales, 1993; Jordan and Morse, 1994; Zhu, Liu, and Mouftah, 1995).

Standard optimal routing algorithms (such as Dijkstra (Evans and Minieka, 1992) provide a satisfactory solution if the required routing decision is based on a single additive QoS parameter (such as cost or time). However, it is known that the routing problem is NP-complete even if the number of QoS parameters is as few as two (shortest weight-constrained path (Garey and Johnson, 1979), pp. 214). Since the number of QoS parameters of an ATM call could be as high as four (cell loss ratio (CLR), cell delay variation (CDV), cell transfer delay (CTD) and bandwidth requirement), the PNNI routing problem is necessarily NP-complete and cannot be solved in the dynamical environment of ATM networks. In order to address the issue of QoS routing in ATM networks, a number of empirical algorithms have been proposed and studied (Guillen, Kia, and Sales, 1993; Jordan and Morse, 1994). A sample routing algorithm is presented in Appendix H of PNNI specification (PNNI 1.0, 1996).

Multiple QoS parameters could be aggregated into a single value which is optimized by the Dijkstra algorithm. The results of such routing are sensitive to the aggregation weights and there are no clear guidelines on how one should choose them (Lee, Hluchyj, and Humblet, 1995). The fallback algorithm was proposed in (Lee, Hluchyj, and Humblet, 1995); it uses sequential Dijkstra algorithms on different QoS requirements in the hope that one of the resulting paths, being optimal for one of the QoS requirements, will be also satisfactory for others. Yet another approach is to use the the administrative weights of network links for routing purposes. The administrative weight is chosen by a network administrator and is useful as an alternative means of optimization (by not using the more conventional criteria like hop count). In a default setting (PNNI 1.0, 1996), where all administrative weights are equal, a routing algorithm based on administrative weights minimizes the hop count for each call. In (Zhu, Liu, and Mouftah, 1995), simulations were done for NSFNET (14 nodes). Dynamic routing was considered for each call with pruning of the links that are unlikely to support the call.

Generally, PNNI routing algorithms should consist of two parts: the first part is a list of precalculated routes and the second part is the list of on-demand routing algorithms which are invoked if none of the precalculated routes satisfies the call's QoS requirements or if the call has been blocked in the previous setup process. PNNI routing algorithms should deliver good performance under diverse circumstances. The appropriate choice of all of these components is still an open issue.

In our previous work (Izmailov et al., 1995; Iwata et al., 1996) we outlined a general structure of a PNNI compatible routing algorithm and studied its performance on simple models. Here we continue the study of PNNI routing algorithm, using more realistic assumptions on network topologies and traffic flows. We keep the general structure of the algorithm the same as in (Izmailov et al., 1995) and elaborate on details and different new versions of the algorithm, testing different traffic and network scenarios. After analyzing our simulations results, we recommend an algorithm which performs very well.

The next sections are organized in the following way. In Section 2, we outline the general structure of our PNNI routing simulator. In Section 3 we describe the specific details of our simulations. In Section 4 we present some of the results of our simulations, and compare various performance metrics of different routing algorithms. In Section 5 we use our simulations results for recommending a specific PNNI routing algorithm, outlining its key performance characteristics. In the final section, we present our conclusions and outline further research plans.

2. PNNI routing simulator

In this section we describe the general structure of our PNNI routing simulator. The PNNI routing simulator (coded in C language) captures some of the key provisions of PNNI routing and signaling specifications, while omitting other features in order to obtain results of simulations in a reasonable time. Here we list the PNNI features implemented in our routing simulator.

- *Static network topology.* The routing simulator simulates a single private network as a lowest level peer group. The network consists of nodes and bidirectional transmission links connecting the nodes. Every link in the network is specified by its bandwidth and its propagation delay. For each type of traffic class, the links of the network also advertise their cell loss ratio (CLR), cell transfer delay (CTD), and cell delay variation (CDV). Each link is assigned its administrative weight by the network administrator. Each node of the network maintains a complete database of the network topology and the networks parameters (CTD, CDV, CLR and administrative weight).
- *Dynamic network topology.* Each link in the network is dynamically characterized by its available cell rate (AvCR). AvCR values change as new calls are established and old calls are terminated (and portions of the link bandwidth are reserved and released, respectively). These link state updates (LSU) inform other nodes on the current QoS parameters of the links and their congestion status. LSU is the key feature needed to support dynamic routing. The nodes of the network update their databases according to the LSU, carried by PNNI Topology State Packets (PTSPs), which are generated by all nodes as periodic time-based updates and as event-driven updates, reflecting significant change of link resources.
- *General structure of routing algorithms.* The routing in PNNI is source-based and connection oriented. If a call requests comes to the source node with its QoS parameters and its target destination, the source node has to calculate and establish a complete path for the call. Only after such a path is found, the connection is setup, and the call is allowed to start the transmission. The path calculation is based on the network topology database, currently available at the source node. The performance metrics advertised by the nodes on the path under consideration are compared with the requested QoS parameters using generic call admission control (CAC). If the advertised performance of the path is satisfactory, then the source node initializes the call setup process. Otherwise, the next algorithm from the routing sequence is used or the call is blocked (if the routing sequence is exhausted). If the call setup fails, another routing attempt could be made, or the call is blocked.
- *Generic call admission control (GCAC).* In order to estimate the ability of a path to accept a call, the QoS requirements of the call are compared with the QoS parameters advertised by links along the path. This is done by generic CAC at the source node. The QoS of the path is calculated by accumulating its additive parameters (CLR, CTD and CDV) and by calculating the minimum of the advertised AvCR for each link in the route. If the total QoS of the path meets the call's QoS requirements and the path has enough bandwidth to carry the call, then the path is considered acceptable for the generic CAC.
- *Call setup and real CAC.* Call setup is modeled as movement of the SETUP packet in the network from the source node to the destination node along the path, calculated by the source node. As the SETUP packet moves, the corresponding parts of the link bandwidth are allocated for the accepted call and are kept reserved until the termination of the call (or receiving the RELEASE packet). While moving along the call setup path, the SETUP packet accumulates aggregated information of the QoS parameters along

the path. If the call cannot proceed to the next node of the path (either because of a mismatch between the generic CAC and the real CAC, or because of the accumulated QoS parameters violate the call's QoS requirements), the RELEASE packet is generated, which is send to the source node, releasing the reserved part of the bandwidth. In case of a CAC mismatch, the source receives the identity of the link that caused the call setup blocking.

- *Granularity of simulations.* The routing simulator simulates traffic flows on the call level, not on the level of individual cells and packets. The simulation of packets is limited to the movements of SETUP, RELEASE and PTSP packets across the network.

3. Simulations

In this section we describe specific details of our simulations that were run on a SGI Challenge L computer using our PNNI routing simulator.

- *Static network topology.* In (Izmailov et al., 1995; Iwata et al., 1996) we studied ANSnet (18 nodes) and randomly connected networks of different sizes (10–20 nodes). In this study, we analyze a larger and a more complex campus network (schematically shown on figure 1). The campus network consists of 5 backbone nodes (switches) fully connected by 10 logical links of 1244Mbps capacity (each logical link consists of two parallel physical links of 622Mbps). The backbone nodes are connected to the central nodes (switches) of 8 domains (departments) with links of 622Mbps. Inside each domain, there are from 8 to 10 nodes, interconnected with 155Mbps links. Some of these intra-domain nodes have direct connections to the backbone 5 nodes via 622Mbps links, bypassing the corresponding central node. In our simulations, default administrative weights 5040 (PNNI 1.0, 1996) were assigned to each link.
- *Dynamic network topology.* Every 15 minutes, each node broadcasts the information about the current AvCR of all the links connected to this node. Also, if a significant change of AvCR occurs, it triggers an event-driven links-state update. The change is considered significant if the new value of AvCR deviates by more than 50% (50% is the default value of ACR_PM (PNNI 1.0, 1996)) from the previously advertised value of AvCR, while being more than 3% (3% is the default value of AvCRmT (PNNI 1.0, 1996)) of the total available bandwidth. The event-driven update cannot be generated unless at least 1 second (1 second is the default value of MinPTSETime (PNNI 1.0, 1996)) elapsed since the previous update. To simplify the simulations and reduce the number of events in the system, we send AvCR updates along the precalculated paths minimizing the propagation delay (instead of the flooding mechanism specified in PNNI). At each node, an update (PTSP packet) is processed for 0.023 seconds (this time was determined from testing) before becoming available in the database of the node. The same delay was assumed for processing the SETUP and RELEASE packets.
- *Traffic structure.* We simulated three types of traffic: UBR, VBR and CBR. We assumed uniform allocation of the load between these three traffic classes: each traffic type constituted a third of the total traffic flow. Traffic was sent from every intra-domain node to every intra-domain node (there are 64 intra-domain nodes in the campus network;

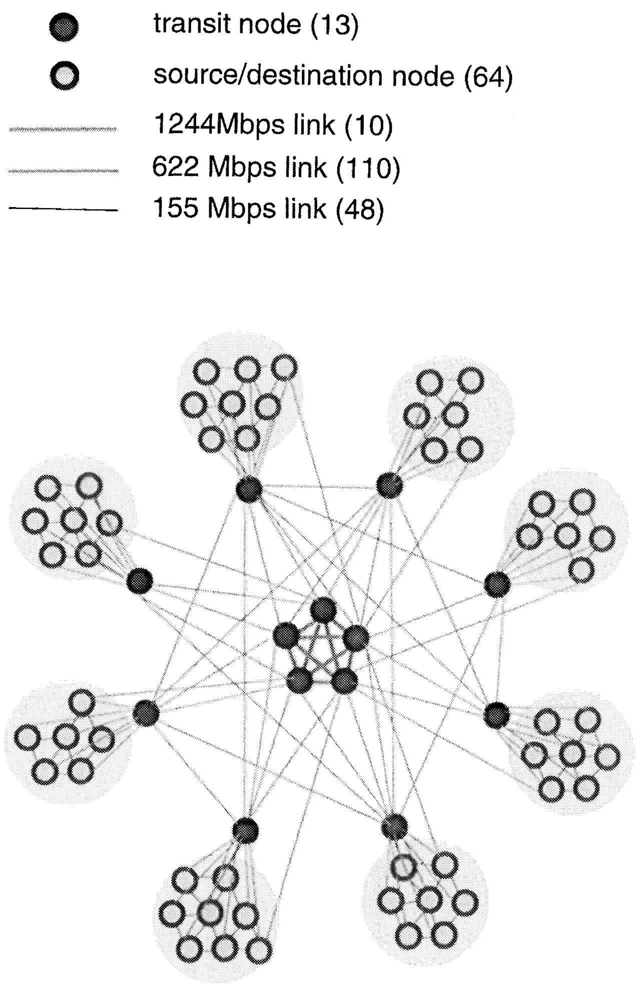


Figure 1. Campus network.

the remaining 13 transit nodes do not generate traffic). The traffic intensity depends on the relationship between the source and destination nodes: we assumed that 50% of the traffic is intra-domain traffic, whereas the other 50% of the traffic goes uniformly to other 7 domains. Since the campus network is not symmetrical, this arrangement generated different traffic loads on different links of the network.

Each traffic type was assumed to have the following CLR and CDV tolerances:

	UBR	VBR	CBR
CLR	10^{-5}	10^{-5}	10^{-8}
CDV	3.0×10^{-3}	1.5×10^{-3}	3.5×10^{-4}

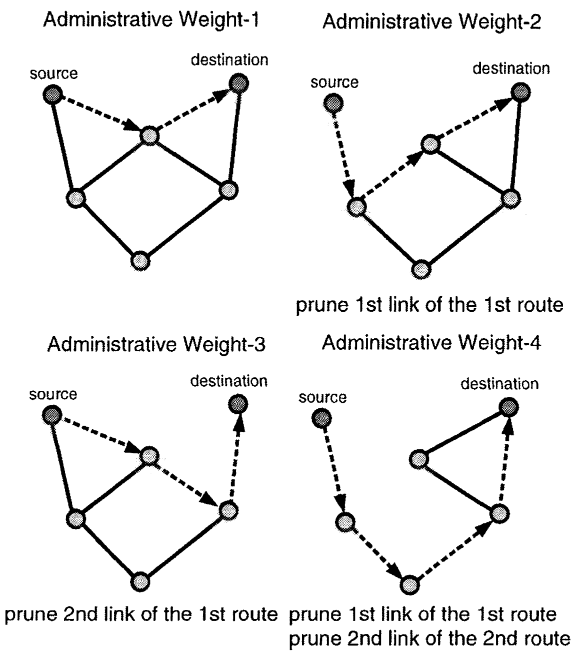


Figure 2. Four precalculated paths.

CTD for all the calls is assumed to be 50% more than the average delay across the network. The same ratio 50% was assumed for advertised CLR and CDV of the links. In other words, a path which is 50% longer than the average path will not be satisfactory in terms of delivering the requested QoS.

Both the call holding time and bandwidth requirements of each call were simulated as exponentially distributed random values. This arrangement generates calls of longer duration less frequently than those of shorter duration. Also, calls requiring less bandwidth occur more frequently than those requiring more bandwidth. A typical example, used for illustrative figures in this paper, is the average bandwidth requirement of 15Mbps with the average call holding time of 5 minutes (various other combinations were simulated as well).

Call interarrival times were simulated as exponentially distributed as random values. By choosing the appropriate average interarrival time, we simulated different network loads. Here load means the ratio of the traffic outflow from each source node to the average bandwidth of the links connected to each node when averaged over all the nodes of the network. Typical examples used for figures in this paper are 50% load and 70% load.

- *Real CAC.* In order to simulate mismatches between the real CAC (implemented at the nodes individually) and the generic CAC (used by the source node for estimating the applicability of a path for routing the call), the simulator generated a random error uniformly distributed from 0% to 10% of the link bandwidth and subtracted the error from the currently advertised AvCR. The error emulates AvCR fluctuations between sending out two consecutive PTSPs.

- *General layout of routing algorithms.* In our simulations, every node stored four precalculated routes to every other node along with the network topology information. The proposed choice of four precalculated paths is based on a selective pruning of some links. The number of pruned links was small, so the network connectivity did not deteriorate much. The following four routes were calculated.

  1. The first route is chosen by Dijkstra algorithm using administrative weight as the cost function.
  2. The second route is chosen by Dijkstra algorithm using administrative weight as the cost function on the network, where the first link of the first route is pruned.
  3. The third route is chosen by Dijkstra algorithm using administrative weight as the cost function on the network, where the second link of the first route is pruned.
  4. The fourth route is chosen by Dijkstra algorithm using administrative weight as the cost function on the network, where the first link of the first route and the second link of the second routes are pruned.

  These four routes were selected in order to produce a diversified pool of routes. Ordering the routes according to their total administrative weight, we denote them as AW1, AW2, AW3 and AW4: AW1 is the shortest route, whereas AW4 is the longest one. Since all the links have the same administrative weight, AW1 is a min-hop route from source to destination.

  Besides the precalculated routes, on-demand attempts were used as well. They were denoted as **1/AvCR on demand** and **AW on demand**. Here **1/AvCR on demand** means that on-demand routing is accomplished by executing the Dijkstra algorithm using inverse AvCR of the links as the cost function, **AW on demand** means that on-demand routing is accomplished by executing the Dijkstra algorithm using administrative weight of the links as the cost function, and **1/AvCR+AW on demand** denotes two sequential on-demand routing attempts, based on 1/AvCR and on administrative weight, respectively. In all on-demand routing attempts, before executing the Dijkstra algorithm, the QoS violating link is pruned from the network (if appropriate), as well as those links that cannot accept the call according to the current database information.

- *Performance metrics.* In order to analyze performance of the simulated routing algorithms, we collected various statistical data on calls-related and network-related performance metrics. The following performance statistics were collected for each call.

  - Blocking rate: this is the key performance metric.
  - Signaling load: average number of messages per call generated during the call setup process. This metric measures the load of signaling on the network by multiple routing and rerouting attempts.
  - Dijkstra rate: the average number of times per call a Dijkstra algorithm was executed, whether or not the call was blocked. For a network with N nodes and E edges, every Dijkstra algorithm requires $O(E \ln N)$ running time (Cormen, Leiserson, and Rivest, 1997) with a reference $5N^3\mu$s of processing time for a network with N nodes on SGI Challenge L computer, so the Dijkstra rate serves as a measure of complexity and costs associated with a particular routing algorithm.

– Efficiency: the ratio of blocking probability to the frequencies of mismatches between the real CAC and the generic CAC. The call blocking may occur either if the route was not found (none of the precomputed and on-demand routes is acceptable), or if the route was found, but was blocked due to a mismatch between the real CAC and the generic CAC. The closer is the ratio to 1, the better is the efficiency of precalculated part of routing: most of the call blockings occur during call setup phase because of CAC mismatches, whereas the probability of not finding the route is relatively small.

All calls are classified into 10 bandwidth slots (referred to as $R = 0, \dots, 10$). Here $R = 0$ corresponds to the calls in the range 0–10Mbps, $R = 1$ corresponds to 20–30Mbps, ..., $R = 9$ corresponds to 90–155, and aggregated statistics is described by $R = 10$. Performance metrics were calculated separately for each bandwidth slot $R = 0, \dots, 9$ and for the aggregated slot $R = 10$.

We also measured network performance by calculating link occupancy statistics (minimum, maximum, average and variance of link occupancy).

4. Results of simulations

In this section we analyze results of our simulations and compare performance of different routing algorithms. The algorithms use different number of precalculated paths and different number of on-demand attempts. They also use different methods of initializing and selecting the paths. We address the issues of optimal number of precalculated paths and on-demand routing attempts, compare different methods of selecting the paths using different performance metrics. By testing them under various scenarios and using various performance metrics, we chose those algorithms that provide superior performance and thus sequentially narrow the scope of algorithms under consideration.

4.1. Number of precalculated paths

Assuming that several precalculated paths can be used in routing, we analyzed the routing efficiency of the number of precalculated paths in the absence of on-demand routing. On the one hand, the more precalculated routes are used, the lower could be the blocking probability. On the other hand, since each source node has to store all the precalculated routes to every destination node in the network, additional precalculated routes require extra memory for their storage. In our simulations, we studied the dependence of the number of precalculated paths and the blocking probability. Figure 3 shows a typical example of improvement in routing efficiency for all 10 bandwidth slots as the number of precalculated routes increases from 1 to 4. As the figure shows, routing efficiency tends to 1 as the number of precalculated routes increases from 1 to 3. However, for calls with moderate requirements for bandwidth (less than 60Mbps), the improvement in blocking probability is insignificant when the number of precalculated routes is further increased further from 3 to 4. For calls with higher requirements for bandwidth, the 4th precalculated route can give

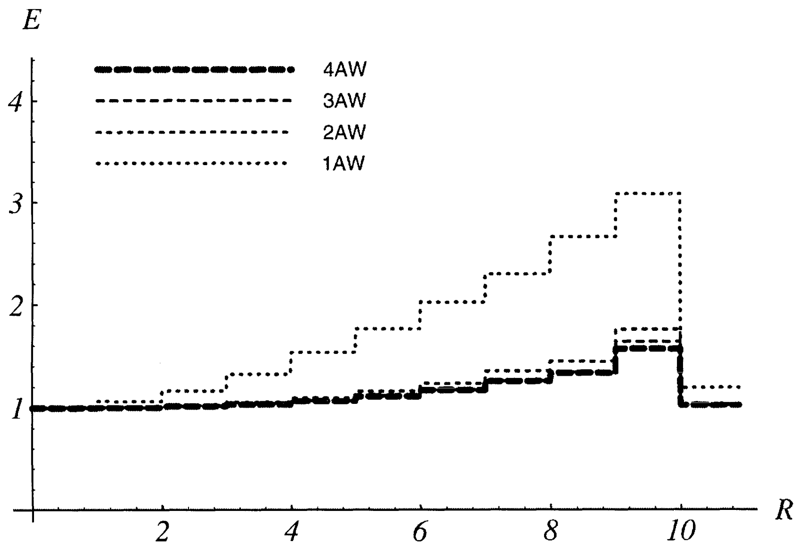


Figure 3. Efficiency of precalculated routing (50% load).

a visible improvement in efficiency. Figure 3 also suggests that further increase of number of precalculated routes will not provide significant improvement, since the efficiency of four precalculated paths is already close to 1: call blockings occur mostly because of outdated link state information.

Since four precalculated paths cover most of the possible precalculated routing, we assumed that the precalculated part of the routing algorithm consists of four precalculated routes in all our subsequent simulations.

4.2. Selection of precalculated paths

There are different methods of selecting the trial paths from the set of four precalculated paths. We considered the following four methods.

- **4AW** Paths are selected sequentially, starting from the first path, until the fourth path.
- **RR** Paths are selected in a round-robin fashion.
- **4random** Paths are selected randomly from the set of four precalculated paths.
- **4AWr** Paths are selected sequentially, with randomization within the groups of paths of the same total administrative weight. Depending on the relative aggregated administrative weights $L_1, \ldots, L_4$ of four different precalculated paths AW1, ... ,AW4, the following 8 cases are possible.

1. (AW1) $\rightarrow$ (AW2) $\rightarrow$ (AW3) $\rightarrow$ (AW4) if $L_1 < L_2 < L_3 < L_4$.
2. (AW1 AW2) $\rightarrow$ (AW3) $\rightarrow$ (AW4) if $L_1 = L_2 < L_3 < L_4$.
3. (AW1) $\rightarrow$ (AW2 AW3) $\rightarrow$ (AW4) if $L_1 < L_2 = L_3 < L_4$.

4. (AW1) $\rightarrow$ (AW2) $\rightarrow$ (AW3 AW4) $L_1 < L_2 < L_3 = L_4$.
5. (AW1 AW2) $\rightarrow$ (AW3 AW4) if $L_1 = L_2 < L_3 = L_4$.
6. (AW1 AW2 AW3) $\rightarrow$ (AW4) if $L_1 = L_2 = L_3 < L_4$.
7. (AW1) $\rightarrow$ (AW2 AW3 AW4) if $L_1 < L_2 = L_3 = L_4$.
8. (AW1 AW2 AW3 AW4) if $L_1 = L_2 = L_3 = L_4$.

Here paths in parentheses are selected randomly. For example, the case 2 means that the selection of 4 paths is done in the following sequence:

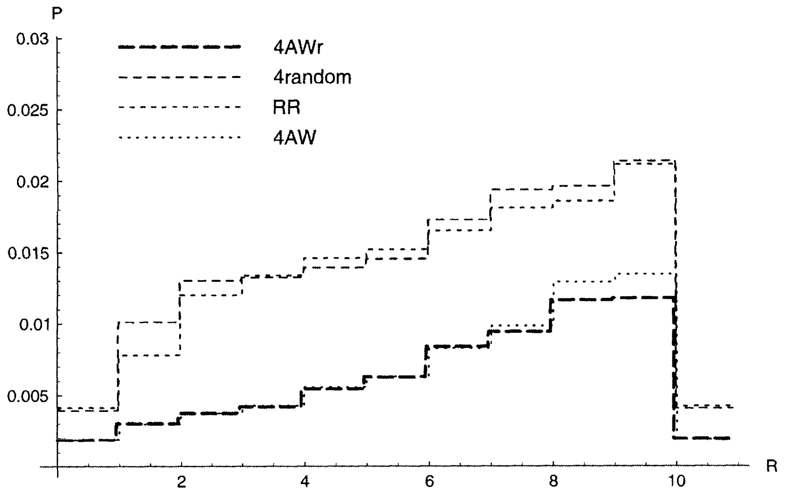
1. randomly chosen path from the set {AW1, AW2},
2. the remaining path from the set {AW1, AW2},
3. the path AW3,
4. the path AW4.

If there is less than four different paths, then the choice of paths is adjusted correspondingly. Figure 4 shows the blocking probabilities for each of the four selection methods. The round-robin (**RR**) and the random (**4random**) methods have significantly worse blocking probabilities than the sequential (**4AW**) and the sequential with randomization (**4AWr**) approaches. The reason is that four precalculated paths differ in quality (the fourth path typically consists of more links than the first one) and both **4random** and **RR** methods make it possible to choose the longest path first when a better path is available. Therefore, the selection of precalculated paths should be done by one of the two other algorithms, **AW** or **4AWr**.

The blocking probability of **4AWr** in slightly less than the blocking probability of **4AW**. The smaller blocking probability of **4AWr** means that more calls are accepted into the network and, since the average bandwidth requirement of a call is 15Mbps, the average link occupancy correspondingly increases (in our simulations, by about 10Mbps). To illustrate this effect, we consider the set of 48 links of 155Mbps capacity and analyze the load of the links in this subnetwork. Figure 5 shows average of AvCR for each of the 48 links during the first 12 hours: As these figures illustrate, **4AWr** accepts and distributes a larger load than **4AW** does. As we will see further, the better load balance of **4AWr** is beneficial if combined with on-demand routing. In our sequential simulations, we used both selection methods **4AWr** and **4AW**.

4.3. On-demand routing

On-demand routing is invoked either if none of the precalculated paths satisfy the QoS requirements, or if the call setup fails and the source node tries to reroute the call. On-demand routing has a potential of reducing the blocking probability, since it can use the current network information more effectively than precalculated routes. On the other hand, on-demand routing is expensive, since it involves an execution of Dijkstra algorithm on the pruned network (the QoS violating link has to be pruned, as well as other links that are not likely to support the call). We simulated algorithms with one and two on-demand routing attempts, and compared their performance with each other and with the purely precalculated routing (without on-demand routing) based on **4AW** algorithm. The followingon-demand



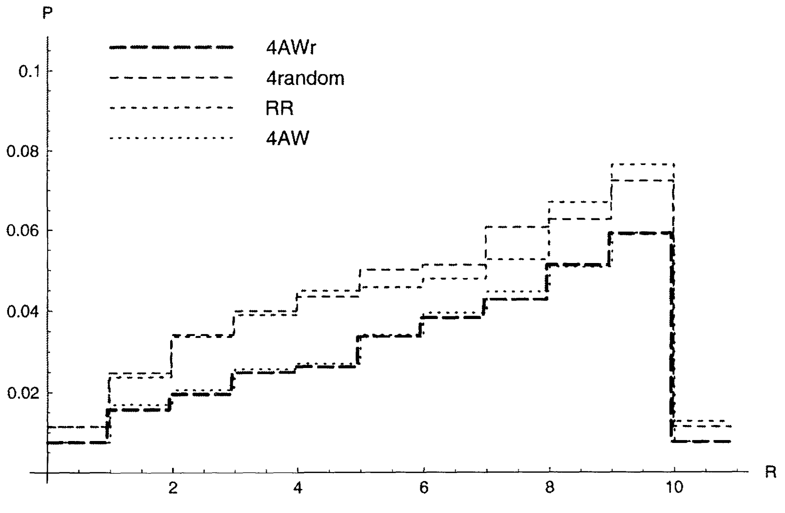


Figure 4. Blocking probabilities for 4 paths (50% and 70% load).

routing algorithms were simulated.

1. **4AW → 1/AvCR on demand**
2. **4AW → AW on demand**
3. **4AW → 1/AvCR+AW on demand**
4. **4AWr → 1/AvCR on demand**
5. **4AWr → AW on demand**
6. **4AWr → 1/AvCR+AW on demand**

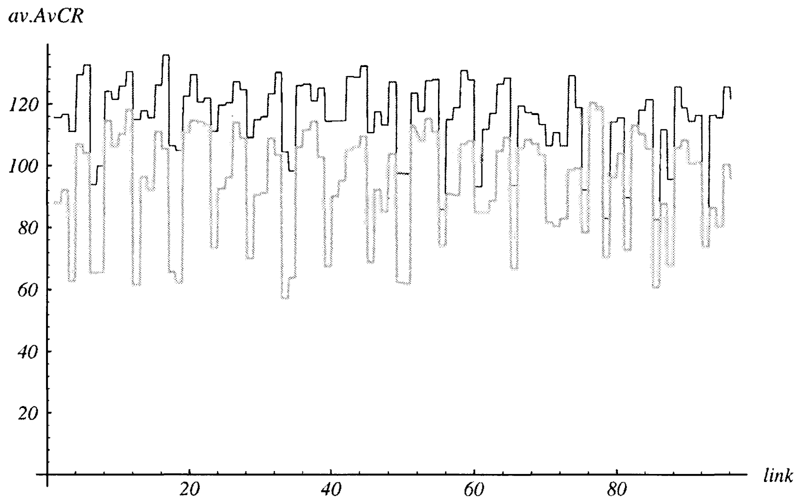


Figure 5. Average of AvCR at each of the 48 155Mbps links of the enterprise network over 12 hours of real time (**4AW** (thin line) and **4AWr** (thick line)).

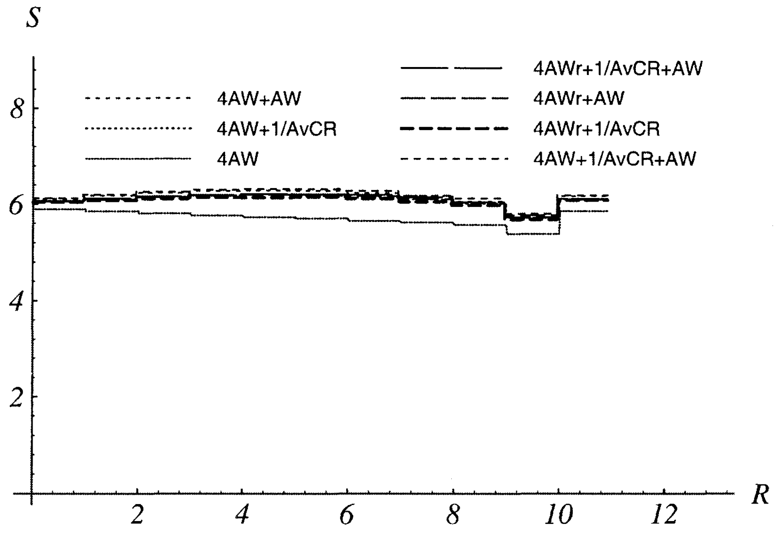
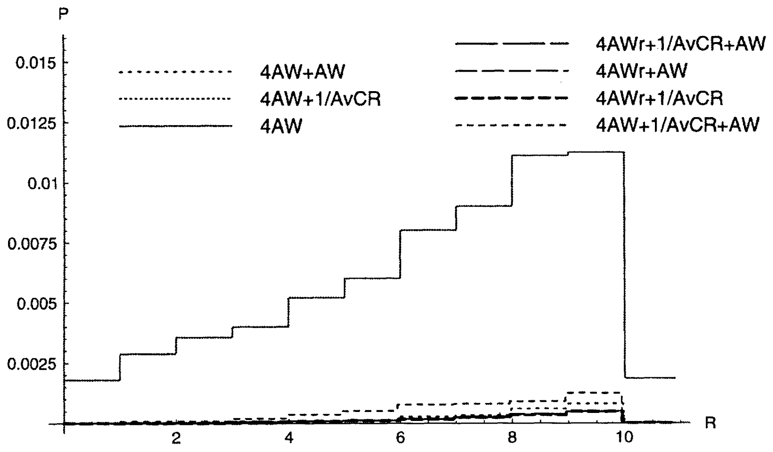


Figure 6. Signaling load for 4 paths with on-demand routing (50% load).

Figures 6–8 show typical behavior of the key performance metrics (signaling load, blocking probability and Dijkstra rate, respectively) for all 6 on-demand routing algorithms and **4AW**.

As figure 6 shows, there is no significant difference in signaling load between different algorithms and for different bandwidth slots. The bulk of the signaling load is produced by the call setup process, and **4AW** produces slightly less signaling load for calls with larger bandwidth requirements. The reason for that is because these calls aremore likely to be



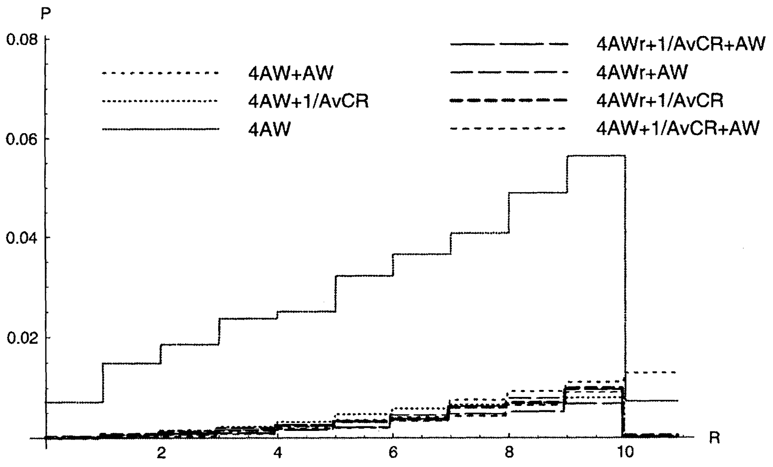


Figure 7. Blocking probabilities for 4 paths with on-demand routing (50% and 70% load).

blocked at the source and they do not generate call setups at all, if the blocking occurs. Other algorithms (with on-demand routing) produce very similar signaling loads with smaller dips for calls with higher bandwidth requirements.

As figure 7 shows, the blocking probability increases as the bandwidth slot R increases: the larger is the bandwidth requirement for the call, the larger is the probability that the algorithm fails to route the call (the same effect is visible on 4). Figure 7 also shows that all 6 on-demand routing algorithms are better than **4AW** in terms of blocking probability. On-demand routing algorithms decrease the blocking probability significantly, up to 4 times for calls with limited bandwidth requirements. However, the differences between 6 on-demand

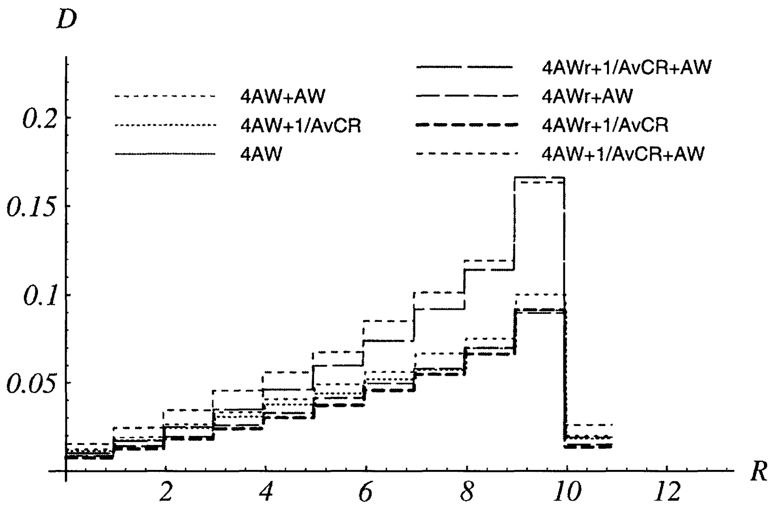


Figure 8. Dijkstra rates for 4 paths with on-demand routing.

routing algorithms are very small. In particular, there is no significant difference between one and two on-demand routing attempts.

On the other hand, as figure 8 shows, an extra on-demand routing attempt adds to the processing time. This observation suggests that we should consider only algorithms with a single on-demand routing attempt, since an extra attempt just uses additional processing time, without really improving the blocking probability.

Analyzing the blocking probability among the remaining 4 algorithms, we conclude that both **4AWr → AW on demand** and **4AWr → 1/AvCR on demand** has slightly smaller blocking probability than **4AW → AW on demand** and **4AW → 1/AvCR on demand**. The reason for that is the load balance provided by **4AWr** is better than that of **4AW**, so the on-demand routing attempt is more efficient for **4AWr**. It also has better network performance (better load balancing).

5. Routing algorithm

In determining a good routing algorithm we take into account the tradeoffs between different performance characteristics. On the one hand, the algorithm has to have a consistently low blocking probability under different circumstances (different network connectivity, different traffic loads and different QoS requirements). An algorithm which is optimal for one case should not become substandard in other cases. Blocking probabilities serve as a measure for this objective. On the other hand, the algorithm should be reasonably inexpensive and should not waste network resources by futile rerouting attempts. Dijkstra rates and signaling load serve as measures for this objective. The overall network performance (load distribution, lack of persistent hot spots and traffic oscillations) is an additional performance measure.

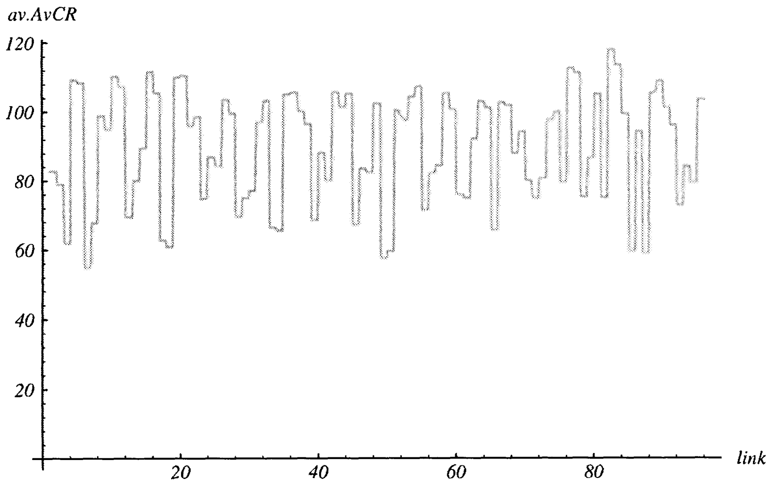


Figure 9. Average of AvCR at each of the 48 155Mbps links of the enterprise network over 12 hours of real time.

The results of simulations discussed above suggest that the most appropriate routing algorithm is **4AWr $\rightarrow$ 1/AvCR on demand**. This routing algorithm is based on four precalculated paths from each source node to each destination node. The precalculated paths are chosen by the Dijkstra algorithm with administrative weight as the cost function on the original network and its pruned subnetworks. Before each execution of the Dijkstra algorithm, a small randomization in administrative weights is introduced. If all the precalculated paths are exhausted, or if the call setup is blocked, a single on-demand routing attempt is executed, using Dijkstra algorithm on inverse AvCR of the links in the pruned network. In all other cases, the call is blocked.

This PNNI routing algorithm provides a significant performance improvements over fixed single path routing (blocking probability is 4–20 times less). As illustrated in figure 3, the algorithm provides efficient routing: most call blocking is due to outdated information, and most of the blocking can be handled by the on-demand routing attempt.

The algorithm exhibits good network performance. It has reasonably even distribution of load among the links (in the case illustrated on figure 9, all links are loaded on average by 80–120Mbps), limited load variance (as figure 10 illustrates) and lack of oscillations (as figure 12 illustrates). Figure 11 shows the behavior of hot spots (the most congested links, defined as the links with minimum AvCR among the 48 links of the subnetwork) during the first 3 hours of real time. As the figure illustrates, there are no persistent hot spots: the maximally loaded links change very frequently.

The implementation of the algorithm requires enough memory at every source node for storing 4 precalculated routes to every destination node. In the case of the campus network, where all paths have at most 4 hops, the required amount of memory is under 1 K (to 64 destination nodes). The implementation of on-demand routing part of the algorithm (used in less than 5% of all cases) requires enough processing time for execution of the Dijkstra algorithm.

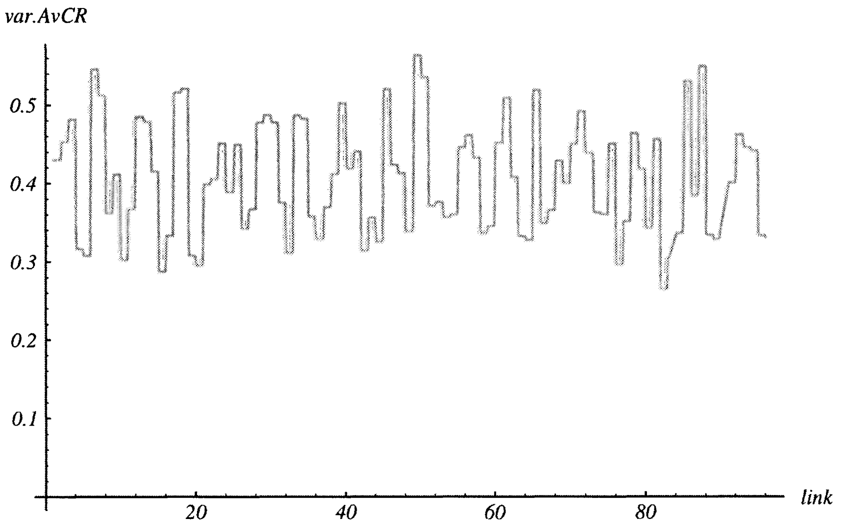


Figure 10. Variance of AvCR at each of the 48 155Mbps links of the enterprise network over 12 hours of real time.

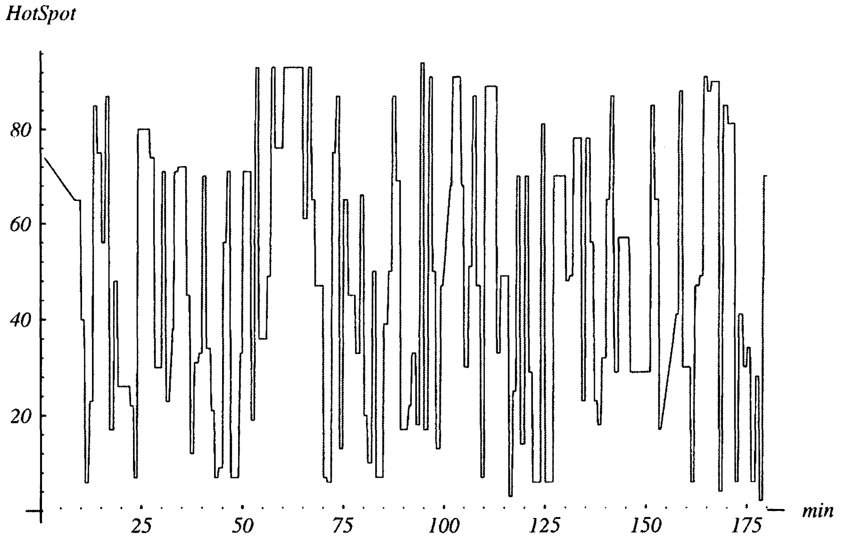


Figure 11. Hot spots (the most congested links) among the 48 links during the first 3 hours of real time.

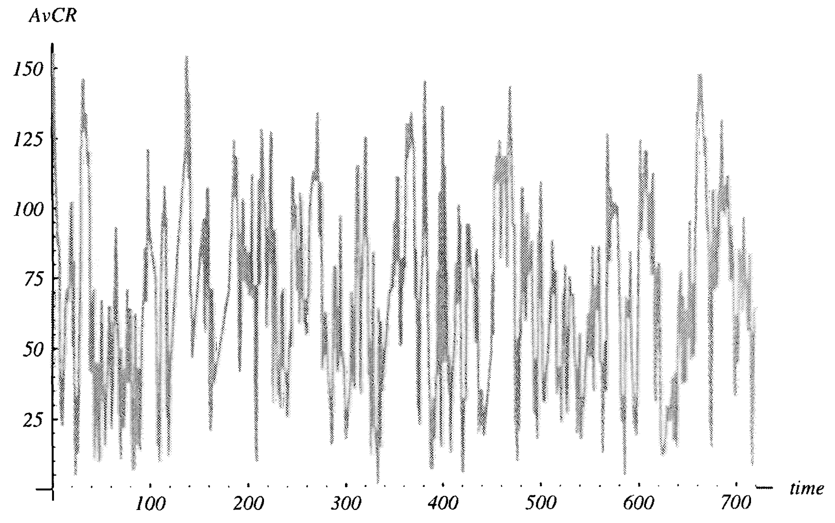


Figure 12. Occupancy of a randomly selected link in the enterprise network during 6 hours of real time.

6. Conclusions

In this paper, we have recommended a good PNNI routing algorithm for routing in private ATM networks. Our recommendation is based on a detailed simulation study. The simulation model quantifies the call blocking probability for a variety of calls with different bandwidth requirements, loads and several choices of the algorithm. The choice of the recommended algorithm is based on a tradeoff between low blocking probability and ease of implementation. As more statistical data become available upon implementation of this routing algorithm, we plan to continue our analysis and to fine-tune the algorithm for better performance in the next release.

Another challenge is to provide recommendations for choosing administrative weights. Currently, all links have the same default administrative weight. Potentially, administrative weight could be used for changes of traffic flows. We plan to address these issues in our future research.

References

Cormen, T., C. Leiserson, and R. Rivest. (1997). *Introduction to Algorithms.* MIT press.

Evans, J.R. and E. Minieka. (1992). *Optimization Algorithms for Networks and Graphs.* New York: Marcel Dekker, Inc.

Garey, M.R. and D.S. Johnson. (1979). *Computers and Intractability—A Guide to the Theory of NP-Completeness.* California, USA: Freeman.

Guillen, A., R.N. Kia, and B. Sales. (1993). "An Architecture for Virtual Circuit/QoS Routing." *Proceedings of 1993 International Conference on Network Protocols.* Los Alamitos, CA.

Izmailov, R., A. Iwata, D.-S. Lee, G. Ramamurthy, B. Sengupta, and H. Suzuki. (1995). "Routing Algorithms for ATM Networks with Multiple Quality of Service Requirements." *Proceedings of the First Workshop on ATM Traffic Management*, Paris, France. pp. 349–356.

Iwata, A., R. Izmailov, D.-S. Lee, G. Ramamurthy, B. Sengupta, and H. Suzuki. (1996). "ATM routing algorithms with multiple QOS requirements for multimedia internetworking," *IEICE Transactions on Communications*, E79-B, 8, pp. 999–1007.

Jordan, T.P., and M.J. Morse. (1994). "Routing Algorithms in ATM Networks." *Proceedings of IEE Eleventh UK Teletraffic Symposium*. London, UK.

Lee, W.C., M.G. Hluchyj, and P.A. Humblet. (1995). "Routing subject to service constraints in integrated communication networks," *IEEE Network*, pp. 46–55.

PNNI 1.0. (1996). Private Network-Network interface Specification Version 1.0 (PNNI 1.0).

Zhu, T., C. Liu, and H.T. Mouftah. (1995). "Dynamic Routing for Multimedia Traffic over ATM Networks." *Proceedings of IEEE Symposium on Computers and Communications*, Alexandria, Egypt, pp. 91–97.

Journal of Heuristics, 6, 39–64 (2000)
© 2000 Kluwer Academic Publishers

Fast, Efficient Equipment Placement Heuristics for Broadband Switched or Internet Router Networks

JOEL W. GANNETT
Telcordia Technologies, 331 Newman Springs Road, Room NVC 3X-323, Red Bank, New Jersey 07701-5699, USA*
email: jgannett@research.telcordia.com

Abstract

Planning and designing the next generation of IP router or switched broadband networks seems a daunting challenge considering the many complex, interacting factors affecting the performance and cost of such networks. Generally, this complexity implies that it may not even be clear what constitutes a "good" network design for a particular specification. Different network owners or operators may view the same solution differently, depending on their unique needs and perspectives. Nevertheless, we have observed a core common issue arising in the early stages of network design efforts involving leading-edge broadband switched technologies such as ATM, Frame Relay, and SMDS; or even Internet IP router networks. This core issue can be stated as follows: Given a set of service demands for the various network nodes, where should switching or routing equipment be placed to minimize the Installed First Cost of the network? Note that the specified service demands are usually projections for a future scenario and generally entail significant uncertainty. Despite this uncertainty, we have found that network owners and operators generally feel it is worthwhile to obtain high-level advice on equipment placement with a goal of minimizing Installed First Cost. This paper reports on a heuristic approach we have implemented for this problem that has evolved out of real network design projects. A tool with both a Solution Engine and an intuitive Graphical User Interface has been developed. The approach is highly efficient; for example, the tool can often handle LATA-sized networks in seconds or less on a workstation processor. By using only nodal demands rather than the more complex point-to-point demands usually required in tools of this sort, we have created an approach that is not only highly efficient, but is also a better match to real design projects in which demand data is generally scant and highly uncertain.

Key Words: telecommunication networks, optimization, network design, equipment placement, switched network, router network, internet

1. Introduction

Planning and designing the next generation of IP router or switched broadband networks seems a daunting challenge considering the many complex, interacting factors affecting the performance and cost of such networks. Generally, this complexity implies that it may not even be clear what constitutes a "good" network design for a particular specification. Different network owners or operators may view the same solution differently, depending on their unique needs and perspectives. For example, some operators may be interested in an unconstrained desert-start solution in which the widest possible collection of network parameters and architectural facets are explored with the goal of attaining the lowest possible

**Telcordia Technologies, Inc.* was formerly known as *Bellcore.*

Installed First Cost for the final design. In other cases, various restrictions might apply that must be accounted for during the optimization search. Some network operators, for example, may be interested in an evolutionary design. Instead of a desert-start scenario, these operators may have an existing infrastructure of switches and communication links that they wish to grow efficiently to accommodate future expected demand growth. Other network operators may be burdened by regulatory restrictions preventing certain money-saving network configurations. Still other operators may have stringent, self-imposed Engineering Guidelines dictating the capacity of backbone links and the minimum and maximum number of backbone links impinging on each switching hub site. In addition, some ATM network operators may have Quality of Service requirements dictating "*N*-hop" architectures; that is, there must exist a path in the network between every pair of nodes that transits no more than N ATM switches ($N = 2$ or 3 is typical). Finally, some network operators may require that their designs obey a panoply of seemingly arbitrary rules. The bases for these rules may be proprietary, in which case the network design consultant may have to apply these rules on "blind faith" while only guessing at their justification.

Despite these complexities, we have observed a core common issue arising in the early stages of network design efforts involving leading-edge broadband switched technologies such as ATM, Frame Relay, and SMDS; or even Internet IP router networks. This core issue can be stated as follows: Given a set of service demands for the various network nodes, where should switching or routing equipment be placed to minimize the Installed First Cost of the network? Stated broadly, this optimization problem involves balancing switching (or routing) equipment cost against homing and backbone transmission costs to decide how switching (or routing) functionality should be distributed throughout the network.

Note that the specified service demands are usually projections for a future scenario and generally entail significant uncertainty. Historically, POTS (Plain Old Telephone Service) was the dominant service carried by public telecommunication networks. In the POTS scenario, in which service demand growth was relatively slow and predictable, the development of sophisticated network planning and design algorithms provided substantial economic payoff to telecom service providers. Today, chaos reigns as the explosive growth of broadband data services—especially those tied to the Internet—leave service providers scrambling to grow network capacity faster than demand. Despite this uncertainty in future demand, we have found that owners and operators of leading-edge carrier or enterprise[1] networks often wish to obtain high-level advice on equipment placement with a goal of minimizing Installed First Cost. Note that the degree of modeling detail or the degree of sophistication in the optimization routines has to be chosen appropriately and carefully to match the relatively broad-brush nature of this problem. Highly sophisticated formal optimization techniques whose results depend critically on fine-grained details of demand data are inappropriate because of the large demand forecast uncertainties (e.g., 100% or more) that are typical for next-generation broadband technologies (McQuillan, 1995).

This paper reports on a heuristic approach we have implemented for such network design problems that has evolved out of real design projects. A tool with both a Solution Engine and an intuitive Graphical User Interface has been developed. The approach and implementation described here is highly efficient; for example, it can often handle LATA-sized networks[2] in seconds or less on a workstation processor. By using only nodal demands rather than the more complex point-to-point demands usually required by tools of this sort, we have created an approach that is not only highly efficient, but also a better match to real design projects in which demand data is generally scant and highly uncertain. This approach provides useful results for network planning and early-stage network design, and is also valuable in other arenas where data is scant but answers are required, such as financial analysis tools used for pricing the tariffs of proposed new services.

In the remainder of this paper, Section 2 describes the problem to be solved with a particular emphasis on the practical issues that have been observed in real network design projects. We emphasize practical issues throughout this paper since it appears that practical issues, while all-important, often get short shrift. Section 3 provides a description of heuristic algorithms the author has devised to address this network design philosophy, while Section 4 describes the experience with implementing these algorithms in a prototype tool that has been successfully used in Telcordia network design projects. Finally, the summary and conclusions are presented in Section 5.

Note: In several places in this paper, the phrase "switch placement" (or variations of it) may be used for brevity. This phrase could equally well read "switch or router placement," which reflects the fact that these techniques can be applied equally well to switch or router networks.

2. Equipment placement problem description

2.1. What is the equipment placement problem?

First, it is important to clarify what is meant by the Equipment Placement Problem. The results described in this paper are oriented toward minimizing the network's Installed First Cost (IFC).

Although criteria other than IFC may matter in some network design projects (e.g., criteria such as Operations Costs or Quality-of-Service goals), such non-IFC criteria are often nebulous and difficult to address in high-level design tools. In practice, minimum IFC solutions do provide useful feedback to network designers, and non-IFC criteria can be addressed in some cases by suitably choosing the cost parameters of the tool's solution model. For instance, Operations Costs are often estimated as a percentage of the Installation Costs of network hardware (with perhaps different percentages for different classes of network hardware). When this approximation is chosen, one can incorporate Operation Costs into the solution by merely adding the appropriate Operation Cost increment to the cost of each class of network hardware.

Basically, there are two physical resources that must be allocated to create high-level minimum IFC designs for IP router or switched broadband networks:

1. *Network switching or routing elements:* We want to know where to place network elements—whether they be ATM switches or multiplexers, Frame Relay switches, SMDS switches, or IP routers. Besides selecting the equipment sites, we want to know how many of each of the various models of equipment should be placed at the selected equipment sites, and how the equipment should be configured with plug-in modules.

In general, we wish to specify an arbitrary collection of equipment models and let the software decide which is best from a cost minimization standpoint. Non-homogeneous solutions, that is, solutions consisting of a mixture of different equipment models, may be acceptable if such a mixture results in lower cost.

2. *Transmission links:* We want to know where to place transmission links and what their capacities should be. That is, we want to identify every pair of nodes that should be connected directly by a transmission link, and what the bit rate of that link should be. When we say "connected directly," we mean that there are no intervening placed network switching or routing elements (see Item 1 above) along the transmission path. *Access* (or *homing*) links carry demand from non-switch (or non-router) nodes to switch (or router) nodes. *Backbone* links carry traffic between switch (or router) nodes. For enterprise networks, we model links as leased lines whose costs are determined by tariffs. For carrier networks, we model links as point-to-point systems with a multiplexer (MUX) at both ends, such as a SONET terminal MUX. Mixtures of leased lines and carrier links may be allowed in some network architectures, so the solution algorithm must be able to investigate these possibilities. The point-to-point system modeling of carrier links may provide only an approximate cost model for cases where the network operator desires to use more complex transport topologies such as SONET protection rings. For such cases, the final transport network architecture would be designed with a special-purpose transport design tool (Cosares et al. 1995) that obtains its input from the output of the Equipment Placement algorithm.

These physical resource allocation decisions are driven by the user's demands on the network, that is, the interconnection services that users will require to support their desired communication needs. We will talk more about network demands below.

2.2. *Practical considerations*

When viewed as a general, abstract computer science problem, the Equipment Placement Problem described above is enormously difficult and complex—even when based on a relatively simple IFC cost metric. Owing to the excessive computation resources that are required by such complex problems, it is virtually impossible to solve these problems exactly (i.e., find the network configuration giving the guaranteed global minimum of the cost metric) in all but the most trivial, artificial examples.

However, there are several practical aspects to these problems that allow us to devise heuristics that, while not guaranteeing a true optimal solution, can be executed efficiently on real network design problems and usually give solutions that are both useful and "good." We shall describe these practical aspects presently.

2.2.1. The myth of readily-available demand data. First, as mentioned above, user demands are the dominant factor controlling the eventual solution in the network design process. In most cases, we are working with projected (future) user demands since we are designing a network that has not yet been built. Thus, there is substantial uncertainty about what the actual demands will be once the network is installed. Projections may be highly uncertain and it is not surprising if they turn out to be inaccurate. But even when we work exclusively with today's demands—as, for example, when we are looking to re-design an existing network—this author has found that there can still be substantial uncertainty as to what the actual demands are. Whether this uncertainty is simply a database access problem, or whether network operators really do not have a complete and correct record of their current network demand, we do not know. The bottom line, though, is that in many design projects it is exceedingly difficult (indeed, virtually impossible) to get reliable data on network demands, either current measurements or future projections. At least, this has been this author's first-hand experience in working on real network design projects.

Given these difficulties and uncertainties surrounding the demand data that drives the eventual network solution, the central practical issue in the Equipment Placement Problem becomes clear:

> It is not appropriate to create a solution procedure and associated architecture style whose answers are critically-dependent on precise demand data details, when in fact network operators cannot provide precise demand data.

In particular, let's look at the issue of so-called "point-to-point" demands vs. so-called "nodal" demands.

2.2.2. Nodal demands vs. point-to-point demands. By "point-to-point" demand, we mean a listing of required connections in the network. Each connection consists of two end points, a so-called "A end" and a so-called "Z end," an indication of the "service" (i.e., type of channel) providing a communication path between those end points, and a count of the number of service instances represented by the point-to-point demand. Since the number of possible pairs of end points in a network grows as N^2 (where N is the number of nodes in the network), point-to-point demand data can grow as the square of the network node count and often becomes voluminous and complex for large networks.[3] Point-to-point demands are illustrated in figure 1(a).

A simpler type of demand data for networks are "nodal" demands. This is simply a list of the count of services at each node. Each such service will be terminated on a switch (or router) somewhere in the network; either at that node if that node is chosen to be a switch (or router) site, or at some other node. Clearly, nodal demand complexity grows only in direct proportion to N. For large networks (large N), this means that nodal demands will be considerably simpler than point-to-point demands. Note that it is trivial to generate a nodal demand set from a point-to-point demand set, if the latter is available: for each node in the network, merely add up the total number of point-to-point demands terminating at that node to obtain the nodal demand at that node. This author has found that network operators can always provide at least an estimate of current or future projected nodal demands for the purpose of network design, while point-to-point demand data is far less available from network operators. For both nodal and point-to-point demand data, this author has found that the quality of the data provided by network operators is often problematic. Not surprisingly, however, the quality of point-to-point demand data (when it is available) tends to be more problematic than the quality of nodal demand data owing to the greater complexity of point-to-point demands. Nodal demands are illustrated in figure 1(b).

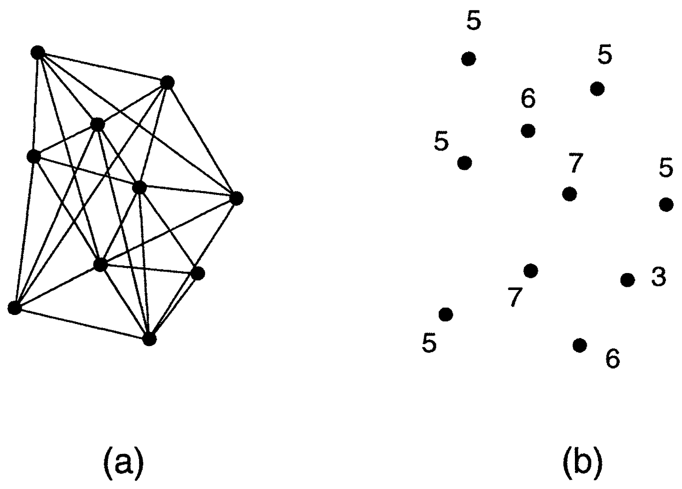


Figure 1. (a) Point-to-point demands. (b) Nodal demands. Each line segment in (a) represents a point-to-point demand, and we assume in this example that the size is a single service unit, such as DS1. The integers in (b) represent the sum of the number of service units in (a) that terminate on that node.

To forecast future demand, one often chooses (for lack of a better estimate) to scale upward existing demand data. Note that scaling nodal demands is relatively straightforward, as it involves increasing the service counts at the nodes in a deterministic or stochastic way. When point-to-point demands are scaled upward, however, one must worry about whether and how to mutate the demand pattern in addition to increasing the total network demand. This entails an added dimension of uncertainty, and it casts doubt on the validity and robustness of any network solution obtained by a paradigm that depends critically on the precise details of the point-to-point demand pattern.

This paper will present a relatively simple solution to the Equipment Placement Problem involving the use of nodal, rather than point-to-point, demands. All other solution procedures this author knows about that address this problem require point-to-point demand data as input, even though such data is usually not available from network operators, while measured and forecasted nodal demands *are* generally available.

2.2.3. Converting nodal demands to point-to-point. Although (as remarked above) it is trivial to convert point-to-point demands to nodal demands, the converse is not at all trivial. In fact, there is generally no *unique* pattern of point-to-point demands consistent with a given set of nodal demands. This is illustrated in figure 2, where four different point-to-point demand patterns are presented, each consistent with the same set of nodal demands. It is possible to narrow the search down to a single point-to-point demand pattern by considering additional problem-specific data (when available) and placing certain restrictions on the properties of the point-to-point pattern (Wasem, Gross, and Tlapa, 1995). However, a residue of uncertainty always surrounds point-to-point demands estimated from nodal demands. The approach described here avoids the uncertainties and complexities of point-to-point demands by simply not requiring them for generating a solution.

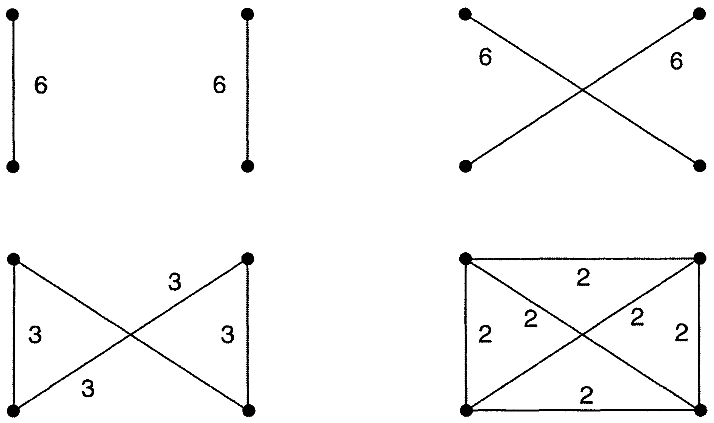


Figure 2. These four point-to-point demand scenarios are all consistent with the same nodal demand scenario: a total of six service units terminating on each node.

2.3. *Relation to other work*

The work described in this paper falls into the general category of algorithmic network design, for which a substantial body of literature exists. Some examples are Gavish and Neuman (1989), Gavish et al. (1989), Mahey et al. (1998), and Gavish and Altinkemer (1986). The present work grew out of the author's difficulty in trying to apply standard techniques, with their rigorous requirements for input data, to real network design consulting projects, where data scarcity is the rule.

Gavish and Neuman (1989), Gavish et al. (1989), and Mahey et al. (1998), for example, require what we call "point-to-point" demands as input, although this author has not yet seen a case where a client could provide a reliable point-to-point demand data set for a proposed, leading-edge, broadband network. Instead, our approach uses what we call "nodal" demands, which clients are able to provide (see the discussion in previous sections). Gavish and Altinkemer (1986), which deals with the topic of local access network design, requires as input the expected traffic between each location and the network "center." This seems to be analogous to what we call "nodal" demands. The technique in Gavish and Altinkemer (1986) then derives (in essence) point-to-point demands from the specified nodal demands and other conditions. The goal of the technique in Gavish and Altinkemer (1986) is the design of a transmission network, while the goal of the present work is equipment placement.

Finally, we are not dealing here with the difficult, unsolved research issues surrounding questions such as how to gauge the amount of bandwidth required on a link to achieve a certain level of Quality of Service (QoS) when a collection of traffic streams from different sources transit that link. (However, our method does support a higher degree of user-selectable bandwidth over-subscription on backbone links versus access links, thus accommodating higher multiplexing gains on backbone links.) Since these QoS issues are, as mentioned, difficult and unsolved, what we have seen in practice is that Engineering

Guidelines are established by network operators to deal with these issues. These Guidelines typically set additive bandwidth allocation requirements for various services that may be different on low-capacity access links versus high-capacity backbone links. Although settling on Guidelines that give the desired network performance is usually an ongoing process of trial and error, no individual network designer working for one of these network operators should fear being taken to task over the eventual network performance if he or she followed the established Guidelines. Our design method described here is meant to fit naturally into this practical, realistic design paradigm.

3. Heuristic algorithms

3.1. General problem description

Briefly, the Equipment Placement problem requires the following inputs:

- A set of network nodes where some or all of the nodes are specified as candidate locations for placing the switches or routers.
- The total demand terminating at each node, which for purposes of the exposition here will contain only unprotected demands.
- Adequate physical information about the nodes and how they are located or interconnected, plus adequate information about transmission systems or tariff charging algorithms, to enable calculation of the cost of transmitting data between any pair of nodes. Among other things, this data is used internally by the software to create a so-called *homing matrix*, where matrix element C_{ij} represents the cost of homing the demand of node i to candidate location j.
- A desired style of backbone network. The backbone network is used for interconnecting the switches (or routers).

The output of this analysis is a network design solution derived with the goal of minimizing Installed First Cost. This solution includes the following elements:

- The switch (or router) placements.
- A homing selection, that is, an indication of which non-switch nodes home to which switch sites.
- A backbone network.
- Solution statistics such as total cost, equipment usage, and cost breakdown.

3.2. Solution calculation

Up to four separate solutions can be calculated. The user can choose which solution the software should calculate and report. Alternatively, the user can choose to have the software calculate all four and report the best (lowest-cost) of the four.

3.2.2. The four basic solutions. The four possible solutions are called W, X, Y, and Z and are described presently.

3.2.2.1. Solution W. The first solution, denoted W, is obtained by simply selecting the lowest-transport-cost homing for each node regardless of other economic factors such as the high cost of excess switching equipment or excess node conversion costs resulting from a large number of switch sites. Generally, this means that the demand at any given node is homed to the candidate switch site closest (in terms of transport distance) to that node. After the lowest-transport-cost homing is selected, which also defines the subset of candidate sites where switches are to be placed,^{4} a backbone network connecting the switch sites is built automatically according to the style indicated by the user. Solution W is then costed and either reported immediately or stored, depending on whether other solutions (see below) will be calculated.

This lowest-transport-cost homing solution can be useful in at least two ways. First, it can serve as a point of comparison. Since manual solutions typically use lowest-transport-cost homing or something close to it, comparing Solution W's cost to that of X, Y, or Z (described below) quantifies the cost savings possible with these optimization techniques. Second, we have seen cases where network designers actually wish to implement the lowest-transport-cost homing solution as their network architecture; they are simply not interested in other homing paradigms for a variety of reasons that may include regulatory restrictions.

3.2.2.2. Solution X. The second solution, denoted X, is obtained by further refining solution W with the *Fast Homing Heuristic* (Section 3.2.3).

3.2.2.3. Solution Y. The third solution, denoted Y, is obtained by further refining solution X with the *Greedy Rehoming Algorithm* (Section 3.2.4).

3.2.2.4. Solution Z. The final solution, denoted Z, is obtained by further refining solution W with the Greedy Rehoming Algorithm.

3.2.3. Fast Homing Heuristic. The Fast Homing Heuristic (FHH) is run on an existing solution (the one denoted W, above) and attempts to reduce the cost of the solution by reducing the number of "hub" sites, that is, sites that have a switch or router. The FHH comprises the following steps:

Algorithm: Fast Homing Heuristic

1. Let Q denote the number of hub sites that have client nodes homed to them with non-zero demand. Denote these hubs by $H[1, 2, \ldots, Q]$. If $Q < 2$, we are done; otherwise, proceed to step 2.
2. Sort the hubs $H[1, 2, \ldots, Q]$ from "largest" to "smallest" (see below for an explanation of "size" in this context). Mark all the hub records as "*NOT ELIMINATED*." Let $k = Q$, that is, k is the index of the smallest hub.
3. Let m, $1 \leq m \leq Q, m \neq k$, be the index of a hub marked "*NOT ELIMINATED*" that has the cheapest *homing cost* (see below) for the demand that is currently homed to hub

   $H[k]$. If there is more than one such cheapest hub, choose the one with the smallest index (that is, choose the largest one).
4. If there is a net savings by rehoming all the demand from $H[k]$ to $H[m]$ (see below for a discussion of how this net savings is calculated), then rehome the demand from $H[k]$ to $H[m]$ and mark $H[k]$ as "*ELIMINATED*."
5. Set $k = k - 1$. If $k < 2$, we are done; otherwise, return to step 3.

Step 2 above orders the switch sites according to "size." Roughly, this means sites with less total demand homed to them are ranked "smaller" (which means a larger index in the array) than sites with more total demand. When two or more sites happen to have the same total demand homed to them, various tie breakers are used such as the number of clients (more clients is taken as a larger size when the total demands of the two client sets are the same). Other tie breakers include the hubbing cost penalty (the node conversion cost of establishing a switch or router site, over and above the cost of equipment and backbone network connections) and the excess switching capacity at the hub.

Steps 3 and 4 eliminate switch sites by rehoming their demand to larger switch sites if such a change appears to give a cost savings. The switch sites are processed sequentially, from smallest to largest. In general, the rehoming entails an increase in *homing cost*, that is, the cost of transporting demand traffic between a customer site and the switch. What is sought, however, are instances where this homing cost increase is more than offset by savings in switching equipment costs, backbone network costs, and node conversion costs. The last represents the savings in *not* having to convert a non-switch node into a switch node (converting a facility to house a substantial new hardware item such as a large switch often entails facility upgrading costs to pay for such things as additional air conditioning, additional electric power mains, etc.). In this algorithm, rehoming costs, node conversion costs, and the cost estimates stored in the backbone cost estimator table are accounted for exactly in making the decision on whether or not to rehome. Switch and switch plugin cost savings, however, are accounted for with only a very simple formula during the solution search to ensure that this algorithm runs quickly.

The simple switch and switch plugin cost savings formula says (essentially) that if the total amount of switching capacity currently in the solution is far in excess of the minimum needed to handle the total global demand (the latter is easily calculated), then essentially all the cost of the switching equipment at a hub will be saved when the hub is eliminated. This follows because there is probably ample excess capacity at other hubs to absorb the demand of the eliminated hub. On the other hand, if the total switching capacity is close to the minimum required, then eliminating a hub will probably result in no net savings in switching equipment because essentially the same amount of equipment will have to be added to other hubs to make up for the capacity lost at the eliminated hub. Between these two extremes, the formula interpolates the savings. The factor used to interpolate the potential switch hardware savings is simply the ratio (switching capacity − minimum switching capacity)/(switching capacity).

3.2.4. Greedy rehoming algorithm.

3.2.4.1. Overview. Similar to the FHH solution search, the Greedy Rehoming Algorithm (GRA) first attempts to eliminate switch sites by rehoming their entire demand to "larger" switch sites. However, the GRA accounts for switch hardware savings with much greater accuracy than done in the FHH. After finishing with the hub elimination phase, the GRA runs one or two types of rehoming strategies in which it tries to reduce cost by rehoming demand in the network. This amounts to packing demand more tightly into the network's switches with the goal of reducing cost by eliminating one or more switches. These rehoming phases are based on subset manipulation and pruning techniques to be described presently, and have highly variable run time.

3.2.4.2. GRA fractional elimination. To understand the fractional elimination phase of the GRA, it should be noted that the GRA (and FHH) assigns a scalar floating-point value to the bandwidth of each service. These abstract bandwidths are normalized to an effective, abstract switch capacity that is derived from the plugin and port configuration of the switch; hence, the sum of the abstract bandwidths of all the service demands at a node is a floating-point number that is equal to the total number of whole plus fractional switches needed to terminate the demand at that node. The least upper bound of the sum of the demands of all the nodes that home to a hub is the total number of switches needed at that hub. The fractional portion of this sum represents a partially utilized switch, that is, a switch that has excess (unused) capacity. If the fractional portion of the demand at one hub can be re-homed to another hub that has enough excess capacity to absorb that demand, then there might be a net savings from this re-homing since one switch would be eliminated from the solution.

This brings up the concept of *weighted subsets*. Say that the total demand at a hub equaled 7.6, that is, the hub has eight switches, and the last of which is only 60% loaded. To get rid of that 60% loaded switch, we would be interested in rehoming subsets of the hub's client nodes whose demands sum up to a floating-point number with fractional part 0.6. But that requirement is probably too stringent; instead, we require the fractional part of the demand sum to be between 0.6 and 0.7. That would guarantee that the last switch at the hub is at least 90% utilized. For example, if the rehomed subset had a demand weight of 2.7, then the remaining demand at the hub would be 4.9: four fully-utilized switches, plus one that is 90% utilized. If the demand sum for a different rehomed subset was 2.6, then the remaining hub demand would be 5.0: five fully-utilized switches.

Continuing with the example, suppose that the candidate rehome subset with total weight 2.6 was itself a subset of the candidate set with weight 2.7. If the GRA determines that there is no economic advantage to rehoming the subset of weight 2.6, then that likely means no economic advantage for the 2.7-weight set, either. That follows because the 2.7-weight set would probably cost more to rehome, being a superset of the 2.6-weight set, while the 2.7-weight set would have no beneficial effects on reducing switch hardware beyond that of the 2.6-weight set. Likewise, if rehoming the 2.6-weight set was economically beneficial, it is unlikely that the 2.7-weight set would offer additional economic benefits. Thus, there is no reason for the GRA to spend time evaluating the 2.7-weight set if it is already evaluating a subset of this set.

To speed the execution of the GRA fractional elimination, we have implemented an efficient technique for synthesizing the required collection of weighted subsets of hub client nodes. Moreover, this code also executes efficient pruning of the weighted subset collection, as exemplified in the previous paragraph. In set theoretic terms, all subsets that are proper supersets of another subset of the collection are pruned from the final collection.

Our experience indicates that this pruning can be expected to give substantially more than a 50% reduction in the number of weighted subsets, resulting in a corresponding decrease in the GRA fractional elimination execution time.

To understand the weighted subset concept better, consider a hub that has nine client nodes with the following scalar demands: (2.1, 1.6, 1.1, 1.05, 0.65, 0.5, 0.4 0.15, 0.05). The sum of these demands is 7.6. We wish to see whether the final (60%-loaded) switch at this hub can be profitably eliminated by rehoming some of its clients to other nodes that have available spare switching capacity. For this we might wish to consider the pruned collection of weighted subsets where the fractional portion of each set weight lies in the range between 0.6 and 0.7. Another way to state this weight restriction is that the set weights must lie in one of the seven intervals $[k.6, k.7]$, where k is an integer between 0 and 6, inclusive. But since the GRA hub elimination phase run previously failed to eliminate this hub, it seems unlikely that values of k in the upper end of this range will result in profitable rehoming. We adopt the following heuristic to save computation time while still concentrating on searching the cases most likely to produce a better solution: we look only at the weight interval [0.6, 0.7], but we base the subset weight calculations on the *fractional* client demands. In this example, the weight vector for the entire set is transformed to (0.1, 0.6, 0.1, 0.05, 0.65, 0.5, 0.4 0.15, 0.05). The four client nodes whose demand exceeded the capacity of one full switch are now counted only for their fractional demand in calculating set weights, so it is possible for them to be included in a rehoming set despite the fact that their full demands exceed the upper limit of the weight interval.[5] The desired collection of subsets based on this transformed weight vector is shown in Table 1. Note that there are 13 members of the pruned subset collection compared to 37 subsets total with weights falling in the range [0.6, 0.7]. These 37 subsets, and their inclusion relationships, are indicated in figure 3. When membership in a set is indicated in tabular form as in Table 1, then the

Table 1. Pruned subset collection for weight interval [0.6, 0.7].

	0.10	0.60	0.10	0.05	0.65	0.50	0.40	0.15	0.05	Weight/ID
1					X					0.65/010
2		X								0.60/002
3						X		X		0.65/0a0
4			X			X				0.60/024
5			X				X	X		0.65/0c4
6	X					X				0.60/021
7	X						X	X		0.65/0c1
8	X		X				X			0.60/045
9				X		X			X	0.60/128
10							X	X	X	0.60/1c0
11			X	X			X		X	0.60/14c
12	X			X			X		X	0.60/149
13				X			X	X		0.60/0c8

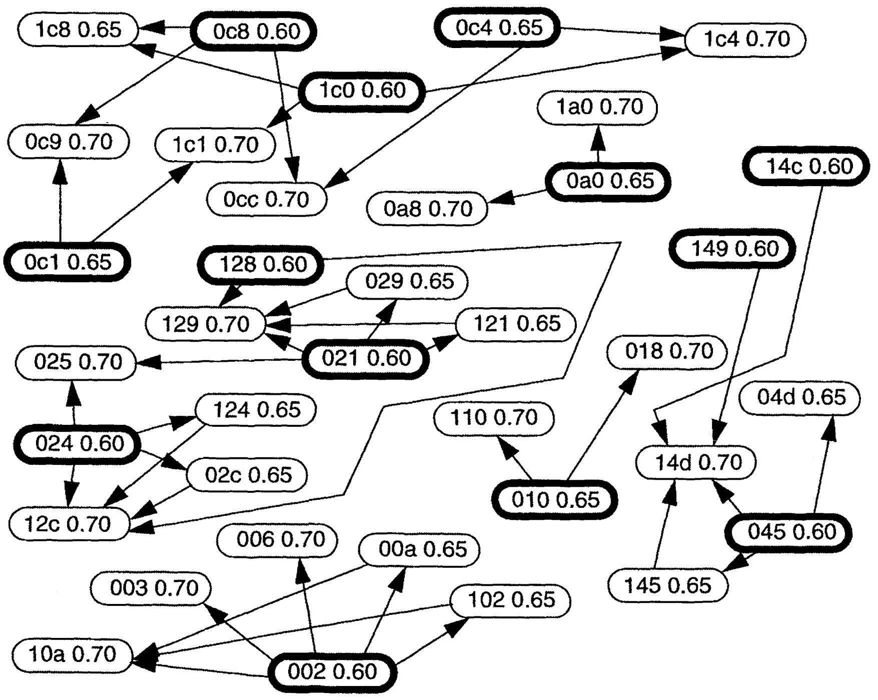


Figure 3. Weighted subset inclusion relationship for the example (0.1, 0.6, 0.1, 0.05, 0.65, 0.5, 0.4 0.15, 0.05) with weight interval [0.6, 0.7]. The arrow indicates inclusion, with the subset at the tail of the arrow *included in* the subset at the head of the arrow. The *pruned collection* of weighted subsets consists of those subsets with no incoming arrows, which are indicated here with heavy outline. Subsets are identified by a three-digit hex number; see text for explanation. The subset weight is also shown.

pattern of X's in each row defines a binary number[6] that uniquely identifies the associated subset. We use this binary number in the present example, expressed in hex notation, as an identification tag for the subsets.

This example is small and the needed subsets can be quickly generated in software by exhaustive enumeration. Such is not possible when a single hub has hundreds or even just dozens of client nodes. Our software tool addresses this issue with an efficient technique for synthesizing pruned collections of weighted subsets without exhaustive enumeration. A high-level description of this algorithm follows. We begin by defining a procedure.

Procedure: `findWeightedSubset`

Input: N client nodes in an indexed array denoted $C[1, 2, \ldots, N]$. Each client $C[i]$ has a nonnegative demand ("weight") associated with it, denoted $C[i].w$. We assume the array $C[1, 2, \ldots, N]$ is sorted from largest to smallest weight. Also, a lower limit L and an upper limit U are specified, $0 \leq L \leq U$. The clients $C[i]$ are marked with a binary flag indicating "INCLUDED" or "NOT INCLUDED." It is required initially that if any of the $C[i]$ are marked "INCLUDED," then the sum of the weights of clients so marked must be strictly less than L. There must exist a client index k between 1 and N inclusive, which is specified as an input, such that every $C[i]$ for $i \geq k$ is marked "NOT INCLUDED."

Output: After the procedure finishes, one or more $C[i]$ for $i \geq k$ may be marked "INCLUDED" such that the sum of the weights of all the "INCLUDED" $C[i]$ is less than or equal to U.

Action: The clients marked "INCLUDED" form a subset, and this procedure may increase the number of elements in that subset by marking additional clients as "INCLUDED." The goal is to produce a member of the pruned collection of weighted subsets described in the text above. Failing that, however, a subset may be produced that will be useful in constructing a subsequent call to `findWeightedSubset` to further search for the desired subsets.

This procedure is composed of the following steps:

1. Set S equal to the sum of the weights of all clients marked as "INCLUDED" (note that $S = 0$ if all clients are marked "NOT INCLUDED").
2. If $S + C[k].w \leq U$, mark $C[k]$ as "INCLUDED."
3. If $C[k]$ is marked as "INCLUDED" and $L \leq S + C[k].w$, the procedure terminates.
4. Set $k = k + 1$. If $k \leq N$, go to step 1; otherwise, the procedure terminates.

Now we are ready to describe the algorithm that finds the complete pruned collection of weighted subsets.

Theorem. *The following Algorithm finds the pruned collection of weighted subsets.*

Algorithm: Finding the Pruned Collection of Weighted Subsets

1. Given N client nodes in an indexed array, denoted $C[1, 2, \ldots, N]$, sorted in descending order of weight. Given also a specified lower limit L and an upper limit U, $0 \leq L \leq U$. All the clients $C[i]$ are marked "NOT INCLUDED" initially. As for the input k of the `findWeightedSubset` procedure, set it to $k = 1$. Now execute the `findWeight-edSubset` procedure.
2. If none of the clients is now marked "INCLUDED," we conclude that the pruned collection of weighted subsets is empty, and the algorithm terminates.
3. Otherwise, the clients marked "INCLUDED" define a subset of clients. Store this subset in the first position of an indefinite-length array of subsets, denoted $M[1, 2, \ldots]$. If the sum of the weights of the elements of this subset lies in the interval $[L, U]$, mark this subset as a member of the pruned collection.
4. Now we use the subsets of $M[1, 2, \ldots]$ as seeds to search further for members of the pruned collection of subsets. We process the members of $M[1, 2, \ldots]$ sequentially, while at the same time allowing $M[1, 2, \ldots]$ to grow by adding more subsets at the end of the $M[1, 2, \ldots]$ array. When we finish, $M[1, 2, \ldots]$ will contain the desired pruned collection of weighted subsets (each one marked with a flag indicating such), possibly

along with other subsets that were generated in the search process. Specifically, in pseudo-C code,

```
• j = 1
• Let $C^*[1, 2, \ldots, N]$ denote separate buffer memory that initially contains a copy of $C[1, 2, \ldots, N]$.
• while (M[j] is not the end sentinel of the array)
    • Set the "INCLUDED/NOT INCLUDED" flags of the elements of the $C^*$ array to reflect the subset encoded in the M[j] record.
    • while ($C^*[1, 2, \ldots, N]$ has at least one client marked "INCLUDED")
        • Set $k$ to equal the index of the client in $C^*$ with the largest index of all those that are marked "INCLUDED." Change the marking of $C^*[k]$ to "NOT INCLUDED." Now set $k = k + 1$.
        • If $k > N$, continue to the next iteration of this while loop.
        • Otherwise, copy $C^*$ to $C$.
        • Execute findWeightedSubset using client input/output array $C$ and inputs $k$, $L$ and $U$.
        • The clients marked "INCLUDED" in $C$ define a subset of clients. Store this subset at the end of $M[1, 2, \ldots]$ if this subset is not already in $M$. If the sum of the weights of the elements of this subset lies in the interval $[L, U]$, mark this subset as a member of the pruned collection.
        • End of inner while loop.
    • j = j + 1
    • End of outer while loop
```

Proof: The subset building procedure embodied in the preceding Algorithm forces the weight sum to be $\leq U$ by construction and guarantees that any constructed subset whose weight sum lies in the interval $[L, U]$ is minimal and is therefore a member of the pruned subset collection. This follows because the subsets are built by adding clients in descending order of weight, with the building process terminating when a client is added that brings the weight sum over the threshold to $\geq L$. Hence, any non-empty proper subset of such a subset would have a weight sum that lies strictly below the L threshold.

Before proceeding with the proof, note that each call to the findWeightedSubset procedure satisfies the requirement that the sum of the elements marked "INCLUDED" when the call is made is strictly less than L. For the first call, this condition is satisfied trivially because there are no elements marked "INCLUDED." Subsets are used as seeds for subsequent findWeightedSubset calls after removing their smallest-weight elements, which guarantees that the weight of the subset is strictly less than L for the same reason as described in the preceding paragraph.

Proving that this algorithm finds all the minimal subsets comprising the pruned subset collection is more involved. We will sketch the proof here. Assume the pruned subset collection is non-empty, and let Q denote a subset in that collection. Assume the elements of Q and any other subset we refer to in this discussion are listed in index order, where the indices are those from the client array $C[1, 2, \ldots, N]$ (which is sorted in descending weight order, as described in the Algorithm). The fact that Q exists at all means that there exist clients whose weight is $\leq U$; indeed, the first (highest-weight) element of Q is a client with weight $\leq U$ since the total weight of Q is $\leq U$ (by hypothesis) and all weights are non-negative. This means that the first call of `findWeightedSubset` will find a non-empty subset P that will be placed into the subset list M, and the first element of P will have an index less than or equal to the index of the first element of Q.

Suppose the index h of the first element of P is strictly less than that of the first element of Q. Then after the first iteration of the outer `while` loop of the algorithm, M contains a new subset whose first element has index $h + 1$ (this was placed in M by the last iteration of the inner `while` loop). Likewise, after some later iteration of the outer `while` loop, M will contain a subset whose first element will have index $h + 2$, etc. Eventually, M will contain a subset whose first element will have the same index as the first element of Q. Let Q^* denote the first such subset to be placed into the list M.

Note that the sum of the first two elements of Q is $\leq U$, since the sum of all elements in Q is $\leq U$, by hypothesis. Since the element weights decrease monotonically with index, it follows from the way the subsets are constructed by the `findWeightedSubset` procedure that the second element of Q^* will have an index less than or equal to that of the second element of Q. By the same argument as was used regarding the first element, it follows that eventually a subset Q^{**} will be placed in M whose first two elements are the same as those of Q. Continuing in this way, it follows that eventually the subset Q itself will be placed in M. □

3.2.4.3. GRA excess placed elimination. The hub elimination phase of the GRA cannot be executed on hub sites where the user has specified a non-zero count of existing or planned switches (by definition, such switches must exist in the final solution—they cannot be eliminated). In lieu of total hub elimination, there might be an advantage to eliminating all excess placed switches at such a hub, that is, all switches placed there by an early pass of the algorithm over and above the existing or planned switches.

Continuing the example from Section 3.2.4.2, suppose the user specified five planned switches for the hub of that example. Since the total hub demand is 7.6, we would be interested in rehoming weighted subsets of client nodes whose weight lies in the range from 2.6 and 2.7. Unlike the Fractional Elimination phase, here we use the full weights of the set elements (clients) to find the weighted subsets, since here the full value of the rehomed demand is as important as its fractional part. The resulting pruned collection of 11 subsets is shown in Table 2.

Although the collection of weighted subsets, which are candidates for rehoming, are different than in the Fractional Elimination case, the procedure for deciding which demands get rehomed to which hubs is exactly the same as in Fractional Elimination. As suggested in Section 3.2.4.1 and as will be described further in the next section, demand is rehomed in a heuristic "smallest-to-largest" order, with a greedy cost decision made at each step regarding whether to use the rehoming of that step or keep the existing homing.

3.2.4.4. GRA calling strategy. The GRA is run twice, as this appears to give results that are generally no worse than a single-pass procedure and occasionally slightly better. In the first pass of the GRA, only "small" hubs are candidates for having their demands rehomed, and the rehoming procedure tries only to rehome each "small" hub's demand to "large" hubs,

Table 2. Pruned subset collection for weight interval [2.6, 2.7].

	2.10	1.60	1.10	1.05	0.65	0.50	0.40	0.15	0.05	Weight
1	X					X				2.60
2	X						X	X		2.65
3		X	X							2.70
4		X		X						2.65
5			X	X		X				2.65
6		X			X		X			2.65
7			X	X			X	X		2.70
8			X		X	X	X			2.65
9				X	X	X	X			2.60
10		X					X		X	2.65
11			X	X			X		X	2.60

thus ensuring that "small" hubs remain "small" as the first pass proceeds. Without this restriction, a "small" hub might receive another "small" hub's demand during rehoming; once a "small" hub is allowed to grow in this manner, it becomes more difficult to eliminate in a later stage of the algorithm's execution. The attempted rehoming proceeds from the very smallest of the "small" hubs up to the largest of the "small" hubs.

In the second pass of the GRA, all hubs are candidates for rehoming and the demand for any given hub can be rehomed to any other hub. Still, however, the procedure is biased toward rehoming the demand of the smallest hubs first and, for any given hub, rehoming its demand to the largest hub available that gives an economic advantage upon rehoming.

3.3. Other issues and refinements

What was described in Section 3.2 were the basic problems addressed by our network design tool along with the basic issues, solution techniques, and algorithms associated with this tool. There are many other issues raised by our approach, and many subtleties and refinements of network design that this tool also addresses. Some of these are mentioned briefly in this Section. A longer version of this paper that elaborates on the points of this Section can be obtained by contacting the author.

3.3.1. Homing paradigm. The homing paradigm used in this approach is all-or-none, meaning that all the unprotected demand at a non-switch node is homed to a single switch site (figure 4). This author's experience indicates that this is generally the kind of homing that network operators want to see in their broadband network architectures. Moreover, all-or-none homing has an economy-of-scale advantage: "fat pipe" links generally have the lowest cost per megabit of capacity, so a single link carrying all the demand of a non-switch node to its target switch site will generally cost less than multiple links carrying portions of the demand to several distinct switch sites. And all-or-none homing has yet

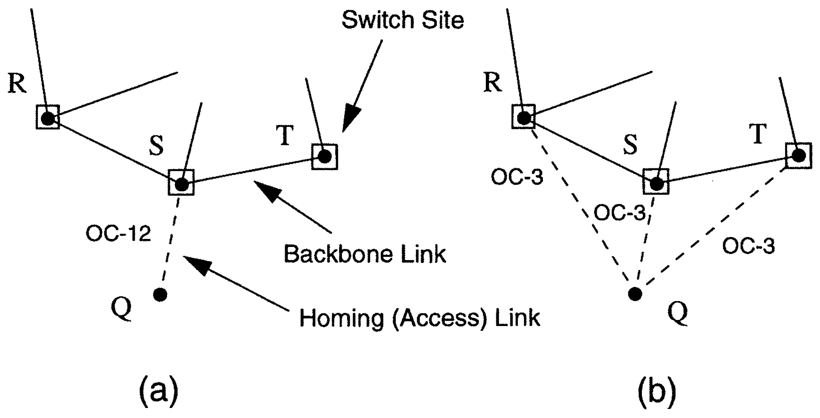


Figure 4. (a) All-or-none homing from node Q. (b) Multiple homing from node Q. In general, the single OC-12 system of (a) will be cheaper than the three OC-3 systems of (b).

more economy-of-scale advantages: it guarantees that all the traffic between any node Q to any remote node P not homing off the same switch site will be carried over the backbone network rather than carried through a secondary multiple-homing link. The "fat pipes" of the backbone network are typically even "fatter" than the homing links, thus offering an even more significant economy-of-scale advantage; so it makes sense to use the backbone links rather than trying to bypass them. And the multiplexing gain offered by asynchronous technologies such as ATM results in still further economy-of-scale advantages for sending traffic over the backbone. This author's experiments using network data from real design projects uncovered no evidence supporting the notion that an optimization algorithm that considers multiple homing can find solutions costing less than our algorithm based on all-or-none homing. It may be possible to build artificial examples where some cost advantage could be obtained with multiple homing, but this author has not seen such examples in real networks.

There is a case, however, when multiple homings *are* generated by our solution technique. That is when demands are specified as requiring *survivability protection.* This issue is discussed at length in the longer draft version of this paper, which is available from the author. Section 4 of this paper shows an example designed with 100% survivability protection (figure 6).

The issues surrounding how to "cover" nodal demands in an economical way with a selection of potential transmission facilities or tariffed services for the purpose of homing those demands to their selected switch are discussed in the longer draft version of this paper, which is available from the author.

3.3.2. Backbone network. The *backbone network* is the network of transmission links that interconnect switch sites (figure 4). The backbone transmission paths may be tariffed services purchased from another service provider with cost dependent on the distance between the switch sites. Or these transmission paths may be transmission systems built and operated by the switching service provider with cost proportional to the length of a complex path through fiber facilities that link the switch sites. The capacity of these backbone links is specified by the user, e.g., one OC-12. All backbone links are assumed to have the same capacity, which sounds restrictive but has in fact matched well with real design projects this author has worked on. The user also specifies the backbone network's style (topology), such as ring, semi-mesh,[7] full mesh, etc. The user may specify some of the links and allow the tool to build the rest, or the user may specify no links and allow the tool to build the entire backbone network. It is unlikely that the user would choose to specify all the backbone links, since that would imply that all the switch sites are already chosen; hence, there would be little left for our tool to design.

Our tool builds a backbone cost estimator table based on the specified backbone network style, statistical properties of the specified topology data, and other data supplied by the user. This table shows the estimated incremental decrease in backbone network cost as switching hubs are eliminated, and it is used during the solution search to account (approximately) for the effect of the backbone network cost on the solution. A more detailed description of the construction of the backbone cost estimator table is given in the longer version of this paper, which is available from the author.

After the switch placement is complete, an actual set of links comprising a backbone network in the user's chosen style is built by the software using straightforward algorithms. For example, if the user chose a ring architecture, then a random node is chosen for the beginning of the ring and each successive link is drawn to the nearest neighbor that is not yet on the ring. This continues up until the last link, which must terminate at the initial node of the ring. Currently, no special techniques are used to try to minimize the length of this final link, and the ring can grow arbitrarily large. The purpose of this backbone network generation is to illustrate for users a possible backbone network in their chosen style. It is expected that users will enter backbone links manually after the switch sites are determined, or use some other backbone network design tool, if backbone network design is a major issue they wish to address (the main purpose of the current tool is switch or router placement, not backbone network design). Given the general data scarcity that has been mentioned in this paper and the need to match this scarcity with appropriate broad-brush solution techniques, our approach to handling backbone networks seems a reasonable match to the problem being solved.

The user can investigate whether the backbone links might become congested under various scenarios using the *Link Advisor*, described next.

3.3.3. Link Advisor. For users that can obtain point-to-point demand data, or who would like to investigate potential capacity bottlenecks in their design under various hypothetical point-to-point demand scenarios, the *Link Advisor* capability is provided. This module routes point-to-point demands using a capacity-sensitive variant of Dijkstra's shortest path algorithm. Based on this routing, potentially congested links can be identified.

Depending on the problem parameters, the results of running such an analysis can be highly dependent on the order in which the point-to-point demands are routed. Although this order dependency may cast doubt on the usefulness of the Link Advisor's analysis, it can also be viewed as reflecting the fact that we usually do not know *a priori* in what order the requests for ATM PVCs (for example) will be received. If desired, the user can change the order that the point-to-point demands are routed to observe how ordering affects network congestion. Routing the demands in bandwidth order, from largest to smallest, is a simple and reasonable heuristic for using link capacity efficiently.

3.3.4. Switch interconnection at a hub site. The solution approach described in this paper can and often does place multiple instances of a switch model at a node (hub site), which raises the issue of how multiple switches at the same hub site intercommunicate to handle the traffic flowing among them.

The answer to this question depends on equipment details. In one particular design project addressed with this tool, the individual ATM switches placed at a hub were in fact units that fit into a common equipment bay. These switches, when placed in the same bay, could intercommunicate through a high-speed backplane. Here there was no inter-switch communication issue as long as the number of switches placed at the node was 16 or less, which was the number that could be accommodated by one equipment bay. Fortunately, the number of switches placed at each node was below the 16 threshold. Had one or more hub sites been above this threshold, it would have been necessary to perform some ad hoc re-modeling of the problem (see next paragraph) and then re-run the solution procedure.

In the more typical scenario, some fraction of switch ports have to be reserved for communication among co-located switches. This fraction is decided by the tool user according to his or her judgement, and may be reflected in the switch model by reducing the effective capacity of the switch to terminate user demands. The degree of interconnectivity among co-located switches can be greater than that among remotely-located switches because of the community-of-interest often implied by geography. Currently our switch placement algorithms have no way to account for this additional interconnectivity. Accounting for switch interconnection at a hub using ad hoc modeling, while certainly an approximation, seems reasonable given the large uncertainty in the demand data that is typical in the design projects we have seen and the relatively broad-brush nature of these analyses.

3.3.5. Still more issues. The software provides support for handling network hierarchy and the placement of multiple switch models. It also allows switches or routers to be modeled in two different ways. One way is as a "black box" with an associated scalar switching capacity, which is also the model used internally by the placement algorithms. The other way is as configurable hardware with a structure of shelves, slots, and plugin modules. The latter is converted internally by the software into an associated "black box" model for use by the placement algorithms. See the longer draft version of this paper, available from the author, for more details.

4. Implementation experience

4.1. The code

A prototype network design tool implementing the philosophy and functionality described in this paper has been developed: the *Strategic Network Layout Planner* (SNLP). The Solution Engine (algorithmic code) for SNLP consists of 21.6K lines of C and implements the functionality described in Section 3. An intuitive Graphical User Interface (GUI) has also been developed for SNLP. The GUI and Solution Engine communicate through 29 disk files, 22 of which are inputs to the Solution Engine and 7 of which are outputs.

4.2. *Performance comments*

SNLP is run mainly on a Sun SPARCstation 5 with 32 Mb of main memory.

Solution X (Section 3.2.2.2), which is generated with the Fast Homing Heuristic (FHH; Section 3.2.3), consistently executes in a few seconds or less for networks between 100 and 200 nodes. It is of no particular consequence how many nodes are specified as candidate switch locations. In our studies, FHH run time proved largely insensitive to the number of candidate switch locations; thus, all nodes were chosen as candidates in most runs.

When the Greedy Rehoming Algorithm (GRA) is used (Solutions Y and Z; Section 3.2.2.3 and Section 3.2.2.4), the total execution time often stays in the range of a few seconds or less for networks of size between 100 and 200 nodes. However, the GRA can sometimes run for minutes or longer on networks in this size range. The execution time behavior of the GRA seems extremely complex, with order-of-magnitude execution time changes resulting from seemingly minor changes in the problem parameters. The GRA has no particular difficulty handling LATA-sized networks where all nodes are specified as candidate switch locations, and has frequently been used to perform such analyses. In fact, this author has occasionally noted seemingly retrograde execution time behavior of the GRA: its execution time has *decreased* dramatically when the number of candidate locations was *increased.*

When the output of the FHH is post-processed by the GRA, as is done for Solution Y (Section 3.2.2.3), it is rare to see cost improvements of more than 3%. And often the GRA can find no improvement at all for the FHH solution. This suggests that the FHH, though it employs a very simple view of switch hardware cost savings, captures the essence of GRA's solution search.

The largest network ever processed by the SNLP Solution Engine consisted of 1681 nodes, where every node was a candidate switch location. This was run on a Sun workstation equipped with 512 Mb of main memory. The Solution Engine's Unix process had a size of 99.8 Mb and took 25 minutes to execute, including processing by the GRA.

4.3. *An example*

Here we demonstrate SNLP on a hypothetical but realistic network design problem involving the deployment of small ATM switches within a LATA that contains 130 COs. Our hypothetical LATA does not represent a real LATA, but it has statistical and geographical properties that make it similar to a real LATA that contains a major urban center.

To save space, we omit most of the details of this network design example. However, we note that the ATM switch model used in these analyses has a base equipment cost, including OC-12 modules for connection to the backbone network, of \$150,000. This switch can accommodate eight plugin modules (\$5500 apiece), each of which can terminate 12 DS3s. Thus a fully-outfitted ATM switch can terminate a total of 96 DS3s. About half the COs in this example have a demand of two DS3s, while the other half have a demand of one DS3. This example uses tariffed (leased line) data transmission with OC-12 backbone links. The backbone network was specified by the user to have the so-called *semi-mesh* style, with the user specifying a minimum degree of three and a maximum degree of four.

This example is modeled on a real design project in which the 3–4 backbone semi-mesh and the OC-12 capacity for all backbone links was specified by the customer. The customer had no interest in exploring whether money could be saved by using lower-capacity links at some places in the backbone. The customer wanted OC-12 because they believed that would safely handle their future, rapidly-growing traffic load for the next few years. Obviously, a different solution would result if we had precise point-to-point demand forecasts (obtained from a well-functioning crystal ball), precise knowledge of what services would win out in the market place, precise knowledge of the Quality of Service required by these services, precise knowledge of what technologies would be available two or three years hence to support QoS, and had the luxury of spending large amounts of expensive engineering time to precisely design the topology and bandwidth resources of this network under such well-defined constraints. In the real world, nothing is known with such certainty and designs have to proceed in a chaotic backdrop where bandwidth demand is exploding while network technology and services seem to evolve at an ever-faster pace.

Figure 5 shows the backbone network for the case when 100% survivability protection is specified. There are eight ATM switches spread among six switch sites. The homings for this solution are shown in figure 6. Note there are two homings for each node, which

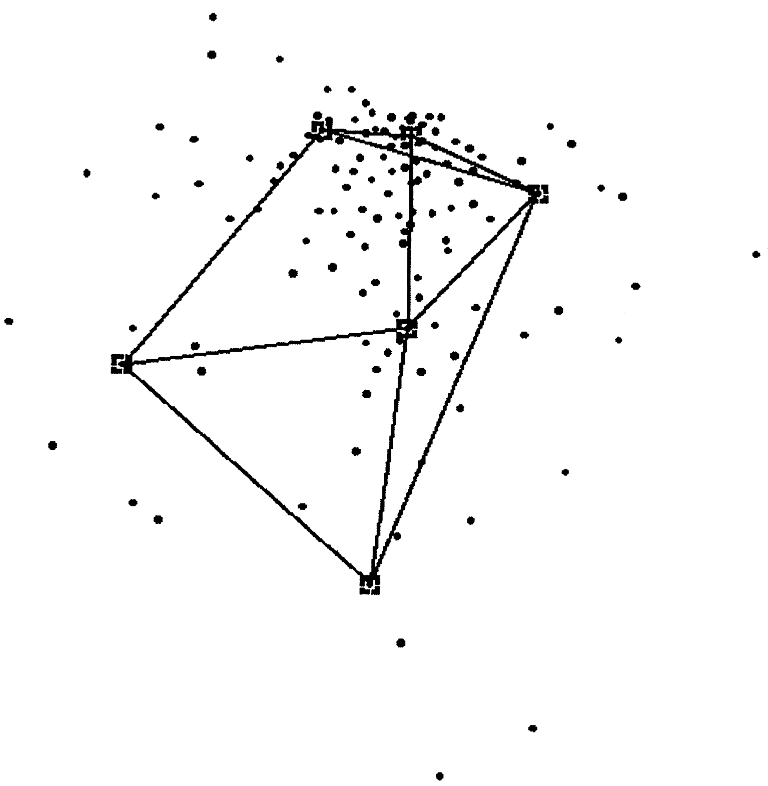


Figure 5. Switch sites and OC-12 backbone network, 100% protection.

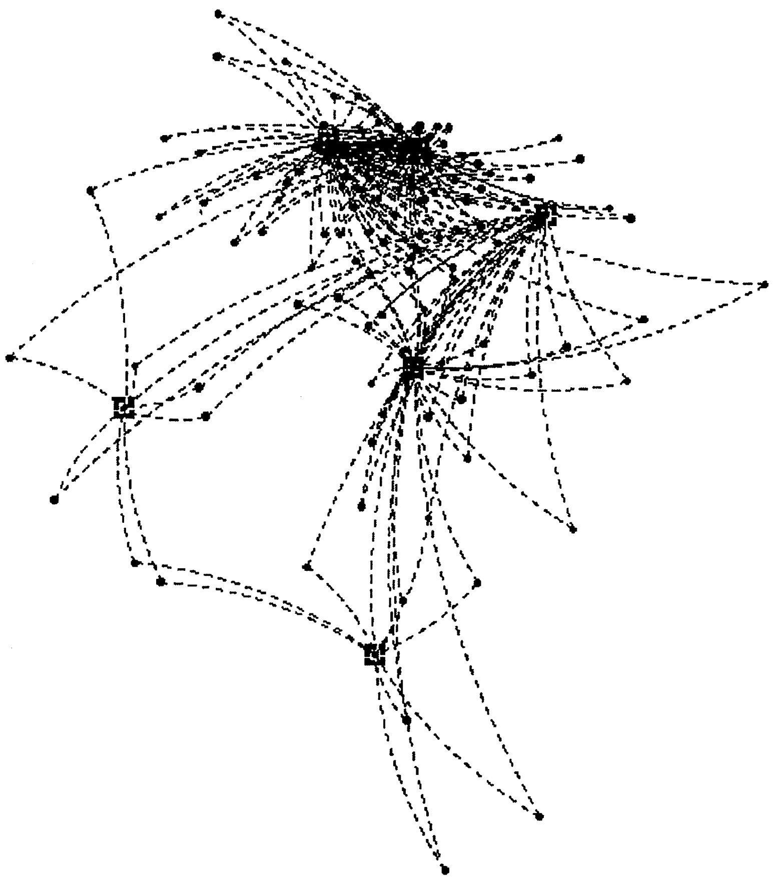


Figure 6. Switch sites and homings, 100% protection. Each homing arc represents either one or two DS3s.

is required to provide the specified node- and link-diverse protection. The total cost of this solution is $46.9 million.

Can the backbone network shown in figure 5 handle the expected traffic? To answer this question, we note that this example is patterned after an analysis we did for a proposed new class of service offering. Since this service had not yet been offered, no information was available about possible point-to-point demand patterns. In fact, it was not even known how heavily loaded the DS3 links might be. This underscores points made earlier in this paper about how network designs must proceed despite scant data. A preliminary analysis done by Telcordia for this new class of service suggested that the DS3 links might be only 10% loaded, in which case the backbone network in figure 5 would have ample capacity.

To quantify the added cost of providing node- and link-diverse survivability in this network, we re-ran SNLP on this example with survivability protection set to zero. The resulting solution is depicted in figure 7. It uses five switch sites with one ATM switch per

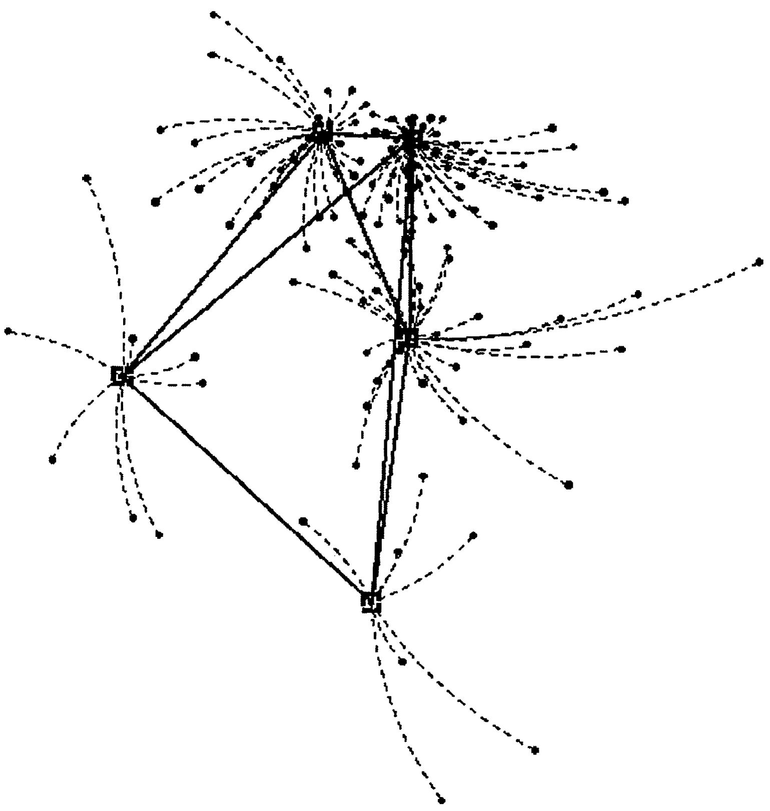


Figure 7. Switch sites, homings, and backbone network with no protection.

site. The total cost of this solution is $29.5 million. So the cost of adding survivability in this scenario is $17.4 million, or a 59% premium over the no-survivability case.

5. Summary and conclusions

A significant and unexpected limitation this author observed in real network design projects was the lack of network demand data. This author has not personally seen a single case where clients were able to provide a useful database of point-to-point demands for a proposed, or even an existing, broadband network or data service. Even the nodal demand data that clients provided was often scant and had the flavor of coarse guesses rather than refined measurements or carefully-researched predictions. In retrospect, it was probably unreasonable to expect anything more; after all, we are dealing with the design of leading-edge network technologies in a backdrop of exploding broadband demand.

Clearly, network optimization techniques whose answers are critically dependent on fine-grained demand details, such as full knowledge of point-to-point demands, are not appropriate in this scenario. The optimization technique described in this paper addresses data scarcity by providing highly-efficient analysis using only *nodal demands*. Because this approach can usually analyze LATA-sized networks in just seconds, many different demand scenarios can be investigated conveniently. Also, the high efficiency of the code allows solution runs where all nodes are candidates for switch placement. This makes the tool simpler to use because the user does not have to spend time trying to pick a small but "good" proper subset of the nodes to specify as candidates. Moreover, the possibility of a poor solution resulting from a poor selection of candidates is eliminated when all nodes can be selected as candidates. In one case, we used this software to analyze a given client network under a large number of different demand scenarios obtained by gradually scaling the base demand upward. This procedure enabled us to formulate an approximate lowest-cost evolutionary path for this network in response to future demand growth. Despite laboring under the handicap of data scarcity, the approach described in this paper has provided valuable money-saving advice to customers on where best to place broadband switching equipment.

Acknowledgments

Marilyn Facchini worked tirelessly in creating the Graphical User Interface for the SNLP prototype. The network displays in Section 4 were obtained from Marilyn's software. Yu Zhao helped debug and enhance SNLP while on internship at Bellcore (now Telcordia) from Cornell University in the fall of 1997. Numerous customers and subject matter experts—too many to mention here—provided valuable insights that helped mold this work. Thanks to Bob Doverspike, formerly of Bellcore and now of AT&T, for encouraging me to publish this work. Finally, I thank the reviewers for providing many helpful comments.

Notes

1. By "enterprise" network, we mean a complex network built, maintained, and used exclusively by a corporation or other organization.
2. That is, networks with size typical of a Local Access Transport Area (LATA), which is usually between 100 and 200 Central Office nodes.
3. For a given class of networks, the number of point-to-point demands can grow roughly as kN^2, where N is the number of nodes and k is a proportionality constant. This can be more than the maximum number of node pairs, $N(N-1)/2$, as parallel demands of different service types can occur between the same pair of nodes.
4. A switch is placed at a candidate switch site if, and only if, nonzero demand is homed to that site.
5. Note that if one of the four large-demand nodes is chosen for rehoming, then its entire demand is rehomed—not just its fractional part.
6. Each X in a row of Table 1 is taken to be a 1 in the binary representation, with the least significant bit in the left column of the table.
7. The *semi-mesh* topology is characterized by the user-specified parameters *minimum degree* and *maximum degree*, which define the minimum and maximum number of links that can impinge on any node in the network.

References

Cosares, S., D. Deutsch, I. Saniee, and O. Wasem. (1995). "SONET Toolkit: A Decision Support System for Designing Robust and Cost-Effective Fiber-Optic Networks," *Interfaces*, 25(1), 20–40.

Gavish, B. and K. Altinkemer. (1986). "Parallel Savings Heuristics for the Topological Design of Local Access Tree Networks," *Proc. IEEE INFOCOM '86*, 130–139.

Gavish, B. and I. Neuman. (1989). "A System for Routing and Capacity Assignment in Computer Communication Networks," *IEEE Trans. Communications* 37(4), 360–366.

Gavish, B., P. Trudeau, M. Dror, M. Gendreau, and L. Mason. (1989). "Fiberoptic Circuit Network Design Under Reliability Constraints," *IEEE J. Selected Areas in Communications*, 7(8), 1181–1187.

Mahey, P., A. Ouorou, L. LeBlanc, and J. Chifflet. (1998). "A New Proximal Decomposition Algorithm for Routing in Telecommunications Networks," *Networks*, 31(4), 227–238.

McQuillan, J. (1995)."Planning Networks without Network Planning," *Business Communications Review*, 25(7), 10–12.

Wasem, O.J., A.M. Gross, and G.A. Tlapa. (1995). "Forecasting Broadband Demand Between Geographic Areas," *IEEE Communications Magazine*, 33(2), 50–57.

Journal of Heuristics, 6: 65–83 (2000)
© 2000 Kluwer Academic Publishers

Virtual Path Design in Service-Specific ATM Networks

IRAJ SANIEE*
Network Design and Traffic Research Grp., Bell Communications Research, 445 South Street, Morristown, NJ 07962, USA
email: iis@bellcore.com

JOEL S. SOKOL
Operations Research Center, Massachusetts Institute of Technology, 1 Amherst Street, Room E40-130, Cambridge, MA 02139, USA
email: jsokol@mit.edu

Abstract

This paper addresses the problem of virtual path management in ATM networks, which is the problem of jointly selecting efficient virtual trunk routes and sizing them to meet end-to-end grade-of-service requirements. The problem is posed over capacitated networks and is formulated as a two-level multi-commodity network flow problem with linear side-constraints (physical layer capacity) and non-linear side constraints (end-to-end/link blocking). Through a variety of examples we show the method (i) generates solutions that agree with engineering judgement, (ii) can solve VP layout management for realistic size networks (of up to 200 nodes) in reasonable time and (iii) provides upper bounds on how far the solution strays from the mathematically optimal design.

Key Words: ATM network management, virtual paths, circuit-switching, multicommodity network optimization, add drop heuristic for combinatorial optimization problems

1. Introduction

Asynchronous Transfer Mode (ATM) is a fast packet switching standard that supports connection-oriented multi-bandwidth services via virtual circuits (VCs). Key consideration in the ATM standard is given to the concept of virtual paths (VPs). These are groups of VCs bundled together to reduce setup, switching and control costs (Addie, Burgin, and Sutherland, 1988; Kanada, Sato, and Tsuboi, 1987; Burgin and Dorman, 1991). Although VPs have not yet been extensively deployed, their use in networks that process a large number of switched VCs, or calls, is fundamental. Many of the quality of service (QoS) concepts and notions developed for ATM networks explicitly depend on the network provider's ability to integrate different services with differing service characteristics over the same physical transport trunks via VPs. One emerging view is that calls with the same QoS requirements are routed over one or more QoS-provisioned VPs which define a partition of one or more transport links with well-defined capacity (D. Mitra and I. Ziedins, 1997). When a connection request is made, the available capacity of the VP (or VPs) is checked

*Current address: Bell-Labs, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974. Email: iis@research.bell-labs.com.

and if sufficient, the connection is routed on that VP (or VPs). Thus blocking is a VP phenomenon and is computed using the VP capacity. Given their fundamental importance, many researchers have investigated the potential properties of VPs and their role in traffic management and service provisioning, see for example (Ahn et al., 1993; Chlamtac, Farago, and Zhang, 1993; Medhi, 1995).

Virtual paths can thus be significant contributors to the efficiency of large ATM networks if they are appropriately set up and dimensioned. Most of the existing work on VP management, however, deals with theoretical issues related to computability of VPs, e.g. (Farago, Chlamtac, and Zhang 1, 2), the call admission control issues related to VPs, e.g., Saito (1994), Sato and Sato (1991), ElWalid and Mitra (1993), or other special cases of the problem, e.g. Monteiro, Gerla, and Fratta (1991), Lin and Cheng (1994) do study the creation of VPs in a comprehensive fashion but solve the problem only in very specialized cases (either each VC is routed on one VP or a VP is routed over a single transport link). Some recent work (Biro et al., 1996) reports use of genetic algorithms for the combinatorial optimization problem associated with (only) the sizing of VPs, but the solution times are too large for the unrealistically small size of the problems solved (6 nodes) for the methods to be of sufficient interest.

As ATM networks grow, it becomes imperative to devise VP management strategies and tactics that are both computationally feasible and effective for the size of networks and volume of calls carried. Our goal is to identify VP creation procedures that are both feasible and effective for moderate and large networks consisting of up to 200 switching nodes. The ideal goal of this research is to enable networks to become autonomous and self-organizing in the sense that based on actual traffic loads, historical records, observed performance and grade-of-service (GoS) parameters, VPs are automatically created, sized and torn down during appropriate minutes, hours or days to provide for the best network throughput and performance.

1.1. Motivation

A Virtual Path (VP) is a direct pipeline between two nodes of a network such that cells that are transported from one end to the other see things as if there was a direct transport link between the two nodes. As defined by its standard, a VP identifier (VPI) of 1 byte is attached to every ATM cell as well as a VC identifier (VCI) of 2 bytes. The VPI tells which exit port of the switch the cell needs to get to while the VCI field identifies the circuit it belongs to. In general, there are substantially more VCs in a network than VPs.

A cell in transit in a VP switch will have only its VPI field scanned, with the VCI field remaining intact. A cell in transition in a VC switch will have both its VPI and VCI fields scanned. Thus, a VC switch has to do more "work" than a VP switch. If a cell is transported over a single physical link, the VPI and VCI fields are read at both ends in any case and no advantage results from establishing VPs. If, on the other hand, cells of a connection need to be routed over multiple physical links and switches then it might be worthwhile to set up an end-to-end VP so that the cells in transit are assigned a fixed VPI.

A VP may be routed on more than one physical edge of the network. That is to say, a VP between nodes i and j may be routed on the physical edges (i, k) and (k, j) to get from i to j. Because VP switching is lower in cost and faster than VC switching, it makes the network more efficient if VPs are built between nodes that are more than one transport link apart and have a large amount of traffic. Furthermore, at the call level, the aggregation of a large number of VCs into one VP will require less reserved bandwidth for the same blocking level as the sum of individual VC capacities put together in addition to VPs making the call admission control simpler. On the other hand, each VP reserves a prespecified amount of capacity from each transport link it is routed on only for its end-to-end demand, thus depriving the network of spare transport capacity at times when the reservation is not used to full advantage. Furthermore, there are a limited number of VPIs available at each switch port. Thus, there is a trade-off in VP creation between ease of call admission control and switching on the one hand and end-to-end reservation of bandwidth on the other.

This trade-off between efficiency of VP switching and loss of capacity to end-to-end VPs is the focus of the mathematical model to be described and the heuristic solution we develop in this paper.

1.2. VP management model

We start with a mathematical description that formally defines the problem that we wish to address, and proceed with a heuristic based on this model which does not involve an exact solution to this formulation.

We define a *flow* to be some bandwidth or offered erlang load between a pair of nodes and a *commodity* to be a request for a flow of specified bandwidth between a given pair of nodes. Erlang load is obtained by multiplying the expected *number* of calls to be made between a pair of nodes by their average bandwidths. In the ATM terminology, such calls may be referred to as the *switched virtual connections. Blocking* of calls occurs, therefore, due to the VP running out of capacity in the same way that a telephony trunk group runs out of 56kbps channels during a busy period. The calculation of a *bandwidth* for each SVC call depends on the traffic profile of the service. These issues are discussed at length in other publications, e.g., ElWalid and Mitra (1993).

There are thus numerous commodities in the network that "see" only virtual edges as they are routed between their individual origins and destinations. However, each VP (virtual edge) is defined as a path along one or more connected physical edges, see figure 1. We

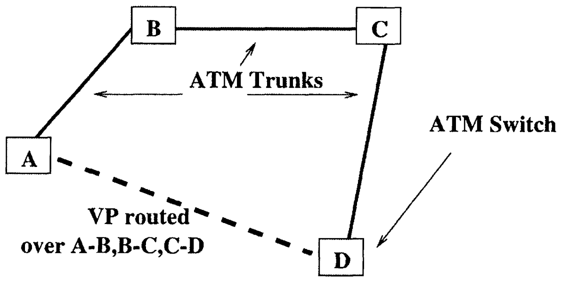


Figure 1. Layout of VPs on ATM trunks.

assume that the network transport capacity is already partitioned for different service QoS levels and thus the VP management problem is to be solved for a given service type only.

The problem of defining the set of virtual edges, sizing them (giving each a capacity), and mapping each of them to a physical path, is provably NP-hard, even without considering the nonlinearity inherent in the blocking probability.

2. Problem statement and mathematical model

2.1. *Problem definition*

Given an undirected capacitated ATM network $G = (N, E)$ with a set K of point-to-point demands, we would like to create a virtual path routing scheme which minimizes the total number of VP-switching operations in the network (to reduce call set-up time) while obeying network capacity restrictions on both the ATM trunks and the nodes. We also specify that the blocking probability for every commodity be at most b per trunk. The model allows splitting of calls between multiple VP trunks since no single path routing constraint is present. Such a constraint would add to the complexity of the model by introducing additional integer variables. However, it is preferable for calls of the same commodity to follow the same path of VP trunks to ensure cell-level QoS. Thus the solution heuristic (see Section 3) attempts to do this when possible.

For every node $i \in N$, we define v_i to be the maximum number of virtual paths that may begin at, end at, or pass through node i. For every ATM trunk $e = (i, j)$, $i, j \in N$, we define $c_e = c_{(i,j)}$ to be the capacity ATM trunk e. Finally, for every commodity $k \in K$, we define d_k to be the *offered load* (in Erlangs) of commodity k.

Let $ER1(b, c)$ be the maximum amount of traffic (in Erlangs) that can traverse an ATM trunk of capacity c such that the blocking probability is no more than b. Similarly, let $ER2(b, d)$ be the minimum capacity that is needed for d Erlangs of traffic to traverse an ATM trunk such that the blocking probability is no more than b. These two nonlinear functions are defined by the Erlang blocking formula $b = \frac{\frac{d^c}{c!}}{\sum_{j=0}^{c} \frac{d^j}{j!}}$, $0 < b < 1$.

2.2. *Assumptions*

We assume that G is a connected graph; otherwise, we can break the problem into subproblems defined on the connected components of G. Also, we assume that the known problem data are (v_i, c_e, and d_k, $\forall i, e, k$). In particular, we assume that the ATM trunks (c_es) are already sized, which is a natural assumption in network management.

2.3. *Models*

We define two models for this problem. The first is a path-based model and the second is a stacked multicommodity flow-based model.

To ensure end-to-end GoS in our models, it is reasonable to divide the most stringent GoS by the practical maximum number of VP trunks traversed, e.g. 3 or 4, and use the resulting value as the VP trunk blocking parameter b. This follows directly from $GoS(O, D) = \prod_{i \in \text{O--D route}}(1 - GoS(VP_i))$ and the simple approximation $GoS(O, D) \approx 1 - \sum_{i \in \text{O--D route}} GoS(VP_i)$ valid for small $GoS(VP_i)$.

2.3.1. Path-based model. Define an ATM path to be a string of ATM trunks $e_1, e_2, \ldots, e_l$ (where l is the length of the path) such that (i) e_i and e_{i+1} share a common endpoint, for all $1 \leq i < l$, and (ii) $e_i \neq e_j$, for all $i \neq j$. Thus an ATM path defines a route between two nodes such that no ATM trunk appears more than once on the route. Thus, every ATM path is a potential VP trunk.

Let P be the set of all possible ATM paths in G. Define I_p to be 1 if path $p \in P$ specifies a VP trunk in the solution, and 0 otherwise. Define s_p to be the size of the VP trunk specified by p. Let $\bar{P}$ be the set of all possible paths of ATM paths in G; thus, $\bar{p} \in \bar{P}$ is a string of ATM paths $p_1, p_2, \ldots, p_l$, where each $p_i \in P$. Thus, $\bar{P}$ is the set of all VP paths. Finally, define $f_{\bar{p}}^k$ to be the amount of flow for commodity $k \in K$ on VP path $\bar{p} \in \bar{P}$. Let M be a sufficiently large number.

We now formulate the problem of minimizing the VP-switching operations (that reduces call set-up times) as follows:

$$\text{minimize} \quad \sum_{k \in K} \sum_{\bar{p} \in \bar{P}} f_{\bar{p}}^k$$

$$\text{subject to} \quad \sum_{\bar{p} \in \bar{P}} f_{\bar{p}}^k \geq d_k \qquad \forall k \in K$$

$$\sum_{\bar{p}: p \in \bar{p}} \sum_{k \in K} f_{\bar{p}}^k - s_p \leq 0 \qquad \forall p \in P$$

$$\sum_{p: e \in p} ER2(b, s_p) \leq c_e \qquad \forall e \in E$$

$$\sum_{p: i \in p} I_p \leq v_i \qquad \forall i \in N$$

$$s_p - MI_p \leq 0 \qquad \forall p \in P$$

$$I_p \in \{0, 1\} \qquad \forall p \in P$$

In this formulation, we minimize the total number of VP-switching operations subject to (in order) the constraints that every commodity is completely accounted for in the routing scheme (1), no VP has more traffic than its size (2), the blocking probability on every edge is at most b (3), the VPI limits are observed at each node (4), no VP can be part of the solution unless it is counted (5) and $f_{\bar{p}}^k$, s_p and I_p are the decision variables. Note that the blocking probability constraints also act as capacity constraints on the edges.

To restrict the problem so that no splitting is allowed, we simply add the constraints $f_{\bar{p}}^k \geq MF_{\bar{p}}^k \, \forall \bar{p} \in \bar{P}, \forall k \in K$, $\sum_{\bar{p} \in \bar{P}} F_{\bar{p}}^k = 1, \forall k \in K$, and $F_{\bar{p}}^k \in \{0, 1\}, \forall \bar{p} \in \bar{P}, \forall k \in K$, where $F_{\bar{p}}^k$ is 1 if commodity k uses path $\bar{p}$ of VP trunks, and 0 otherwise.

Notice that if we ignore the concavity (in s_p) of the Erlang function ($ER2(b, s_p)$ by approximating $ER2(b, s_p)$ with s_p, then this model is a linear integer program. However, due to the size of the sets P and (especially) $\bar{P}$, it is extremely large. For example, if G is merely a complete graph on 5 nodes, then $|P| = 160$ and $|\bar{P}| = 4, 185, 760$.

2.3.2. Stacked multicommodity flow-based model. Due to the size of the Path-Based Model, we define a second model, based on the concept of multicommodity flows. We view the commodities as flowing only on the virtual network, and define a multicommodity flow problem as such. Simultaneously, considering the total amount of flow on each VP trunk as the demand for that VP, we also define a multicommodity flow problem on the physical network, using the VPs as commodities.

We denote the origin and destination of an ATM trunk, a VP trunk, or a commodity by $O(\cdot)$ and $D(\cdot)$. In the case of an undirected trunk or commodity, we arbitrarily assign one endpoint to be the origin and the other to be the destination.

For the moment, we assume that only one VP trunk can exist between any pair (ordered pair, for a directed problem) of endpoints. Then, we make the following definitions: $z_{(m,n)}$ is 1 if a VP trunk exists between nodes m and n, and 0 otherwise. $x^k_{(m,n)}$ is the fraction of the demand of commodity k that travels on the VP trunk between nodes m and n. $y_e^{(m,n)}$ is 1 if the VP trunk between nodes m and n is routed on ATM trunk e, and zero otherwise.

Now we formulate the problem as follows:

$$\text{minimize} \quad \sum_{k\in K}\sum_{(m,n)} d_k x^k_{(m,n)}$$

$$\text{subject to} \quad \sum_{n\in N} x^k_{(m,n)} - \sum_{n\in N} x^k_{n,m} = \begin{cases} 1, & \text{if } m = O(k) \\ -1, & \text{if } m = D(k) \\ 0, & \text{otherwise} \end{cases} \qquad \forall k \in K, \quad \forall m \in N$$

$$x^k_{(m,n)} \in \{0,1\} \qquad \forall (m,n), \quad \forall k \in K$$

$$x^k_{(m,n)} \le z_{(m,n)} \qquad \forall (m,n), \quad \forall k \in K$$

$$\sum_{j\in N} y^{(m,n)}_{(i,j)} - \sum_{j\in N} y^{(m,n)}_{(j,i)} = \begin{cases} z_{(m,n)}, & \text{if } i = m \\ -z_{(m,n)}, & \text{if } i = n \\ 0, & \text{otherwise} \end{cases} \qquad \forall i, \quad \forall (m,n)$$

$$y^{(m,n)}_{(i,j)} \in \{0,1\} \qquad \forall (i,j), \quad \forall (m,n)$$

$$\sum_{(m,n)}\sum_{j} y^{(m,n)}_{(i,j)} + \sum_{(m,n)}\sum_{j} y^{(m,n)}_{(j,i)} \le v_i \qquad \forall i \in N$$

$$\left(\sum_{(m,n)}\sum_{k\in K} ER2\big(b, d_k x^k_{(m,n)}\big)\right) y_e^{(m,n)} \le c_e \qquad \forall e$$

$$z_{(m,n)} \in \{0,1\} \qquad \forall (m,n)$$

To allow splitting, we can relax the x-variables so that they can take any value on the interval [0, 1].

Note that this is not a linear model, even if we replace the Erlang function with a linear approximation, because the last constraint contains the product of two variables. However, the model has other characteristics which we can take advantage of to construct a good heuristic solution method. Specifically, if we assume that we have a pre-determined set of VP trunks (thus, the values of the z-variables are set), then the first three sets of constraints of our model define a multicommodity flow problem on the x-variables, and the next three sets of constraints define a multicommodity flow problem on the y-variables. Only the last (nonlinear) set of constraints links the x- and y-variables together.

For simplicity, we call these two multicommodity flow problems the x-problem and the y-problem.

3. Heuristic solution methodology

We have created a two-stage heuristic to find a good solution to this problem. In the first, or *buildup* stage, we begin with an incomplete (perhaps empty) set of VP trunks and iteratively add new VP trunks until a feasible routing of all commodities is found. In the second, or *add-drop* stage, we begin with a feasible solution and, by iteratively adding and dropping VP trunks from our solution, we attempt to find progressively less-costly solutions.

In both stages, we attempt to solve the x- and y-problems by successively solving shortest path problems for each VP trunk and each commodity. For any problem of the type we address, if the ATM trunk capacities and node VPI limits are large enough, then this method will give us the optimal solution. For tighter problems, we have found that this method generally yields good solutions, and has the advantage that even a large set of shortest path problems can usually be solved much faster than a single multicommodity network flow problem.

3.1. Preprocessing

3.1.1. Zero-demand edges. Occasionally, it is necessary for a solution to include a VP trunk which is routed on a single physical edge. Because our heuristic only considers VP trunks whose endpoints are those of a network commodity, a simple preprocessing step is necessary. For each edge $(i, j) \in E$ for which no commodity from i to j exists, we create commodity (i, j) and assign it demand 0. In this way, we ensure that the direct VP from i to j on edge (i, j) will be considered by our heuristic, without affecting any of the capacity constraints.

3.1.2. Zero-VP nodes. Not all nodes have the proper equipment to be an endpoint of a VP trunk. If such a zero-VP node i has no commodities with positive demand whose origin or destination is i, then we set the VPI limit v_i to infinity and delete any zero-demand commodities with origin or destination i. Thus any number of VP trunks may pass through this node, but none may originate or end there.

It is also possible for a zero-VP node j to have one or more commodities with positive demand. In this case, we create a new node $\hat{j}$ and a new edge $(j, \hat{j})$. We re-define all commodities (with positive demand) with endpoint j, changing their endpoint to $\hat{j}$ instead. Finally, we set $v_{\hat{j}} = 1$ and preprocess node j as above, now that it has no positive-demand commodities.

In this way, we allow any number of VP trunks to pass through node j, but we force all positive-demand commodities with endpoint j to be routed on a single VP trunk to a positive-VP node, from which they can be routed along VP trunks to their other endpoint. This models a "hub-and-spokes"-type of network in which only the hub nodes are VP-capable.

Note that the zero-VP node preprocessing step comes after (and thus takes precedence over) the zero-demand edges preprocessing step.

3.2. Building a feasible solution

We begin each iteration of the buildup heuristic with a (possibly empty) set of existing VP trunks. We route these VP trunks on the physical network (subject to the VPI limits at each node) in random order using a shortest path or "hop" (Dijkstra's) algorithm. Then, based on the capacity constraints on each ATM trunk, we route as many commodities as possible on the virtual network.

Normally, no splitting of commodity routes is allowed. However, if after a (user-defined) number of iterations have passed without finding a feasible solution, an iteration takes place in which splitting is allowed. During these iterations, a commodity or commodities may be split and partially routed. In this case, however much demand can be routed on the shortest path is, and the rest of the demand becomes a "new" commodity (with the same endpoints) which is routed separately. Everytime a trunk is saturated it becomes invisible to the shortest path algorithm for the remaining flows.

Once the initial routing is complete, we remove all VP trunks which have no commodities routed on them. We then iteratively add new VP trunks by finding (randomly) a commodity which is not routed and creating, where possible, a direct VP trunk between its endpoints.

Because the goal of the buildup stage is to create a feasible solution, we do not allow the addition of VP trunks or the routing of commodities which would create infeasibilities with respect to the VPI or capacity constraints.

We repeat this buildup stage until a fully-feasible solution (a solution in which all constraints are satisfied and all commodities are completely routed) is found.

We found that the randomized order is necessary to avoid cycling. Without randomization, the same VP trunks could be added and then deleted from our test examples at every iteration, with no progress toward feasibility ever being made.

3.3. Improving the solution

Once we have a feasible solution, we heuristically attempt to improve the quality of the solution by iteratively adding and dropping VP trunks from the solution. Before the add and drop phases, we route all commodities (in random order) on the existing set of VP trunks.

When routing commodities at this stage, we allow temporary infeasibilities with respect to the VPI and capacity constraints. When all routing is finished, we take any VP trunks and commodities with pass through a node or edge whose constraint is violated, and reroute them. We repeat this process until all constraints are satisfied.

In the add phase, using a value function defined by the size and current routed distance (on the virtual network) between endpoints of each commodity, we order all commodities which are routed on more than one VP trunk and, where possible (depending on VPI and capacity constraints) add a direct VP trunk for each commodity and route that commodity.

At the end of the add phase, we drop all VP trunks which have no commodities. Additionally, we have a drop threshold such that we drop VP trunks whose removal does not all more than our threshold value to the overall cost of the solution. The drop threshold is initially set to 10% of the total demand in the network, and decreases by a factor of 10 approximately every two iterations until it reaches zero. The nonzero drop threshold allows us to explore different localities of the feasible region rather than remaining in a single (perhaps bad) locality.

4. Testing the heuristics

We performed a number of tests on our heuristic solution methods. The first class of tests was a set of small "tricky" problems, and the second was a large network problem (66 nodes, 96 edges, 488 commodities). We also compared our solutions with dynamic and on-line routing schemes to determine if the VPs were effective.

4.1. Performance on specialized test problems

Each of the following problems on which we tested our heuristic had some characteristic which made the VPs intuitively simple to design. Our purpose was to determine how well the heuristic performed in these difficult cases, which could occur as subgraphs within a large problem.

In all test cases, no initial set of VP trunks was specified; all solutions were created from scratch by our heuristic.

Our first test was on a large network (see Section 4.2) in which neither the VPI limits nor the edge capacities were binding constraints. Thus, the optimal solution was for every commodity to have its own VP trunk directly from its origin to its destination. Our heuristic successfully created this solution.

The second set of tests we performed was on networks in which the VPI limits were tight but the capacity was effectively unlimited. We tested one linear network in which the set of demands was complete (see figure 2). In this case 4 VPs were created: a VP from 1-2 with capacity 4 (to route demand 1-2, 1-3, 1-4 and 1-5), a VP from 2-3 with capacity 6 (to route 1-3, 1-4, 1-5, 2-3, 2-4, 2-5), a VP from 3-4 with capacity 6 (to route demands 1-4, 1-5, 2-4, 2-5, 3-4, 3-5) and a VP from 4-5 with capacity 4 (to route 1-5, 2-5, 3-5, 4-5). Another linear network had a specialized set of demands which allowed for a low-cost solution (see figure 3). We also tested a more-highly connected network (see figure 4) in which the only feasible solution was a spanning tree. In all three cases, our heuristic was successful, although it required one or more extra runs with different random seeds in order to find the solution.

Our third set of test cases included networks in which the capacity constraints dictated the characteristics of the solution, while the VPI limits were large enough to be effectively unlimited. We created a simply-solved network (see figure 5), tightened the edge capacities to force VP trunks to be shared by the commodities (see figure 6), and then tightened the capacities even more (see figure 7) to force a split solution. We also tested a single-commodity problem (see figure 8) in which the capacities were so tight that three distinct

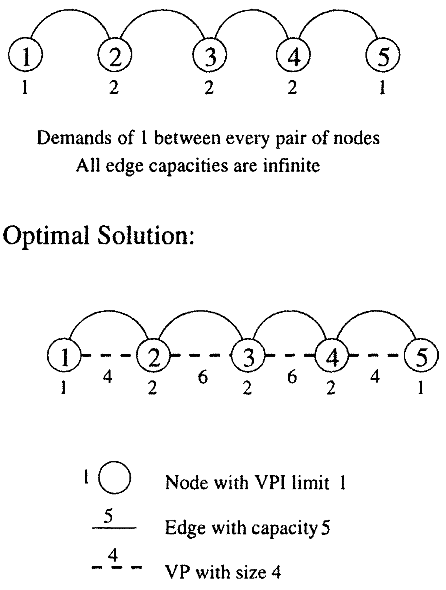


Figure 2. Linear network with complete demands.

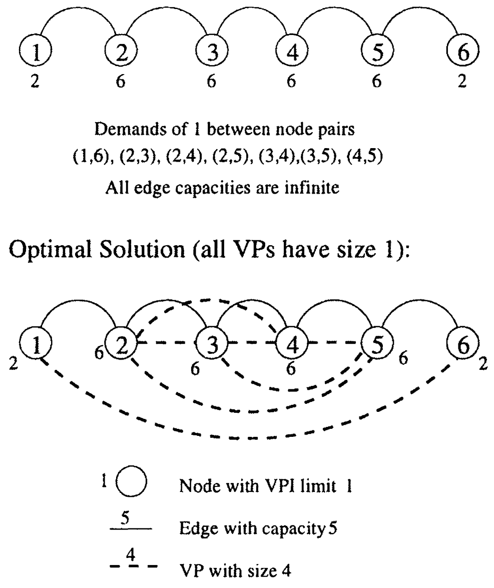


Figure 3. Linear network with specialized demands.

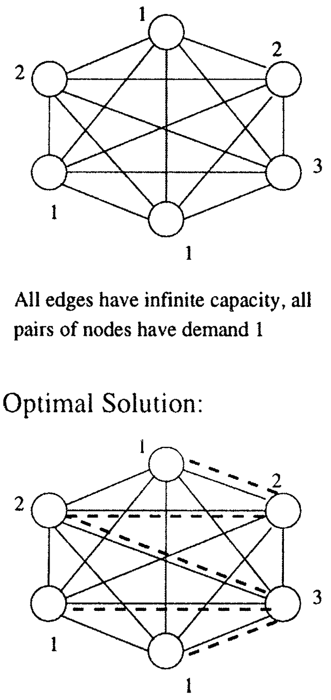

1
2
2
3
1
1
All edges have infinite capacity, all pairs of nodes have demand 1
Optimal Solution:
1
2
2
3
1
1

Figure 4. Highly connected network.

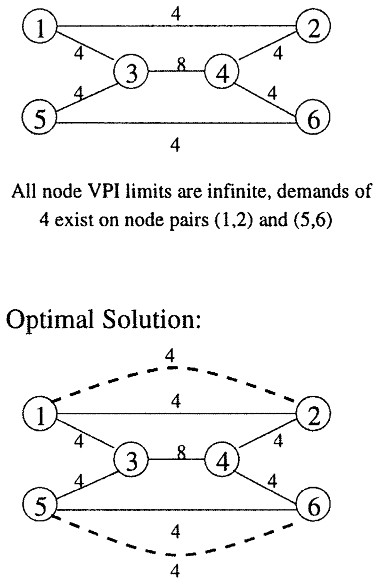
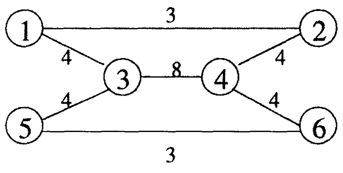

4
4
4
8
4
4
4
All node VPI limits are infinite, demands of 4 exist on node pairs (1,2) and (5,6)
Optimal Solution:
4
4
4
8
4
4
4
4
4

Figure 5. Network with two simple non-overlapping VPs.



All node VPI limits are infinite, demands of 4 exist on node pairs (1,2) and (5,6)

Optimal Solution:

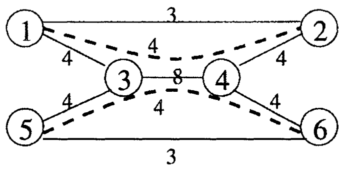
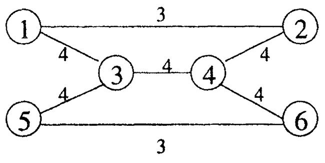


Figure 6. Network with two VPs sharing the network edge 3–4.



All node VPI limits are infinite, demands of 4 exist on node pairs (1,2) and (5,6)

Optimal Solution:

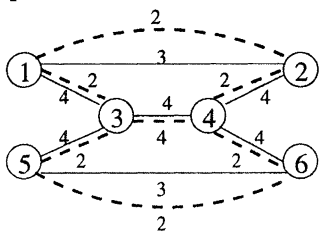
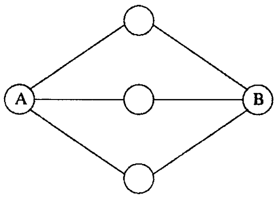


Figure 7. Network with multiple non-overlapping VPs.



All node VPI limits are infinite, all capacities are 5, a demand of 12 exists between A and B

Optimal Solution:

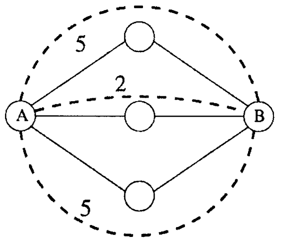
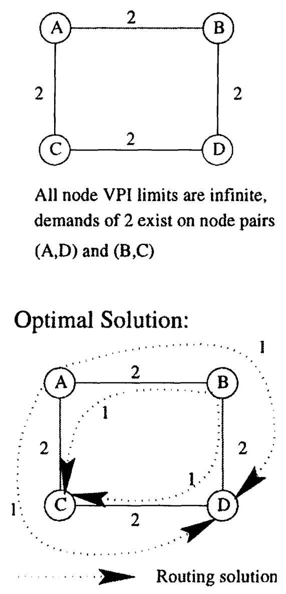


Figure 8. Network with tight capacities forces multiple VPs to be created.

VP trunks on three distinct physical paths had to be created to route the one commodity. Again, our heuristic successfully found the solutions to all of these problems.

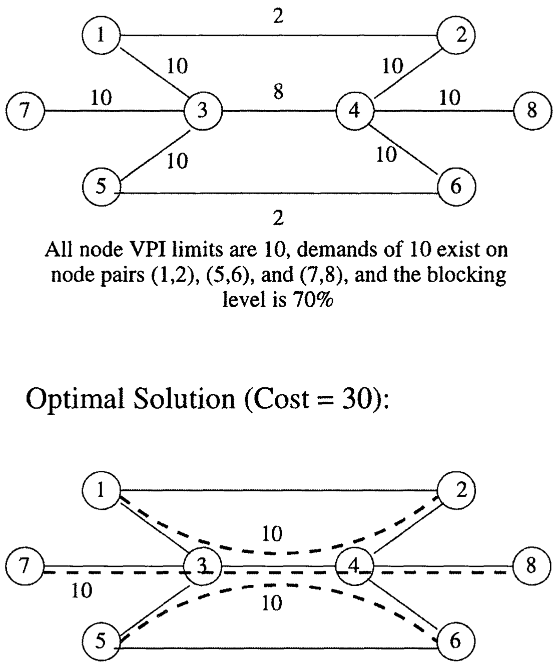
For our fourth test, we contrived an example which we knew our heuristic would not be able to solve (see figure 9). In general, we considered a single ring topology with two commodities such that any simple path between the endpoints of one commodity passed through exactly one of the endpoints of the other commodity. Every ATM trunk had the same capacity (with enough capacity to handle an even number (at least two units) of demand), and both commodities had just enough demand to singlehandedly use all the capacity of an ATM trunk. The only feasible solution is to split both commodities; however, because routing a commodity will bring any ATM trunk on which it travels to zero remaining capacity, the heuristic will not be able to split either commodity. In this case, it is the routing, not the VP selection, which causes the difficulty. A simple solution to this problem is to modify the initial data file so that one of the commodities is split into two: one with demand $n - 1$, where n was its original demand, and one with demand 1. When we made this simple change, our heuristic was able to solve every test problem of this type.

The final test we performed was on a network in which the capacity and VPI constraints both played a role in determining the solution. The first set of constraints we used (see figure 10) allowed an unshared solution. The heuristic successfully found the correct solution. The 70% blocking level occurs because of the 30 units of traffic that is ultimately offered to ATM trunk (3, 4) and other links have blocking much below this level. When we tightened the capacity (see figure 11), we expected to force a shared solution. However, the heuristic found a way to exploit high VPI limits, and found a better solution than we had thought existed. When we tightened all of the VPI limits (see figure 12), the heuristic successfully found the unique (split and shared) solution.



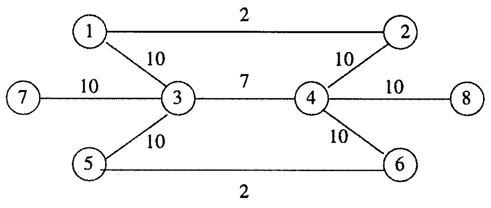
A
B
C
D
2
2
2
2
All node VPI limits are infinite, demands of 2 exist on node pairs (A,D) and (B,C)
Optimal Solution:
1
1
1
1
Routing solution

Figure 9. Network with split routing.



1
2
3
4
5
6
7
8
2
10
10
10
8
10
10
10
2
All node VPI limits are 10, demands of 10 exist on node pairs (1,2), (5,6), and (7,8), and the blocking level is 70%
Optimal Solution (Cost = 30):
10
10
10

Figure 10. Network with unshared VPs.



All node VPI limits are 10, demands of 10 exist on node pairs (1,2), (5,6), and (7,8), and the blocking level is 70%

Optimal Solution (Cost = 70):

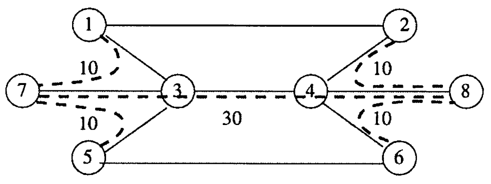
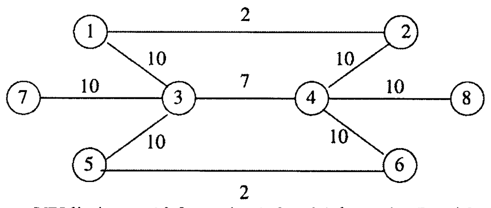


Figure 11. Network with tight capacities does not force shared VPs.



VPI limits are 10 for nodes 1-6 and 1 for nodes 7 and 8, Demands of 10 exist on node pairs (1,2), (5,6), and (7,8), Blocking Level is 70%

Optimal Solution (Cost = 90):

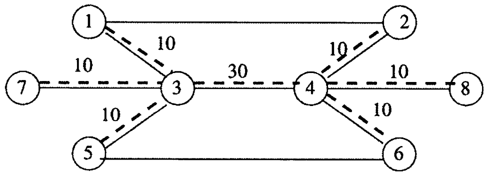


Figure 12. Network with tight capacities results in split and shared VPs.

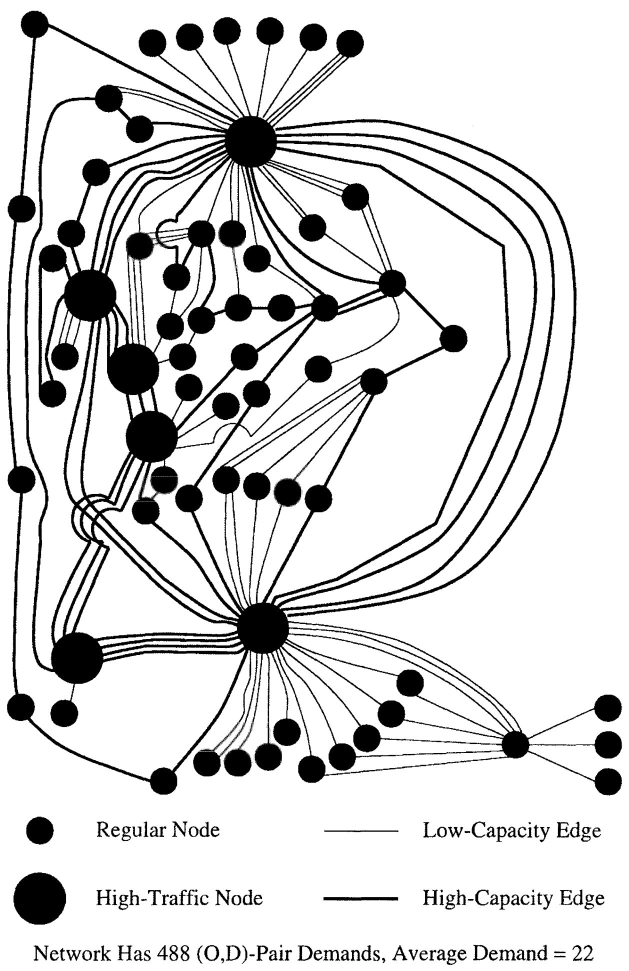


Figure 13. VP design for a real medium-sized core ATM network.

4.2. Performance and robustness for a large network

Using real network topology and demand data, we used a Bellcore network planning tool to generate edge capacities for a 66-node, 96-edge, 488-commodity network (see figure 13).

For this problem we solved the LP relaxation of the second problem formulation given in Section 2.3.2 in which the VPI constraints were also relaxed to attain a lower bound. We then applied our heuristic to this problem, starting with an empty VP set. The heuristic created a solution with a cost of 13086, which is about 22% from the above lower bound (assuming a single VP trunk for each commodity) of 10756. However, the lower bound is provably unattainable due to not meeting the VPI limits. When the VPI limits are added to the LP formulation, the heuristic solution was found to be within 20% of this new bound.

We also tested the heuristic on the same problem, but starting with a VP trunk set identical to the ATM trunk set E. Thus, for every ATM trunk $(i, j) \in E$, there was a VP trunk from i to j in the initial VP trunk set. With this simple start, our heuristic created a solution costing 11549, which is about 7% from the unattainable lower bound, and therefore roughly 5–6% from the optimal solution.

In both cases, the final solution contained approximately 250 VPs.

For the examples given, the run times ranged from a fraction of a second (5 node linear network) to 10s of minutes for the 66 node example. In order to test the robustness of our heuristic, we solved the problem 10 times, each with successively higher demands (ranging from 1% to 10% increases over the initial demand data). We calculated two measures of robustness: the percentage of VP trunks that were the same between two solutions, and the weighted average size difference percentage between similar VP trunks in the different solutions. The second measure is calculated as follows: for every VP trunk which is the same between two solutions, multiply the absolute value of the size difference by the size of the VP trunk in the smaller-demand solution. Sum over all VP trunks, and divide by the sum of the squares of the sizes of the similar VP trunks.

When we started the heuristic from scratch (with an empty set of initial VP trunks), we found that 80–85% of the VP trunks were generally the same, but that their size varied by 15–30%.

On the other hand, starting the heuristic with the ATM trunks as initial VP trunks led to over 90% of the VP trunks being the same between any two demand levels (from 0% to 10% inflation), with VP trunk size varying from inflation difference by no more than 1% in most cases, and never by more than 2.5%.

4.3. Comparison with dynamic and online competitive routing

We selected a 15 node, 64 (physical) edge network in which 5 nodes where core and fully-connected (with link capacities) and the remaining 10 had one or more capacitated physical links to the core nodes. There was demand between all pairs of nodes. Using the point-to-point loads, we attempted to use (i) state-dependent routing of Ott and Krishnan (1992) and (ii) online competitive routing (Awerbuch et al., 1994) over the physical trunks and compare the results with the blocking associated with the VPs designed by our code. The idea was to show the effectiveness of VP creation in increased throughput and reduction in blocking.

State dependent routing (SDR) as coded by Krishnan could not be used since there were several commodities requiring at least three trunks for routing, and this available implementation assumed all commodities had a one-trunk or at most two-trunk routes available to them. We doubt, however, that in networks where routes are no more than two hops long there is much gain in VP creation and management.

Online routing, based on Bellcore's implementation of the Plotkin-like competitive scheme (Andrews, Krishnan, etc.), showed an average blocking of 13% if the physical links were used as trunks. Setting the per VP-trunk blocking at 5%, we obtained a VP layout design that routed all demands without exceeding the capacity of the physical trunks. Over 90% of the demands were routed on 1 or 2-trunk routes, resulting in overall blocking of less than 11% overall. Thus the VP map created resulted in improved throughput as measured by the Bellcore's implementation of the online competitive routing. This improved performance is to be attributed to the efficiency and blocking improvements introduced by VP trunks in networks in which non-negligible amounts of demands have to traverse multiple ATM trunks. Naturally, the efficiency of VP creation is diminished when the underlying ATM trunk network has (say) at most 2-hop routes for any commodity. In other words, a critical advantage of VPs in sparse networks is that they help reduce the induced cost of a path (Kelly, 1986; Girard, 1993) through commodity switching, as seen by commodities.

We conclude that over networks with large hop counts for a significant number of node pairs and in which utilization is non-negligible, VP management significantly improves blocking and thus call admission while for low utilization networks in which all node pairs have one or two hop routes, the effectiveness of VP creation and management is not as large.

5. Conclusion

In this paper, we have considered the problem of establishing and routing VP trunks in an ATM network partitioned to carry QoS-sensitive services. We formulated the problem as a layered multicommodity network flow problem with nonlinear constraints and created a shortest-path-based heuristic algorithm to solve it iteratively. We tested our heuristic on a number of special cases which can form the *building blocks* of difficult large problems, and found that the heuristic was able to successfully find the optimal solution (or at the main characteristics of the correct solution) in each case, with occasional preprocessing steps. Tests of the proposed algorithm on a real network (topology, edge capacities and demands) extracted from a Bellcore planning tool show that when seeded with VP trunks corresponding to the physical edges of the network our heuristic provides a good solution to the VP trunk definition and routing problem, coming within 7% of the (unattainable) lower bound for the problem and 5–6% of optimality. This is particularly true when VP trunks were initially selected to coincide with the physical edges. Further, the seeded heuristic is robust with respect to small (up to 10%) increases in demand, with most cases showing no more than a 1% difference in the sizes of similar VPs (after adjustment for increased demands) and no less than 91% of VP trunks being similar, even for the most marked (10%) fluctuation in demands. These examples also showed the efficiency of VP creation in networks in which the average end-to-end demand has to traverse more than one or two physical trunks. Clearly, the effectiveness of VP management is reduced in networks in which either direct trunks or at most two trunk routes exist between most, if not all, end nodes.

Acknowledgment

The authors wish to thank the anonymous referees for detailed and helpful comments.

References

Addie, R.G., J.L. Burgin, and S.L. Sutherland. (1988). "B-ISDN Protocol Architecture," *Proc. Globecom '88*, Vol. 2, Florida, pp. 716–720.

Ahn, S., R.P. Tsang, S.-R. Tong, and D.H.C. Du. (1993). "Virtual Path Layout Design on ATM Networks," *IEEE Infocom '94*, Vol. 1, Canada, pp. 192–200.

Awerbuch, B., Y. Azar, S. Plotkin, and O. Waarts. (1994). "Competitive Routing of Virtual Circuits with Unknown Duration," *Proc. of 5th ACM-SIAM Symp. Disc. Algo.* pp. 321–327.

Biro, J., A. Farago, T. Tron, and M. Boda. (1996). "Neural Networks for Logical Partitioning of ATM Networks," *IEEE Globecom '96*, Vol. 1, London, pp. 745–750.

Burgin, J. and D. Dorman. (1991). "Broadband ISDN Resource Management: The Role of Virtual Paths," *IEEE Comm. Mag.* 29(9), 44–48.

Chlamtac, I., A. Farago, and T. Zhang. (1993). "How to Establish and Utilize Virtual Paths in ATM Networks," *IEEE ICC'93*, Vol. 3, Geneva, pp. 1368–1372.

Chlamtac, I., A. Farago, and T. Zhang. (1994). "Optimizing the System of Virtual Paths," *IEEE/ACM Trans. on Networking* 2(6), 581–587.

Girard, A. (1993). "Revenue Optimization of Telecommunication Networks," *IEEE Trans. on Communications* 41, 583–591.

ElWalid, A. and D. Mitra. (1993). "Effective Bandwidth of General Markovian Traffic Sources and Admission Control of High Speed Networks," *IEEE/ACM Trans. Networking* 1, 329–343.

Kanada, T., K. Sato, and I. Tsuboi. (1987). "An ATM-Based Transport Network Architecture," *IEEE COMSOC* Workshop, Osaka, pp. 2–2.

Kelly, F.P. (1986). "Blocking Probabilities in Large Circuit-Switched Networks," *Adv. Appl. Prob.* 18, 473–505.

Lin, F.Y.-S. and K.-T. Cheng. (1994). *IEEE Globecom '94*, Vol. 2, pp. 777–782.

Medhi, D. (1995). "Multi-Hour, Multi-Traffic Class Network Design for Virtual Path-Based Dynamically Reconfigurable Wide-Area ATM Networks," *IEEE/ACM Trans. on Networking* 3(6), 809–818.

Mitra, D. and I. Ziedins. (1997). "Hierarchical Virtual Partitioning: Algorithms for Private Networking," *Proc. IEEE Globecom '97*, Vol. 3, pp. 1784–1791.

Monteiro, J.A.S., M. Gerla, and L. Fratta. (1991). "Statistical Multiplexing in ATM Networks," *Perf. Eval.* 12, 157–167.

Ott, T. and K. Krishnan. (1992). "Separable Routing: A Scheme for State-Dependent Routing of Circuit-Switched Telephone Traffic," *Annals of OR* 35, 43–68.

Saito, H. (1994). *Teletraffic Technologies in ATM Networks*. Boston, MA: Artech House.

Sato, Y. and K.-I. Sato. (1991). "Virtual Path and Link Capacity Design for ATM Networks," *IEEE JSAC* 9, 104–111.

Journal of Heuristics, 6, 85–105 (2000)
© 2000 Kluwer Academic Publishers

Design of Stacked Self-Healing Rings Using a Genetic Algorithm

MOR ARMONY*
New York University, New York, New York 10012, USA

JOHN G. KLINCEWICZ
AT&T Labs, Middletown, New Jersey 07748, USA

HANAN LUSS†
Telcordia Technologies, Piscataway, New Jersey 08854, USA

MOSHE B. ROSENWEIN†
Merck-Medco Managed Care, L.L.C., Franklin Lakes, New Jersey 07417, USA

Abstract

Ring structures in telecommunications are taking on increasing importance because of their "self-healing" properties. We consider a ring design problem in which several stacked self-healing rings (SHRs) follow the same route, and, thus, pass through the same set of nodes. Traffic can be exchanged among these stacked rings at a designated hub node. Each non-hub node may be connected to multiple rings. It is necessary to determine to which rings each node should be connected, and how traffic should be routed on the rings. The objective is to optimize the tradeoff between the costs for connecting nodes to rings and the costs for routing demand on multiple rings. We describe a genetic algorithm that finds heuristic solutions for this problem. The initial generation of solutions includes randomly-generated solutions, complemented by "seed" solutions obtained by applying a greedy randomized adaptive search procedure (GRASP) to two related problems. Subsequent generations are created by recombining pairs of "parent" solutions. Computational experiments compare the genetic algorithm with a commercial integer programming package.

Key Words: network design, telecommunications networks, SONET rings, integer programming, combinatorial optimization

1. Introduction

In recent years, developments in synchronous (SDH/SONET) transmission products have led to considerable interest in using self-healing rings (SHRs) in telecommunications networks. The term "self-healing" refers to the fact that service can be restored very rapidly (within milliseconds) in the event of a single link or node failure.

In an SHR, nodes that have equipment called Add-Drop Multiplexers (ADMs) are linked in a cycle so that traffic can flow between every pair of nodes. Depending on the amount of

*Work was partially performed while employed as an Intern at AT&T Labs.
†Work performed while employed at AT&T Labs.

traffic and on the capacity of the SHR, it is possible that a single SHR will be insufficient to serve all the traffic for a given set of nodes. In such a situation, one common approach is to use so-called "stacked" rings (also called "parallel" or "concentric" rings). In a stacked ring design, several SHRs are routed around the same ring topology. Each SHR would then serve some subset of the nodes. Our model assumes that one of the nodes is designated as a hub node that is to be served by each of the SHRs.

Depending on the situation, different operational assumptions or restrictions on the stacked ring design may be appropriate. We consider a situation with the following features:

- Each node may be served by multiple SHRs, with a separate ADM for each SHR. A fixed cost is incurred for each ADM.
- A given traffic demand between a pair of nodes i and j must follow a single route on the rings. This represents common practice in many systems.
- The (single) hub node can be used to exchange traffic between SHRs. A cost is incurred per unit of traffic exchanged.
- There is a limit on the number of ADMs allowed per SHR. Likewise, there is a fixed ADM capacity that limits the amount of traffic carried on any given SHR.

The problem is to determine to which rings each node should be connected, and how traffic should be routed on the rings. We will refer to this problem as the Stacked Ring Design Problem (SRDP). The objective is to optimize the tradeoff between the cost of ADMs and the cost of exchanging traffic at the hub.

Earlier work has considered related problems with differing assumptions. Harder (1996), for example, developed linear programming approaches for models in which no traffic exchange is permitted between the stacked rings. Klincewicz, Luss, and Yan (1998) consider a situation with multiple ring topologies, where each non-hub node is to be on a single SHR. Laguna (1994) developed a tabu search heuristic for a model in which traffic between a pair of nodes may be split among multiple routes. In general, there is a fast-growing literature on problems of network design using SHRs. For example, Wasem, Wu, and Cardwell (1994) and Cosares et al. (1995) describe computer-aided design procedures for survivable synchronous networks, and Luss, Rosenwein, and Wong (1998) describe an algorithm for selecting inter-networked rings from among a large set of candidate rings.

The solution approach that we develop for SRDP utilizes a genetic algorithm (GA). GA is a metaheuristic that works with a "population" of many potential solutions. Using procedures that are based on analogies to principles of population genetics, such as "survival of the fittest", it creates successive generations of potential solutions. The best individual solution that is encountered in the course of the successive generations is kept as the final heuristic solution to the problem. The GA approach has been applied to various combinatorial problems; see, for example, Dowsland (1996), Goldberg (1989) or Reeves (1993). In particular, recent applications of GA to telecommunications problems include Cox et al. (1996) and Davis et al. (1997). Comparisons with CPLEX, a commercial integer programming package, indicate that our GA heuristic generates excellent solutions to SRDP.

In order to generate "seed" solutions for the GA, we utilize a greedy randomized adaptive search procedure (GRASP) for two related problems. Since GRASP heuristics contain a random element, multiple heuristic solutions can be generated.

Section 2 provides more detailed descriptions of SHRs and stacked ring designs. A mathematical formulation of SRDP is described in Section 3. Section 4 gives an overview of the GA metaheuristic in general. The particular GA heuristic that has been developed for SRDP is then described in Section 5. Section 6 provides an illustrative example and Section 7 discusses our computational results. Section 8 provides final remarks.

2. Self-healing rings (SHRs)

Figure 1 depicts a self-healing ring (SHR), with seven nodes arranged in a cycle and linked by a pair of optical fibers. An Add-Drop Multiplexer (ADM) is located at each node.

An SHR has both "working capacity" and "protection capacity". "Working capacity" is used to serve demand under ordinary circumstances; "protection capacity" is used in the event of a single link or node failure. The "capacity" of an SHR refers to the "working capacity". Each ADM around a ring must have the same capacity.

Our model assumes that the type of SHR being used is a so-called *two-fiber uni-directional ring*. In a two-fiber uni-directional ring, traffic in the two fibers travels in opposite directions. The fiber in one direction, say clockwise, is the "working" fiber, used to serve all the traffic. The traffic between any pair of nodes i and j $(i < j)$ utilizes the entire working fiber to accommodate demand from i to j and from j to i. As a result, the capacity of the working

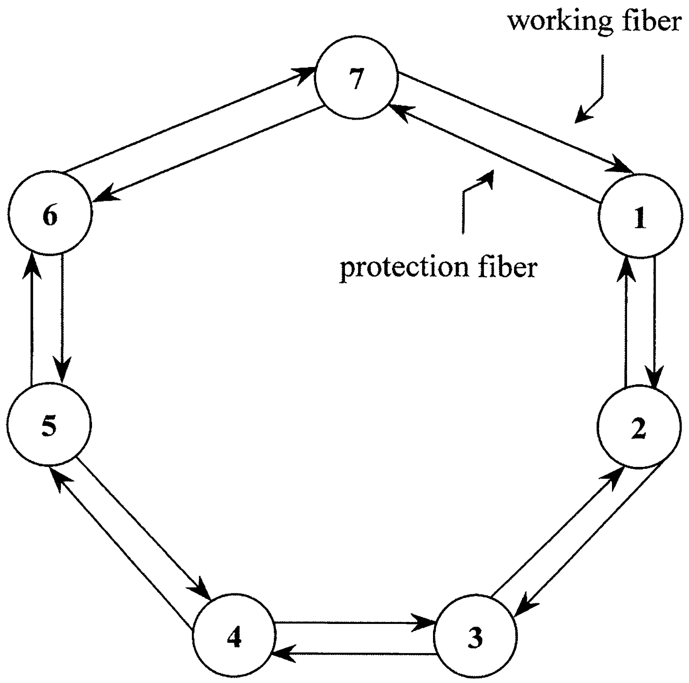


Figure 1. A self-healing ring.

fiber on a uni-directional ring must be as large as the sum of all the point-to-point demands that are carried on the ring. The fiber in the opposite direction is used for protection. In the event that one of the links in the ring is cut, traffic can travel on this protection fiber in the opposite direction to reach its destination. For example, suppose the link between nodes 3 and 4 is cut. Then the traffic from node 1 to 5 may be routed on the working fiber from 1 to 3, switched to the protection fiber at node 3, and routed on the protection fiber from node 3 to 5.

We note that there are also other types of rings, called bi-directional rings, in which there is working capacity in both the clockwise and counterclockwise directions. For bi-directional rings, the amount of capacity utilized by a set of demands depends on how demands are routed around the ring. Algorithms to "balance" traffic loads among the links of bi-directional rings are given, for example, in Cosares and Saniee (1994), Harshavardhana, Johri, and Nagarajan (1996), Myung (1997), and Schrijver (1998). For more detailed explanations of SHRs, see, for example, Wu (1992). Extensions of our model to bi-directional rings are discussed in Section 8.

As indicated above, the algorithm described in Section 5 is used to design a set of stacked, uni-directional rings. In a set of stacked rings, there is a single topological ring that links all the nodes. Each stacked ring will follow the route of the topological ring, but only connect a subset of the nodes.

To illustrate, consider figure 2. There are ordinary nodes 1, 2, 3, 4, 5 and 6, plus a designated hub node 7. One stacked ring connects nodes 1, 2, 4, 6 and 7; the other stacked ring connects nodes 1, 3, 5, 6 and 7. Traffic between nodes 3 and 4, for example, travels between node 4 and hub node 7 on the first stacked ring and between node 7 and node 3 on the second stacked ring. Thus, each demand unit between nodes 3 and 4 uses a capacity unit on both rings. We assume that the demand between a given pair of nodes does not exceed the capacity of a single SHR. (Otherwise, fully-utilized, dedicated rings for the corresponding pair of nodes can be installed.)

3. Mathematical formulation

In this Stacked Ring Design Problem (SRDP), we consider two types of costs: a fixed cost h per ADM, and a cost t per unit of demand that is exchanged at the hub node. The cost per ADM tends to encourage fewer ADMs, and, hence, more interconnections between rings. The interconnection cost, however, tends to encourage more traffic within each stacked ring, and, hence, more ADMs.

We consider a fixed number R of stacked uni-directional rings. There are N nodes $(1, 2, \ldots, N)$. The hub node (node N) is assigned to all of the rings. The other nodes can be assigned to multiple stacked rings; however, there is an upper bound b on the number of nodes per ring (not including the hub node).

Let c be the ADM capacity, i.e., the maximum demand that each ring can carry. We use the convention that, for every two nodes $(1 \leq i < j \leq N)$, there is a demand $d_{ij} \geq 0$ from i to j and from j to i, thus using up d_{ij} working capacity units along the entire ring.

Variables x_{ik} define ADM assignments to rings, where $x_{ik} = 1$ if node i is assigned to ring k and $x_{ik} = 0$ otherwise $(i = 1, 2, \ldots, N-1$ and $k = 1, 2, \ldots, R)$. Variables y_{ijkl} define the

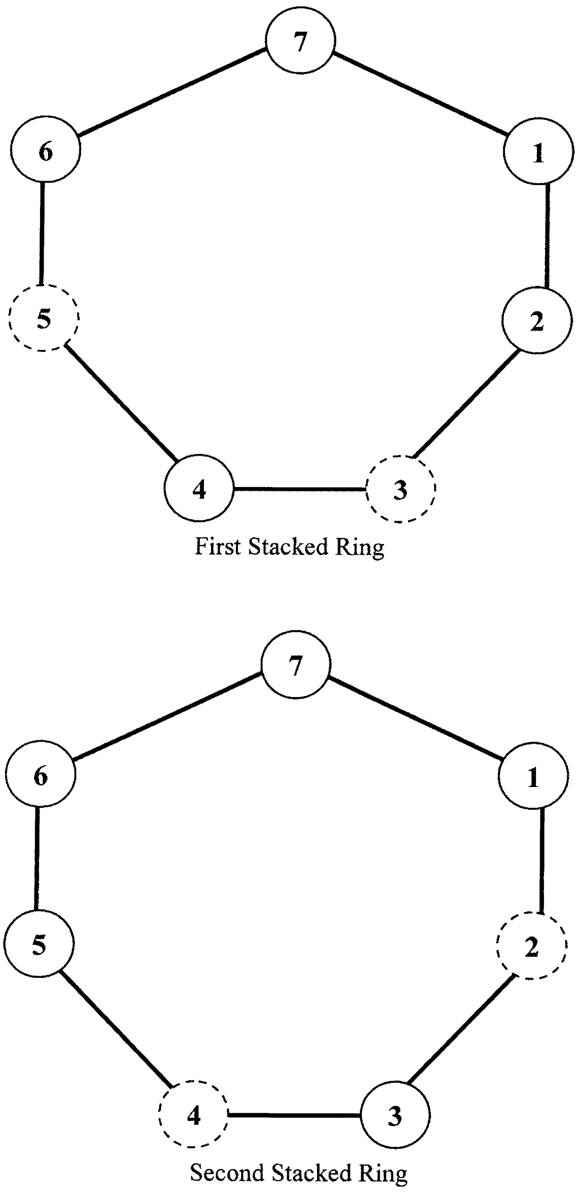


Figure 2. Stacked rings.

demand routing, where $y_{ijkl} = 1$ if demand d_{ij} is routed between node i on ring k and node j on ring l and $y_{ijkl} = 0$ otherwise ($1 \leq i < j \leq N$ and $k, l = 1, 2, \ldots, R$). If $y_{ijkk} = 1$, then the demand d_{ij} between nodes i and j is routed on the same ring, ring k. If $y_{ijkl} = 1$ for $l \neq k$, then the demand d_{ij} between nodes i and j is routed on both rings k and l, using d_{ij} capacity units on both rings.

The problem can be formulated as follows:

SRDP

$$\text{Minimize}\left[t\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\sum_{k=1}^{R}\sum_{\substack{l=1\\l\neq k}}^{R}d_{ij}y_{ijkl}+h\sum_{i=1}^{N-1}\sum_{k=1}^{R}x_{ik}\right] \tag{1}$$

so that:

$$\sum_{k=1}^{R}x_{ik}\geq 1 \quad \text{for } i=1,2,\ldots,N-1 \tag{2}$$

$$\sum_{i=1}^{N-1}x_{ik}\leq b \quad \text{for } k=1,2,\ldots,R \tag{3}$$

$$\sum_{k=1}^{R}\sum_{l=1}^{R}y_{ijkl}=1 \quad \text{for } 1\leq i<j\leq N \tag{4}$$

$$\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\left[d_{ij}y_{ijkk}+\left(\sum_{\substack{l=1\\l\neq k}}^{R}d_{ij}(y_{ijkl}+y_{ijlk})\right)\right]\leq c \quad \text{for } k=1,2,\ldots,R \tag{5}$$

$$y_{ijkl}\leq x_{ik} \quad \text{for } 1\leq i<j\leq N \text{ and } k,l=1,2,\ldots,R \tag{6}$$

$$y_{ijkl}\leq x_{jl} \quad \text{for } 1\leq i<j\leq N \text{ and } k,l=1,2,\ldots,R \tag{7}$$

$$x_{ik},y_{ijkl}\in\{0,1\} \quad \text{for } 1\leq i<j\leq N \text{ and } k,l=1,2,\ldots,R. \tag{8}$$

The terms in the objective function (1) represent the interconnection costs and ADM costs, respectively. The cost of the ADMs at the hub node is a constant hR and is, therefore, not included (determining an optimal number R of rings will be discussed in Section 7). Constraints (2) indicate that each node should be on at least one stacked ring. Constraints (3) indicate that there are at most b nodes (excluding the hub node) on any given stacked ring. Constraints (4) ensure that all demands are routed. The capacity constraints on the stacked rings are enforced using constraint (5). Constraints (6) and (7) ensure that $y_{ijkl}=0$ if either $x_{ik}=0$ or $x_{jl}=0$. Constraints (8) indicate that x_{ik} and y_{ijkl} are binary variables. (Note that if the demand d_{ij} were allowed to be split among multiple routes, then the decision variables y_{ijkl} would simply be defined as continuous variables between 0 and 1.)

4. The Genetic Algorithm metaheuristic

A Genetic Algorithm (GA) consists of five primary components:

Encoding of a problem solution as a "chromosome". Each string ("chromosome") consists of a sequence of characters ("genes"). For example, this might be a sequence of zeros and ones. Let L represent the length of the sequence. To apply GA to a particular problem, it is necessary to devise an encoding scheme that provides a mapping between potential solutions to the problem and possible sequences of length L.

Choice of initial population. Each "generation" consists of a set of possible solutions, encoded as "strings" or "chromosomes". To start the algorithm, an initial set of chromosomes must be generated. Typically, these are generated randomly. However, the initial population can also include some so-called "seed" solutions that have a particular simple structure or that can be generated by a fast heuristic.

A fitness function. Each potential solution must be evaluated according to a fitness function. Better solutions have higher fitness. The fitness function will be used to choose chromosomes to serve as "parents" for the next generation of solutions.

Creation of the next generation. Given a set of feasible solutions (that is, the current generation), a new set (the next generation) must be created. First, a number of the best solutions from the current generation can be automatically carried over into the next generation. These are called "elite" solutions. The rest are created by "recombinations" of chromosomes from the current generation. A single recombination occurs as follows:

- A pair of chromosomes (that is, a set of "parents") from the current generation is chosen in a random fashion.
- Two new solutions ("children") are created from the parent solutions. For example, as is done in our implementation, a crossover point n, where $1 \leq n < L$, is first chosen randomly. The first n genes of the first parent are combined with the last $L - n$ genes of the second parent to create one child, and the first n genes of the second parent are combined with the last $L - n$ genes of the first parent to create a second child. This is known as "one-point crossover".
- Mutations can occur in the children. That is to say, with some low probability, say 1%, changes can occur in the genes of the children. For example, if each gene is a bit with 0/1 values, then with probability 1%, a given bit is changed from 1 to 0 or vice versa.
- As necessary, adjustments for feasibility, etc. can be made in the children.

The recombination process continues until the size of the next generation is equal to the size of the current generation.

Stopping rule. Typically, the decision on when to stop is made *a priori* by fixing the number of generations. Alternatively, it can depend on the number of successive generations for which no improvement in the best fitness function value has been found.

The best solution encountered in the course of the generations is kept as the final solution. For more detailed discussion of GAs and their applications, see, for example, Dowsland (1996), Goldberg (1989) or Reeves (1993). Applications of GAs to telecommunications problems include Cox et al. (1996) and Davis et al. (1997).

5. A Genetic Algorithm for stacked ring design

5.1. Overview

We refer to the particular genetic algorithm that we propose for SRDP as GA-SRDP. The features of GA-SRDP are described below.

Encoding a solution to SRDP. We use the following encoding. Each chromosome consists of a sequence of 0/1 bits of length $L = (N - 1)R$. Each bit represents a variable x_{ik}, which equals 1 if node i has an ADM on ring k, and 0 otherwise. The first $N - 1$ bits correspond to nodes on ring 1, the second $N - 1$ bits correspond to nodes on ring 2, etc. Thus, there is a unique mapping between bit strings of length L and possible ADM placements.

To fully specify a solution, however, values must also be given for the y_{ijkl} variables that indicate how demands are to be routed. Given a chromosome that specifies values for x_{ik} variables, we determine values for y_{ijkl} variables by means of a heuristic. The heuristic assigns each demand to a single route between two ADMs. First, it attempts to assign demands that can be routed on a single ring. Next, it assigns remaining demands to routes that utilize two rings. Details of the heuristic are given in Section 5.2.

Choice of initial population. The initial population consists of a small number of "seed" solutions in addition to a set of randomly generated solutions. The seed solutions represent heuristic solutions to two related problems. For some seeds, we generate solutions where each non-hub node is on just one stacked ring. For others, we generate solutions where sufficient ADMs are included so that no traffic is exchanged between rings. These seeds are feasible to SRDP as formulated in Section 3, but represent, in a sense, "extreme" cases. The heuristics for seed solutions are described in Section 5.3.

In order to generate initial random solutions, we consider nodes one at a time, in turn. For each node i, a random number q is chosen, where $1 \leq q \leq R$. Node i is assigned to q rings, chosen at random from among those rings that have less than b nodes assigned thus far. In the event that only q' such rings are available, where $q' < q$, then node i is also assigned to $q - q'$ of the "full" rings. Any infeasibilities that result are resolved after all nodes have been processed. First, if, in a randomly generated solution, a ring k' serves more than b nodes, then nodes are chosen one-by-one, at random, to be removed from ring k' until the constraint is met. Next, each node i is checked to make sure that it is served by a least one ring. If not, it is assigned randomly to a non-full ring. If no non-full ring is available, a node j that is served by multiple rings is chosen at random. A ring k' serving j is chosen at random, and node i is assigned to ring k' in place of j.

The fitness function. Each solution must be evaluated according to a fitness function. Recall that a solution consists of the x_{ik} values as specified in a given chromosome, together with the y_{ijkl} values generated by the heuristic in Section 5.2. The fitness function for a given solution m has two components:

- *Objective function value v_m.* The objective function value consists of the sum of ADM costs plus the cost of traffic that is exchanged at the hub, as given in the objective function (1).
- *Penalty cost p_m.* For some chromosomes, it may be impossible to route the traffic without violating the capacity constraints (5). The heuristic in Section 5.2 is designed to route all traffic demands; if necessary, constraints (5) will be violated. A user-specified penalty cost is incurred, for each given ring, per unit of traffic in excess of c units. This penalty cost should be large enough so that infeasible solutions would not be included among the elite solutions. In practice, we found that setting the penalty cost equal to the ADM cost worked well.

The fitness function f_m for a given solution m is defined as $f_m = V - v_m - p_m$, where V is the worst solution value, i.e., $V = \max_m\{v_m + p_m\}$. Thus, better solutions have greater fitness.

Creation of the next generation. In creating the next generation of solutions, we use "elite" solutions plus "recombinations", as described in Section 4. For "recombinations", pairs of "parents" must be chosen. In our implementation, the first parent is chosen randomly, with probabilities that are proportional to their fitness function values. That is the probability of selecting solution $\bar{m}$ as a parent is equal to $f_{\bar{m}} / \sum_m f_m$. The second parent is, then chosen randomly from among the current generation according to a uniform probability distribution. This is a common selection method in various GA implementations. (Choosing both parents using probabilities proportional to fitness values can be problematic in the event that several similar solutions, with high fitness values, are present in the current generation. In this case, these similar solutions may be frequently chosen as parents and create multiple children similar to themselves. Choosing the second parent by a uniform distribution increases the diversity in the population.)

Once the "children" are created and possible mutations occur, it may be necessary to adjust the solutions for feasibility. This "genetic repair" is accomplished in the same manner described above for initial random solutions.

Stopping rule. In order to provide a standard basis of comparison for computation times and results across various problems and parameter settings, we employed a fixed number of generations as a stopping rule for our computational experiments. Details are given in Section 7.

5.2. Routing demands

The heuristic routes demands d_{ij} one at a time. After each assignment, the heuristic updates the remaining available capacity on each ring. It also updates the list of feasible routes for each unassigned demand d_{ij} that, if used, would not violate capacity constraints (5). The heuristic first assigns demands with a single feasible route. It then gives priority to demands that can feasibly be routed on a single ring. A description of the heuristic is as follows:

Demand routing heuristic

Step 0. *Initialize possible routes*: A given string or chromosome specifies values for the x_{ik} variables, which indicate which nodes are served by each ring. Based on these node-ring assignments, determine, for each demand d_{ij}, the set of all feasible routes.

Step 1. *Assign demands with one feasible route*: If demand d_{ij} has a single feasible route with node i served by ring k and node j served by ring l, then set $y_{ijkl} = 1$. (Note that it is possible that $k = l$.) These represent "forced" assignments.

Step 2. *Update feasible routes*: Update, for each unassigned demand d_{ij}, the sets of feasible routes that, if used, would not violate capacity constraints (5). If some d_{ij} now has only a single feasible route, return to Step 1.

Step 3. *Assign single-ring routes*: Consider unassigned demands with feasible single-ring routes. If none available, go to Step 5. Otherwise, among those with the fewest number of feasible routes, choose the one with largest d_{ij} value and assign it to a single feasible ring k. (That is, set $y_{ijkk} = 1$.) If there is more than one feasible single-ring assignment, choose the ring k having the greatest remaining capacity.

Step 4. *Update*: Update, for each unassigned demand d_{ij}, the set of feasible routes. If some d_{ij} now has a single feasible route, return to Step 1, otherwise, return to Step 3.

Step 5. *Assign two-ring routes*: Consider unassigned demands that have feasible routes. If none exists, go to Step 7. Otherwise, among those with the fewest number of feasible routes, choose the one with the largest d_{ij} and route it on a feasible set of rings k and l. (That is, set $y_{ijkl} = 1$.) If more than one set of rings (k, l) is feasible, choose the set that would minimize the maximum demand assigned to either ring k or ring l.

Step 6. *Update*: Update, for each unassigned demand d_{ij}, the set of feasible routes. Go to Step 5.

Step 7. *Make infeasible assignments if necessary*: If no unassigned demands remain, STOP; the solution is specified. Otherwise, the unassigned demands d_{ij} have to be routed as best as possible. They are sorted in decreasing order of d_{ij} and processed one-by-one, selecting the route that minimizes the infeasibility incurred.

A problem related to SRDP considers a situation in which each demand d_{ij} could, in fact, be split among multiple routes; see Laguna (1994). The resulting routing problem (SRDP with specified values for the variables x_{ik}) would then be a linear program. However, since the routing problem needs to be solved for each proposed solution (e.g., 100 times per generation), the computation time, using a standard linear programming solver, may become excessive. Alternatively, the heuristic above can be readily modified to allow for demand splits among multiple routes.

We also note that the Demand Routing Heuristic can be utilized independently. For example, after rings are installed, the demand pattern may change somewhat over time. This heuristic can then be used to route the new demands, given the configuration of the rings.

5.3. Heuristics for seed solutions

As indicated in Section 5.1, there are two types of seed solutions. In the first type, each node is served by a single stacked ring. The heuristic to generate the seed solutions of this type is based on a special case of the algorithm described by Klincewicz, Luss, and Yan (1998). Specifically, it modifies that algorithm to create a greedy randomized adaptive search procedure (GRASP). In GRASP, items to be processed are chosen at random from a list of candidates. Because of this random element, the heuristic can be repeated several times, yielding different seed solutions. For an overview of GRASP heuristics, see Feo and Resende (1995).

The procedure attempts to create R subsets of nodes by incrementally combining various groups of nodes. Specifically, it attempts to consolidate a group containing node i with a group containing node j, if there is a large d_{ij} value. Note that since each given node will only be in a single group, that group must handle all traffic that originates or terminates at that node. In this way, we can determine when a group would violate the traffic constraint.

Generating seed solutions where each node is on a single ring

Step 1. *Initialize*: Order demands d_{ij} in decreasing order.

Step 2. *Process each demand in turn*: The "candidate list" is a list containing the largest unprocessed demands. Let q equal the remaining number of unprocessed demands and let s be a user-specified parameter that indicates the maximum length of the candidate list. The candidate list thus has length equal to $\min(s, q)$. Choose a demand d_{ij} at random from the candidate list and process it. There are three cases:

- *Case a*: *Neither node in the pair currently belongs to a group.* In this case, open a new group consisting of i and j.
- *Case b*: *One node belongs to a group; the other does not.* Say i belongs to a group. Node j is added to the group, provided that the new group does not violate the node constraint (3) or traffic constraint (5). Otherwise, j is put in a group by itself.
- *Case c*: *Each node in the pair belongs to a different group.* Combine the two groups into one, provided that the new group does not violate the node constraint (3) or traffic constraint (5). Otherwise, the groups are unchanged.

Repeat Step 2 until no unprocessed demands remain.

Step 3. *Combine groups as necessary*: If more than R groups remain, go through all possible pairings of the existing groups in arbitrary order. If a particular pairing is feasible, combine the two groups into a single group. Repeat until either no further feasible combinations are possible, or there are R groups. If, after exhausting feasible combinations, there are still more than R groups, it is necessary to make infeasible combinations. Specifically, take the two smallest groups (in terms of number of nodes) and combine them into a single group; repeat this as often as necessary. Note that this may result in a solution that violates the traffic constraint (5). The fitness function would then include a penalty cost component. Solutions that violate node constraint (3) would undergo "genetic repair", as described above for randomly-generated solutions.

The other type of seed solution assigns each node to enough rings so that there is no exchange of traffic necessary at the hubs. Again, a GRASP procedure is used so that multiple seed solutions can be generated. Our GRASP heuristic is based on bin-packing. There are R bins, corresponding to R stacked rings. Each bin starts with capacity c. As demands d_{ij} are added to the bin, capacity is used up. Nodes, corresponding to the demands that are added to each bin, are likewise associated with the bin. The heuristic can be stated as follows.

Generating seed solutions with no traffic exchange

Step 1. *Initialize*: Order all demands d_{ij} in decreasing order. Initially, all demands are unassigned. Also, keep a list of demands that cannot be assigned feasibly; initially, this list is empty.

Step 2. *Choose demands*: The "candidate list" is a list containing the largest unprocessed demands. Let q equal the remaining number of unprocessed demands, and s' be a user-specified parameter that indicates the maximum length of a candidate list. The candidate list has length equal to min (q, s'). If the candidate list is empty, go to Step 6. Otherwise, choose a pair (i, j) at random from the candidate list to be processed, and go to Step 3.

Step 3. *Process demand*: Find the bin k with the most utilized traffic capacity, such that including d_{ij} would not violate capacity constraint (5) and such that including nodes i and j with nodes already associated with bin k would not violate the node constraint (3). If there is no such feasible bin, add this demand to the list of demands that cannot be assigned feasibly and go to Step 2. Otherwise, add the demand d_{ij} to bin k. If either i or j, or both, are not already associated with bin k, then associate them with bin k. Go to Step 4.

Step 4. *Continue to fill chosen bin*: Create a candidate list, consisting of those largest remaining demands, such that including the demand in bin k would not require adding additional nodes, and such that including the demand would not violate the capacity constraint (5). Let q' be the number of such demands. Then the candidate list has length equal to min (q', s'). Randomly choose a demand from the candidate list and include it in bin k. Repeat until it is not feasible to continue. If there are already b nodes in bin k, go to Step 2. Otherwise, go to Step 5.

Step 5. *Add a node to chosen bin*: In a manner similar to Step 4, create a candidate list of demands that can be feasibly included in bin k by adding just one node. If this list is empty, go to Step 2. Otherwise, randomly choose a demand from the candidate list. Include it, and its required node, in bin k and return to Step 4.

Step 6. *Assign infeasible demands*: If the list of demands that cannot be feasibly assigned is empty, then STOP. Otherwise, sort this list in decreasing order. Process each demand, in turn, from this list. That is, assign each demand to the least full bin to which the demand can be assigned, without violating node constraint (3). If there is no such bin, pick a bin at random and assign the demand there (thereby violating constraint (3) in the process). STOP when all demands have been processed. Solutions that violate constraint (3) would then undergo "genetic repair", as described above for randomly-generated solutions.

6. Illustrative example

To illustrate the problem, we present a 10-node, 2-ring example, with realistic demand data. The demands to be routed are listed at the top of figure 3. We assume node 10 serves as the hub, and that available capacity on each ring is 48 units. This, for example, would correspond to a standard OC-48 ring. In this case, we assume demand is measured in OC-1 units.

Two solutions are presented. In Solution 1, every demand is routed on a single ring, i.e., no exchange of demand between rings is permitted. (This is the type of seed solution generated by the second GRASP heuristic described above.) In this solution, there are 46 units of demand routed on Ring 1. In addition to the ADM at the hub, there are 5 additional

Node Pair	Demand	Solution 1 Ring Used	Solution 2 Ring Used
1-2	2	1	1
1-4	12	1	1
1-7	2	2	1,2
1-8	1	1	1
1-10	10	1	1
2-3	2	2	2
2-5	1	1	1
2-8	13	1	1
2-9	3	2	2
3-4	11	2	2
4-5	7	1	1
4-9	2	2	2
6-7	2	2	2
7-9	21	2	2
7-10	4	2	2

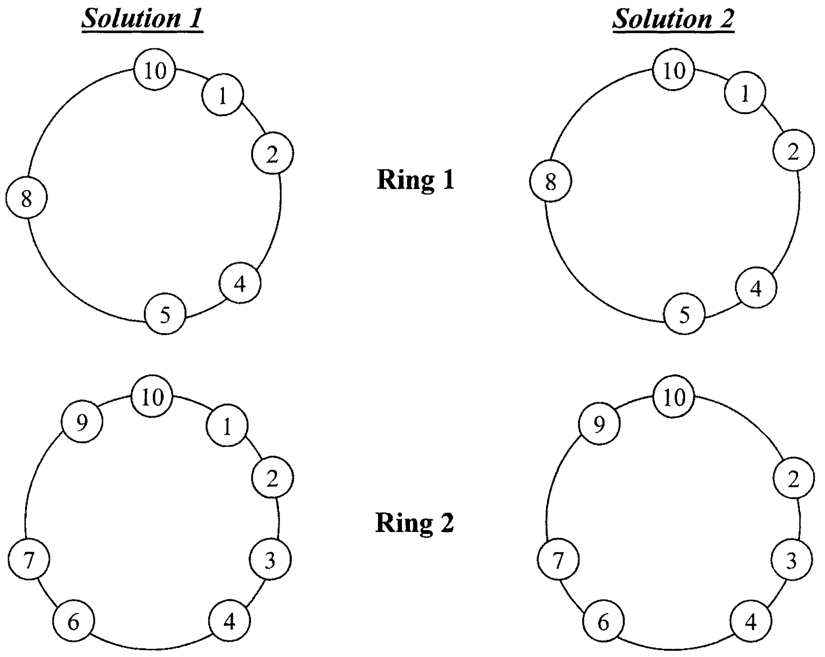


Figure 3. Illustrative example.

ADMs on Ring 1. There are, likewise, 47 units of demand routed on Ring 2. In addition to the ADM at the hub, there are 7 additional ADMs on Ring 2. Thus, there are 12 ADMs in total, excluding those at the hub. In addition to the hub, there are 3 nodes (nodes 1, 2 and 4) appearing on both rings.

By contrast, Solution 2 allows traffic to be exchanged at the hub. In particular, 2 units of demand between nodes 1 and 7 are now exchanged at the hub and routed on both rings.

Because of this, node 1 no longer requires an ADM on Ring 2. Thus, in Solution 2, Ring 1, with 5 ADMs (excluding the hub ADM), carries 48 units of demand and Ring 2, with 6 ADMs (excluding the hub ADM), still carries 47 units of demand. Only 11 ADMs, excluding those at the hub, are required in total, with only 2 non-hub nodes (nodes 1 and 2) appearing on both rings. If we assume a realistic cost ratio of $h/t = 50$, then we see that allowing traffic to be exchanged at the hub can reduce the total cost.

We note that 2-ring solutions in which every non-hub node appears on only one ring would not be feasible for this example. (Such solutions are generated by the first GRASP heuristic.) Because of the amount of traffic that would have to be exchanged at the hub in this case, three rings would be required in order to have a feasible amount of ring capacity.

7. Computational results

The GA-SRDP algorithm described in Section 5 was implemented in a C language program and tested in a UNIX® environment on a Sun Sparc Server 690MP, using the Sun OS 4.1.4 operating system.

Comparisons were made between GA-SRDP and the CPLEX™ mixed integer programming software package, developed by CPLEX Optimization, Inc. The CPLEX package was used to solve the integer programming (IP) formulation of the problem given in Section 3, as well as a mixed integer programming formulation (MIP) in which the y_{ijkl} variables were allowed to take on fractional values. We note an interesting implementation issue. Given an optimal solution, an alternate optimum can be obtained by simply permuting the order of the rings. Multiple optima such as this can increase the computing time required by a branch-and-bound algorithm, such as the one contained in CPLEX. To differentiate among these permuted solutions, a perturbation term $k\varepsilon_1$ can be added to the ADM costs of each ring k. This will induce the ring with the most nodes to be designated as ring 1, etc. To further distinguish between rings having the same number of nodes, an additional perturbation term $i\varepsilon_2$ can be added to the ADM cost at node i. Thus, the above cost h per ADM can be replaced by $h + k\varepsilon_1 + i\varepsilon_2$ for an ADM at node i on ring k, where $0 < \varepsilon_2 < \varepsilon_1 \ll h, t$.

Table 1 summarizes the comparisons among the GA, IP and MIP solutions. For the experiments reported in this table, the following parameters were used in GA-SRDP: Population size of 100 chromosomes, number of generations equal to 100, an elite number of 5, a total of 6 seed solutions—3 of each type, "candidate list" size $s = 3$ and $s' = 3$ for the GRASP heuristics, and a penalty cost equal to the ADM cost.

The problems have either 10 or 15 nodes (including the hub) and either 2 or 3 rings. For each pair of nodes, an integer demand value was randomly generated between 0 and U, where U is the maximum possible demand. For $N = 10$ and $R = 3$, we set $U = 12$; for $N = 15$ and $R = 3$, we set $U = 6$; and for $N = 15$ and $R = 2$, we set $U = 4$. These numbers were chosen to yield problems in which the minimum capacity utilization would be in the range of 0.6–0.7.

Even for these relatively small problems, considerable computation time was required by CPLEX. For the first seven problems, the MIP solution was identical to the IP solution, but the MIP required much less CPU time than the IP. For the first problem and the sixth problem, the IP and MIP required much less CPU time than for other problems of the same

Table 1. Comparisons of solutions among the IP, MIP and Genetic Algorithms (capacity $c = 150$, ADM cost $h = 50$, inter-ring routing cost $t = 1$).

Input parameters				Solution values			CPU times (sec.)			GA solution characteristics			
Nodes N	Rings R	Max. nodes (exc. Hub) b	Total demand $\sum_{i,j} d_{ij}$	IP	MIP	GA	IP	MIP	GA	No. of ADMs (exc. hubs)	Inter-ring traffic	Capacity utilization	Generation found
10	3	5	250	572	572	574	547	397	30	9	124	83	54
10	3	5	304	659	659	674	3713	3032	37	11	124	95	92
10	3	5	294	660	660	660	7579	4226	38	11	110	90	78
10	3	5	295	626	626	635	2419	1130	36	10	135	96	43
15	2	12	222	820	820	868	2767	342	77	16	68	97	26
15	2	12	200	771	771	774	340	79	75	14	74	91	75
15	2	12	250	895	895	897	>5 h	510	73	17	47	99	44
15	2	12	281	1017	1016	1018	>5 h	462	107	20	17	99	57
15	3	8	320	–	1010	975	–	>5 h	95	17	125	99	92
15	3	8	328	–	1105	1053	–	>5 h	91	19	103	96	95
15	3	8	386	–	1256	1160	–	>5 h	99	22	60	99	64
15	3	8	391	–	1254	1196	–	>5 h	113	23	46	97	71

size. For these problems, we note that the optimal solution assigned each non-hub node to just a single stacked ring. For problems with N = 15 and $R = 3$, even the MIP was not able to terminate after 5 hours of CPU time. The MIP solution values displayed for these problems represent the best MIP solutions encountered up to this point. Because of the time required for the MIP, the IP was not run for these problems. An attempt to improve these run times by incorporating simple valid cuts led to only negligible improvements.

The GA-SRDP heuristic gave excellent results for the problems in Table 1. For the first eight problems, GA-SRDP averaged within about 1.3% of the optimal IP solution, generated by CPLEX, in a small fraction of the CPU time required by the IP and MIP software. If one were to exclude the fifth problem, in which GA-SRDP was 5.8% from the optimum, GA-SRDP averaged within 0.7% of the optimal IP solution. For the problems with $N = 15$ and $R = 3$, the GA-SRDP heuristic found solutions in an average of about 100 seconds that were approximately 5% better than the best solution encountered by the MIP in 5 hours.

The cost ratio of $h/t = 50$ led to high capacity utilization on the stacked rings. That is, a considerable amount of traffic is routed on two rings in order to reduce the number of required ADMs. All except the first problem (where total demand is relatively low and each non-hub node has only one ADM) have a capacity utilization in excess of 90%. The solutions to the last two problems in Table 1 also contain rings in which the constraint (3) on the number of nodes per ring is tight, representing highly constrained problems.

Table 2 reports results of experiments in which GA-SRDP was used to solve larger-size problems with up to 50 nodes and 7 stacked rings. Demands for these problems were generated randomly. For each possible origin-destination pair, a zero demand was generated

Table 2. Genetic Algorithm solutions (capacity $c = 250$, max nodes per ring $b = 20$, costs $h = 50$, $t = 1$).

Input parameters				Solutions		Solution characteristics			
Nodes N	Rings R	Total demand $\sum_{i,j} d_{ij}$	No. of positive d_{ij}'s	Solution value	CPU time (sec)	No. of ADMs (exc. hubs)	Inter-ring traffic	Capacity utilization	Generation found
20	3	472	139	1320	163	21	225	93	88
30	3	529	140	1946	208	33	201	97	82
40	3	551	158	2443	273	50	187	98	95
50	3	538	152	2992	333	51	196	98	96
20	5	769	190	1742	401	26	442	97	90
30	5	850	204	2632	610	45	382	99	92
40	5	865	204	3501	743	63	351	97	95
50	5	886	208	4321	933	80	321	97	75
20	7[a]	960	190	2133	718	34	433	93	93
30	7	1265	244	3647	1142	66	347	92	90
40	7	1342	267	4651	1470	85	401	99	79
50	7	1285	238	5841	1706	109	391	96	99

[a]Only 6 rings are used.

with probability p; otherwise, an integer demand between 1 and U was chosen using a uniform distribution. For 3-ring problems, we set $U = 5$; for 5-ring problems, we set $U = 7$; and, for 7-ring problems, we set $U = 9$. A value of p was chosen for each problem so that the expected number of positive demands for 3-ring problems would be 150, for 5-ring problems would be 200, and for 7-ring problems would be 250. (The exceptions were 20-node problems with 5 or 7 rings, in which $p = 0$ and all 190 origin-destination pairs had positive demand.) These parameter settings again yielded problems in which the minimum capacity utilization would be in the range of 0.6–0.7.

The computation times indicated in Table 2 for these problems are still reasonable. These times range from about 3 to 6 minutes for the 3-ring problems, 6 to 16 minutes for the 5-ring problems, and 12 to 28 minutes for the 7-ring problems. The required CPU time depends on the number of nodes N, the number of stacked rings R and the number of positive demands. Note, in particular, that the demand routing heuristic in Section 5.2 depends on the number of positive demands and, indirectly, on the number of the stacked rings, since an increased number of rings would often tend to increase the number of possible routings for a given demand. The routing heuristic must be executed for each solution generated and represents the major part of the computational effort. Hence, if all possible d_{ij} were positive ($p = 0$), the computation time would increase quadratically with N and may become excessively large for large N.

In our examples, for a given R, the number of positive demands is about the same. Thus, the computation time increases at a somewhat less than linear rate with N. For example, for $R = 5$, the CPU time for $N = 30$ is 610 seconds and, for $N = 50$, it is 933 seconds, an increase of 53%. For fixed N, the CPU time increases at a faster than linear rate with R. For example, for $N = 20$, the CPU time for $R = 5$ is 401 seconds and, for $R = 7$, is 718 seconds, an increase of 79% (the number of positive demands is 190 in both examples). In other examples where R was increased, the increase in CPU time was even greater, since the number of positive demands increased as well.

The solutions in Table 2, like those in Table 1, exhibit a high capacity utilization (all in excess of 90%). This indicates that the solutions include inter-ring traffic routings as much as possible, and thereby tend to minimize the number of ADMs used. Note that the solution for $N = 20$ and $R = 7$ included only 6 rings that actually had non-hub nodes connected to them. This is in keeping with the pattern of high capacity utilization solutions. In general, however, since the problem formulation assumes a fixed number of rings R, and the GA-SRDP heuristic is based on this assumption, it is best, if one seeks to determine an "optimal" number of stacked rings, to run the algorithm for various values of R. Such a parametric analysis can then take into account tradeoffs associated with variable numbers of rings, such as ADM costs at the hub node, or other operational costs per stacked ring.

Note, in Table 2, that the best solutions tended to be found in "later" generations. (For 8 out of 12 problems, the solution was found in generation 90 or later.) In Table 3, we summarize the results obtained when the problems with $R = 5$ were run for additional generations. In this table, solution values were normalized, with the best solution found in 300 generations having the value 100. As can be seen, solutions can be improved by increasing the number of generations, although the amount of improvement tapers off. Between generations 50 and 100, solutions improved an average of 7%. Between generations

Table 3. Genetic Algorithm solutions as a function of number of generations (capacity $c = 250$, max nodes per ring $b = 20$, costs $h = 50, t = 1$).

Input parameters				Normalized solutions for 50–300 generations					
Nodes N	Rings R	Total demand $\sum_{i,j} d_{ij}$	No. of positive d_{ij}'s	50	100	150	200	250	300
20	5	769	190	113	105	103	100	100	100
30	5	850	204	114	106	105	103	102	100
40	5	865	204	118	109	105	103	100	100
50	5	886	208	111	108	105	102	102	100

Table 4. Comparison of Genetic Algorithm solutions and seed solutions (capacity $c = 250$, max nodes per ring $b = 20$, costs $h = 50, t = 1$).

Input parameters				Seed solutions		GA solutions		
Nodes N	Rings R	Total demand $\sum_{i,j} d_{ij}$	No. of positive d_{ij}'s	Solution value	CPU time (sec)	Solution value	CPU time (sec)	% Improvement
20	3	472	139	1650	<1	1320	163	20.0
30	3	529	140	2850	<1	1946	208	31.7
40	3	551	158	3143	1	2443	273	22.3
50	3	538	152	5091	1	2992	333	41.2

100 and 200, solutions improved an average of 5%. However, between generations 200 and 300, the average improvement was only 2%. Computation times are roughly proportional to the number of generations, and can be extrapolated, based on the values in Table 2.

As described above, the initial population includes feasible seed solutions generated by GRASP heuristics. These heuristics, as described in Section 5.3, create solutions either (i) in which only a single ADM per non-hub node is allowed, or, else, (ii) in which no traffic exchange occurs. To illustrate the value of GA, as well as the value of allowing both traffic exchange at the hub and multiple ADMs per node, we present in Table 4 a comparison of the best GRASP seed solutions and the final GA solution. (In these examples, the best GRASP seed solution was always of the second type, i.e., in which no traffic exchange occurs at the hub.) We consider, in particular, in Table 4, the 3-ring problems from Table 2. For each of these, we list the best GRASP seed solution and the required CPU time to generate them (one second or less), the final solution generated by GA and the required CPU time, and the percentage improvement of the GA solution over the seed solutions. As can be seen, the percentage improvement is quite large, ranging from 20% to over 40%, with an average of 28.8%.

As with other GA heuristics, it is necessary for the user to set various algorithmic parameters, such as number of chromosomes, number of generations, number of seeds and number of elite solutions. Some experimentation is needed for a particular set or class of problems to

Table 5. Genetic Algorithm solutions for different ADM cost values ($N = 20$, $b = 20$, $\sum_{i,j} d_{ij} = 769$, 190 positive d_{ij}'s, $t = 1$).

Input parameters			Solutions		Solution characteristics				
ADM cost h	Capacity c	Rings R	Solution value	CPU time (sec.)	No of ADMs (exc. hubs)	Inter-ring traffic	Rings used	Utilization	Generation found
1	250	5	44	1066	44	0	4	77	98
10	250	5	409	1019	39	19	4	79	129
25	250	5	1024	787	33	199	4	97	89
50	250	5	1664	781	24	464	5	99	177
100	250	5	2951	755	25	451	5	98	199
150	250	5	4219	932	25	469	5	99	186
1	300	5	40	1037	40	0	3	85	26
10	300	5	380	918	38	0	3	85	116
25	300	5	1022	619	19	547	5	88	92
25	300	4	896	683	32	96	3	96	184
25	300	3	904	700	32	104	3	97	188
50	300	5	1499	633	19	549	5	88	0
100	300	5	2448	588	19	548	5	88	56

determine parameters that will yield robust solutions, and an appropriate tradeoff between average solution quality and computation time.

In Table 5, we present results for problems with different cost ratios h/t and different ring capacities c, where the ADM cost h is changed and the inter-ring traffic cost t is kept fixed at $t = 1$. Also, we have set the number of generations at 200. As noted above, an ADM cost of $h = 50$, for the problems in Tables 1 and 2, yielded solutions with high capacity utilization. We expect this pattern of solutions to change as h and c change. For example, as h decreases, we would expect more ADMs in the solution and, hence, less inter-ring traffic and lower capacity utilization. Likewise, for fixed h, making the capacity constraint less stringent by setting a larger c should result in lower cost.

These expected patterns are borne out in Table 5. The same set of $N = 20$ nodes and the same demand data are used throughout. Costs h are varied from 1 through 150, and c values of 250 and 300 are used. Consider the examples with $c = 250$. When $h = 1$ (relatively negligible ADM cost), the number of ADMs is sufficiently large to have zero inter-ring traffic. At the other extreme, when $h \geq 50$, the number of ADMs is reduced as much as possible, resulting in ring utilizations of over 98%. Due to the ring capacity limit of $c = 250$, it is not possible to reduce the number of ADMs further. However, when $c = 300$ and $h \geq 50$, the number of ADMs is reduced to 19, i.e., a single ADM for each non-hub node.

As discussed above, parametric analysis of R can be important. For $h = 25$ and $c = 300$, values of $R = 5, 4$ and 3 were tested. With $R = 4$ and $R = 3$, a significant improvement in the solution was found. If, in addition, one were to consider the costs of ADMs at the hub, then the solution with $R = 3$ ($904 + 3 \times 25 = 979$) is preferred to the solution with

$R = 4$ ($896 + 4 \times 25 = 996$). GA-SRDP can readily be modified to include, in the fitness function, ADM costs at the hub node for each ring that actually carries traffic. Indeed, the modified algorithm found, for the example above with $R = 5$, a good solution that uses only 3 rings. However, the modified algorithm did not perform that well in all examples. In general, the GA-SRDP algorithm presented in this paper is designed to find good solutions for a given value of R, rather than optimizing over R. Hence, parametric analysis of R is important. The example above with $R = 5$ illustrates the case where the selected R is too large. However, the selected R may also be too small. Although not obvious, the minimum required ADMs may be smaller when R is larger, even when the hub ADMs are counted. We demonstrate this through a simple example. Let $N = 5$, $R = 3$, $c = 3$, $d_{ij} = 1$ for all $i < j < N$ and $d_{iN} = 0$ for all i. It can be readily shown that the minimum number of non-hub ADMs that can carry all the demands is 6 (e.g., $x_{11} = x_{22} = x_{33} = x_{41} = x_{42} = x_{43} = 1$ and all other $x_{ik} = 0$). The corresponding routing variables can easily be determined. However, when $R = 4$, the smallest number of non-hub ADMs is reduced to 4 (e.g., $x_{11} = x_{22} = x_{33} = x_{44} = 1$ and all other $x_{ik} = 0$), resulting in 2 fewer non-hub ADMs.

As with any heuristic, suboptimal solutions can be found. For example, consider the solutions generated for $c = 250$ and $h = 100$ and 150. If one takes the same ADM placements and routings as in the solution obtained for $c = 250$ and $h = 50$ (24 ADMs instead of 25 ADMs) and adjusts the ADM cost to $h = 100$, one obtains a value of 2864 (an improvement of 2.9%, compared to the value 2951 when $h = 100$ was used in the algorithm). Likewise if the hub cost is adjusted to $h = 150$, the solution value is 4064 (an improvement of 3.7%, compared to the value 4219 when $h = 150$ was used in the algorithm). In general, since for large cost ratios h/t the ring utilization should be close to 100%, the heuristic may not always find the smallest number of ADMs that achieves this utilization. Although the solutions above were found in "later" generations (177 or larger), solving these examples with 300 generations provided only negligible improvements (0.1% or less) in the solution values.

8. Final remarks

In this paper, we presented a Stacked Ring Design Problem (SRDP) and proposed a genetic algorithm (GA-SRDP) to solve this problem. The solution structure of SRDP can be encoded in a vector of fixed length, where each element is a zero-one variable, thus making a genetic algorithm an attractive solution approach. Indeed, the computational results confirm this assessment. Our solution approach also demonstrates how various heuristics can be successfully combined within a genetic algorithm framework; in this case, we used GRASP heuristics for seed solutions as well as a specialized heuristic for demand routing.

This paper assumes a single hub. This implies, however, that, if the hub fails, traffic cannot be exchanged between SHRs. Therefore, in some applications, it may be desirable to utilize two hubs. The algorithmic framework that is presented here for SRDP can be adapted, with only moderate effort, to the two-hub case. It would require, in particular, some modification to the Demand Routing Heuristic.

As mentioned in Section 2, an important extension to SRDP, open for further research, is the consideration of bi-directional rings. In terms of the mathematical formulation, this implies adding to the routing variables two additional indices to represent clockwise and counterclockwise routing in the relevant rings. In GA-SRDP, this implies using a more complex demand routing heuristic that uses load balancing in order to utilize the available capacity most effectively. However, as this heuristic is used to evaluate all generated solutions, the computation time may become prohibitively large.

Other problems of interest that are open for further research include problems with additional constraints, such as a limit on the number of rings to which a node may be assigned. More general architectures, such as networks in which there are more than one topological ring, also need to be addressed. The latter problem extends the model in Klincewicz, Luss, and Yan (1998), which considers multiple topological rings connected at a hub node, where each non-hub node has only a single ADM. A genetic-based approach may provide one promising approach for such problems.

References

Cosares, S., D.N. Deutsch, I. Saniee, and O.J. Wasem. (1995). "SONET Toolkit: A Decision Support System for Designing Robust and Cost-Effective Fiber-Optic Networks," *Interfaces* 25, 20–40.

Cosares, S. and I. Saniee. (1994). "An Optimatization Problem Related to Balancing Loads on SONET Rings," *Telecommunication Systems* 3, 165–181.

Cox, Jr., L.A., L. Davis, L. Lu, D. Orvosh, X. Sun, and D. Sirovica. (1996). "Reducing Costs of Backhaul Networks for PCS Companies Using Genetic Algorithms," *Journal of Heuristics* 2, 201–216.

Davis, L., L.A. Cox, Jr., W. Kuehner, L. Lu, and D. Orvosh. (1997). "Dynamic Hierarchical Packing of Wireless Switches Using a Seed, Repair and Replace Genetic Algorithm," *Journal of Heuristics* 3, 187–206.

Dowsland, K.A. (1996). "Genetic Algorithms—A Tool for OR?," *Journal of the Operational Research Society* 47, 550–561.

Feo, T.A. and M.G.C. Resende. (1995). "Greedy Random Adaptive Search Procedures," *Journal of Global Optimization* 6, 109–133.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley.

Harder, J. (1996). "SONET Ring—Design with Cutset Inequalities and Other Speedups: Some Computational Experience," In *Proceedings of the Fourth International Conference on Telecommunications Systems*, Nashville, TN.

Harshavardhana, P., P.K. Johri, and R. Nagarajan. (1996). "A Note on Weight-Based Load Balancing on SONET Rings," *Telecommunications Systems* 6, 237–239.

Klincewicz, J.G., H. Luss, and D.C. Yan. (1998). "Designing Tributary Networks with Multiple Ring Families," *Computers and Operations Research* 25, 1145–1157.

Laguna, M. (1994). "Clustering for the Design of SONET Rings in Interoffice Telecommunications," *Management Science* 40, 1533–1541.

Luss, H., M.B. Rosenwein, and R.T. Wong. (1998). "Topology Network Design for SONET Ring Architecture," *IEEE Transactions on Systems, Man and Cybernetics* 28, 780–790.

Myung, Y.-S., H.-G. Kim, and D.-W. Tcha. (1997). "Optimal Load Balancing on SONET Bidirectional Rings," *Operations Research* 45, 148–152.

Reeves, R. (1993). *Modern Heuristic Techniques for Combinational Problems*. Oxford: Blackwell.

Schrijver, A., P.D. Seymour, and P. Winkler. (1998). "The Ring Loading Problem," *SIAM Journal of Discrete Mathematics* 11, 1–14.

Wasem, O.J., T.-H. Wu, and R.H. Cardwell. (1994). "Survivable SONET Networks—Design Methodology," *IEEE Journal on Selected Areas in Communications* 12, 205–212.

Wu, T.-H. (1992). *Fiber Networks Service Survivability*. Boston, MA: Artech House.

Journal of Heuristics, 6, 107–130 (2000)
© 2000 Kluwer Academic Publishers

A Zoom-In Approach to Design SDH Mesh Restorable Networks

MARIO PICKAVET* AND PIET DEMEESTER
University of Gent-IMEC, Department of Information Technology, Sint-Pietersnieuwstraat 41, 9000 Gent (Belgium)
email: mario.pickavet@intec.rug.ac.be

Abstract

Mesh restorable networks based on SONET (Synchronous Optical Network, standard optical transmission technology widely accepted and implemented in North America) or SDH (Synchronous Digital Hierarchy, the European standard currently adopted by the major European telecom operators) are an economically attractive solution in areas where high demand and high connectivity are involved (Wu, 1995). In these networks, the reconfiguration capability of the digital cross connect systems (DCS) allows to reroute the demand affected by network failures. The degree of sharing of spare capacity in networks based on this architecture is high.

This paper presents a heuristic algorithm for solving the near-optimal design of SDH mesh-type link restorable networks, i.e. determining the network topology and assigning the capacity to transport the demand in normal situations and to allow full link restorability in case of single link failures. The algorithm is based on a Zoom-In technique, a novel approach which forms a compromise between sequential and integrated techniques. The different building blocks of the algorithm are tested extensively and compared with other results mentioned in literature. Comparison of the simulation results for the overall design problem with other solution techniques indicates that the Zoom-In method is a quite promising approach, able to combine the accuracy of integrated approaches with the calculation speed of sequential approaches.

Key Words: capacity design, Genetic Algorithm, link restoration, topology design, Zoom-In technique

1. Introduction

The problem studied in this paper is the design of SDH mesh restorable networks.[1] Given is a set of node locations and a VC-4 node-to-node demand (1 VC-4 corresponds to a bit rate of 155 Mbit/s). Asked is to determine between which nodes a link must be installed (topology design), how to route the demand (routing design) and which capacity should be assigned to each link (capacity design). The installed capacity must be sufficient to carry the demand, not only when all network components are functioning properly, but also in case of a single link failure. The mechanism used to reroute the demand affected by a link failure is link restoration: the affected connections are rerouted between the endpoints of the failing link (see (Wu, 1992; Sato, 1996)).

Installing a link with a certain VC-4 capacity between two nodes comprises the installation of a cable between those nodes and a sufficient number of fiber systems. A fiber system

*Research Assistant of the Fund for Scientific Research—Flanders (F.W.O.-V, Belgium).

consists of a line system and eventually regenerators at regular distances along the cable to upgrade the attenuated and distorted signal. Accordingly, three cost components were considered (including equipment cost as well as installation cost): the cable cost, the line system cost and the regenerator cost. The cable cost is roughly proportional to the distance between the nodes. With respect to the cost of the line systems and regenerators, we considered STM-1 (155 Mbit/s), STM-4 (622 Mbit/s) and STM-16 (2.5 Gbit/s) fiber systems. Economies of scale play a role in the selection of line systems. For instance, an STM-4 offers a capacity which is four times as high as that of an STM-1 at less than double the cost (a similar relationship holds between STM-16 and STM-4). As a result, when the VC-4 demand offered to the network is relatively high, one may expect a predominant use of STM-16 line systems, STM-4's and STM-1's only being installed where STM-16's would be severely underused. The cost of the regenerators depends on the type of used fiber system, but also shows a stepwise dependence on the length of the cable.

To measure the performance of a potential solution method for the complex network design problem described above, several aspects must be taken into account. The most important element is, of course, the quality of the found solutions (i.e., a low cost). However, also other features, that are related to the available computing resources such as the calculation time and the memory requirements, turn out to be indispensable to make a correct comparison between different algorithms. Despite the off-line nature of a network design process, the calculation speed is important because it restricts the size of problem instances that can be solved within a "reasonable" time. Also the memory requirements of an algorithm might pose a restriction on the size of problems that can be solved in practice. Depending on the particular algorithm and the available resources, either the calculation time or the memory usage can be the most limiting factor.

A number of related network design problems has been considered in literature. On one hand, the spare capacity assignment problem in mesh restorable networks has been treated by many authors, see for instance Grover, Bilodeau, and Venables (1991), Herzberg (1993), Herzberg, Bye, and Utano (1995), and Poppe and Demeester (1998). Starting from a given network topology and a given routing in the failureless state, these papers describe algorithms to allocate spare capacity for different failure types and restoration mechanisms. On the other hand, our problem is also related to the design of multi-commodity survivable networks, studied for instance in Minoux (1981), Gavish et al. (1989), and Stoer and Dahl (1994). These papers assume that there is no relationship between the routing of the demand in the failureless state and the routing of the demand in case of a failure. The formulations they studied impose the feasibility of a multi-commodity flow in each of the network states (the failureless state and each of the failure states), which is valid under the assumption that there is no restriction on the way the network may be reconfigured after a failure has occurred. This rerouting strategy contrasts with the link restoration situation, where a clear relationship is noticed between the routing in the failureless state and the routing in each of the failure states: the traffic which is not affected by the (link) failure cannot be rerouted and the affected traffic can only be rerouted between the endpoints of the failing link.

This paper presents a novel algorithm that combines the topology design and capacity assignment in case of a strict relationship between the routing in the failureless state and the failure states, based on the link restoration paradigm. In Poppe and Demeester (1997) a similar problem is described and an Integer Linear Programming based algorithm is presented. Further on, we will compare the results of the two approaches whenever possible.

The paper is structured as follows. The two main building blocks of the algorithm are a type of Genetic Algorithm which solves a more abstract capacitated survivable network design problem and an algorithm to allocate the spare capacity, described in Sections 2 and 3, respectively. The overall algorithm (Section 4) is based on a Zoom-In strategy, a technique that forms a compromise between a sequential approach and an integrated approach. In Section 5 some numerical examples are described and attention is paid to the relative importance of the different phases of the algorithm. The major building blocks of the algorithm are tested in Section 6 and the main assumptions on which the Zoom-In algorithm is based are highlighted. The quality of the overall solution is compared with the results of other techniques. Section 7 concludes the paper.

2. Abstract Capacitated Survivable Network Design (ACSND)

One of the modules used in the overall algorithm (described in Section 4) is an algorithm to solve a more abstract network design problem ACSND. The original problem stated in Section 1 is therefore slightly adapted to speed up the algorithm.

Firstly, a linearized cost model is used. This leads to a representation of the cost of a network as a sum of link costs that are bilinear functions of the length and the capacity of the link:

$$c(N) = \sum_{e \in E(N)} [c_1 \cdot l(e) + c_2 \cdot u(e) + c_3 \cdot l(e) \cdot u(e)]$$

where N is a network, $c(N)$ is the cost of the network N, $E(N)$ is the set of links of the network N, $l(e)$ is the length of the link e, $u(e)$ is the capacity provided on link e, c_1, c_2 and c_3 represent cable, line system and regenerator cost, respectively.

Secondly, a more abstract routing strategy is used, which does only indirectly incorporate the spare capacity needed for link restoration. The link capacities are calculated by rescaling the demand matrix (e.g. multiplying all node-to-node demands by a factor $\eta = 1.2$) and dividing the traffic along the two shortest link-disjoint paths. The philosophy behind this bi-routing strategy is that in case of a single link failure at least a fraction $\eta/2$ will not be affected, while the remaining fraction $1 - \eta/2$ can be (partly or entirely) restored using some spare capacity available in the network if $\eta > 1$.

These assumptions lead us to a straightforward optimal capacity assignment procedure for a fixed topology:[2] using a minimum cost flow algorithm (Ahuja, 1993) each demand d is routed through the network with link costs $c_2 + c_3 \cdot l(e)$ (these are the only cost components which depend on the capacity assignment and hence on the routing of the traffic) and link capacities $d/2$ (to ensure bi-routing along link-disjoint paths). This allows a fast evaluation of the total network cost $c(N)$ for a certain topology, so that we can concentrate on the choice of a good topology.

To solve the ACSND problem, a Genetic Algorithm enhanced with some deterministic optimization routines is used. We refer to Pickavet et al. (1997) for a detailed description of the algorithm. The main building blocks of the ACSND algorithm are shown in figure 1.

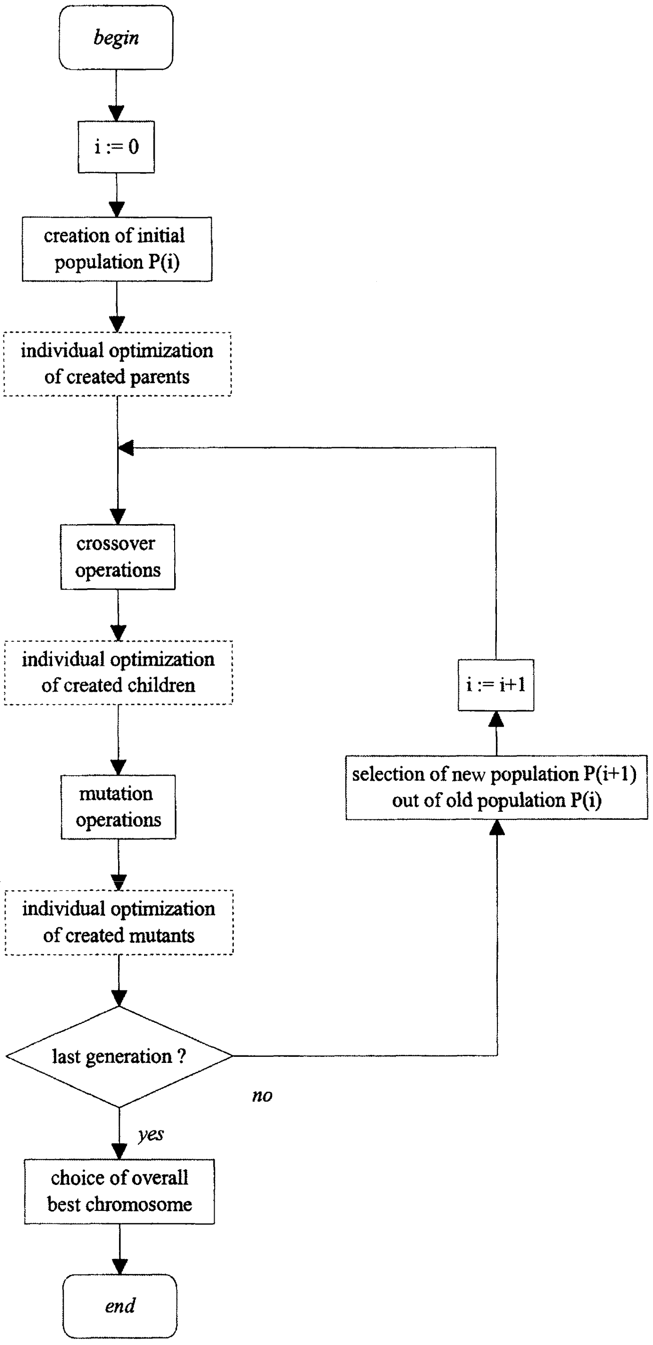

begin
i := 0
creation of initial population P(i)
individual optimization of created parents
crossover operations
individual optimization of created children
mutation operations
individual optimization of created mutants
last generation ?
no
yes
selection of new population P(i+1) out of old population P(i)
i := i+1
choice of overall best chromosome
end

Figure 1. Abstract capacitated survivable network design (ACSND) algorithm.

A Genetic Algorithm (Holland, 1975) is a general optimization technique which mimics the genetic evolution of species. The main difference with other A. I. approaches, as for instance Simulated Annealing or Tabu Search, is that a Genetic Algorithm deals with a 'population' of solutions (a solution is called a 'chromosome') rather than with a single solution. In case of the ACSND problem, a chromosome is in fact a network topology. The objective is to create topologies with a minimal overall network cost $c(N)$ (and with at least link-connectivity 2).

The network topologies in the initial population are created by randomly selecting some links. Some chromosomes are added to this population by applying some crossover and mutation operations. A crossover operation combines two existing chromosomes to create a new (and hopefully better) one, whereas a mutation operation creates a new chromosome by modifying an existing one. Out of this extended population, consisting of so-called parents, children and mutants, a new population is selected according to the 'survival of the fittest' principle: high quality chromosomes (i.e., topologies with at least link-connectivity 2 and a low network cost $c(N)$) have a higher probability of being chosen for the next generation. This sequence of crossover operations, mutation operations and selection of a new generation is repeated until no better solutions are found during a number of generations. Out of all the generations, the best solution is retained as the final solution of the ACSND problem.

It is important to recognize the building blocks (Holland, 1975) of a high quality solution, in order to define a crossover operation that combines the good characteristics of the parents and removes the bad characteristics. We finally opted for a sort of cut-and-paste operation on the network topologies: we replace a 'bad' part[3] of one topology by the corresponding 'good' part of another topology. This is illustrated on a 8-node example problem in figure 2. The 'weakness' of the basic parent network is the single link between nodes v_4 and v_5 leading to a low reliability: if that link fails, the network will be divided in two not-connected parts. By replacing a part of the network around that weakness (all the links that are incident to the nodes v_4 and/or v_5, dashed lines in basic parent network) by the corresponding part of another network (all the links that are incident to the nodes v_4 and/or v_5, full lines in auxiliary parent network), we create a child network which is hopefully more reliable than both of its parents. To decide which crossovers will take place in a population, we first check which parent chromosomes are complementary, i.e., the bad parts of the two topologies should be non-overlapping. From these candidate pairs of complementary parents, a fixed number of pairs is randomly selected, for each pair a basic parent network is randomly assigned and one child chromosome is created per pair.

When defining a suitable mutation operation, it is important to introduce some drastic adaptations to the parent network that allow the Genetic Algorithm to escape from any local minima. The quality of the mutant network is hereby only of secondary importance. We opted for a type of rotation operation. First, the centre of gravity of all nodes is calculated and the nodes of the network are ordered (in a circular way) according to their polar angle from the centre of gravity. The links in the mutant network are then obtained by rotating the node numbering over one position and redrawing the parent network links between the same node numbers in the mutant network. This is illustrated in figure 3 on a 8-node problem.

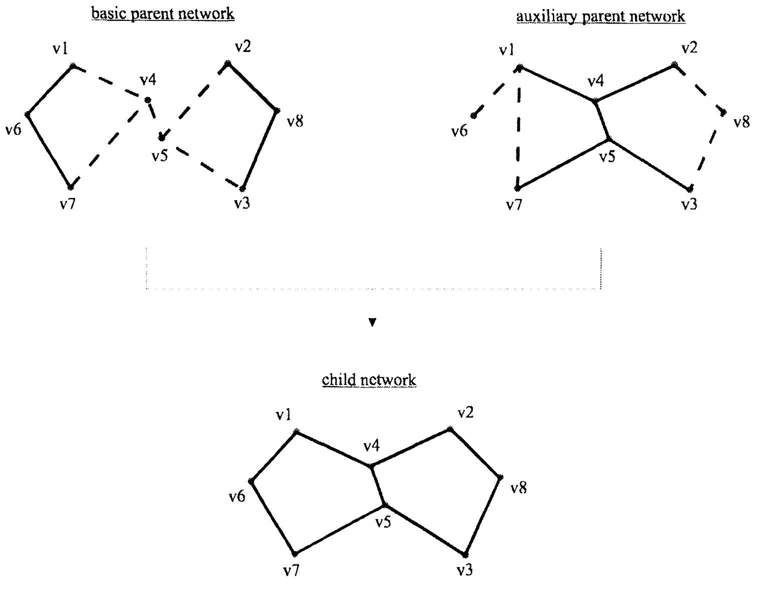


Figure 2. Example of crossover operation.

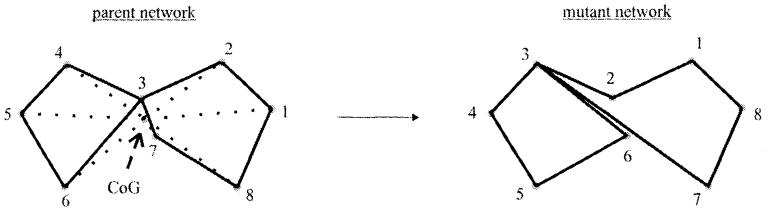


Figure 3. Example of mutation operation.

The parent networks that will be used for a mutation operation are randomly chosen. Usually the mutant chromosomes in a population form only a small fraction of the total population.

The Genetic Algorithm described above enables us to explore the ACSND search space in a thorough and efficient way, but lacks some strong directives towards link-connectivity 2 of the created topologies. For instance, the link-connectivity of children networks created with the crossover operation is often lower than 2. Therefore, in contrast with the standard Genetic Algorithm paradigm, some deterministic routines were introduced to optimize the chromosomes individually (see figure 1). The most important optimization routines are:

1. On one hand, to ensure a link-connectivity of at least 2, each node of the network should have at least degree 2. Hence, links will be added to interconnect nearby nodes with

Table 1. Typical values for the Genetic Algorithm parameters.

Number of parents per population	20–40
Number of children per population	20–40
Number of mutants per population	5–10
Number of generations	5–20

a low degree. On the other hand very high node degrees usually lead to high network costs, so (long) links between two nodes with a high node degree may be removed.

2. Topologies containing traversing links—i.e. links that are crossing each other—usually lead to a higher network cost than topologies without traversing links, without providing extra connectivity. Hence, two traversing links are omitted and replaced by non-traversing links (if these links are not present yet).
3. If the line system costs (c_2) are high when compared to the cable cost (c_1), introducing additional links in a network with link-connectivity 2 may decrease the overall network cost (due to shorter routing paths).
4. After the calculation of the link-connectivity matrix of the topology, links may be added between nodes with a mutual connectivity below 2. On the other hand, long links that interconnect nodes with a very high mutual connectivity may be removed to reduce the network cost. If needed, this process of evaluating the connectivity of the network and adding/removing some links can be repeated.

The routines 1, 2 and 3 are rather rough heuristics to correct some major shortcomings of a chromosome quickly, while routine 4 is a refined method using more CPU time. The influence of these 4 routines on the quality of the final solution was tested extensively and turned out to be quite significant. The importance of including problem specific methods in a Genetic Algorithm to increase the search efficiency has also been observed for a wide range of other optimization problems (Michalewicz, 1997).

Some typical values for the most important Genetic Algorithm parameters are shown in table 1. In general, the algorithm turns out to be very robust and not very sensitive to the exact parameter values, so fine tuning of the values is not very useful.

3. Spare Capacity Allocation (SCA)

The SCA algorithm starts from a fixed topology and a fixed working capacity assignment. The objective is to allocate enough spare capacity to allow full link restorability of single link failures at minimum cost (using one modularity m, e.g. $m = 16$ if we use STM-16 fiber systems). Moreover, to be suited for use in the Zoom-In approach (see Section 4), the algorithm should be able to provide rather good solutions quickly and very good solutions during a long run.

The algorithm consists of three phases: a minimal spare capacity allocation procedure, a rough spare capacity addition and a tightening phase (see figure 4).

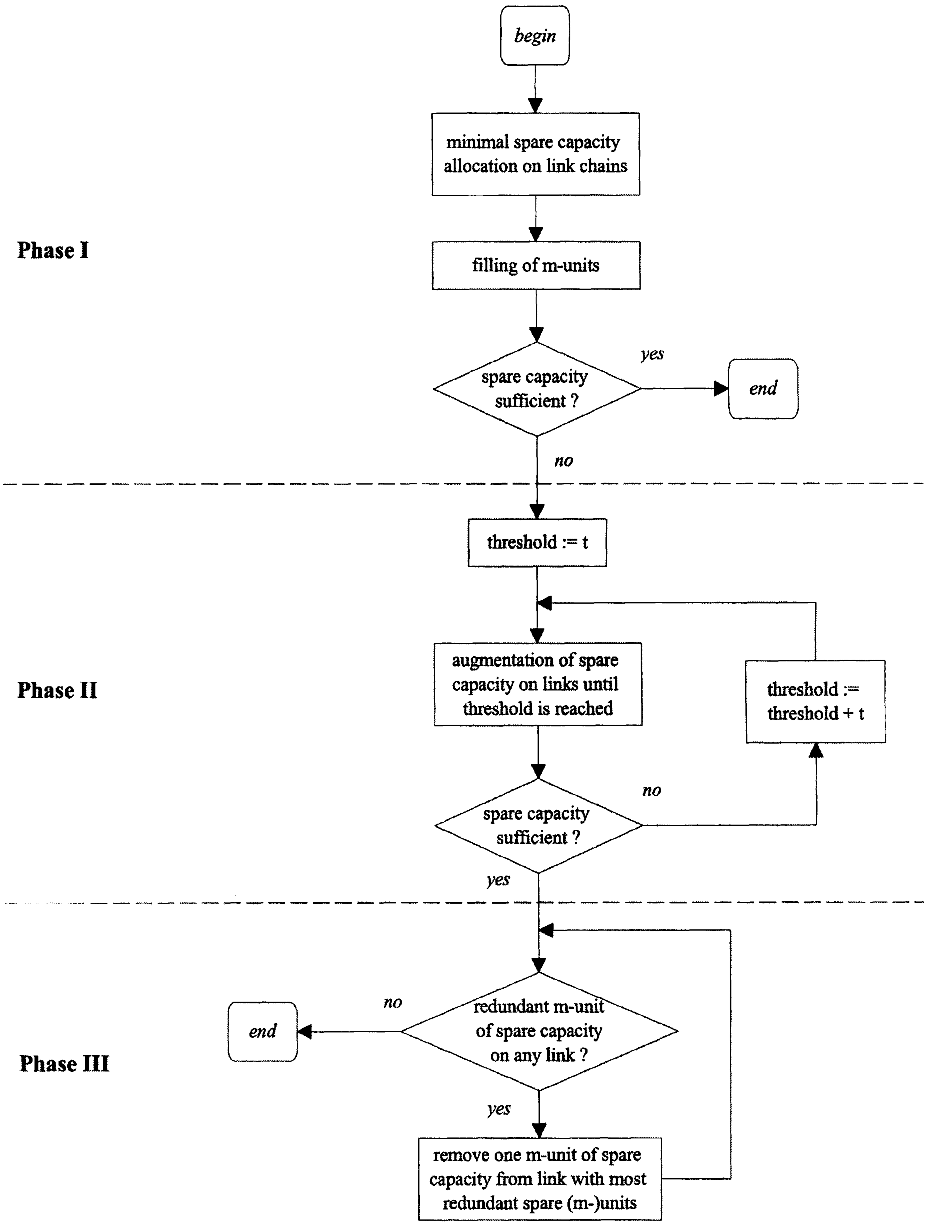

begin
minimal spare capacity allocation on link chains
Phase I
filling of m-units
spare capacity sufficient ?
yes
end
no
threshold := t
Phase II
augmentation of spare capacity on links until threshold is reached
threshold := threshold + t
spare capacity sufficient ?
no
yes
Phase III
redundant m-unit of spare capacity on any link ?
no
end
yes
remove one m-unit of spare capacity from link with most redundant spare (m-)units

Figure 4. Spare capacity allocation (SCA) algorithm.

3.1. *Phase I: minimal spare capacity allocation*

Starting from the working capacity assignment on each link, a simple and extremely fast method can be used to assign some initial spare capacity on the links. Indeed, if several links $l_1, l_2, \ldots, l_i$ form a chain,[4] the spare capacity $s(l_x)$, $1 \leq x \leq i$, on the link l_x must be at least as big as the working capacity $w(l_y)$ on any other link l_y of the chain (due to the principle of link restoration):

$$s(l_x) \geq \max_{1 \leq y \leq i, y \neq x} w(l_y) \quad 1 \leq x \leq i$$

Furthermore the spare capacity on each link l_x can be augmented without any additional cost as long as no additional line system is needed:

$$s(l_x) := m \left\lceil \frac{s(l_x) + w(l_x)}{m} \right\rceil - w(l_x)$$

After these spare capacity additions, we check whether all single link failures are fully link restorable (using a maximum flow algorithm (Goldberg, 1988)). If so, the found spare capacity allocation is optimal. If not, some spare capacity will be added in phase II.

3.2. *Phase II: spare capacity addition*

If the spare capacity allocation found in phase I turns out to be insufficient, we define an initial threshold t on the spare capacity $s(l_x)$ on each link l_x. If $s(l_x) \leq t$, we add one or more m-units[5] of spare capacity on each link until the threshold is reached:

$$s(l_x) := s(l_x) + m \cdot \max\left(\left\lceil \frac{t - s(l_x)}{m} \right\rceil, 0\right)$$

If not all single link failures are fully link restorable, we define new thresholds $2t, 3t, \ldots$ until full restorability is reached. Usually the initial threshold t is chosen rather large (e.g. 20% of the average working capacity on a link), resulting in a second phase which is quite fast (only a few threshold adaptations are needed) but inaccurate (inefficient usage of spare capacity).

3.3. *Phase III: spare capacity tightening*

This last and most time-consuming phase starts from the solution found in the second phase and tries to remove some m-units of spare capacity from the links. We repeatedly search for the link (or one of the links) with most redundant units of spare capacity, by changing the rerouting patterns in such a way that the considered link is avoided as much as possible, while honoring the spare capacity restrictions on the other links. We then remove one m-unit of spare capacity from the link with most redundant units. This procedure is repeated until no redundant m-units of spare capacity are left. The philosophy behind removing only one m-unit on a link at a time instead of removing as much m-units as possible is based on the intuitive notion that, by acting in a less greedy fashion, the final result will generally show a more uniform distribution of the spare resources throughout the network, leading to a more efficient sharing of spare resources.

To speed up the tightening phase, we allow a rough, approximate search for the link with most redundant spare units. For instance, in order to decide from which link an m-unit of spare capacity should be removed, we can compare the links with respect to the number of spare m-units or $m/2$-units on a link that are completely redundant (granularity m or $m/2$ instead of 1). This rough SCA algorithm is much faster and does usually not effect the solution quality significantly.[6]

Remark: The extension of the algorithm towards multiple facility types is straightforward. For instance, let us assume that the three facility types STM-16, STM-4 and STM-1 are considered and that the cost of an STM-16 is twice the cost of an STM-4 and four times the cost of an STM-1 (economy of scale principle). Then, since the STM-16 fiber systems are the most cost-effective option (lowest cost per capacity unit), we first execute the three phases of the SCA algorithm for $m = 16$, resulting in a capacity allocation with only STM-16 line systems. Then the tightening phase III is executed again, to replace some STM-16 systems by STM-4 and/or STM-1 systems, by adapting the rerouting pattern if necessary. Since the removal of an STM-16 by one STM-1 provides now the highest cost saving per removed capacity unit, we start by repeatedly searching for a link with 15 redundant units of capacity and replacing one of the STM-16s on this link by an STM-1, leading to a cost saving corresponding to 0.75 STM-16s. If no links with 15 redundant units are left, we repeatedly search for links with 12 redundant units (replacing an STM-16 by an STM-4, cost saving 0.5 STM-16s) and finally we check the links with redundancy 11 (replacing an STM-16 by an STM-4 and an STM-1, cost saving 0.25 STM-16s). It must be stressed that this multi-facility extension of the SCA algorithm is not a simple link-by-link optimization of the single-facility STM-16 solution, since an adaptation of the rerouting pattern is possible at each facility replacement.

4. Zoom-In algorithm

A sequential three-phase approach to handle the SDH mesh restorable network design problem (defined in Section 1) seems natural. One could start by designing a suitable link topology, connecting all the nodes while satisfying certain connectivity constraints to ensure some reliability (see for instance Monma and Shallcross (1989)). Once the topology is fixed, one could design the routing pattern and assign sufficient working capacity on each link. In the third phase, one starts from a given topology and working capacity assignment to calculate the needed spare capacity on the links to ensure full link restorability of single link failures (see for instance Grover, Bilodeau, and Venables (1991)). The advantage of such a sequential approach is its calculation speed: by dividing the overall design problem in three subproblems that are solved independently, the final solution is found relatively quickly (or, stated otherwise, we can design bigger SDH networks within a reasonable calculation time). However the interdependence of the three subproblems is neglected, e.g. the topology choice has an influence on the capacity costs and the routing of the working traffic has a certain impact on the spare capacity cost. As a consequence, the separation of the three subproblems might lead to low quality solutions (i.e., high costs).

To overcome this disadvantage, one could tackle the design problem as a whole: the relations between the three subproblems described above are fully taken into account. Such an integrated approach makes it the possible to create high quality solutions. However, the required resources, calculation time and memory requirements, are usually much higher than for the sequential approach. As a consequence, only smaller problem instances can be solved in practice. To design bigger networks simplifying assumptions or premature termination of the algorithm must be applied to reduce the calculation time and/or memory requirements. These measures however affect the solution quality.

The problems encountered with these two fundamentally different approaches give rise to a new approach which forms a compromise between the sequential and the integrated approach: a Zoom-In strategy. The 'zooming-in' philosophy is applied with respect to both the accuracy of the solution and the search effort. At the first stage(s) of the algorithm a rough solution is constructed, based on simplifying assumptions. As the algorithm proceeds, more accurate models are used and the search effort is increased. The final stage of the algorithm takes all problem details into account and performs a thorough and detailed search.

The Zoom-In strategy is also reflected in the evolution of the optimization techniques that are used in the algorithm. Global construction heuristics are used in the beginning, eventually combined with some random elements to get a more global view of the solution space. Afterwards, the emphasis shifts towards local improvement techniques where the nature of searching becomes more deterministic.

The Zoom-In approach tries to combine the advantages of both the sequential approach—its calculation speed—and the integrated approach—its high quality solutions. To achieve this, a careful choice has to be made of the problem details to be considered in each phase of the algorithm. This decision is typically based on a trade-off: considering an additional aspect of the problem improves the solution quality, but also leads to an augmentation of the required time and memory.

An additional advantage of the Zoom-In strategy is its flexibility. Depending on the time or memory available, some problem details can be ignored during one or more phases. On one hand, for small problem instances, we can incorporate a large number of problem details from the beginning of the algorithm, leading to high quality solutions. On the other hand, the Zoom-In approach is also useful for much bigger problem instances, since we can ignore some problem details in the first phases of the algorithm to limit the calculation time without losing too much on solution quality.

To apply the Zoom-In strategy to the SDH network design problem, the interactions between topology design, working capacity assignment and spare capacity assignment must be studied carefully. We finally opted for a four-phase algorithm (see table 2).

Table 2. Zoom-In approach applied to SDH network design problem.

	Function	Cost model	Capacity assignment
I	Creation of basic topology	Simplified	Approximation
II	Local optimization of topology	Simplified	Approximation
III	Local optimization of topology	Exact	Correct
IV	Refinement of capacity assignment	Exact	Correct, refined

The first and usually most important phase is a Genetic Algorithm which provides a good basic topology, based on a simplified cost model and an approximate capacity assignment procedure. The quality of this basic topology is checked and eventually improved in a second phase. The third phase considers the correct cost model and capacity assignment procedure. Again, the influence on the topology is checked. The working and spare capacity assignment is refined in the last phase, using a fixed topology.

4.1. Phase I: creation of basic topology

In a first phase, a starting topology is built and an estimate of the needed link capacities is made. The cost of the line systems is approximated by a function which is proportional to the capacity on the link—based on the STM-16 fiber system costs—and the regenerator cost is assumed to be proportional to both the capacity and the length of the link. Furthermore, several simulations have shown that applying a bi-routing strategy represents a rather good approximation for the exact spare capacity assignment procedure based on link restoration, as long as the total amount of needed capacity remains roughly the same (i.e., if the factor η, see Section 2, is properly chosen). A typical example is shown in figure 5.

Based on a randomly chosen topology, the factor η was set to match the total network costs for the bi-routing and link restoration case. For this fixed value of η, the total network

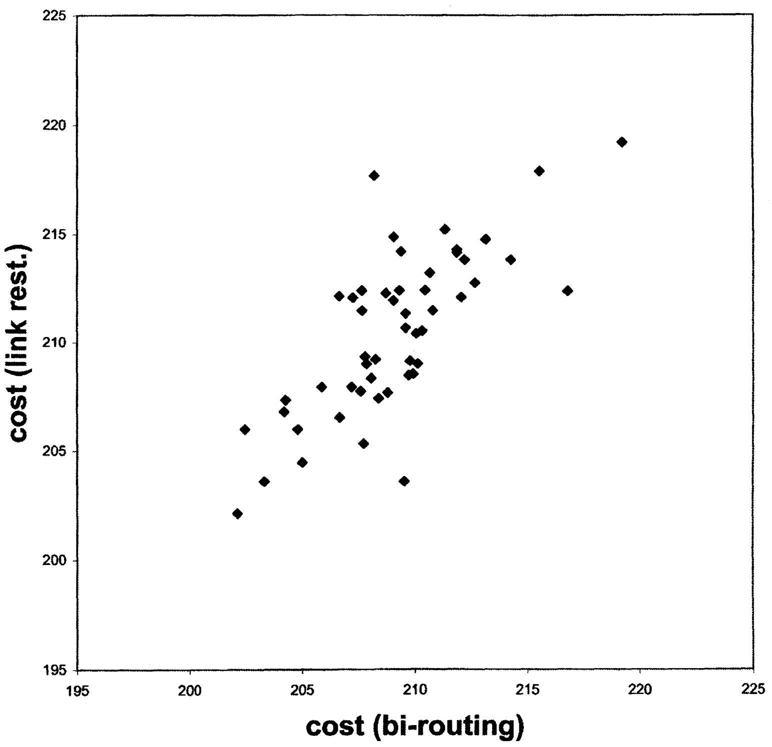


Figure 5. Network cost for a random set of topologies: bi-routing case vs. link restoration case.

costs were then compared for a number of topologies, obtained by randomly applying some minor modifications on the initial topology. Most scattered points in the figure are located close to the first bisector, which illustrates the high correlation of the network cost $c(N)$ between the bi-routing case and the link restoration case. This relationship allows us to reduce the calculation time by appling the bi-routing strategy in the first phase of the algorithm. The Genetic Algorithm described in Section 2 is used to solve the resulting ACSND problem.

4.2. *Phase II: local optimization of topology (optional)*

In a second phase, we still consider the ACSND problem. We try to improve the topology found in the first phase using local improvement techniques. Several possibilities were implemented: removing a link, adding a link or replacing a link by a new link. To restrict the number of possible link replacements several criteria could be used, e.g., only substituting a link l_x by a link l'_x if l_x and l'_x have a common node, introducing an upper bound on the length of the link l'_x, introducing an upper bound on the angle between the links l_x and l'_x.

The decision which local optimization techniques will be used is based on a trade-off between the desired quality of the final solution and the calculation time. This second phase may be omitted if the Genetic Algorithm already performed a thorough search through the solution space, which usually yields a solution that cannot be improved anymore by the above mentioned local optimization techniques. Especially if the relative importance of cable installation cost is high when compared with the line system cost, the Genetic Algorithm outperforms the local optimization technique by far.

4.3. *Phase III: local optimization of topology using more detailed description*

The third phase considers the correct cost model (as described in Section 1) which incorporates the discrete nature of capacity (STM-1, STM-4 and STM-16) and length (regenerators placed at regular distances). Furthermore, the exact capacity allocation procedure is applied: the working capacity suffices to route the demand through the network during normal operation and the spare capacity allows to reroute the traffic affected by a single link failure using a link restoration scheme. This phase is meant to examine the influence of the changes with respect to the cost model and capacity assignment on the topology. We again try to improve the topology found in the second phase by removing, adding or replacing a link.

Extremely long calculation times to evaluate the overall network cost for a certain topology are avoided by fixing the routing strategy for the demand (working capacity along shortest path) and by applying the rough SCA algorithm (see Subsection 3.3). Nevertheless, the evaluation time is usually much higher than for the bi-routing based capacity assignment procedure used in the first two phases. This indicates the benefit of running the second phase before starting the third phase: if necessary, some serious shortcomings on the topology can be solved quickly in the second phase and only the ‘finishing touch’ on the topology is performed by the slower third phase.

4.4. Phase IV: refinement of capacity assignment

This final phase leaves the topology unchanged and refines the working and spare capacity assignment, respectively.

For the working capacity assignment, we consider two possible paths for each node-to-node demand: the shortest path and the shortest path which is link-disjoint from this shortest path.[7] By dividing each demand between its two possible paths, we try to obtain a cheaper overall (working and spare) capacity allocation. The refinement of the working capacity is performed in two steps. First we try to reduce the overall cost by switching a node-to-node demand completely from its present path to its alternative path and recalculating the spare capacity based on this altered working capacity allocation, using the rough SCA algorithm. Since the switching of a node-to-node demand from its shortest path to its alternative path tends to be more effective if the alternative path is as short as possible, the node-to-node demands are enumerated according to increasing length difference between shortest and alternative path. This procedure is repeated until any switching of a node-to-node demand yields a solution which is at least as expensive as before: a local optimum is reached. In the second step, we examine what happens when we switch one unit from the present path to the alternative path or vice versa, until no improvements can be found anymore. Since no fractional division of demands is allowed, the working capacity refinement stops here. Note that this working capacity refinement is in fact also based on the Zoom-In strategy: we first examine some drastic changes on the working capacity and only if the local optimum is reached, the influence of small changes is checked.

Once the working capacity is fixed, we try to refine the spare capacity allocation by examining the influence of the granularity of the redundant spare units (see Subsection 3.3, refined tightening phase). The tightest spare capacity allocation is retained.

5. Numerical examples

The Zoom-In algorithm was tested on several quite different problem situations: we studied situations where no cables are present yet (so-called ‘green-field’ or ‘desert’ problem instances) as well as situations with some cables already installed and we varied the scale of the network (international, national and regional networks), the size of the network (8 to 50 nodes) and the density of the demand matrix. The calculation times ranged from a few seconds to about 24 hours,[8] depending mainly on the size of the network, but also on the number of commodities and the relative importance of cable cost versus line system cost. For all these problem instances, the memory requirements stayed largely below the available resources.

As an example, two 20-node green-field problem instances are discussed in detail below: a national and a regional network. The problems are identical with respect to the node locations and demand matrix (39 commodities, ranging from 1 to 20 VC-4’s), but the scale is different: the network diameters are 300 km and 30 km, respectively. The cable cost is 2 MBF/km. Three types of fiber systems were considered, the corresponding costs are mentioned in table 3. The regenerator interleaving distance is 50 km, independent from the fiber system.

Table 3. Cost of line systems and regenerators for different fiber systems.

	Line systems (MBF)	Regenerators (MBF)
STM-1	1.0	0.025
STM-4	1.8	0.050
STM-16	3.5	0.100

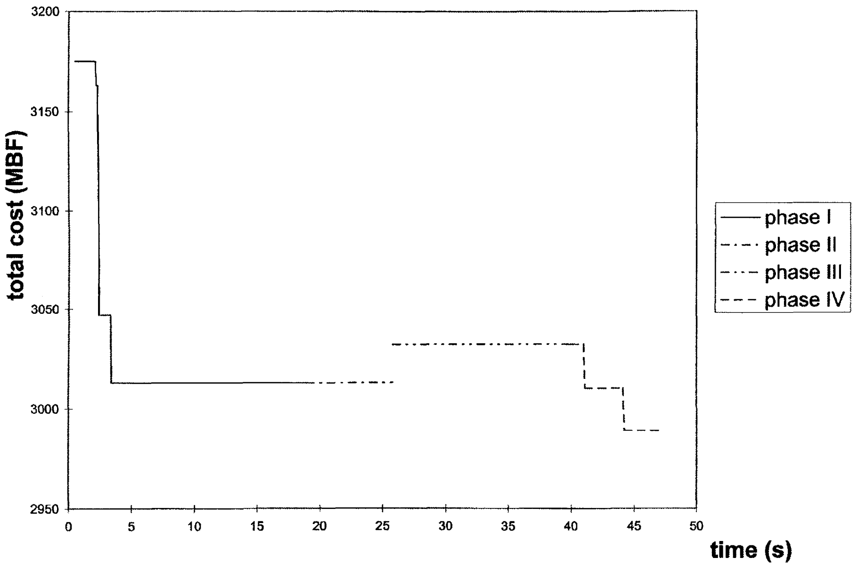


Figure 6. Evolution of the overall network cost (national problem instance).

5.1. *Case study I: national network*

The evolution of the overall cost of the temporary solution is shown in figure 6. The Genetic Algorithm (phase I) founds a first feasible solution with a cost of 3175 after 0.5 seconds. Gradually, better solutions are found until an excellent solution with a cost of 3013 is encountered after 3.4 seconds. No better solutions are found until the end of the Genetic Algorithm and the found solution turns out to be optimal when compared with ‘neighbor’ networks with one link more, one link less or one replaced link (phase II). After 25.8 seconds the first two phases of the Zoom-In algorithm are finished. From the third phase on, the correct cost model and capacity assignment procedure are introduced. This transition of cost model and capacity assignment procedure leads to a gap in the cost evolution from 3013 to 3032. Again, the algorithm searches for better neighboring networks (phase III), in vain. In the last phase, the capacity assignment is refined, leading to a first improvement by switching the working path of a node-to-node demand completely to its alternative path

Table 4. Components of national network.

Component	Number	Cost (MBF)
Cable	1200.4 km	2400.8
STM-1 line systems	1	1
STM-1 regenerators	1	0.025
STM-4 line systems	0	0
STM-4 regenerators	0	0
STM-16 line systems	165	577.5
STM-16 regenerators	109	10.9
Total		2990.225

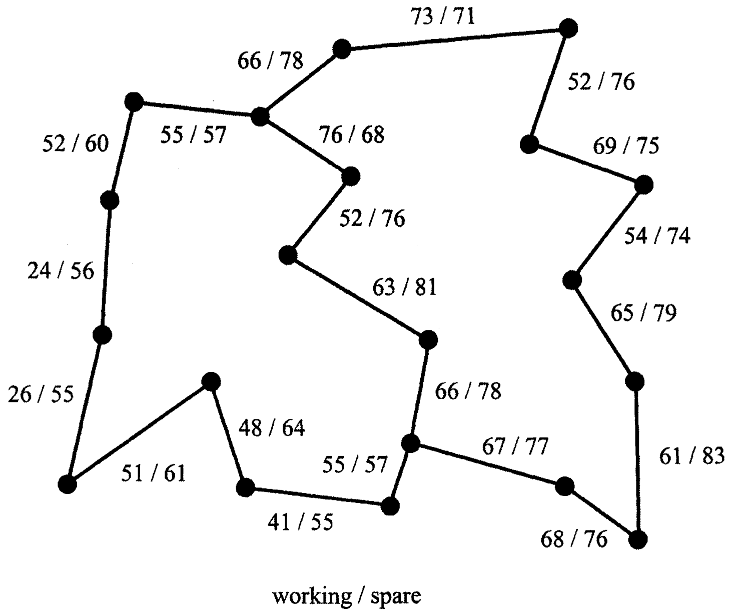


Figure 7. National network design (topology, working and spare capacity).

(cost 3010) and a second improvement is performed by switching 3 units from a working path to an alternative path (cost 2990). Varying the granularity of the redundant spare units in the SCA algorithm does not yield better solutions and the program terminates after 47.2 seconds. The final topology and capacity allocation are shown in figure 7, the different components of the overall cost are mentioned in table 4.

When examining the evolution of the overall network cost, we notice that the topology found by the Genetic Algorithm in phase I is not improved by the second and third phase.

This is caused by several reasons. Firstly, several experiments showed that the Genetic Algorithm performs best in situations where the relative importance of the cable cost versus the line system cost is high—as is the case here: ratio 80%/19%, see table 4—, leading to sparse networks. Furthermore, a local optimization technique based on adding, removing or replacing a link is more powerful in the opposite situation of dense networks.[9] This examination of the strengths and weaknesses of the first two phases explains the lack of improving moves in phase II. Since we are dealing with a cost dominated by the cable installation here, the adaptation of the capacity assignment procedure from a bi-routing strategy towards a working and spare capacity allocation based on link restoration has no big influence on the cost of a solution. As a consequence, the topology found in phase I turns out to be locally optimal with respect to the correct capacity assignment procedure too (phase III).

When considering the type of line systems used to build the network (see table 4), a clear predominance of STM-16 line systems is noticed. This is due to the high demand requirements on one hand and the economy of scale principle on the other hand: two STM-4 line systems (providing 8 VC-4 channels) are more expensive than 1 STM-16 line system (providing 16 VC-4 channels). As a consequence, the working paths and certainly the spare paths strive towards a distribution throughout the network that fills the available STM-16 fiber systems as good as possible and uses as few STM-4 and STM-1 line systems as possible.

The cost of the regenerators is only 0.4% of the total cost, hence the influence on the topology choice and capacity allocation is almost negligible. Regenerators are only important when international networks with high demands are considered (long distances combined with big capacity requirements).

5.2. *Case study II: regional network*

When tackling the 30 km diameter problem instance, the overall cost of the temporary solution changes as shown in figure 8.

The final topology and capacity allocation are shown in figure 9. Table 5 mentions the different components of the overall cost (no regenerators are needed: short distances).

Now the relative importance of cable cost versus line system cost is much smaller (due to the smaller distances): the final solution shows a ratio of 52%/48%. As a consequence the topology is much denser (see figure 9).

When we compare the evolution of the algorithm for the regional network (figure 8) with the national network (figure 6), some striking differences are noticed. Firstly, the total calculation time is much bigger for the regional network: a problem instance with about a fifty-fifty ratio of cable cost versus line system cost is usually much harder to solve than a problem instance dominated by the cable cost. Secondly, the relative importance of the four phases has changed. Whereas the second and the third phase were in fact useless for the national problem instance, these phases clearly show their use for this example. Indeed, local optimization techniques based on adding, removing or replacing a link become more powerful when we are dealing with dense networks.

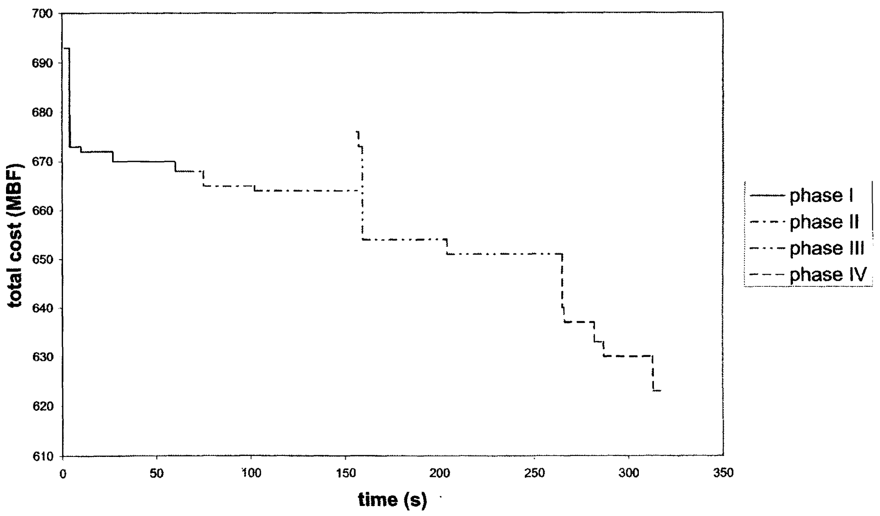


Figure 8. Evolution of the overall network cost (regional problem instance).

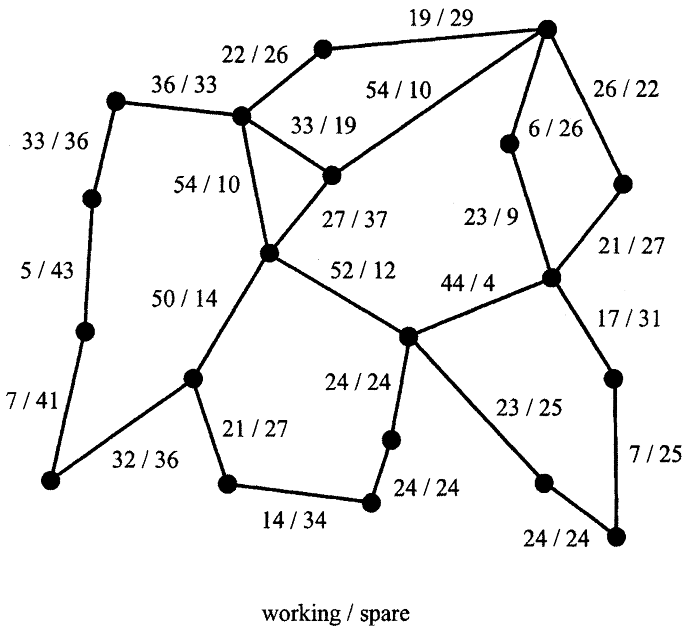


Figure 9. Regional network design (topology, working and spare capacity).

Table 5. Components of regional network.

Component	Number	Cost (MBF)
Cable	161.5 km	323.0
STM-1 line systems	2	2
STM-4 line systems	4	7.2
STM-16 line systems	83	290.5
Total		622.7

6. Comparison with other techniques

In this section the results of the main building blocks of the algorithm and the overall Zoom-In algorithm are compared with some results found in literature.

6.1. ACSND problem

One possible way to trace the performance of a Genetic Algorithm is to compare the obtained results with optimal solutions found with Integer Linear Programming techniques (see for instance also Karunanithi and Carpenter (1997) for such a comparison). However, due to the intrinsic complexity of the ACSND problem, only for some special cases (e.g. the Traveling Salesman Problem) we were able to find guaranteed optimal solutions using an integer programming code based on the work done in Stoer (1992) and Grötschel, Monma, and Stoer (1995). In these simple cases, the Genetic Algorithm reached the global optimum rather quickly. In more general cases an optimality check was not possible. However, an examination of the evolution of chromosomes in the Genetic Algorithm shows that the best solution is not found 'by accident': a lot of chromosomes, mostly obtained after several crossover operations, are nearly as good as the best solution.[10] Furthermore, the strength of the Genetic Algorithm for sparse networks, eventually combined with the strength of the local optimization search in phase II for dense networks, gives rise to an algorithm which is suited for a broad spectrum of ACSND problems.

6.2. Spare capacity assignment

The SCA algorithm allocates the spare capacity in a network with link modularity (e.g. 16 for STM-16 fiber systems) to allow full link restorability of single link failures. This problem was also tackled in Sakauchi, Nishimura, and Hasegawa (1990) and Grover, Bilodeau, and Venables (1991), describing a LP approach and a heuristic algorithm SLP for the near-optimal assignment of spare capacity, respectively. These papers mention a problem instance with 11 nodes and 23 links based on OC-12 link modularity. The solution based on a LP approach uses 164 OC-12 fiber systems (see figures 3 and 4 in (Sakauchi, 1990)), the SLP algorithm yields a slightly better solution with 163 fiber systems (see figure 3b in

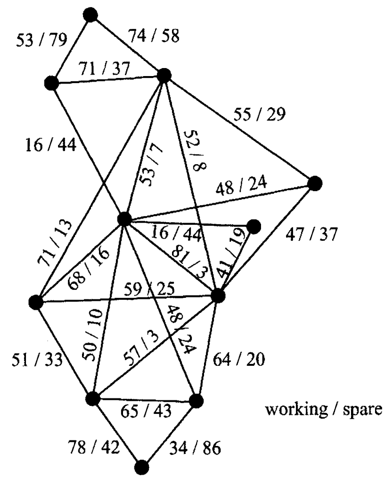


Figure 10. Spare capacity allocation (mod. 12) by SCA algorithm.

(Grover, 1991)). We tested the refined[11] SCA algorithm on this problem instance, yielding a capacity assignment that uses 163 OC-12 fiber systems (see figure 10). This solution is capacity-equivalent but not identical to the SLP solution and slightly better than the LP based solution, using very few calculation time resources (1.1 seconds).

6.3. *Working and spare capacity assignment*

The overall capacity assignment procedure, as described in Subsection 4.4, is based on the assumption that considering only two disjoint paths for the working traffic suffices to create a cost-effective capacity distribution. Stated otherwise, considering a much higher degree of bifurcation of the working traffic (i.e. considering a higher number of possible working paths) usually won't lead to a big additional reduction of the overall capacity cost. In fact, the philosophy behind this approach is the flexible nature of spare capacity, which allows to make efficient use of the available capacity due to the imperfect filling of the last m-unit on a link by the working capacity and moreover allows to compensate for local accumulations of working capacity.

To verify this assumption, some experiments were carried out on a set of random problem instances. We compared the results with an ILP method which constructs the optimal solution of a related problem: the cheapest capacity allocation is constructed which allows a routing of the traffic under normal conditions and provides enough spare capacity to ensure full link restorability of single link failures, when *fractional* flow variables are allowed (Poppe, 1997). The set of possible solutions of the latter problem includes all possible solutions of the SDH capacity problem—which requires integer flow variables—, hence the solution found with the ILP technique provides a lower bound for the SDH problem. The average gap between the solutions found with the method of Subsection 4.4 and the ILP solutions was 1.1%. The ILP solutions usually contained a lot of non-integer

Table 6. Comparison of (working + spare) capacity assignment with ILP lower bounds.

	Lower bound (ILP)	Found solution (phase IV)	Gap (%)
Case study I	166	166	0.0
Case study II	86	87	1.2

flow variables, so the actual gap between the found solution and the optimal solution might even be smaller. From these experiments, we can conclude that considering only two disjoint paths for the working traffic usually has little or no influence on the solution quality.

As an example, the number of STM-16 line systems obtained in the (slightly adapted)[12] case studies I and II (see Section 5) were also compared with the lower bounds obtained by the ILP method, see table 6.

6.4. *Overall problem: design of SDH mesh restorable network*

The Zoom-In approach can be described as an integrated approach with some adaptations to avoid excessive calculation time. The algorithm described in Section 4 is in fact based on two main assumptions:

1. The exact type of capacity assignment (based on bi-routing in phases I and II, exact working and spare assignment in phase III and IV) has no big influence on the choice of the topology or at least, the drawbacks of a topology choice based on an approximate capacity assignment can be eliminated almost completely by local improvement of the topology in phase III.
2. The fast capacity allocation procedure used in phase III (working capacity along shortest path, rough SCA algorithm to assign spare capacity) is sufficiently accurate to measure the influence of the capacity on the topology choice. As a consequence, the refined capacity allocation procedure in phase IV can be carried out on a fixed topology.

To give an impression of the impact of these assumptions on the calculation time, we estimated the time needed to design a SDH mesh restorable network for case study I (see Subsection 5.1) when all capacity allocation refinements were considered from the beginning of the algorithm: 2300 seconds instead of 47 seconds! It is clear that this integrated approach does not leave us any hope of tackling large problem instances (e.g. 50 nodes) within a reasonable calculation time.

To check whether the assumptions mentioned above are justified, some experiments were carried out to get an indication of the quality of the overall solution. Due to the intrinsic complexity of the overall problem, verifying the optimality of the solutions of the Zoom-In algorithm is only possible for small problem instances. The algorithm was tested on a problem instance with 8 nodes, 13 candidate links and 13 commodities, taken from Gavish et al (1989). This problem was solved to optimality in Poppe and Demeester (1997), using

Table 7. Comparison of solution quality of ILP algorithm and Zoom-In algorithm (same calculation times).

	ILP solution (MBF)	Zoom-In solution (MBF)	Gap (%)
Case study I	3038	2981	1.2
Case study II	649	633	2.5

a sophisticated branch-and-cut-and-price algorithm, based on an integrated ILP approach. The result obtained with the Zoom-In algorithm was 562.2 (calculation time: 9.1 seconds), the global optimum for this problem instance.

The Zoom-In algorithm and the branch-and-cut-and-price were also compared for a set of bigger random problem instances. Due to time and memory restrictions it was not possible to search the whole branching tree for these bigger problems: the ILP algorithm was ended after examining a number of branching nodes. To be able to compare the quality of the results, the calculation time of the Zoom-In algorithm was also restricted (by decreasing the number of possible moves in local optimization procedures) not to exceed the time needed by the ILP algorithm. On the average, the solutions found with the Zoom-In algorithm were 1.3% cheaper than the ILP solutions (standard deviation: 0.85%). The simulation results show the intrinsic advantage of a Zoom-In approach when compared to an integrated approach: by carefully balancing the impact of introducing a simplifying assumption on the solution quality on one hand and the calculation time on the other hand in each phase of the algorithm, a method can be developed which produces better results than an integrated approach within the same time. Furthermore, the memory requirements of the Zoom-In algorithm are very modest, even for large problem sizes (e.g., 50 nodes). In realistic situations, the calculation time will be the limiting factor.

As an example, the results for the two (slightly adapted)[13] case studies in Section 5 are compared in table 7.

7. Conclusions

An algorithm has been developed for the near-optimal design of SDH mesh restorable networks. The technique is based on several building blocks such as a type of Genetic Algorithm to design an initial topology, local optimization procedures to adapt the topology to a specific capacity assignment procedure and a SCA algorithm to assign spare capacity to ensure full link restorability of single link failures. To combine these building blocks into an overall method, a Zoom-In strategy is applied: the algorithm gradually moves from a rough solution based on an approximate problem description towards a refined solution taking all problem details into account. This novel approach, which forms a compromise between a sequential and an integrated method, aims at combining the advantages of both approaches.

The algorithm was tested extensively on several different problem instances and the results were compared with other techniques mentioned in literature. These experiments show that the Zoom-In algorithm is a quite promising technique which is able to deliver high quality solutions using modest time and memory requirements. Furthermore, the used approach provides a flexible framework with respect to changing problem formulations: usually only some minor modifications are needed to tackle related problems. Ongoing work is studying the introduction of other restoration mechanisms and modeling more complex failure scenarios.

Acknowledgment

The authors like to thank Fabrice Poppe for providing Integer Linear Programming solutions to compare and test the performance of the Zoom-In algorithm. Part of this work has been supported by the Flemish Government through the IWT project ITA/950214/INTEC and by the European Commission through the ACTS project AC205/PANEL.

Notes

1. The described algorithm can also be used to design similar networks, e.g. based on the SONET technology.
2. With at least link-connectivity 2, to allow bi-routing along link-disjoint paths.
3. A bad part of a topology is a part that typically leads to solutions with link-connectivity <2 or a high network cost.
4. A chain $l_1 - l_2 -, \ldots, -l_i$ is an ordered list of links where l_x and l_{x+1} have a common node and the degree of this node is exactly 2, $\forall x \in \{1, 2, \ldots, i-1\}$.
5. An m-unit of capacity is a block of m capacity units (corresponding to one fiber system).
6. Another possibility to speed up the tightening phase is to add an intermediate phase between phase II and III, where we remove all m-units of spare capacity that are not at all used by the rerouting determined at the end of phase II (for any link failure). However, such an intermediate phase usually does not significantly reduce the spare capacity of phase II, since the considered rerouting pattern at the end of phase II is typically using up most of the spare capacity in the close neighborhood of the failing link and little or no spare capacity far from the failing link. This inefficient usage of the spare resources can not be corrected by the intermediate phase.
7. The requirement of the second path to be link-disjoint with respect to the first path is based on the intuitive notion that this choice allows a thorough redistribution of the working capacity allocation, if needed.
8. All calculation times mentioned in this paper are measured on a Pentium Pro 200 Mhz PC with 32Mbyte internal memory.
9. As an example, we also studied the situation where phase I was omitted: phase II starts from a topology without links and moves towards better networks using local optimization. This way, the first feasible result was obtained after 7.5 seconds (cost 4101) and the final result of phase II was 3047 after 130 seconds, i.e. 1.1% worse than the result based on phase I using up much more calculation time. Several tests on other random problem instances confirmed this behavior, clearly indicating that the Genetic Algorithm generally outperforms the local optimization technique by far in the case of sparse networks.
10. We refer to Pickavet et al. (1997) for a more detailed discussion of the solution quality obtained with the Genetic Algorithm.
11. I.e. in phase III of the SCA algorithm, we repeatedly remove one 12-unit of spare capacity from the link with most redundant spare *units*.
12. We considered only STM-16 fiber systems and the regenerator cost was neglected. As can be seen from the results in section 5 (Tables 3 and 4), these restrictions have no big influence on the final result.
13. Again, only STM-16 fiber systems were considered and the regenerator cost was neglected.

References

Ahuja, R.K., T.L. Magnanti, and J.L. Orlin. (1993). *Network Flows: Theory, Algorithms and Applications.* Englewood Cliffs, New Jersey: Prentice Hall.

Gavish, B., T. Trudeau, M. Dror, M. Gondreau, and L. Mason. (1989). "Fiberoptic Circuit Network Design Under Reliability Constraints," *IEEE Journal on Selected Areas in Communications* 7(8), 1181–1187.

Goldberg, A.V. and R.E. Tarjan. (1988). "A New Approach to the Maximum Flow Problem," *Journal of the ACM* 35, 921–940.

Grötschel, M., C.L. Monma, and M. Stoer. (1995). "Design of Survivable Networks." In M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser (eds.), *Handbooks in Operations Research and Management Science, Vol. 7*, Amsterdam: North-Holland, pp. 617–672.

Grover, W.D., T.D. Bilodeau, and B.D. Venables. (1991). "Near Optimal Spare Capacity Planning in a Mesh Restorable Network." In *Proc. of IEEE Globecom*, pp. 2007–2012.

Herzberg, M. (1993). "A Decomposition Approach to Assign Spare Channels in Self-Healing Networks." *Proc. of IEEE Globecom*, pp. 1601–1605.

Herzberg, M., S. Bye, and A. Utano. (1995). "The Hop-Limit Approach for Spare Capacity Assignment in Survivable Networks," *IEEE/ACM Transactions on Networking* 3(6), 775–784.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Karunanithi, N. and T. Carpenter. (1997). "SONET Ring Sizing with Genetic Algorithms," *International Journal on Computing and Operations Research* 24(6), 581–591.

Michalewicz, Z. (1997). *Genetic Algorithms + Data Structures = Evolution Programs*. London: Springer-Verlag.

Minoux, M. (1981). "Optimum Synthesis of a Network with Non-Simultaneous Multi-Commodity Flow Requirements." In P. Hansen (ed.), *Studies on Graphs and Discrete Programming*. North-Holland, pp. 269–277.

Monma, C.L. and D.F. Shallcross. (1989). "Methods for Designing Communications Networks with Certain Two-Connected Survivability Constraints," *Operations Research* 37(4), 531–541.

Pickavet, M., F. Poppe, J. Luystrermans, and P. Demeester. (1997). "A Genetic Algorithm for Solving the Capacitated Survivable Network Design Problem." In *Proc. of Fifth International Conference on Telecommunication Systems*, pp. 71–76.

Poppe, F. and P. Demeester. (1997). "The Design of SDH Mesh Restorable Networks: Problem Formulation and Algorithm." In *Proc. of Fifth International Conference on Telecommunication Systems*, pp. 88–97.

Poppe, F. and P. Demeester. (1998). "Economic Allocation of Spare Capacity in Mesh Restorable Networks: Model and Algorithm." In *Proc. of Sixth International Conference on Telecommunication Systems*, pp. 77–86.

Sakauchi, H., Y. Nishimura, and S. Hasegawa. (1990). "A Self-Healing Network with an Economical Spare-Channel Assignment." In *Proc. of IEEE Globecom*, pp. 438–443.

Sato, K.-I. (1996). *Advances in Transport Network Technologies, Photonic Networks, ATM and SDH*, Norwood: Artech House.

Stoer, M. (1992). *Design of Survivable Networks*. New York: Springer-Verlag.

Stoer, M. and G. Dahl. (1994). "A Polyhedral Approach to Multi-commodity Survivable Network Design," *Numerische Mathematik* 68, 149–167.

Wu, T.H. (1992). *Fiber Network Service Survivability*, London: Artech House.

Wu, T.H. (1995). "Emerging Technologies for Fiber Network Survivability," *IEEE Commun. Mag*. 33(2), 58–74.

Journal of Heuristics, 6: 131–148 (2000)
© 2000 Kluwer Academic Publishers

Heuristics for Distribution Network Design in Telecommunication

GERALDO R. MATEUS*, HENRIQUE P.L. LUNA* AND ADRIANA B. SIRIHAL
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Av. Antonio Carlos, 6627 31270-010-Belo Horizonte-MG-Brasil
email: mateus@dcc.ufmg.br; pacca@dcc.ufmg.br; abs@dcc.ufmg.br

Abstract

A distribution network problem arises in a lower level of an hierarchical modeling approach for telecommunication network planning. This paper describes a model and proposes a lagrangian heuristic for designing a distribution network. Our model is a complex extension of a capacitated single commodity network design problem. We are given a network containing a set of sources with maximum available supply, a set of sinks with required demands, and a set of transshipment points. We need to install adequate capacities on the arcs to route the required flow to each sink, that may be an intermediate or a terminal node of an arborescence. Capacity can only be installed in discrete levels, i.e., cables are available only in certain standard capacities. Economies of scale induce the use of a unique higher capacity cable instead of an equivalent set of lower capacity cables to cover the flow requirements of any link. A path from a source to a terminal node requires a lower flow in the measure that we are closer to the terminal node, since many nodes in the path may be intermediate sinks. On the other hand, the reduction of cable capacity levels across any path is inhibited by splicing costs. The objective is to minimize the total cost of the network, given by the sum of the arc capacity (cables) costs plus the splicing costs along the nodes. In addition to the limited supply and the node demand requirements, the model incorporates constraints on the number of cables installed on each edge and the maximum number of splices at each node. The model is a NP-hard combinatorial optimization problem because it is an extension of the Steiner problem in graphs. Moreover, the discrete levels of cable capacity and the need to consider splicing costs increase the complexity of the problem. We include some computational results of the lagrangian heuristics that works well in the practice of computer aided distribution network design.

Key Words: distribution network, network design, lagrangian heuristics

1. Introduction

The urban telecommunication system is composed by intricate networks that enable the communication of hundreds of thousands of customers. The hierarchical organization of this network plays a major role, in as much as optimized levels of customers concentration enables the substantial economies of scale of increasing transmission bandwidth. The design of such networks with an hierarchical modeling approach is also a consequence of the fact that people solve complex problems by solving higher level problems first and then solving

*Research supported by Conselho Nacional do Desenvolvimento Científico e Tecnológico—CNPq, and by Telecomunicações de Minas Gerais—TELEMIG, Brasil.

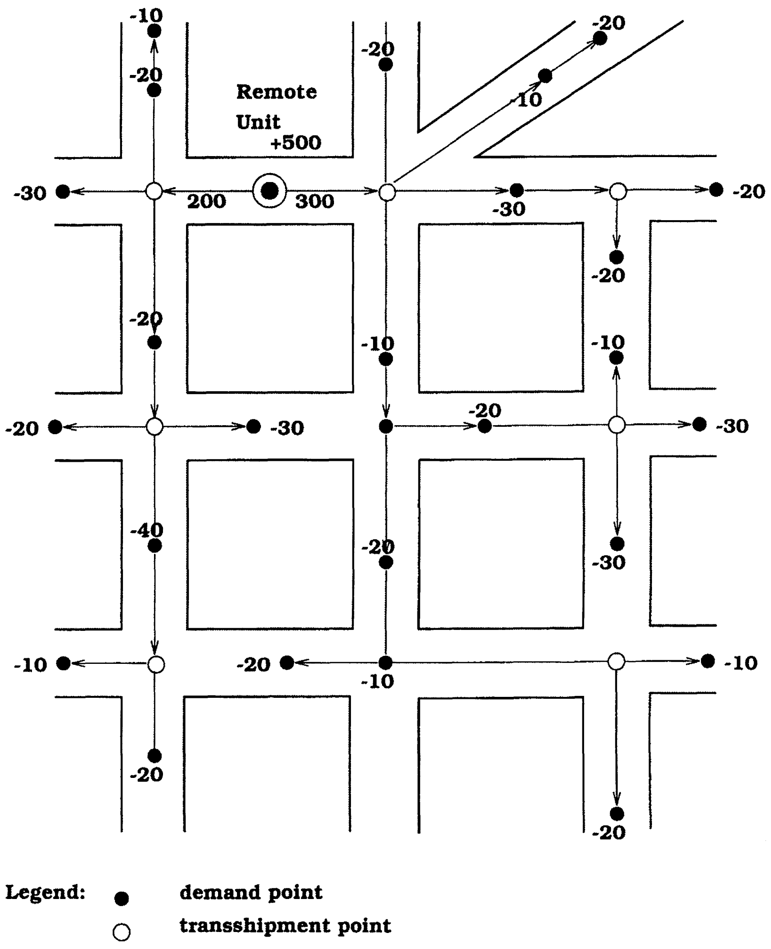


Figure 1. Spatial demands of a distribution network.

resulting lower level problems. A distribution network problem arises in a lower level of the hierarchical organization of the telecommunication system. The problem can be viewed as an extension of a capacitated single commodity network design problem, as illustrated in figure 1. The commodity flow in an arc can be interpreted as the equivalent number of telecommunication lines that links a switching center to the customers served across that arc. The problem assumes that customers are clustered and connected to terminal boxes (sinks), normally installed in electricity poles. The problem also assumes given the location and capacity of optical remote units (sources), which are supposed to be connected to the switching centers. We need to install adequate capacities on the arcs to route the required flow to each sink. This paper describes a model and a lagrangean heuristic for the computer assisted design of a distribution network.

A method where a large problem is broken up into several smaller problems is normally implicit in the literature concerning specific models of telecommunication network design

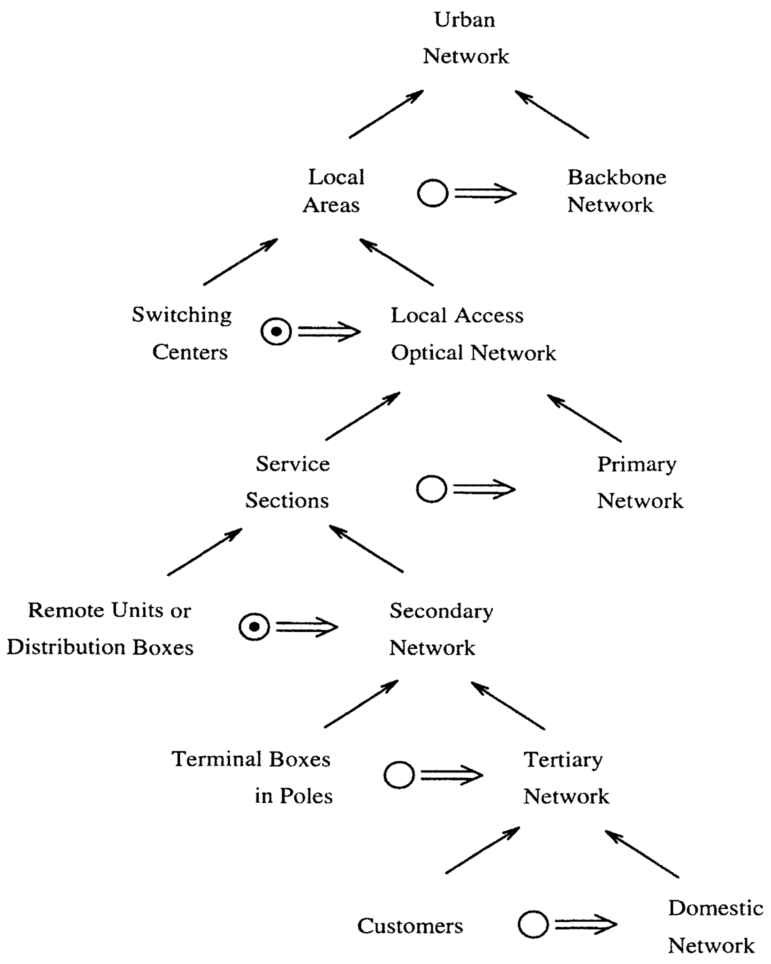


Figure 2. Hierarchical structure of an urban telecommunication network.

(Gavish, 1982, 1983, 1991; Minoux, 1989). In order to provide a better description of the distribution network context we prefer to follow here the practice of working explicitly with an hierarchical design organization (Balakrishnan, Magnanti, and Mirchandani, 1994; Mateus, Cruz, and Luna, 1994). Figure 2 depicts the hierarchical structure of an urban telecommunication network. The symbol $\bigcirc \Longrightarrow$ means that the element at left is a generic node of the network described at right, and the symbol $\odot \Longrightarrow$ means that the element at left is the *root* node of a *tree* network referred at right. The first level of the figure states that the *urban space* is partitioned in *local areas*. Each local area is served by a *switching center.* The communication among local areas is performed by a *backbone network.* The *switching center* is the element of linkage between the *backbone network* and the *local access network.* Remark that a *switching center* is a node representing a local area in the *backbone network* and that it is also the *root* node of the correspondent *local access network.*

We are in this paper concerned with the second phase of the local access network design process. Figure 2 shows that, in order to provide access for the customers assigned to each switching center, we need to perform three levels of network design:

1. At the first level, each local area is partitioned in *service sections*, and the *primary network* provides optical access of the service sections to the assigned switching center.
2. At the second level, each service section is partitioned in *terminal sections*, and the *secondary network* provides the access of these terminal sections. An *optical remote unit* or a *distribution box* is the element of linkage between the *primary* and the *secondary* networks.
3. At last, the *tertiary network* encompasses the *domestic networks* that assure the links of the customers to assigned terminal boxes.

The distribution network studied here concerns a particular local access network design problem, that in figure 2 is associated with the secondary network. The literature on local access network design problems covers a variety of settings, which raises issues of dimensioning, topological design and routing. Balakrishnan and colaborators (1991) discuss several local access network design formulations (see also Magnanti, Mirchandani, and Vachani, 1993; Bienstock and Günlük, 1995). The local access design can be made either according to a Steiner tree (Luna, Ziviani, and Cabral, 1987; Balakrishnan, Magnanti, and Wong, 1989) or else be based in an extension of the minimum spanning tree problem (Gavish, 1983, 1991; Hochbaum and Segev, 1989; Mirzaian, 1985). In any case it results a NP-hard problem. In the first case we express the real existence of intermediate or transshipment nodes, but we face the difficulty of a NP-hard subproblem, thus looking for approximate solutions of the Steiner problem in graphs (Hakimi, 1971; Aneja, 1980; Wong, 1984; Maculan, 1987; Beasley, 1989; Duin and Volgenant, 1989; Agrawal, Klein, and Ravi, 1994). In the second case we deal with a restricted modeling approach, but we can take advantage of the fact that a greedy algorithm is able to find optimal solutions of the minimum spaning tree subproblem (Kruskal, 1956).

A complementary problem arising in local access design of computer and communication systems concerns the location of facilities. The facilities may be switching centers, remote units, concentrators, distribution or terminal boxes (Minoux, 1989; Gavish, 1991). Normally a facility is the *root* (supply) node of a local access *tree* network, and it is one of many demand nodes of a higher level network, that may be a *tree*, a *ring* or a *multiconnected* network. We can be inspired by classical methods to solve facility location problems, either in capacitated versions (Sá, 1969; Beasley, 1988; Mateus and Luna, 1992) or in uncapacitated versions (Balakrishnan, Magnanti, and Wong, 1989). Distribution network planning consists of both locating the optical remote unit and selecting adequate cables to route the required flow to each terminal box. We assume here that a previous phase has located the remote units (or distribution boxes) in the wider area of a switching center. We address here the topology and dimensioning aspects of the distribution network, thus focusing the smaller area of one or some service sections.

We have not found in the literature the formulation proposed here. The model is a NP-hard combinatorial optimization problem because it is an extension of the Steiner problem in graphs. Moreover, the need to install standard capacities and the need to consider cable splicing costs result in an extremely difficult integer programming model. We propose a lagrangian relaxation heuristic where the dual variables are updated by a subgradient method. We have introduced valid inequalities to improve the lower bounds. We have observed that even small problems are intractable by commercial software and also that the problem complexity increases when the existing network must be considered in the solution procedure. The heuristic generates acceptable solutions in the practice of distribution network design, because it imitates human designs in a certain sense. But, unfortunately, high gaps between lower and upper bounds for many instances do not support any assertion on the optimality of these solutions. We report some computational tests with real distribution networks.

This paper is organized as follows. Section 2 introduces the network design problem and presents the mathematical programming formulation. Section 3 presents the algorithm and Section 4 summarizes extensive computational experiments. The paper has also a concluding section.

2. Distribution network design problem

The linkage of the urban communication customers is nowadays performed by fiberoptics for the backbone and for the primary networks. For the secondary (distribution) and the tertiary networks we can have fiber and/or copper cables, and the models should acommodate any possible configuration: *fiber-to-the-home, fiber-to-the-curb* or traditional copper links. In any case the distribution network is composed by cables with standard capacity levels, normally given by a number of pairs of fiberoptics or copper wires. In large cities the topology of the backbone network is a *multiconnected* network, and the *tree* is the basic topology of the local access networks. In such cases we can say that the urban primary network is a *forest*, where each switching center is the *root* and remote units are *leaves* of a local area *tree*. The secondary network is a *forest*, where each remote unit is the *root* and terminal boxes are *leaves* of a service section *tree*. And the tertiary network is a *forest*, where the terminal box of each selected pole is the *root* and customers' computer and communication equipments are *leaves* of a terminal section *tree*.

Our model focuses the details of a minor area concerning some *trees* of the secondary network *forest*. The model is a complex extension of a capacitated single commodity network design problem. We are given a network containing a set of sources with maximum available supply, a set of sinks with required demands, and a set of transshipment points. We need to install adequate capacities on the arcs to route the required flow to each sink. Capacity can only be installed in discrete levels, i.e., cables are available only in certain standard capacities. Economies of scale induce the use of a unique higher capacity cable instead of an equivalent set of lower capacity cables to cover the flow requirements of any link. The reduction of capacity levels across any path is inhibited by splicing costs, that may be compensated by economy on cable costs as long as are reduced the flow requirements across the path. The objective is to minimize the total cost of the network, given by the sum of the arc capacity (cables) costs with the splicing costs along the nodes. In addition to the limited supply and the node demand requirements, the model incorporates constraints on the number of cables installed on each edge and the maximum number of splices at each node.

The Distribution Network Design Problem can be defined over a graph $G = (N, A)$, where N is the set of nodes and A is the set of *directed* arcs. Let $S \subset N$ be a set of connection facilities (optical remote unities and or distribution boxes), which are supply nodes with capacity a_i, $\forall i \in S$. Let also $T \subset N$ be a set of intermediate or transshipment nodes and $D \subset N$ be a set of terminal boxes or demand nodes with capacity $d_i > 0, \forall i \in D$. Each node $i \in N$ has a unitary cost e_i^m to splice a cable of capacity level t^m, $m \in M$. For each arc $(i, j) \in A$ there is a unitary cost c_{ij}^m of using the arc with a cable of capacity t^m. We consider $c_{ij}^m = 0$ and $e_i^m = 0$ for all cables and splices in the existing network. The optimization problem consists of dimensioning adequate capacities in a subset of arcs in order to meet the demand in each terminal box with minimal total cost. The mathematical programming problem is:

$$\min \sum_{m\in M} \sum_{(i,j)\in A} c_{ij}^m y_{ij}^m + \sum_{m\in M} \sum_{i\in N} e_i^m z_i^m \tag{1}$$

subject to:

$$\sum_{(i,j)\in\Gamma^+(i)} x_{ij} - \sum_{(j,i)\in\Gamma^-(i)} x_{ji} \leq a_i, \quad \forall i \in S \tag{2}$$

$$\sum_{(i,j)\in\Gamma^+(i)} x_{ij} - \sum_{(j,i)\in\Gamma^-(i)} x_{ji} = 0, \quad \forall i \in T \tag{3}$$

$$\sum_{(i,j)\in\Gamma^+(i)} x_{ij} - \sum_{(j,i)\in\Gamma^-(i)} x_{ji} = -d_i, \quad \forall i \in D \tag{4}$$

$$\sum_{m\in M} t^m y_{ij}^m \geq x_{ij}, \quad \forall (i, j) \in A \tag{5}$$

$$\sum_{(i,j)\in\Gamma^+(i)} y_{ij}^m - \sum_{(j,i)\in\Gamma^-(i)} y_{ji}^m \leq z_i^m, \quad \forall i \in N, \forall m \in M \tag{6}$$

$$\sum_{m\in M} (y_{ij}^m + y_{ji}^m) \leq 3, \quad \forall (i, j) \in A \tag{7}$$

$$x_{ij} \geq 0, \text{integer}, \quad \forall (i, j) \in A \tag{8}$$

$$y_{ij}^m \in \{0, 1, 2, 3\}, \quad \forall (i, j) \in A, \quad \forall m \in M \tag{9}$$

$$z_i^m \in \{0, 1, 2, 3\}, \quad \forall (i) \in N, \quad \forall m \in M \tag{10}$$

where we have the following sets and parameters: S the set of connection facilities (optical remote units or distribution boxes); T the set of intermediate nodes; D the set of demand nodes or terminal boxes; M the set of cable capacity levels; $\Gamma^+(i)$ the set of arcs $(i, j) \in A, \forall j \in N$; $\Gamma^-(i)$ the set of arcs $(j, i) \in A, \forall j \in N$; c_{ij}^m unitary cost of a cable of modularity $m \in M$ on arc $(i, j) \in A$; e_i^m splicing cost of a cable with capacity level $m \in M$ on node $i \in N$; t^m cable capacity for the level $m \in M$; a_i supply capacity in node $i \in S$; d_i demand in node $i \in S$. And where we have the following variables: x_{ij} flow through the arc $(i, j) \in A$; y_{ij}^m number of cables of capacity t^m, $m \in M$, installed on arc $(i, j) \in A$; z_i^m number of cables of capacity t^m that are spliced on node $i \in N$.

The objective function minimizes the sum of the investment costs in cables together with the working costs of splicing these cables in the distribution network. The inequality (2) and the Eqs. (3) and (4) are the flow conservation relations for the supply, transshipment and demand nodes. Constraints (5) express the capacity requirements to meet the flow at each arc. Constraints (6) measure the number of cables of capacity t^m that are spliced on each node i. Equations (7) impose the maximum number of cables on each edge. The set of constraints (8), (9) and (10) assures that the variables are integer and nonnegative. Remark that, for an urban aesthetic reason, a distribution network design practice limit to 3 the maximum number of cables (in either direction) on each edge. As a consequence, for each type m of cables, the number of splices z_i^m is between 0 and 3.

Note that the model is an extension of a capacitated network design problem. The network is *directed* and a positive flow $x_{ij} > 0$ indicates the *number of telecommunication lines* that is served across the arc (i, j) in the direction of a path linking the connection facility to a terminal node. Of course, the direction here is a matter of design, permitting to identify cables and splices across the path of each line from the switching center to the final customer. This does not prevent the use of full duplex links in the distribution network, in such a way that if a cable is installed between two nodes i and j then both the traffic (in *erlangs*) from i to j and from j to i can flow on the same cable, as it is the classical case for voice communication.

The model assumes that a particular terminal box can be supplied from more than one connection facility. This is not really a good practice. Although in some cases the designer can accept a terminal box served by two connection facilities, our implementation admits the interference of the designer to rule out such exceptions. Remember that the model is a computer aided design tool, and so it is expected that the role of the designer is very important to arbitrate on such exceptions and also to accommodate the existing network in the project. By the way, the y variables associated with the existing network can be expressed in the formulation with null costs for all current cables and splices. In this case, each type of installed cables can be selected, $y_{ij}^m \neq 0$, or else rejected in the solution, $y_{ij}^m = 0$.

The originality of this capacitated network design problem is that it takes into account the cable splicing costs. The work force to perform the linkage of cables with different capacities represent about twenty percent of the total cost of a distribution network. Remark that we account the cost to splice any cable in its starting point. That is, the number of spliced cables of capacity t^m that is considered at node i is given by the difference between the number of type *m outgoing* cables emanating from node i minus the number of type *m incoming* cables at node i. If for a certain capacity level the number of incoming cables exceeds the number of outgoing cables, then there is no need to consider a splicing cost already accounted in an originating node. It is not the case of a distribution network, but remark that a similar modeling approach could perhaps account the costs of (multiplex/demultiplex) equipments required at the nodes to transfer traffic from one type of cable to another cable with different transmission bandwidth.

The model is a NP-hard combinatorial optimization problem because it is an extension of the Steiner problem in graphs. The discrete levels of cable capacity and the need to consider the splicing costs increase the complexity of the problem. The above formulation is prohibitively expensive for real problems. The number of constraints and integer variables are very high, as in the presented examples. Branch and bound or other optimal methods may not be practicable when dealing with such large problems. We are thus suggesting heuristics in the next section.

3. A lagrangian relaxation algorithm

We present a lagrangian relaxation heuristic with a subgradient procedure to update the set of Lagrange multipliers. The lagrangian relaxation method has been successfully applied to approximate many combinatorial optimization problems. The heuristic generates lower and upper bounds for the optimal objective function value. The lower bounds are obtained by relaxing a set of constraints. Associated with the lower bounds we can generate feasible solutions and upper bounds for the original problem.

Let *LB* be the best lower bound and *UB* be the best upper bound in a certain iteration. The lagrangian procedure consists of iteratively modifying the Lagrange multipliers by the subgradient method. If *LB* is approximately equal to *UB* then an optimal solution have been obtained, otherwise we have a duality gap (Fisher, 1981, 1985; Geoffrion and McBride, 1978) and we can not assure optimality of the obtained solution.

In the next subsections we introduce the lower and upper bounds and the subgradient procedure.

3.1. *Lower bound*

A lower bound for the proposed model can be easily obtained by relaxing complicating constraints. In our case, a lower bound is obtained by dualizing the constraints (5) and (6). In this article we report the computational procedure for this relaxation.

Let $\alpha_{ij} \geq 0, \forall (i,j) \in A$ be the set of multipliers associated with constraints (5), and $\beta_i^m \geq 0, \forall i \in N$ and $\forall m \in M$, be the set of multipliers associated with constraints (6). The relaxed problem is given by:

$$L(\alpha,\beta) = \min \left[\sum_{(i,j)\in A} \alpha_{ij} x_{ij} + \sum_{m\in M} \sum_{(i,j)\in A} \left(c_{ij}^m - \alpha_{ij} t^m + \beta_i^m - \beta_j^m\right) y_{ij}^m \right. \\ \left. + \sum_{i\in N} \sum_{m\in M} \left(e_i^m - \beta_i^m\right) z_i^m \right] \tag{11}$$

subject to: (2), (3), (4), (7), (8), (9) and (10).

For any given set of multipliers α_{ij} and β_i^m, the lagrangian problem can be decomposed into three subproblems, such that $L(\alpha,\beta) = L_{P1} + L_{P2} + L_{P3}$. The subproblem P_1 corresponds to a minimum cost network flow procedure. The second and third subproblems can be solved by inspection.

3.1.1. Subproblem P_1. The subproblem P_1, including the variables x_{ij}, is given by:

$$L_{P_1} = \min \sum_{(i,j)\in A} \alpha_{ij} x_{ij} \tag{12}$$

subject to: (2), (3), (4) and (8).

Subproblem P_1 can be solved by a minimum cost network flow procedure. We have applied here a primal simplex algorithm (Bradley, Brown, and Graves, 1977). Let x^r be the optimal solution to the subproblem P_1.

3.1.2. Subproblem P_2. Subproblem P_2, including variables y_{ij}^m, has the following formulation:

$$L_{P_2} = \min \sum_{m \in M} \sum_{(i,j) \in A} \left(c_{ij}^m - \alpha_{ij}\, t^m + \beta_i^m - \beta_j^m\right) y_{ij}^m \qquad (13)$$
$$\text{subject to: (7) and (9)}$$

The subproblem can be solved by inspection. The y_{ij}^m variables are selected in increasing order of their coefficients in (13), and we obtain the y^r optimal solution to subproblem P_2.

We include a valid additional constraint to subproblem P_2 to improve the lower bound. Our approach is motivated by the minimum number of arcs in a feasible solution. This constraint imposes the minimum number of arcs (MNA) to be selected in P_2 to accomodate an implicit design for the original problem. We can test different constraints. In this sense, we make MNA equal a lower bound for the minimum number of arcs in a feasible solution, defined over a generalized Steiner problem where all costs are 1, as in Agrawal, Klein, and Ravi (1994). Moreover, it consists in finding a collection of arc-disjoint cuts in this particular Steiner network. In this article we include only one additional constraint, we select at least $|D|$ arcs, $MNA = |D|$, where $D \subset N$ is the set of demand nodes.

3.1.3. Subproblem P_3. Subproblem P_3, including the variables z_i^m, is much simpler and is given by:

$$L_{P_3} = \min \sum_{m \in M} \sum_{i \in N} \left(e_i^m - \beta_i^m\right) z_i^m \qquad (14)$$
$$\text{subject to: (10)}$$

Again the subproblem can be solved by inspection. Let z^r be the optimal solution to the subproblem P_3.

3.2. Upper bound

A feasible design for the original problem is an upper bound for the optimal value of the objective function (1). A feasible solution is obtained from the solutions of the three subproblems. The first subproblem provides a feasible flow x^r. Remark that any flow in any arc can always be covered by a maximum of three cables of the higher capacity level because they correspond to a total value greater than the capacity of the connection facility. Assuming the feasible flow x^r and beginning at each origin node, the upper bound procedure moves to the destination nodes. For each arc (i, j) in the path from an origin to a terminal node, compare the fixed cost to install one, two or three cables, with sufficient capacity to meet the flow, plus the splice costs for these cables, with the fixed cost of maintaining the same incoming cables in node i. Check all the feasible sets of cables, select the minimum cost set and repeat until every flow is met. In this procedure, the current cables and the cables selected in subproblem P_2 take priority to new elements. After that, we need to calculate only the feasible values for the set of variables z_i^m. These values are given based on the constraints (6):

$$z_i^m = \sum_{(i,j)\in\Gamma^+(i)} y_{ij}^m - \sum_{(j,i)\in\Gamma^-(i)} y_{ji}^m, \quad \forall i \in N, \forall m \in M$$

3.3. *Initial solution*

The solution procedure receives, from the location phase, the set of distribution boxes, and the set of terminal boxes assigned for each distribution box. An initial solution is then obtained for each distribution box separately, by solving a minimum cost network flow problem for each distribution box i with its supply capacity a_i and the demand nodes given by the set of terminal nodes assigned to it. Let x^0 be the feasible flow to meet the demand. Therefore, the initial solution is obtained by the same procedure presented for the upper bound assuming the feasible flow x^0.

3.4. *Subgradient procedure*

For each set of lagrangian multipliers α_{ij} and β_i^m, we obtain a lower bound. Therefore, the tightest lower bound which can be achieved is the solution of the lagrangian dual problem:

$$\max_{\alpha,\beta\geq 0} L(\alpha, \beta) \tag{15}$$

Several optimization methods have been suggested for solving the dual problem. We adopt a subgradient procedure. The subgradient method used here consists of iteratively modifying the values of the multipliers. In the nth iteration the multipliers are updated by:

$$\begin{aligned}\alpha^{n+1} &= \max\{0, \alpha^n + s^n h^n(y^r, x^r)\}\\ \beta^{n+1} &= \max\{0, \beta^n + s^n g^n(y^r, z^r)\}\end{aligned}$$

where s^n is a step size calculated by:

$$s^n = \rho \frac{UB - L(\alpha^n, \beta^n)}{\|h^n\|^2 + \|g^n\|^2} \tag{16}$$

ρ is a scalar restricted to $0 < \rho \leq 2$ and $h^n(y^r, x^r)$ and $g^n(y^r, z^r)$ are the subgradient vectors given by:

$$h_{ij}(y^r, x^r) = x_{ij} - \sum_{m\in M} t^m y_{ij}^{m^r}, \quad \forall (i, j) \in A$$

$$g_i^m(y^r, z^r) = \sum_{(i,j)\in\Gamma^+(i)} y_{ij}^{m^r} - \sum_{(j,i)\in\Gamma^-(i)} y_{ji}^{m^r} - z_i^{m^r}, \quad \forall i \in N, \forall m \in M$$

We follow basically the approach of Geoffrion (Geoffrion and McBride, 1978) and Fisher (1981, 1985) in setting $\rho = 2$ initially. We half ρ if the best lower bound do not increase within $2n$ subgradient iterations, following in this case what has also been done by Beasley (1988). In our algorithm we only consider that the lower bound has increased if an improvement of 1% is achieved.

4. Computational results

The heuristic method was tested on two real networks, both selected as case studies for the telecommunication company. The problems were carried on a set of values given by the users: number of nodes, number of arcs, cable capacities, cable and splice costs, demands and graph structure.

The proposed algorithm was implemented in C and the tests were ran on a SUN SPARCstation 10 with 64Mb of RAM. We present the results for the following data:

- Network 1: 126 nodes, 272 directed arcs, 104 current cables in 88 arcs, 22 splices in 15 nodes;
- Network 2: 217 nodes, 520 directed arcs, 184 current cables in 147 arcs, 73 splices in 47 nodes;
- distribution box supply capacity: 600
- capacities, cable and splice costs: see Table 1 (provided by the telecommunication company for the current monetary unit);
- demand capacities and graph structure given by the users.

We remark that such real networks are quite sparse, with the original graph having a number of undirected edges about 10% and 20% larger than a spanning tree. This is because many demand nodes are not at intersections of streets, they are adjacent to only two edges (see figure 1). A demand or intermediate node in a corner rarely has more than four adjacent edges. Of course the above data assume that all the opportunities for problem reduction have already been done, i.e., obvious ways to fix variables, combine nodes, eliminate arcs, etc.

Table 1. Capacities, cable and splice costs.

Capacity	Cost per meter	Splice cost
10	3.20	59.43
20	4.02	52.50
30	4.51	55.62
50	6.84	61.72
75	8.13	69.39
100	9.04	76.96
150	12.29	97.38
200	15.66	120.27
300	22.15	150.71

We have performed a wide range of test cases. A test case is defined over different aspects to be analyzed: the number of distribution boxes, the existing network and the influence of some parameters as the number of arcs and the total demand.

We have also analysed two different relaxations in our computational results. A first relaxation, *Relaxation 1*, was proposed in Section 3. The duality gap is calculated by $100(UB - LB)/UB$. The heuristic suggests an acceptable solution in small computation time. But, the gap between the upper and lower bounds is yet very high for a great number of subgradient iterations. Then, we fixed the maximum number of subgradient iterations equal to 200. After that, we have fixed the x variables equal to the best solution obtained in the first 200 iterations. Let x^* be this best solution. Supposing $x = x^*$ in the proposed model, the problem reduces to the y and z variables and a new relaxation can be obtained relaxing only constraints (6). The new relaxed problem provides the *Relaxation 2*, that is also based on lagrangian relaxation. The new heuristic can be applied in the same sequence presented for Relaxation 1. The solution for the new relaxed problem is *not* a lower bound for the proposed model, but this alternative relaxation becomes easier to be solved. The lagrangian function in this case provides a "Cost Estimator" (CEst) that is a lower bound for the cost of any solution for which the flow variables are fixed in the values x^* given by Relaxation 1. The maximum number of subgradient iterations is fixed equal to 1000 for Relaxation 2, where we have registered in all tables the CEst and the Error values, the last being the estimated error with respect to an optimal solution for the flow variables fixed in x^*.

Table 2 summarizes the results for Network 1 with 6 distribution boxes selected in the location phase, and considering the existing network. Table 3 also presents the results for

Table 2. Network 1: Number of candidate DB = 6, with the existing network.

			Relaxation 1				Relaxation 2				
N. of DB	DB/Total demand	Initial solution	N. of iterat.	UB	LB	Gap (%)	N. of iterat.	UB	CEst	Error (%)	Time (s)
1	12/190	2499.43	200	2499.43	1056.98	58	120	2499.43	2474.66		29.29
1	26/410	12178.68	200	12178.68	7275.75	40	966	12178.66	11907.52	2	157.26
1	34/190	1479.30	200	1479.30	553.77	63	15	1479.30	1464.98		17.26
1	39/140	1546.55	200	1546.55	773.49	50	44	1546.55	1531.58		20.31
1	80/570	4942.35	200	4942.35	1755.19	64	1000	4935.06	4793.01	3	229.85
1	113/190	3623.14	200	3623.14	1899.74	48	204	3623.14	3587.21		36.60
2	12, 26/600	14678.11	200	14445.79	7946.94	45	1000	14434.50	14151.29	2	222.60
2	26, 34/600	13657.97	200	13568.15	7660.07	44	548	13568.15	13434.81		134.99
2	34, 39/330	3025.85	200	3025.85	1326.21	56	25	3025.85	3004.62		35.4
2	39, 80/710	6427.18	200	6427.18	2459.35	62	618	6427.18	6363.30		183.59
2	80, 113/760	8565.49	200	7823.54	3000.76	62	1000	7779.24	7552.50	3	312.29
3	12, 26, 34/790	16157.40	200	15819.34	8339.00	47	631	15819.34	15661.32		245.98
3	26, 34, 39/740	15204.52	200	15012.53	8305.92	45	241	15012.53	14862.72		112.34
3	34, 39, 80/900	7968.20	200	7968.20	2918.34	51	180	7968.20	7890.31		123.55
3	39, 80, 113/900	10112.04	200	9376.19	3696.75	61	939	9325.79	9157.04	2	340.51

Table 3. Network 1: Number of candidate DB = 8, without the existing network.

			Relaxation 1				Relaxation 2				
N. of DB	DB no./Total demand	Initial solution	N. of iterat.	UB	LB	Gap (%)	N. of iterat.	UB	CEst	Error (%)	Time (s)
1	12/240	4278.21	200	4278.21	2154.21	50	22	4278.21	4235.89		21.21
1	26/370	12415.12	200	12415.12	8153.55	34	67	12316.01	12193.56		31.71
1	34/210	2794.47	200	2794.47	1521.27	46	22	2794.47	2766.53		18.68
1	39/160	2569.40	200	2569.40	1480.51	42	13	2569.40	2547.21		17.85
1	60/140	2617.56	200	2617.56	1177.31	55	13	2610.27	2586.16		17.09
1	80/140	4038.77	200	4038.77	2061.85	49	9	4038.77	4006.69		16.60
1	95/100	2032.77	200	2032.77	998.08	51	7	2032.77	2017.29		13.92
1	113/330	9300.17	200	9300.17	5083.86	45	29	9300.17	9208.48		23.33
2	12, 26/610	16640.75	200	16640.75	10047.26	40	67	16640.75	16480.53		46.49
2	12, 34/450	7072.68	200	7072.68	3515.61	50	33	7072.68	7005.33		58.37
2	12, 69/380	6895.77	200	6895.77	3152.29	54	27	6895.77	6833.48		58.79
2	26, 34/580	15209.59	200	15209.59	9423.15	38	43	15123.62	14977.22		69.37
2	34, 39/370	5363.87	200	5363.87	2885.96	46	25	5363.87	5314.09		54.56
2	39, 69/300	5186.96	200	5186.96	2510.90	52	19	5186.96	5141.30		28.25
2	69, 80/280	6656.36	200	6656.36	2990.78	55	15	6655.11	6588.94		53.21
2	80, 95/240	6015.92	200	6015.92	2872.86	52	10	6015.92	5962.26		22.35
2	95, 113/430	11280.36	200	11280.36	5857.72	48	19	11280.36	11176.10		28.34
3	12, 26, 34/820	19435.22	200	19435.22	11326.24	42	55	19435.22	19243.54		80.06
3	12, 34, 69/590	9690.24	200	9690.24	4484.32	54	34	9690.24	9598.86		132.16
3	34, 69, 80/490	9450.79	200	9450.79	4318.88	54	24	9449.58	9361.66		113.69
3	39, 95, 113/590	13902.34	200	13902.34	7147.93	49	26	13902.34	13770.91		128.64
3	69, 80, 95/380	8525.37	200	8525.37	3794.30	55	14	8525.37	8445.68		34.16
3	80, 95, 113/570	15319.13	200	15319.13	7636.49	50	20	15319.13	15175.11		37.75

Network 1 with 8 distribution boxes given by the location phase, but it does not consider the existing network. The first column is the number of distribution boxes (DB) considered in the distribution network design. This number varies between 1 and 3 boxes. The second column presents the node numbers associated to the studied distribution boxes and the same total demand for their local networks. The initial solution procedure was presented in Section 3. The column labeled Time is the computation time in seconds for both relaxations.

The heuristic generates effective designs and the results illustrate the efficacy of the algorithm. But, the bounds are usually rather far apart, with gaps of the order of 40% to 50%. We make the point that, for a given set of x values, the Relaxation 2 produces optimal y and z values in almost all cases. The final solution (UB) is generally very close to the initial solution. The result suggests that this is a good solution, and may be explained, in part, because the test networks are sparses, as we have observed for the case of urban networks. The algorithm performance is improved when the existing network is not considered. A

Table 4. Network 1: Number of candidate DB = 10, with the existing network.

			Relaxation 1				Relaxation 2				
N. of DB	DB no./Total demand	Initial solution	N. of iterat.	UB	LB	Gap (%)	N. of iterat.	UB	CEst	Error (%)	Time (s)
1	26/580	14737.31	200	14737.31	8996.82	39	367	14659.23	14513.96		95.22
1	80/600	7945.69	200	7945.69	4023.62	49	347	7923.08	7815.98		102.37
1	113/510	12149.97	200	12149.97	6395.78	47	531	12149.97	12029.53		106.80
2	26, 113/1090	26872.42	200	25870.57	14648.91	43	1000	25823.96	25411.96	2	344.73
2	26, 80/1180	22659.00	200	22659.00	13740.15	39	340	22659.00	22135.87		303.02
2	80, 113/1110	20095.66	200	20095.66	10258.94	49	475	20095.66	19712.27	2	232.16
3	26, 80, 113/1690	34870.69	200	34870.69	19678.53	44	588	34870.69	34259.26		425.51

Table 5. Network 1: Number of candidate DB = 10, without the existing network.

			Relaxation 1				Relaxation 2				
N. of DB	DB no./Total demand	Initial solution	N. of iterat.	UB	LB	Gap (%)	N. of iterat.	UB	CEst	Error (%)	Time (s)
1	26/600	19415.80	200	19123.84	12549.75	34	71	18956.54	18768.16		43.25
1	69/590	17808.57	200	17808.57	10567.85	41	31	17791.87	17614.83		40.83
1	113/500	17548.94	200	17548.94	10093.21	42	100	17498.85	17326.67		45.05
2	26, 69/1190	37279.98	200	37022.95	22710.26	39	126	36848.91	36481.54	2	146.38
2	26, 113/1100	36912.16	200	36672.78	22244.91	39	123	36498.20	36133.53		102.27
2	69, 113/1090	35413.13	200	35413.13	20249.58	43	94	35370.86	35021.35	2	99.54
3	26, 69, 113/1690	54828.92	200	54489.51	32299.81	41	159	54336.84	53794.64		115.86

faster solution time is obtained and the duality gaps are still large, though they are consistently smaller. Tables 4 and 5 compare the influence of the existing network in a set of test networks with the same topology.

Tables 6 and 7 shows the results for Network 2, considering the existing network in both cases. The number of distribution boxes to serve the total demand of the network is 6 in Table 6 and 12 in Table 7. The solutions are also quite acceptable from the users point of view. The results confirm the effectiveness of the initial solution. The number of current cables rejected in the final solution is small. We can conclude that the use of the existing network has been a contributing factor to reduce the network cost.

We have also performed computational tests using dense networks. Table 8 presents the results for a problem instance with 150 nodes and the number of directed arcs ranging from 330 to 600, without the existing network. The results demonstrate that, for the studied example, the number of arcs does not have a strong influence on the required computation time, but that the cost reduces significantly with the number of arcs.

Table 6. Network 2: Number of candidate DB = 6, with the existing network.

DB number	49	53	76	149	150	49–53	53–76	49–53–76
Total demand	480	600	370	590	240	1080	970	1450
Nodes/Arcs	53/214	67/272	33/94	53/174	21/66	119/492	99/392	151/612
Init. solution	3919.33	10334.91	4497.44	22265.42	6508.70	14254.24	14832.35	18751.69
Iterations rel. 1	114	114	200	200	200	114	200	200
Gap (%)	63	67	55	38	39	66	48	52
Iterations rel. 2	1000	593	1000	679	414	790	888	1000
Error (%)	3	4	2	1	1	5	4	4
Cost	3919.33	10289.32	4497.44	22243.57	6403.84	14019.86	14786.76	18706.10
Time (s)	197.2	152.23	145.27	159.78	66.86	324.62	349.56	609.88
N. current cables	48	58	17	30	13	106	74	123
N. current splices	26	22	1	5	4	50	24	51
N. cables proposed	11	34	13	53	17	44	47	58
N. splices proposed	8	27	13	31	13	35	40	48
N. cables rejected	0	0	0	1	1	1	4	4

Table 7. Network 2: Number of candidate DB = 12, with the existing network.

DB number	49	53	76	149	49–53	53–76	49–53–76
Total demand	490	410	240	220	900	650	1140
Nodes/Arcs	54/218	47/190	25/70	25/96	100/414	71/268	124/492
Init. solution	3919.33	3193.60	4112.27	2455.76	7112.93	7305.87	11225.20
Iterations rel. 1	114	143	200	117	114	200	200
Gap (%)	63	73	60	65	67	69	67
Iterations rel. 2	1000	441	1000	371	1000	1000	1000
FGap (%)	3	1	1	16	2	2	2
Cost	3919.33	3193.60	4112.27	2455.76	7112.93	7305.87	11225.20
Time (s)	204.28	101.88	121.38	51.65	345.23	269.82	453.21
N. current cables	49	41	9	25	90	50	99
N. current splices	27	15	1	0	43	18	46
N. cables proposed	11	12	14	10	23	26	37
N. splices proposed	8	8	12	11	16	20	28
N. cables rejected	0	0	1	0	0	1	1

Table 8. Influence of the number of arcs.

Number of arcs	Number of DB	Time (s)	Cost
330	4	3.91	31726.70
400	4	4.39	25557.93
500	4	4.51	20347.22
600	4	4.85	18356.22

5. Summary and conclusions

Decision support systems for engineering design in telecommunication require an hierarchical organization of optimization models. The rising complexity of integrated computer and communication systems imposes for the emerging problems a coherent divide-and-conquer solving strategy. Optimal and heuristic algorithms must be adapted to the different classes of models, with consistent transference of information and decision among the models. We have showed a typical hierarchy of decision levels, and we have focused here the lower levels of local access network design.

We insist on the importance of flow formulations for telecommunication network design. Our view is that this class of telecommunication problems is a flow extension of the Steiner tree problem, thus including as particular cases the alternative approaches based on the minimum spanning tree problem. We retain a capacitated single commodity network design problem, with an extension that is a referencial model to design the capilar part of the local access networks.

The distribution network design problem presented in our paper is an extremely difficult problem. As an extension of the Steiner tree problem it is a NP-hard combinatorial optimization problem. The need to install capacity in discrete levels and the incorporation of splicing costs make difficult the exact solution of the integer programming model. The computational results show that, from the theoretical point of view, it is very difficult to prove the optimality of the solutions. Heuristics is perhaps the only practical solution methodology for such a complex problem faced by telecommunications network providers.

The computational results show fast solution times, thus suggesting that the heuristic can solve the large scale problems met in practice. A careful comparison with human designs has also suggested that the system performs well. We have obtained good feasible solutions in low processing time, thus attending the planners expectation with regard to both the efficiency and usability quality criteria. The tests performed with dense networks demonstrate that, for the studied cases, the number of arcs does not have a strong influence on the computation time, and that the cost reduces significantly with the number of arcs. Both facts suggest that the designer should be incentivated to increase the density of the input graphs, in spite of the sparse nature of the urban networks. Automated mapping can help in this direction.

From the theoretical point of view, the results could be improved if it were complemented by new lower bounds, cutting plane algorithms and reduction tests (Bienstock and Günlük, 1995; Magnanti, Mirchandani, and Vachani, 1993). An anonymous referee remembered that, for a simpler problem in Balakrishnan, Magnanti, and Wong (1989), a "weak" lagrangian relaxation was used to generate upper and lower bounds. The duality gaps were substantial; however, when a more accurate lower bounding procedure was used, the upper bounds were shown to be good. There are several strategies that might be devised for this more complicated model, but most of the approaches appear very difficult to be implemented. Researchers have long recognized that, for capacitated problems, optimal solution methods that use only the standard flow-based problem formulation similar to the one described in this paper are ineffective. The literature suggests that strong formulations, based on results from polyedral combinatorics, appear to provide the best means to solve these problems. Our lagrangian relaxation satisfies the so-called integrality property. Hence, the best possible lagrangian lower bound cannot exceed the optimal linear programming value, which in this case is weak. The point that we study now for some simplified models is that, instead of working with heuristics based on weak single commodity flow formulations, we might perhaps have better results with strong multicommodity flow formulations. The combinatorial nature and the large scale induced by such redundant formulations might then be treated by Benders decomposition, as an alternative to the resolution of the very large scale linear programming relaxations.

References

Agrawal, A., P.N. Klein, and R. Ravi. (1994). "When Trees Collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks." Technical Report CS-90-32, Department of Computer Science, Brown University, Providence, Rhode Island 02912.

Aneja, Y.P. (1980). "An Integer Linear Programming Approach to Steiner Problem in Graphs," *Networks* 10, 167–178.

Balakrishnan, A., T.L. Magnanti, and P. Mirchandani. (1994). "Modeling and Heuristic Worst-Case Performance Analysis of Two-Level Network Design Problem," *Management Science* 40, 846–867.

Balakrishnan, A., T.L. Magnanti, A. Shulman, and R.T. Wong. (1991). "Models for Planning Capacity Expansion in Local Access Telecommunication Networks," *Annals of Operations Research* 33, 239–284.

Balakrishnan, A., T.L. Magnanti, and R.T. Wong. (1989). "A Dual-Ascent Procedure for Large-Scale Uncapacitated Network Design," *Operations Research* 37, 716–740.

Beasley, J.E. (1988). "An Algorithm for Solving Large Capacitated Warehouse Location Problems," *Journal of the Operational Research Society* 33, 314–325.

Beasley, J.E. (1989). "An sst-Based Algorithm for the Steiner Problem in Graphs," *Networks* 19, 1–16.

Bienstock, D. and O. Günlük. (1995). "Computational Experience with a Difficult Mixed-Integer Multicommodity Flow Problem," *Mathematical Programming* 68, 213–237.

Bradley, G.H., G.G. Brown, and G.W. Graves. (1977). "Design and Implementation of Large Scale Primal Transshipment Algorithms," *Management Science* 24, 1–34.

Duin, C.W. and A. Volgenant. (1989). "Reduction Tests for the Steiner Problem in Graphs," *Networks* 19, 549–567.

Fisher, M.L. (1981). "The Lagrangean Relaxation Method for Solving Integer Programming Problems," *Management Science* 27, 1–18.

Fisher, M.L. (1985). "An Application Oriented Guide to Lagrangean Relaxation," *Interfaces* 15, 10–21.

Gavish, B. (1982). "Topological Design of Centralized Computer Networks—Formulations and Algorithms," *Networks* 12, 355–377.

Gavish, B. (1983). "Formulations and Algorithms for the Capacitated Minimal Directed Tree," *Journal of the ACM* 30, 118–132.

Gavish, B. (1991). "Topological Design of Telecommunication Networks—Local Access Design Methods," *Annals of Operations Research* 33, 17–71.

Geoffrion, A.M. and R. McBride. (1978). "Lagrangean Relaxation Applied to Capacitated Facility Location Problems," *AIIE Transactions* 10, 40–47.

Hakimi, S.L. (1971). "Steiner's Problem in Graphs and its Implications," *Networks* 1, 113–133.

Hochbaum, D.S. and A. Segev. (1989). "Analysis of a Flow Problem with Fixed Charges," *Networks* 19, 291–312.

Kruskal, J.B., Jr. (1956). "On the Shortest Spanning Tree of a Graph and the Travelling Salesman Problem," *Proc. Amer. Math. Society* 7, 48–50.

Luna, H.P.L., N. Ziviani, and R.M.B. Cabral. (1987). "The Telephonic Switching Centre Network Problem: Formalization and Computational Experience," *Discrete Applied Mathematics* 18, 199–210.

Maculan, N. (1987). "The Steiner Problem in Graphs," *Annals of Discrete Mathematics* 31, 185–212.

Magnanti, T.L., P. Mirchandani, and R. Vachani. (1993). "The Convex Hull of Two Core Capacitated Network Design Problems," *Mathematical Programming* 60, 233–250.

Mateus, G.R., F.R.B. Cruz, and H.P.L. Luna. (1994). "Algorithm for Hierarchical Network Design," *Location Science* 2, 149–164.

Mateus, G.R. and H.P.L. Luna. (1992). "Decentralized Decision-Making and Capacitated Facility Location," *The Annals of Regional Science* 26, 361–377.

Minoux, M. (1989). "Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications," *Networks* 19, 313–360.

Mirzaian, A. (1985). "Lagrangian Relaxation for the Star-Star Concentrator Location Problem: Approximation Algorithm and Bounds," *Networks* 15, 1–20.

Sá, G. (1969). "Branch-and-Bound and Approximate Solutions to the Capacitated Plant Location Problem," *Operations Research* 17, 1005–1016.

Wong, R.T. (1984). "A Dual Ascent Algorithm for the Steiner Problem in Directed Graphs," *Mathematical Programming* 28, 271–287.

Journal of Heuristics, 6: 149–166 (2000)
© 2000 Kluwer Academic Publishers

LP-Based Heuristic Algorithms for Interconnecting Token Rings via Source Routing Bridges

V. SRIDHAR
Management Department, The Kogod College of Business Administration, American University, Washington DC 20016-8044, USA

JUNE S. PARK
Department of Management Sciences, 108 Pappajohn Business Administration Bldg., The University of Iowa, Iowa City, IA 52242-1000, USA
email: june-park@uiowa.edu

BEZALEL GAVISH
Owen Graduate School of Management, Vanderbilt University, Nashville, TN 37203, USA

Abstract

We develop a method to determine the topology of a network that interconnects a number of token rings using source routing bridges. The purpose is to compute a topology that provides low response delays for network users at a minimal cost of bridge installations. We formulate this network design problem as a mixed binary integer linear program. We develop effective heuristic algorithms. The algorithms exploit the topology and routing solutions of the linear programming relaxation in a sophisticated manner which we believe is new in the literature. The model incorporates performance issues, such as network stability, bridge overflow, back pressure effect and broadcast storm, that are specific to the underlying communication technology. By formally incorporating these performance issues, we tighten the model formulation and improve the quality of the LP bound considerably. Computational results are reported for problems with up to 20 token rings and 190 potential bridge locations.

Key Words: source routing bridge, capacitated network design, multicommodity flow, mixed integer linear program, LP-based heuristics

I. Introduction

A network of local area networks (LANs) can provide enterprise-wide connectivity and ease of information access for all users in an organization. The telecommunication industry offers public networking services such as frame relay and switched multi-megabit data service (SMDS) which can be used to interconnect LANs that are dispersed over a wide area. LANs dispersed within a geographically limited area such as a building or a campus are usually interconnected using bridges, LAN switches, routers, and private backbone networks such as fiber distributed data interface (FDDI) rings or more recently asynchronous transfer mode (ATM) switches.

The network administrator of an organization must configure the LAN interconnection network so that business applications requiring inter-LAN communications are supported without excessive delays. Each instance of network congestion leading to a slow-down or an unavailability of the network may cause the loss of many business transactions which could amount to millions of dollars. The total capital investment in a campus network might be only a couple of million dollars; nonetheless, in most organizations, the telecommunication budget is tightly controlled, and most network administrators strive to find a lower cost networking solution.

The purpose of this paper is to develop a method for configuring a network of token rings that are connected by *source routing* (SR) *bridges* (Perlman, 1992). The method is to help the network administrator find the interconnection of token ring LANs that provides high performance to users at a low cost of configuration.

There are about 20 million token ring nodes worldwide today (Love and Lynch, 1998). Token rings have a number of advantages over Ethernets including predictable maximum delay, higher degree of fault-tolerance, easier trouble shooting and network management. As a result, token rings are priced higher than Ethernets. However, equipment costs account for only about 20% of network expense, while network services account for 35% and administration 45% according to Strategic Networks Consulting (SNCI) (ASTRAL, 1996). Companies, especially Fortune 1000 companies whose mission-critical business operations heavily rely on mainframes and System Network Architecture (SNA)-based, online transaction processing systems, invest in the added value of token rings because there are quantifiable benefits to their businesses, and they often save money in overall cost of ownership.

SR bridges have been the primary method for transporting SNA traffic across token rings. Two bridging standards have been specified by the IEEE 802.1d Committee: SR bridge for connecting token rings and *spanning-tree routing bridge* (a.k.a. *transparent bridge*) for connecting both Ethernets and token rings. SR bridges have some significant advantages over transparent bridges: (i) SR bridges support mesh and even parallel redundant connections, and allow traffic load to be automatically shared among the available paths. Transparent bridges can utilize only a spanning-tree subset of all available bridges for routing, thus providing only one path for each communicating LAN pair. (ii) The time taken to recover from network faults may stretch into minutes with transparent bridges (because the bridges must reconfigure the spanning tree), whereas with SR bridges network traffic can automatically be routed around the faults.

In response to the increasing demand for greater bandwidths to individual users in token ring environments, token ring (TR) switches and high-speed token ring (HSTR) technologies have emerged (IEEE, 1998; Taylor, 1997). The TR switch is a multi-port bridge that implements source routing on a microprocessor chip. HSTRs can deliver a data rate of 100Mbps to 1Gbps to each station. HSTRs are mainly used to provide the dedicated link between TR switches, between a TR switch and a router, or between a TR switch and a high-performance server. All HSTRs are backward compatible with classical token rings and support source routing.

A typical example of using SR bridges is illustrated in figure 1. A cluster of token rings are interconnected by SR bridges. Each cluster is connected to a TR switch port or a router port. In a large campus area, several TR switches and routers may be deployed, and interconnected by HSTRs and/or a backbone (such as a FDDI ring or an ATM switch). The campus backbone is in turn connected to wide area networks such as the Internet, a frame relay network and

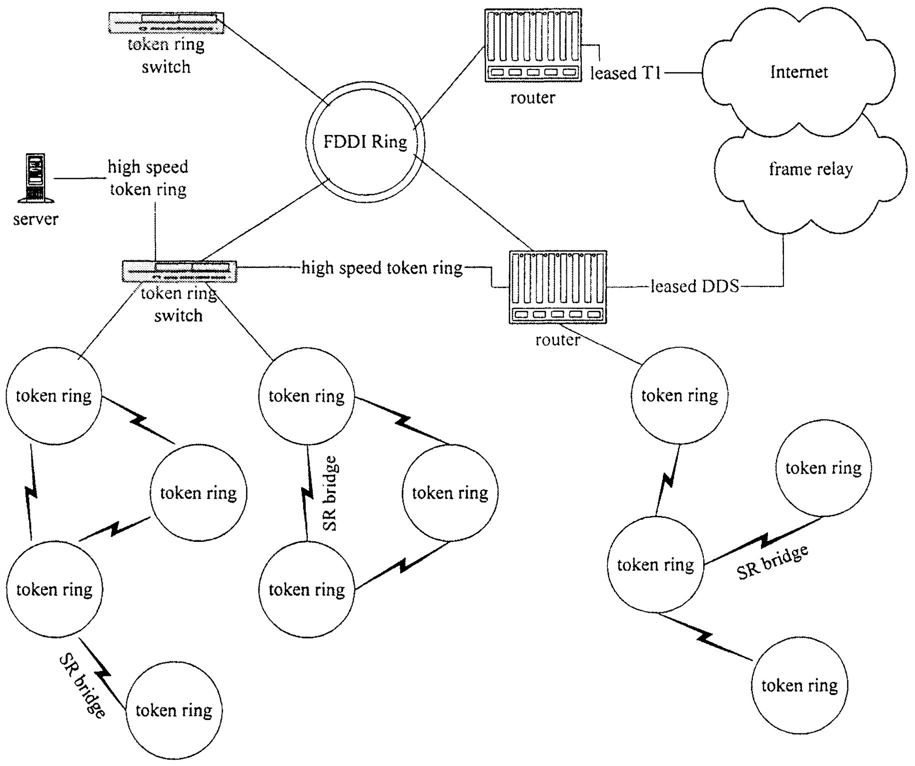


Figure 1. Campus network with source-routing bridged subnets.

point-to-point leased digital links. The network design problem considered in this paper focuses on the design of a SR bridged subnet. Each bridged subnet typically represents a separate geographic area (e.g., a building) or a small group of organizational units.

Most SNA-based organizations have invested in many SR bridges, while they need to purchase new TR switches and HSTRs as the communication demand increases. Interconnecting token rings using SR bridges may cost much less, but has a disadvantage compared to interconnecting them using TR switches. SR bridges create *transit traffic* (to be discussed in detail in the next section) on token rings that they connect. The transit traffic may consume significant portions of the token rings' bandwidths. Therefore the SR bridged network requires a careful topology design to avoid ring congestions. If properly designed, however, SR bridged subnets can support a considerable amount of inter-LAN communication at little extra cost—by utilizing existing bridges and unused slack capacities of token rings.

In the next section we describe the problem to be solved and review related studies in the literature. In Section III, we discuss performance issues that are unique to LAN interconnection networks, and more specifically, to the network connecting token rings using SR bridges. In Section IV we formulate the model. In Section V we present solution algorithms. Computational results are reported in Section VI. Section VII contains concluding remarks.

II. Problem description and literature review

As shown in figure 2, LANs and potential bridge locations can be regarded as nodes and arcs of a fully connected undirected graph, respectively. We provide a guideline for choosing a bridge product to consider for installation on each arc. We then present a formal model and heuristic algorithms to determine the topology of the SR bridged network. The topology is a connected and spanning subgraph of the fully connected graph. A possible topology is shown in figure 2 by bold lines.

The network design model is a mixed binary integer linear program. The model incorporates performance issues, such as network stability, bridge overflow, back pressure effect and broadcast storm, that are specific to the underlying communication technology. The network design heuristics find a topology which has a low cost of bridge installations and supports all inter-LAN communication requirements without causing network congestion. By formally incorporating the performance issues, we tighten the model formulation and improve the quality of the linear programming (LP) bound considerably. The heuristic algorithms exploit the topology and routing solutions of the LP relaxation in a sophisticated manner which we believe is new in the literature.

While there have been several studies on the interconnection of LANs using bridges (Ersoy and Panwar, 1993; Fetterolf and Anandalingam, 1991, 1992a, 1992b; Gupta and

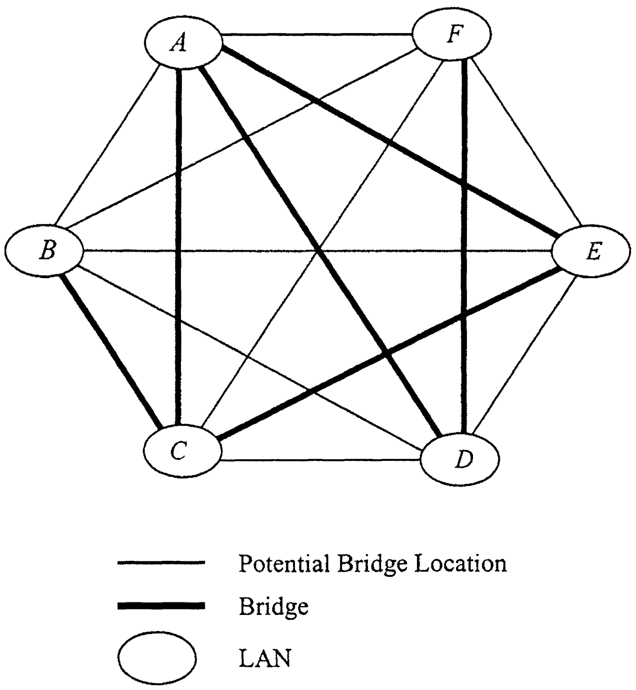


Figure 2. An example of token ring interconnection network.

Ross, 1991; Kaefer, 1995; Kaefer and Park, 1995, 1998), all these considered *spanning-tree routing bridges* (Perlman, 1992) and restricted the network topology to a tree. Our problem, on the other hand, must allow a mesh topology and dynamic routing since we consider source routing bridges. In a companion paper (LeBlanc et al., 1996) we solved the problem of interconnecting token rings via SR bridges using a simulated annealing algorithm which employed the flow deviation method (Fratta, Gerla, and Kleinrock, 1973) for the routing subproblem. In this paper, we develop LP-based heuristic algorithms that are computationally more efficient than the simulated annealing algorithm, and provides both upper and lower bounds for the optimal cost of the network. LAN interconnection design models with *survivability* considerations can be found in (Kaefer, 1995; Kaefer and Park 1995, 1998; Lee et al., 1995). Studies on the LAN-WAN interconnection using bridges, routers and ATM switches can be found in (Fetterolf and Anandalingam, 1992a, 1992b; Park et al., 1996; Boisson, Strilka, and Sutter, 1994).

Our problem falls in the category of capacitated network design problems (CNDP). CNDP is significantly different from the uncapacitated network design problem (UNDP) or the uncapacitated, survivable network design problem (USNDP) (Monma and Shallcross, 1989). The uncapacitated (survivable) NDP is to find a minimum cost (two-)connected graph spanning a given set of nodes. The CNDP must find simultaneously both the minimum-cost, connected spanning graph and the feasible routing of commodities on the graph so that none of the links (and nodes) are overloaded with traffic flow. The main contribution of this paper, on the algorithm side, is the re-routing algorithms (i.e., *two-hop*, *max-flow* and *LP re-routing* algorithms) and the way how they are incorporated into the DROP algorithm. All the re-routing algorithms utilize LP solutions in sophisticated manners which are effective and we believe are new in the literature. Linear programming and cutting planes were used in (Clarke and Anandalingam, 1995) to solve an USNDP, but in quite different contexts of model and solution strategy.

III. Network performance issues

In a bridged network of LANs, the *total traffic load* on a LAN consists of the *intra-LAN traffic load* plus the *inter-LAN traffic load*. The inter-LAN traffic load on a LAN consists not only of the message traffic originating or terminating at the LAN, but of all the inter-LAN traffic that goes past the LAN en route. For example, in figure 2, if a station in LAN A sends 100 message packets to a station in LAN B via LAN C, then these 100 packets add to the traffic load on LAN C. Thus, a LAN's channel capacity is consumed by three traffic components: intra-LAN traffic, inter-LAN traffic originating or terminating at the LAN, and inter-LAN traffic passing through the LAN. The last component will be called the *transit traffic* at a LAN.

For each LAN one can determine, using simulation or numerical approximation, the *effective channel capacity*, i.e., the maximum traffic load that allows message queues at LAN stations to attain an equilibrium in steady state (Park, Bartoszynski, and Rosenkrantz 1995, Park and Kang, 1993). The mean packet delay in a token ring increases only marginally as the traffic load approaches the effective channel capacity, but grows exponentially beyond that point (Bux, 1985). Therefore, the *total traffic load* on each LAN must not exceed the LAN's effective channel capacity. Since the intra-LAN traffic load and the inter-LAN traffic originating and terminating at each LAN are given as data, the amount of *transit traffic* entering each LAN must be controlled by the topology design and flow assignment. Once we determine the topology, however, traffic flow is assigned by the source routing mechanism so that congestion is avoided as much as possible.

Source routing is similar to the virtual circuit routing, where the communication path between source and destination stations (e.g., path $B \rightarrow C \rightarrow A \rightarrow D \rightarrow F$ from a station in LAN B to a station in LAN F in figure 2) is set up before starting each communication session. Source routing bridges provide better *congestion control* than spanning-tree routing bridges (Gerla and Kleinrock, 1988). However, the performance of a source-routing network can be severely degraded if a LAN or a bridge becomes congested (Bux and Grillo, 1985).

A bridge monitors all packets transmitted on both LANs it connects, and captures those packets whose source and destination are not on the same LAN. This is called bridge *filtering*. Captured packets may be buffered briefly and then transmitted on the other LAN, which is called *forwarding*. The speed of bridge filtering is usually fast enough not to cause packet losses. In fact, very few bridge products have a *per-port filtering rate* smaller than the interfacing LAN's effective channel capacity. On the other hand, a bridge's *forwarding rate* is often smaller than the interfacing LAN's effective channel capacity. A bridge discards arriving packets when its buffer is full. The discarded packets must be retransmitted by the sender, thus increasing the end-to-end delay. Therefore, to avoid network congestion and excessive response delay, the traffic passing through each bridge per unit time should be limited below the forwarding rate of the bridge.

It should be noted that the congestion in a LAN propagates to other LANs due to the limited buffer space in bridges. This is known as *back pressure effect* (Nishida et al., 1986). For example, in figure 2, if LAN A is congested, the buffers of bridges AC, AD and AE may fill up completely. Stations in other LANs will not succeed in sending messages to the congested LAN A due to the bridge overflows. Messages overflowed from the bridges must be retransmitted increasing the load in the message-originating LANs. The latter LANs may in turn become congested, and their congestion will then cause congestion of other LANs. To avoid this back pressure effect, the maximum throughput of each bridge must be kept smaller than the attached LANs' *slack capacities* as well as the bridge's forwarding rate. By the slack capacity we mean the effective channel capacity minus the intra-LAN traffic load.

Since bridges operate at the MAC layer, the source station has to perform *flooding* of *route discovery packets* in selecting the path for a communication session. In flooding (or called *all-routes broadcast*), the number of route discovery packets returning to the source LAN equals the number of simple paths connecting the source and destination LANs. Consequently, in interconnecting LANs using SR bridges, one of the major concerns is preventing *broadcast storm*—the overloading of LANs and bridges with an excessive number of route discovery packets. Broadcast storm can occur if the network topology is too dense. For example, in figure 2, there are only two simple paths ($A \rightarrow C \rightarrow B$) and ($A \rightarrow E \rightarrow C \rightarrow B$) between LANs A and B. However, if there are 20 LANs and they are fully connected using 190 bridges, the number of simple paths between each pair of LANs is 1.74×10^{16}. It takes more than 5×10^{4} years to transmit 1.74×10^{16} route discovery packets in a 4 Mbps ring, since the ring can transmit at most 10,000 route discovery packets per second (Soha and Perlman, 1988). The topology of the LAN interconnection network must be designed so as not to suffer from this problem. An effective way to prevent broadcast storm is to restrict the *network cardinality* as discussed in detail in the next section.

IV. Network design model

The model uses the following parameters: a set of LANs, V; the effective channel capacity (or a prespecified, maximum allowable throughput) l_i of each LAN $i \in V$; the intra-LAN traffic load, γ_i, on each LAN $i \in V$; inter-LAN traffic requirements λ_w for all $w \in W = \{(s, t) \in V \times V, s \neq t\}$ (assuming without loss of generality that all LANs communicate among one another); the forwarding rate, b_{ij}, of the candidate bridge considered for connecting LANs i and j; and the cost, c_{ij}, of installing the candidate bridge between LANs i and j which includes the bridge cost and wiring cost. Also needed to be predetermined is the maximum cardinality (or *density*) of the network topology (i.e., the total number of installed bridges) denoted by $\bar{n}$, which is used to prevent broadcast storm. The source and destination nodes of commodity w are denoted by $O(w)$ and $D(w)$, respectively.

Let $G = (V, A)$ be an undirected complete graph with $A = \{\{i, j\} : i, j \in V, i \neq j\}$. The network topology is defined by a connected and spanning subgraph $H = (V, E)$ of G. Bridges are installed on $E \subseteq A$. The model has two types of decision variables: topology design variables $\boldsymbol{y} = [y_{ij}, \{i, j\} \in A] \in \{0, 1\}^{|A|}$ such that $y_{ij} = 1$ if $\{i, j\} \in E$ and 0 otherwise; and flow variables $\boldsymbol{x} = [x_{ij}^v, v \in V, \{i, j\} \in A]$ that model the flow of commodity v on link $\{i, j\}$ in the direction from i to j where commodity v represents all the inter-LAN traffic supplied from node v and demanded at all other nodes, $V - \{v\}$. (In the sequel, the terms *bridge* and *link* are used interchangeably; so are *LAN* and *node*.) The supply of commodity v is given by $\sum_{w:O(w)=v} \lambda_w$. The demand of commodity v at node $i \neq v$ equals λ_w for $w = (v, i)$.

Note that flow costs are zero because the LAN interconnection network is privately owned. If, given any link, the unit cost of flow on the link is identical for all different commodities, then one can aggregate all commodities originating from the same node into a single commodity (Gondran and Minoux, 1988). Since zero flow costs are a special case of that condition, all the inter-LAN traffic originating from LAN v (i.e., all w such that $O(w) = v$) can be aggregated into a single commodity, which is also denoted by the node index v. This aggregation reduces the problem size considerably.

The network design problem, called [P], is formulated as follows:

$$\text{[P]} \quad \text{minimize } z = \sum_{\{i,j\} \in A} c_{ij} y_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in V \setminus i} \left(x_{ij}^v - x_{ji}^v\right) = \begin{cases} \sum_{w:O(w)=v} \lambda_w & \text{if } i = v \\ -\lambda_w \text{ for } w = (v, i) & \text{if } i \neq v \end{cases} \quad \forall i \in V - \{i^\circ\},\ \forall v \in V \tag{2}$$

$$\sum_{j \in V \setminus i} \sum_{v \in V \setminus i} x_{ij}^v \leq l_i - \gamma_i - \sum_{w:O(w)=i} \lambda_w - \sum_{w:D(w)=i} \lambda_w \qquad \forall i \in V \tag{3}$$

$$\sum_{v \in V} \left(x_{ij}^v + x_{ji}^v\right) \leq y_{ij} \min(b_{ij}, l_i - \gamma_i, l_j - \gamma_j) \quad \forall \{i, j\} \in A \tag{4}$$

$$\sum_{\{i,j\} \in A} y_{ij} \leq \bar{n} \tag{5}$$

$$y_{ij} \in \{0, 1\} \quad \forall \{i, j\} \in A \tag{6}$$

$$x_{ij}^v, x_{ji}^v \geq 0 \quad \forall \{i, j\} \in A, \ \forall v \in V. \tag{7}$$

The objective (1) captures the total cost of installing bridges. Constraints (2) specify the flow conservation equation for each commodity at each node. Note that for each commodity we keep only $(|V| - 1)$ equations ignoring one equation corresponding to an arbitrarily chosen node i°. By deleting one equation per commodity, we can remove linear dependency in constraint set (2), which often greatly facilitate solving the LP relaxation of [P]. Constraints (3) are node capacity constraints. The left-hand side of inequality (3) represents the total out-flow of *transit traffic* from LAN i (which equals the total in-flow of transit traffic into the LAN). The right-hand side represents the *residual channel capacity* of the LAN that can be used to accommodate *transit traffic* flow.

In order to prevent the back pressure effect, arc capacity constraints (4) require that the flow through a bridge must be limited below its *maximum allowable throughput rate* given by $\min(b_{ij}, l_i - \gamma_i, l_j - \gamma_j)$. Even if we use constraints $\sum_{v \in V}(x_{ij}^v + x_{ji}^v) \leq y_{ij} b_{ij}$ in place of (4), they together with constraints (3) can prevent the back pressure effect; in other words, those constraints and constraints (3) together mathematically imply constraints (4). Using the logical throughput bound instead of the physical forwarding capacity, b_{ij}, in constraints (4), however, has a profound effect on the quality of the solution of the LP relaxation. For example, in our computational experiments with 10-node problems, the optimal value of the LP relaxation was found to increase by up to 75% by using the throughput bound instead of the physical forwarding capacity.

The candidate bridge product considered for installation on arc $\{i, j\}$ is determined *a priori* using the following criteria: First, the per-port filtering rates of the bridge must be greater than l_i and l_j, respectively, since otherwise a packet loss may occur frequently. The forwarding rate b_{ij} must be greater than the smaller of the two incident LANs' slack capacities, i.e., $b_{ij} > \min(l_i - \gamma_i, l_j - \gamma_j)$. Otherwise, LANs' bandwidth cannot be fully utilized because bridges become the bottleneck. The network administrator should select the lowest cost bridge product meeting the two criteria (and possibly additional criteria) for each pair of LANs. If the two selection criteria are satisfied, the right-hand side of constraints (4) can be replaced by $y_{ij} \min(l_i - \gamma_i, l_j - \gamma_j)$.

Constraint (5) restricts the network cardinality to prevent broadcast storm. Recall that broadcast storm is caused by an excessive number of route discovery packets returning to the source LAN. That number is equal to the number of simple paths connecting the origin and destination LANs. Let $S(w)$ be the number of simple paths between an origin-destination (OD) pair w. Counting $S(w)$ for an OD pair is a *#P-complete* problem (Valiant, 1979). We developed an exhaustive enumeration algorithm, which is a modification of the k-shortest path algorithm (Yen, 1971), to count $S(w)$ in an arbitrary graph. Define $\bar{S} = \max_{w \in W} S(w)$. We want to limit $\bar{S}$ so that no LANs are overloaded by broadcast storm.

Table 1. Effect of topology on broadcast storm: relationship between $|V|$, $|E|$, and $\bar{S}$.

$\|V\|$	$\|E\|$	$\bar{S}$	Delay
10	20	265	0.0338
10	25	1535	0.1957
10	30	5655	0.7210
10	35	17900	2.2823
10	40	51505[a]	6.5669
10	45	109601	13.9740
15	20	50	0.0064
15	25	514	0.0655
15	30	3824	0.4876
15	35	25009[a]	3.1886
15	40	119991[a]	15.2990
15	45	488963[a]	62.3430
15	50	1962014[a]	250.1600
20	20	2	0.0003
20	25	40	0.0051
20	30	620	0.0791
20	35	7612	0.9705
20	40	48768[a]	6.2179
20	45	365629[a]	40.6618
20	50	1911253[a]	243.6848

[a]The path counting algorithm was stopped after two hours of CPU time. Therefore, the reported value of $\bar{S}$ is actually a lower bound on $\bar{S}$.

Given the network size (i.e., number of nodes), quantity $\bar{S}$ depends on the network topology. Intuitively, the denser the topology, the greater $\bar{S}$. The effect of network topology on $\bar{S}$ is studied in Sridhar (1994). We report here a summary of results from the numerical experiments conducted in that thesis. First, topology, $H = (V, E)$, was represented by three parameters: number of nodes $|V|$, number of links $|E|$, and $\sigma(d)$ representing the standard deviation of node degrees. In the numerical experiments, networks with varying parameters were generated and $\bar{S}$ was computed for each network.

From extensive experiments we found that, for any $|V|$ and $|E|$, the greatest $\bar{S}$ was observed when $\sigma(d)$ was between 0.5 and 1. In Table 1, we report $\bar{S}$ computed for networks with varying $|V|$ and $|E|$. All networks reported on in Table 1 had $\sigma(d)$ in the range of [0.5, 1]. The table also shows the time delay (in seconds) in processing $\bar{S}$ route discovery packets on a 4Mbps ring, which can be regarded the worst case delay for setting up a path.

Suppose a limit of 5 seconds is set by the network administrator for the route discovery delay. It can be seen from Table 1 that if $|E| \leq 35$, then the delay is less than 5 seconds in a

10-node network. It is interesting to see that to meet the 5-second delay limit we also need $|E| \leq 35$ in 15 and 20-node networks. In computational experiments (see Section VI), we limited network cardinality $|E|$ below 35.

It should be noted that another way to prevent broadcast storm is to set a limit on the number of hops in the routing path. However, the IEEE 802.5 Standard allows up to 13 hops for source routing. With this hop limit, every simple path can be used in a network with up to 14 nodes. As such, the standard hop limit does not prevent broadcast storm.

V. Network design heuristics

Like other capacitated network design problems (Magnanti, Mirchandani, and Vachani, 1995), the LP relaxation of [P] has a significantly lower value than [P]. For example, the LP optimal value was only about 20% of the optimal value of [P] for a 10-node problem. Due to the large integrality gap, when we applied the mixed integer programming code of Optimization Subroutine Library (OSL) (IBM, 1991), it could not find an optimal solution for 10-node problems in ten hours of CPU time.

Therefore, we developed a solution procedure consisting of three successive heuristic algorithms called HEUR1 to HEUR3, which can generate a high-quality feasible topology starting with a LP solution within a reasonable amount of CPU time for problems with up to 20 nodes and 190 potential bridge locations. Before describing these heuristics, we define some notation first: OS[$\cdot$] denotes an optimal solution of problem [$\cdot$], and OV[$\cdot$] the optimal value of problem [$\cdot$]; [LP] denote the LP relaxation of [P]; [P($\boldsymbol{y}$)] denotes problem [P] with constraint (5) and (6) dropped and with $\boldsymbol{y}$ fixed; and finally, a topology $E \subseteq A$ is said to be *routing-feasible* if the corresponding characteristic vector $\boldsymbol{y}$ makes [P($\boldsymbol{y}$)] feasible. A topology $E \subseteq A$ is *feasible* to [P] if E is *routing-feasible* and $|E| \leq \bar{n}$. Note that [P($\boldsymbol{y}$)] is a linear program that finds a feasible routing of flow $\boldsymbol{x}$ not violating node and link capacities (3) and (4) given a topology $\boldsymbol{y}$.

First, we solve [LP]. HEUR1 uses the fractional $\boldsymbol{y}$ solution in the LP optimal solution as a measure of the *merit* of installing a bridge on each arc. Using this measure, HEUR1 first constructs a spanning tree on G and then builds a feasible topology E by expanding the tree. Because of constraint (5) (i.e., $|E| \leq \bar{n}$), constructing a feasible topology is difficult when the overall inter-LAN traffic demand is high relative to the total slack capacity of LANs. If HEUR1 finds a feasible topology, then HEUR2 (*Drop* heuristic) and HEUR3 (*Exchange* heuristic) are applied in succession to reduce the cost of topology while maintaining the feasibility. If HEUR1 fails in finding a feasible topology, then HEUR2 starts with a fully connected network and tries to drop as many links as possible without violating *routing-feasibility* (i.e., without violating constraints (2) to (4)). If the final topology meets constraint (5), then HEUR3 is invoked to reduce the cost. HEUR1 and HEUR2 do not guarantee finding a feasible solution for a feasible problem. However, they could find feasible solutions even for the most difficult problem instances in our computational experiments as reported in the next section.

In the following description of solution algorithms, $\boldsymbol{y}$ always represents the characteristic vector of E, and an update of $\boldsymbol{y}$ always implies the corresponding update of E, and vice versa. The cost of topology E is denoted by $C(E) = \sum_{\{i,j\}\in E} c_{ij}$.

Step 1. (*Construction of an MST*) Solve [LP]. Let $(\bar{x}, \bar{y})$ = OS[LP] and $\underline{z}$ = OV[LP]. Assign a weight of $1/\bar{y}_a$ to each arc $a \in A$. Using these arc weights find a MST, $T = (V, E)$, on G. If $C(E) < \underline{z}$, add more bridges in Step 2 since tree T is infeasible. Otherwise, go to Step 3.

Step 2. (*Topology Expansion*) Sort arc set $(A - E)$ in decreasing order of $\bar{y}_a$. Let $L = (a_1, a_2, \ldots, a_{|A-E|})$ be the sorted list of arcs. Let $h = \min[\bar{n} - |E|, \min\{m : \sum_{l=1}^{m} c_{a_l} + \sum_{a \in E} c_a \geq \underline{z}\}]$. Update E by setting $y_{a_l} = 1$ for $1 \leq l \leq h$ (so that $|E| \leq \bar{n}$ and $C(E) \geq \underline{z}$).

Step 3. (*Feasibility Test*) Check if [P($\boldsymbol{y}$)] is feasible using the dual simplex method. (From the second time this step is invoked, start dual simplex with the previously saved basis B which is dual feasible to [P($\boldsymbol{y}$)], sharply reducing the number of pivots required). If feasible, save E and go to HEUR2. If not feasible, do the following: if $|E| = \bar{n}$, set $E = A$ (a fully connected topology) and go to HEUR2; otherwise, save the final basis as B and go to *Step 4* to add more bridges.

Step 4. (*Topology Expansion*) To select arcs to add bridges to, compute $k = \min[\bar{n} - |E|, \max\{h + 1, \max\{l > h : y_{a_l} \geq \alpha y_{a_h}\}\}]$ where α is a tunable parameter between 0 and 1 (With a smaller α, more arcs are selected for adding bridges). Update E by setting $y_{a_l} = 1$ for $h + 1 \leq l \leq k$. Set $h = k$. Go to *Step 3*.

Figure 3. HEUR1—Build heuristic.

A step-by-step description of HEUR1 is given in figure 3. In Step 1, in order to find a high-quality LP solution whose value is close to OV[P], one may employ a disaggregate formulation of [LP] (Wolsey, 1989) using flow variables $[x_{ij}^w, w \in W, \{i, j\} \in A]$ instead of $[x_{ij}^v, v \in V, \{i, j\} \in A]$ which model the flow of commodity w on arc $\{i, j\}$ in the direction from i to j. Also, one can employ a cutting plane procedure that dynamically adds to [LP] valid inequalities such as *cut set inequalities* (Magnanti, Mirchandani, and Vachani, 1995) and inequalities $y_{ij} \geq (x_{ij}^w + x_{ji}^w)/\lambda_w$ for $w \in W$, $\{i, j\} \in A$ (Padberg, Van Roy, and Wolsey, 1985) (see Sridhar (1994) for more details). However, the performance of our heuristics is not very sensitive to the quality of the LP solution which HEUR1 starts with, as can be seen from computational results in the next section.

HEUR2 iteratively applies a *Drop* operation which tries to delete the least utilized link by re-routing the flow on that link to other links. Define $\boldsymbol{f} = [f_{ij}, \{i, j\} \in E : f_{ij} = \sum_{v \in V}(x_{ij}^v + x_{ji}^v)]$ to represent the inter-LAN traffic flow on $H = (V, E)$. Given a topology E, HEUR2 determines an optimal routing $\boldsymbol{f}$ on H by solving the routing model [P*($\boldsymbol{y}$)] which is problem [P($\boldsymbol{y}$)] with an objective function $\sum_{i \in V} \sum_{j \in V \setminus i} \{c_{ij} / \min(b_{ij}, l_i - \gamma_i, l_j - \gamma_j)\} \sum_{v \in V} (x_{ij}^v + x_{ji}^v)$ to minimize. Problem [P*($\boldsymbol{y}$)] is a *minimum cost multicommodity flow problem* which induces flow of commodities into those links whose setup costs are cheaper relative to their capacities. After obtaining $\boldsymbol{x}$ = OS[P*($\boldsymbol{y}$)], HEUR2 tries to remove existing links in E one at a time starting with the arc with the least amount of flow and continuing in the increasing order of f_{ij}.

Let E' be the topology generated by deleting link $\{s, t\}$ from a *routing-feasible E*. To determine routing-feasibility of E' we first check a *necessary* condition $-E'$ must be connected and spanning. One can use an MST algorithm to check connectivity (Sridhar, 1994). If E' is connected and spanning, compute $\boldsymbol{\eta} = [\eta_i, i \in V]$ where $\eta_i = l_i - \gamma_i - \sum_{w:O(w)=i} \lambda_w - \sum_{w:D(w)=i} \lambda_w - \sum_{v \in V \setminus i} \sum_{j \in V \setminus i} x_{ij}^v$ represents the residual node capacity after supporting the flow $\boldsymbol{f}$ (i.e., slacks of constraints (3) at OS[P*($\boldsymbol{y}$)]). Also compute $\boldsymbol{\kappa} = [\kappa_{ij}, \{i, j\} \in E]$

where $\kappa_{ij} = \min(b_{ij}, l_i - \gamma_i, l_j - \gamma_j) - f_{ij}$ represents the residual link capacity (i.e., *slacks* of constraints (4) at OS[P*(y)]).

We can then check any of the following three *sufficient* conditions for routing-feasibility of E'. Suppose E is routing-feasible and $E' = E - \{s, t\}$. Then E' is routing-feasible if

(i) the current flow f_{st} on the deleted link $\{s, t\}$ can be re-routed using two-hop (s, t)-paths in E' without violating link and node capacities κ and η, or
(ii) the maximum flow from s to t on the network $H' = (V, E')$ with link capacities κ and node capacities η is greater than or equal to f_{st}, or
(iii) problem [P*(y', $\tilde{\boldsymbol{x}}_{st}$)] constructed as follows is feasible: Given an optimal $\boldsymbol{x}$ for [P*(y)], define $V_1 = \{v : x^v_{st} = 0\}$, i.e., a set of commodities not flowing on link $\{s, t\}$. Define $\tilde{\boldsymbol{x}}_{st} = [x^v_{ij}, v \in V_1, \{i, j\} \in E]$, i.e., the flow of commodities V_1 on $H = (V, E)$. Then [P*(y', $\tilde{\boldsymbol{x}}_{st}$)] is problem [P*($y'$)] with $\tilde{\boldsymbol{x}}_{st}$ fixed.

Condition (ii) implies (i), but not vice versa. Condition (iii) implies (ii), but not vice versa. Condition (ii) can be checked by solving the maximum flow problem with given link and node capacities (see Gondran and Minoux (1988) for an algorithm to solve the maximum flow problem with both link and node capacities). The complexity of checking conditions (i) and (ii) for a deleted link is of order $O(|V|)$ and $O(|E'| \cdot |V|^2)$, respectively. Checking condition (iii) requires solving [P*(y', $\tilde{\boldsymbol{x}}_{st}$)] by the simplex method. We call the applications of conditions (i), (ii) and (iii) for deleting a link *two-hop re-routing, max-flow re-routing*, and *LP re-routing*, respectively. In HEUR2 we first apply the *two-hop re-routing* to every link in E to see if each can be deleted, and then use the *max-flow re-routing* and then the *LP re-routing* to see if any of the remaining links can be further deleted.

The two-hop re-routing or the max-flow re-routing leaves superfluous commodity flows on the network after deleting a link, thus consuming more link and node capacities than are necessary. Therefore, if the two-hop (resp. max-flow) re-routing, after having deleted one or more links, encounters a link not removable, the optimal routing on the current topology is recalculated (using Step 1 in the algorithm) to clear the network of superfluous flows. After that, the two-hop (resp. max-flow) re-routing is applied again to existing links to check if more links can be deleted. The algorithm terminates if none of the existing links can be deleted by the three re-routing methods. The algorithm of HEUR2 is provided in figure 4. HEUR3 iteratively performs an Exchange operation which replaces an existing link by a lower-cost link while maintaining feasibility of the topology; see figure 5.

VI. Computational results

In this section we report our computational experience with the network design heuristics described in the previous section. In the computational experiment we conduct a stress test with the heuristic algorithms solving a range of easy to very difficult problem instances to measure the solution quality and the algorithm run time. All the algorithms were coded in FORTRAN and run on an IBM 3090 model 400J. IBM's OSL (IBM, 1991) was used for solving linear programs.

Test problems are generated as follows. All nodes are assumed to be 4Mbps token rings, and l_i was set to 3.2Mbps, or 80% of the data rate. The bridge forwarding rate, b_{ij}, was set

```
Step 0. (Initialization) Let $E$ be the topology passed from HEUR1.
Step 1. (Optimal Routing on $H = (V, E)$) If $E$ is a tree, go to Step 6. Otherwise, solve $[\mathrm{P}^*(\boldsymbol{y})]$ by the dual simplex method (using the previously saved basis $B$ as the starting basis from the second time this step is invoked). Let $\boldsymbol{x}$ and $B$ be the optimal solution and the optimal basis of $[\mathrm{P}^*(\boldsymbol{y})]$. Compute $\boldsymbol{f}$ using $\boldsymbol{x}$. Update arc and node capacities $\kappa$ and $\eta$.
Step 2. (Arc Sorting) Sort arc set $E$ in increasing order of $f_{ij}$. Let $m = |E|$, and let $L = (a_1, a_2, \ldots, a_m)$ be the sorted list of arcs. Go to Step 4 if Step 3 has been completed.
Step 3. (Topology Reduction by Two-Hop Re-routing) If all arcs in list $L$ have been considered for removal by two-hop re-routing, then Step 3 is completed; go to Step 1 if any arc has been removed since the last computation of the optimal routing by Step 1, and go to Step 4 otherwise. Let $\{s, t\}$ represent the first arc in list $L$ that has not been considered yet. If the new candidate topology $E' = E - \{\{s, t\}\}$ is not connected, repeat Step 3. Otherwise, apply two-hop re-routing to determine if $E'$ is routing-feasible. If so, update $E = E'$, update $\kappa$ and $\eta$ by subtracting new flows along the two-hop $(s, t)$-paths, and repeat Step 3. If not, either go to Step 1 or repeat Step 3 depending on whether any arc in $L$ has been removed since the last execution of Step 1.
Step 4. (Topology Reduction by Max-Flow Re-Routing) This step is similar to Step 3. If all arcs in $L$ have been considered, go to Step 5. Let $\{s, t\}$ be the first arc in $L$ not considered yet. If $E' = E - \{\{s, t\}\}$ is not connected, repeat Step 4. Otherwise, apply max-flow re-routing. If $E'$ is routing-feasible, update $E = E'$, update $\kappa$ and $\eta$ by subtracting new flows along the $(s, t)$-paths, and repeat Step 4. If not, either go to Step 1 or repeat Step 4.
Step 5. (Topology Reduction by LP Re-Routing) Set $\{s, t\} = \arg\min\{f_{ij}, \{i, j\} \in E\}$. If $E' = E - \{\{s, t\}\}$ is not connected, go to Step 6. Otherwise, apply LP re-routing to determine if $E' = E - \{\{s, t\}\}$ is routing-feasible. If so, update $\boldsymbol{x} = \boldsymbol{x} + \mathrm{OS}[\mathrm{P}^*(\boldsymbol{y}', \tilde{\boldsymbol{x}}_{st})]$, compute $\boldsymbol{f}$ using $\boldsymbol{x}$, update $E = E'$, and repeat Step 5; otherwise, go to Step 6.
Step 6. (Termination) If $|E| > \bar{n}$, stop: a feasible topology has not been found. Otherwise, set $\bar{z} = C(E)$, save $E$, and go to Step 0 in HEUR3.
```

Figure 4. HEUR2—Drop heuristic.

```
Step 0. (Initialization) Let $E$ be the feasible topology obtained from HEUR2. Let $\underline{z}$ and $\bar{z}$ be the lower and upper bounds obtained from HEUR1 and HEUR2, respectively.
Step 1. (Arc Sorting) Let $Q = \{(a_1, a_2) : a_1 \in E, a_2 \in A - E, c_{a_1} \geq c_{a_2}, C(E \cup \{a_2\} - \{a_1\}) \geq \underline{z}\}$. Sort $Q$ in decreasing order of $c_{a_1} - c_{a_2}$. Let $L = (q_1, q_2, \ldots, q_{|Q|})$ be the sorted list. Set $l = 1$.
Step 2. (Topology Change by Link Exchange) If $l = |Q|$, then stop. Let $q_l = (a_1, a_2)$. Set $E' = E \cup \{a_2\} - \{a_1\}$. If $E'$ is not connected, then set $l = l + 1$ and repeat Step 2. Check feasibility of $[\mathrm{P}(\boldsymbol{y}')]$ by the dual simplex method starting with the previously saved basis $B$. If feasible, save the final basis as $B$, update $E = E'$, update $\bar{z} = C(E)$ if $C(E) < \bar{z}$, set $l = l + 1$, and go to Step 1. Otherwise, set $l = l + 1$ and repeat Step 2.
```

Figure 5. HEUR3—Exchange heuristic.

to 4Mbps for all arcs. The upper bound on the total number of bridges, $\bar{n}$, was set to 35 for all test problems. The remaining parameters $|V|$, c_{ij}, λ_w and γ_i were varied across problem instances.

Through computational experiments we found two factors to have most significant impact on the problem difficulty and the structure of the feasible topology: the problem size $|V|$, and the ratio of the total inter-LAN traffic demand, $\sum_{w \in W} \lambda_w$, to the total slack capacity of LANs, $\sum_{i \in V}(l_i - \gamma_i)$ (called the *DC ratio* in short). Three different problem sizes are considered: 10, 15 and 20 nodes. All pairs of nodes are allowed to be connected so that there are 45, 105 and 190 zero-one variables, respectively, for the three problem sizes. Four different ranges of the DC ratio are considered which are labeled as 'A' through 'D': A[10%, 15%], B[40%, 45%], C[70%, 75%] and D[90%, 95%]. Bridge installation costs c_{ij} are generated based on the information in bridge vendor catalogs (Information Handling Services 1993, Ziff-Davis Publishing, 1993).

A total of 8 problems are generated for each network size by varying the DC ratio across the four categories and using two different seed numbers for the random number generator. Each problem instance is identified by a code such as '10B2'. This ID refers to the second random instance of the 10-node problem with DC ratio in B range.

Table 2 shows the gap between the cost of the feasible network obtained by the heuristics and the cost of the LP relaxation solution (i.e., the LP bound) measured as a percentage of the lower bound, and the number of bridges in the feasible network. The LP bound was obtained by solving [LP] with the two types of valid inequalities mentioned in Section V added dynamically. We used the disaggregate formulation of [LP] for 10 and 15 node problems. We used the aggregate formulation for 20-node problems, since otherwise it required an excessively long CPU time.

The network design heuristics are very effective when the DC ratio is either very low or very high. When DC ratio is in A or D range, the gap is less than 5% for all problems

Table 2. Performance of the network design heuristics.

Problem ID	Number of bridges	Gap (%) = 100(feasible value-LP bound)/LP bound
10A1	9	0.0
10A2	9	0.7
10B1	10	20.8
10B2	10	14.3
10C1	18	111.6
10C2	18	97.4
10D1	32	4.5
10D2	33	3.2
15A1	14	1.1
15A2	14	0.7
15B1	19	42.3
15B2	18	44.2
15C1	34	85.2
15C2	33	126.0
20A1	19	1.3
20A2	19	2.7
20B1	26	64.7
20B2	27	87.9

with 10 to 20 nodes. Problems with DC ratio in B and C ranges are most difficult to solve, showing the gap ranging between 14–126%. However, from observing the trends of both upper and lower bounds over the entire range of DC ratio, it seems clear that large gaps are due to poor qualities of lower bounds. With 10 node problems, for instance, the upper bound increases steadily as the DC ratio increases. However, the lower bound shows little increase over the ranges A through C and then shows a sudden jump in range D, resulting in large gaps for B and C ranges. As such, the network design heuristics seem to yield near-minimum cost topologies for all problem categories.

As expected, the number of installed bridges increases as DC ratio increases. With DC ratio in A range, the network topology is a tree for all problem sizes. At a higher DC ratio, the network cardinality is greater for larger problems. For 10-node problems, the network cardinality reaches 10 when the DC ratio is in B range, 18 in C range and 32–33 in D range; for 15-node problems, it reaches 18–19 in B range, and 33–34 in C range; and for 20-node problems, 26–27 in B range. With $\bar{n}$ set to 35 for all test problems, it seems that 15-node (resp. 20-node) problems become infeasible once the DC ratio increases to D (resp. C) range.

Table 3 shows the improvement of the network cost by major steps of the heuristics. Each figure in the table is an average across the two random problems in each problem category. Columns (1) to (3) show the cost improvement by the two-hop, the max-flow and the LP re-routing in HEUR2, respectively, and Column (4) by the exchange operation in HEUR3.

For problems with DC ratio in A or D range and with 10 or 15 nodes, the initial feasible topology obtained from HEUR1 is very good as can be observed from Table 3, and little

Table 3. Percentage cost improvements in various steps of the network design heuristics.

Problem category	HEUR2 *Step 3* (two-hop) (1)	HEUR2 *Step 4* (max-flow) (2)	HEUR2 *Step 5* (LP) (3)	HEUR3 (link exchange) (4)	Total (5)
10A	0.0	0.0	0.0	2.6	2.6
10B	10.3	0.0	0.0	0.0	10.3
10C	10.8	0.0	0.0	0.0	10.8
10D	2.3	0.0	0.0	0.2	2.5
15A	0.0	0.0	0.0	1.4	1.4
15B	6.3	0.0	1.9	0.0	8.2
15C[a]	66.7	7.8	1.7	0.7	76.9
20A	22.2	0.0	0.0	24.2	46.4
20B[a]	81.3	6.4	0.3	0.0	88.0

Column (1): 100{Initial network cost obtained from HEUR1—Cost after *Step 3* of HEUR2}/Initial cost.
Column (2): 100{Cost after *Step 3* of HEUR2—Cost after *Step 4* of HEUR2}/Initial cost.
Column (3): 100{Cost after *Step 4* of HEUR2—Cost at completion of HEUR2}/Initial cost.
Column (4): 100{Cost at completion of HEUR2—Cost at completion of HEUR3}/Initial cost.
Column (5): Total cost improvement = (1) + (2) + (3) + (4).
[a]HEUR1 did not find a feasible topology; therefore, HEUR2 started with a fully connected network.

improvement is made by the subsequent heuristic algorithms. For problems with DC ratio in B and C ranges, the two-hop re-routing is very effective for reducing the network cardinality and cost. The max-flow re-routing could eliminate considerably more links when HEUR2 started with a fully connected network. The LP re-routing was not so effective in further removing the links. The exchange operation in HEUR3 also contributes to cost reduction. The contribution was remarkable for a 20-node problem.

The CPU time spent for obtaining a LP solution in HEUR1 was 20 seconds–4 minutes and 7 minutes–3.5 hours for 10 and 15 node problems, respectively, for which the disaggregate formulation of [LP] was used for strengthening the LP bound. The CPU time was 1–6 minutes for 20-node problems for which the aggregate formulation was used. The quality of the initial LP solution does not have a critical effect on the quality of the feasible solution found by the heuristics as can be seen in Tables 2 and 3. For instance, for problem category $20A$ which started with poorer LP solution than problem categories $10A$ and $15A$, the heuristics attained much greater cost reductions (see Table 3) leading to a gap of less than 3% (see Table 2). Also, for problem categories $15C$ and $20B$ for which HEUR1 could not find feasible topologies due to poor qualities of starting LP solutions, HEUR2 effectively found low-cost feasible topologies by removing as many as 72 and 164 links, respectively, from the fully connected topology.

We limited the total run time of the heuristics to 3 hours for each problem. The CPU time spent by the heuristics (excluding the time for obtaining the initial LP solution) was 1 second–2 minutes, 5 seconds–45 minutes and 6 minutes–3 hours for 10, 15 and 20-node problems, respectively.

VII. Conclusion

We have presented a topology design model for interconnecting token rings using source routing bridges. A feasible solution to the model yields a topology that guarantees no traffic congestion. We found that the feasible topology is extremely sensitive to the traffic condition measured by the ratio of total inter-LAN traffic demand to total slack capacity of LANs (DC ratio). Bridges use LANs' slack capacities in routing inter-LAN traffic. Since the bridge throughput should be limited below the slack capacity of each incident LAN to avoid back pressure effect, it is the total slack capacity of LANs that most critically affects the cardinality and cost of a feasible topology.

Most SNA-based organizations already have investments in SR bridges. Therefore, there is a strong economic motivation to utilize them as a substitute for token ring switches. The bridged subnet is especially attractive when the DC ratio is small so that a high-performance subnet can be constructed using a small number of bridges. Our computational experience shows that when the DC ratio is below 20%, the optimal topology is a tree. Our topology design heuristics are effective in finding a near-minimal cost, high-performance topology whenever the DC ratio is not too high to admit a feasible topology. Moreover, the heuristics are robust, i.e., not much dependent on the quality of the starting LP solution.

In cases when the heuristics cannot find a feasible topology due to a high DC ratio and a tight limit on the network cardinality, one should consider using a token ring switch in addition to bridges. It should be noted that if a feasible topology cannot be found using source routing bridges, then the spanning-tree routing bridge is not a feasible option, either.

Spanning-tree routing bridges cannot be used unless there exists at least one tree topology that can support all inter-LAN traffic, and the non-existence of a feasible topology with source routing bridges implies the non-existence of a feasible tree topology.

References

ASTRAL (Alliance for Strategic Token Ring Advancement and Leadership). (1996). "An Educational Guide to Token Ring." (http://www.astral.org/astralwp.html).

Boisson, N., O. Strilka, and A. Sutter. (1994). "A Simulated Annealing Algorithm for LANs Interconnection Networks Design." In *Proc. 2nd Int'l Conf. Telecom. Syst.*, Vanderbilt Univ., Nashville, TN.

Bux, W. (1985). "Performance Issues." In G. Goos and J. Hartmanis (eds.), *Lecture Notes in Computer Science 184: Local Area Networks: An Advanced Course.* Berlin, Germany: Springer-Verlag, pp. 108–161.

Bux, W. and D. Grillo. (1985). "Flow Control in Local Area Networks of Interconnected Token Rings," *IEEE Trans. Commun.* 33, 1058–1065.

Clarke, L.W. and G. Anandalingam. (1995). "A Bootstrap Heuristic for Designing Minimum Cost Survivable Networks," *Computers and Operations Research* 22, 921–934.

Ersoy, C. and S.S. Panwar. (1993). "Topological Design of Interconnected LAN/MAN Networks," *IEEE J. Select. Areas Commun.* 11, 1172–1182.

Fetterolf, P.C. and G. Anandalingam. (1991). "Optimizing Interconnection of Local Area Networks: An Approach Using Simulated Annealing," *ORSA J. Computing* 3, 275–287.

Fetterolf, P.C. and G. Anandalingam. (1992a). "A Lagrangian Relaxation Technique for Optimizing Interconnection of Local Area Networks," *Operations Res.* 40, 678–688.

Fetterolf, P.C. and G. Anandalingam. (1992b). "Optimal Design of LAN–WAN Internetworks: An Approach Using Simulated Annealing," *Annals Operations Res.* 36, 275–298.

Fratta, L., M. Gerla, and L. Kleinrock. (1973). "The Flow Deviation Method: Approach to Store-and Forward Computer Communication Network Design," *Networks* 3, 97–133.

Gerla, M. and L. Kleinrock. (1988). "Congestion Control in Interconnected LANs," *IEEE Network* 2, 72–76.

Gondran, M. and M. Minoux. (1988). *Graphs and Algorithms.* New York: Wiley.

Gupta, S. and K.W. Ross. (1991). "Performance Modeling and Optimization of Networks of Bridged LANs," *Queueing Syst.* 9, 113–132.

IBM. (1991). *Optimization Subroutine Library: Guide and Reference.* Release 2, Kingston, NY.

IEEE. (1998). IEEE 802.5 Documents. (http://p8025.york.microvitec.co.uk/802.5/documents/).

Information Handling Services. (1993). *Vendor Master Directory and Vendor Catalogs on CD-ROM.* Englewood, CO.

Kaefer, F. (1995). "Algorithms for Designing Survivable LAN-Interconnection Networks Using Transparent Bridges." Ph.D. Thesis, Univ. of Iowa, Iowa City, IA.

Kaefer, F. and J.S. Park. (1995). "A Method for Interconnecting LANs with Survivability Considerations." In *Proc. 3rd Int'l Conf. Telecom. Syst.*, Vanderbilt Univ., Nashville, TN.

Kaefer, F. and J.S. Park. (1998). "Interconnecting LANs and a FDDI Backbone Using Transparent Bridges: A Model and Solution Algorithms," *INFORMS J. Computing* 10, 25–39.

LeBlanc, L., J.S. Park, V. Sridhar, and J. Kalvenes. (1996). "Topology Design and Bridge-Capacity Assignment for Interconnecting Token Ring LANs: A Simulated Annealing Approach," *Telecom. Syst.* 6, 21–43.

Lee, F.C.Y., G. Anandalingam, and L. Clarke. (1995). "Design of Survivable LAN-LAN Internetworks with a Hierarchical Topology." Working Paper 94-20, Dept. of Sys. Eng., Univ. of Pennsylvania, Philadelphia, PA.

Love, R.D. and J.J. Lynch. (1997). "Token Ring Migration: A White Paper." IBM Networking Hardware Division, Research Triangle Park, NC. (http://www.hstra.com/articles.html).

Magnanti, T.L., P. Mirchandani, and R. Vachani. (1995). "Modeling and Solving the Two Facility Capacitated Network Loading Problem," *Operations Res.* 43, 142–157.

Monma, C.L. and D.F. Shallcross. (1989). "Methods for Designing Communications Networks with Certain Two-Connected Survivability Constraints," *Operations Research.* 37, 531–541.

Nishida, T., M. Murata, H. Miyahara, and K. Takashima. (1986). "Congestion Control in Interconnected Local Area Networks." In R.L. Pickholtz (ed.), *Local Area and Multiple Access Networks.* Rockville, MD: Computer Science Press, pp. 107–136.

Padberg, M.W., T.J. Van Roy, and L.A. Wolsey. (1985). "Valid Linear Inequalities for Fixed Charge Problems," *Operations Res.* 33, 842–861.

Park, J.S., R. Bartoszynski, and W.A. Rosenkrantz. (1995). "Stability of p-Persistent CSMA/CD" *ORSA J. Computing* 7, 149–159.

Park, J.S., and K. Kang. (1993). "Delay Analysis of Multidimensional Queuing Process in CSMA/CD Local Area Networks," *Telecom. Syst.* 1, 217–242.

Park, J.S., L. LeBlanc, and B.H. Lim. (1996). "Interconnecting Remote LANs Using SMDS: A Model and solution algorithms." In *Proc. 4th In'l Conf. Telecom. Syst.* Nashville, TN.

Perlman, R. (1992) *Interconnections: Bridges and Routers*. Reading, MA: Addison-Wesley.

Soha, M. and R. Perlman. (1988). "Comparison of Two LAN Bridge Approaches," *IEEE Network* 2, 72–76.

Sridhar, V. (1994). "Optimal Design of Interconnected Local Area Networks." Ph.D. Dissertation, Univ. of Iowa, Iowa City, IA.

Sridhar, V., B. Gavish, and J.S. Park. (1995). "Reliable Interconnection of LANs Using Source Routing Bridges." In *Proc. 3rd ORSA Telecom. Conf.*, Boca Raton, FL.

Taylor, M. (1997). "High Speed Token Ring: A Technology White Paper." Madge Networks, San Jose, CA, (http://www.hstra.com/articles.html).

Valiant, L.G. (1979). "The Complexity of Enumeration and Reliability Problems," *SIAM J. Computing* 8, 410–421.

Wolsey, L.A. (1989). "Strong Formulation for Mixed Integer Programming: A Survey," *Math. Programming* 45, 173–191.

Yen, J.Y. (1971). "Finding the k Shortest Loopless Paths in a Network," *Management Sci.* 17, 712–716.

Ziff-Davis Publishing. (1993). *Data Sources—the Complete Computer Product Book: Hardware and Data Communications Volume*. New York, NY.

Journal of Heuristics, 6, 167–181 (2000)
© 2000 Kluwer Academic Publishers

Detecting and Preventing Routing Problems in the Planning Process of CCSS#7 Networks

GUSTAVO PARAMÉS, JAVIER CALATRAVA AND GONZALO CARRIÓN
TELEFÓNICA I+D, Emilio Vargas St., 6. 28043 Madrid, Spain
email: gustavo@tid.es; jcj@tid.es; cgr@tid.es

Abstract

Common Channel Signalling System #7 (CCSS#7) has become nowadays widely used by most of the operators which are strongly dependant on its performance. This paper develops next issues. First, usual problems with routing in CCSS#7 networks that can not be solved by the protocol are described, and a checking algorithm based on modified dynamic programming techniques is proposed. And secondly, the problem of updating routing tables of nodes and several alternatives to solve it, are presented and compared. Two kind of solutions to solve this second problem are proposed and tested. Conclusions are presented.

Key Words: signalling, CCSS#7, loops, network planning, heuristic, dynamic programming, travelling salesman problem, TSP, genetic algorithms

1. Introduction

Common Channel Signalling System #7 (CCSS#7) has become nowadays widely used by most of the operators. The flexibility provided by this ITU standard (ITU-Q700, 1993) is considered essential in a competitive environment for the quick development of new services.

CCSS#7 is in fact a packet switched network with two operation modes, connection and connection-less oriented, that allows sending messages between nodes (Signalling Points, SP). CCSS#7 is used by the PSTN (Public Switched Telephone Network), IN (Intelligent Network) or GSM (Global System for Mobile Communications) for the managing of calls and accessing to databases.

Because of the strong dependence of these networks on the reliability of the signalling network, CCSS#7 has been provided with some mechanisms for error detection and control, as well as procedures for congestion control and rerouting in case of network failures (Jabbari, 1992). Nevertheless, some problems can arise at the network level, that are not fully solved by these procedures.

This paper explains these problems and their implications in the planning and provisioning processes. First, problems in the routing of CCSS#7 networks are described and analysed. Secondly, an algorithm based on dynamic programming techniques for checking the routing tables proposed at the planning level is presented. Finally, the problem of updating in the network the planned structures is described and several algorithms are also proposed for its solution.

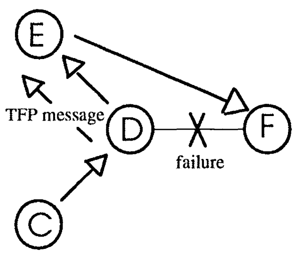
2. Routing problems in CCSS#7 networks

In accordance with the ITU recommendations (ITU-Q700, 1993), the nodes in CCSS#7 are named Signalling Points (SP), and a Signalling Transfer Point (STP) is an SP that can route messages coming from other SP/STPs. The possible paths between two SPs are named signalling routes and are defined as a sequence of SP/STPs linked by link-sets. Finally, a signalling relation is defined between two SPs when there is a need of communication between them.

Routing can be described at the planning level by a set of tables defining the way in which the signalling messages are routed to a specific destination, for each SP. These tables point to the next node in the routing of the message. Several alternatives with different priorities can be specified in the same SP for the same destination, taking into account the possibility of failures in network elements. Load sharing can also be described with these tables, by means of the definition of several alternatives with the same priority; in this case, the traffic will try to be equally forwarded among these alternatives. An alternative of lower order is only used if the higher priority options are all in a failure situation. Thus, the signalling routes between the two SPs of a signalling relation are defined by means of the elements of the routing tables specified for the SP destination, of all the SP/STPs involved as routing nodes.

A simple mechanism is used in order to expand the information of failures (ITU-Q704, 1996). This mechanism (Transfer Prohibited Procedure) uses a specific kind of control message, the Transfer Prohibited Message (TFP message). When a STP D can not use the usual option for routing a signalling message to a destination node F (because of a failure in the link-set), it sends the message through the second alternative (E). It also sends to this STP a TFP message with an indication for the STP of no sending traffic to the destination node through it (figure 1). When an STP has no more valid options, it sends TFP messages to all the adjacent Signalling Points (figure 2). When an SP/STP C receives a TFP message from a STP D related to a destination SP F, C knows it should not send messages to F through D.

In spite of all the facilities provided to avoid and to solve failure situations, the protocol is characterised by some weakness due to the possibility of a wrong working of the network in some specific circumstances (Houck et al., 1994).

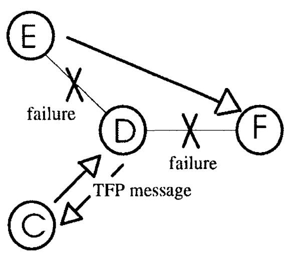


C	Destination	Next 1	Next 2
	F	D	

D	Destination	Next 1	Next 2
	F	F	E

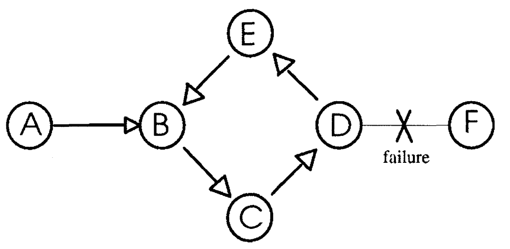
E	Destination	Next 1	Next 2
	F	F	

Figure 1. A TFP message is sent from D to E related to destination F, when the usual option is not available. E is defined as the alternative for the destination F when DF link-set is in a failure situation.



C	Destination	Next 1	Next 2
	F	D	
D	Destination	Next 1	Next 2
	F	F	E
E	Destination	Next 1	Next 2
	F	F	

Figure 2. A TFP message is sent from D to C related to destination F, when F can not be reached from D.

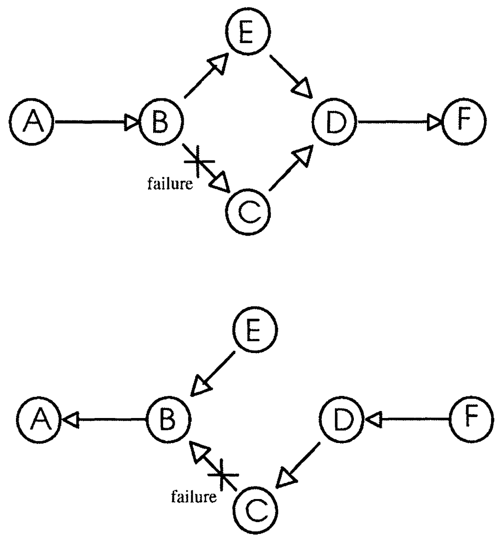


A	Destination	Next 1	Next 2
	F	B	
B	Destination	Next 1	Next 2
	F	C	
C	Destination	Next 1	Next 2
	F	D	
D	Destination	Next 1	Next 2
	F	F	E
E	Destination	Next 1	Next 2
	F	B	

Figure 3. Circular routing example in case of failure of a link-set. Routing tables present two alternatives.

Thus, it is possible that some problems can arise in the routing in a specific network situation defined as a combination of network element failures:

- **Circular Routing.** In this situation, a message sent to a SP F reaches a STP B and, after some "hops", it arrives repeatedly to B (figure 3). Taking into account that routing is decided based on the destination, these messages will not finish their loop until the network situation changes. Some other protocols, with routing also based on the definition of routing tables, provide some mechanisms for solving this situation, such as a time flag. CCSS#7 protocol can solve some simple situations by means of TFP messages but in general, this problem has to be avoided at the planning level. The importance of the circular routing is not only that the message can not reach its destination, but the congestion it can produce, having a strong effect on other messages. Theoretically, the problem can arise even without network failures, due to a wrong definition of routing tables.
- **Lack of Bidirectionality** is produced when the messages sent from a SP origin, A, can reach their SP destination, F, but the messages sent from F to A can not arrive to their destination (figure 4). CCSS#7 protocols require a "dialogue" between the origin and



A	Destination	Next 1	Next 2
	F	B	

B	Destination	Next 1	Next 2
	F	C	E
	A	A	

C	Destination	Next 1	Next 2
	F	D	
	A	B	

D	Destination	Next 1	Next 2
	F	F	
	A	C	

E	Destination	Next 1	Next 2
	F	D	
	A	B	

F	Destination	Next 1	Next 2
	A	D	

Figure 4. Lack of bidirectionality example in case of failure of a link-set. Routing tables present two alternatives.

destination nodes (A and F). So, repeated attempts of A to obtain response from B can also produce a congestion state. This anomaly can happen when there is at least one signalling route in the signalling relation A-F that is not defined, in the opposite direction, for the signalling relation F-A. In this circumstance, a set of failures can be found that blocks all the routes between F and A, in such a way that this route is still possible between A and F.

- **Routing Tables Not Fully Defined.** In this case, a message sent to a SP destination, D, reaches a SP/STP with no routing element for D. In this case, this message will be lost. It could happen that there is no way for the origin node to reach the destination D, in any network situation.

All these problems should be avoided in the planning process by means of a proper definition of the routing tables. This definition is also related to quality objectives of CCSS#7:

- ITU specifies a maximum number of STPs in the routing of the messages, in order to avoid an excessive delay in the communication (ITU-Q709, 1993). If this number is exceeded, a **Too Long Route** anomaly is produced.

The ITU recommendations related to the management of CCSS#7 (Glitho, 1996; ITU-Q753, 1993) define some procedures at the network level in order to identify the above anomalies. These procedures are based on the testing of all the signalling routes (sequences of STP/SPs) by means of the sending of test messages (Glitho, 1996). Several disadvantages have been identified for this approach. Some of them are related to the overload it produces. It has also to be taken into account that this procedure is not always implemented in SPs. But mainly, due to the heavy consequences of some of these problems in the client networks (GSM, IN or PSTN), they should be previously prevented.

3. Detecting routing problems in the planning process

Due to the strong dependence of the PSTN, IN and GSM networks on the working of the CCSS#7 network, the planned routing tables have to be checked carefully before implementing them in the nodes. Every feasible network situation, that is to say, every combination of network element failures, should be analysed because of the heavy impact of some of the routing problems on these networks.

Manual tests can only be carried out if a very regular network structure is used. Computer tests are, in this way, needed. The simulation of all the combinations of network failures is computationally problematic. The usual approach to the problem is based on the full exploration of all the possible routes for all the defined signalling relations. This process can be simplified taking into account that routing is normally accomplished based on just the destination. In this way, dynamic programming techniques (Nemhauser and Wolsey, 1988) are a good candidate for such a task.

Let be defined a generic problem as the determination of a sequence of T decision variables for T periods, with a set of valid states for each one; it can be solved as a dynamic programming problem (DP problem) if the determination of the decision variable for the period t does not depend on the variables for the periods $1, \ldots, t-1$, and only on the decision variables and states for the periods $t+1, \ldots, T$. Therefore, if n is the number of nodes in the CCSS#7 network, n decision variables and n states are defined for each destination, in each node. The decision variable is defined as the possibility of routing problems from the node to the destination. The states are defined as the set of paths from the node to the destination. In this way, the state of a node is calculated as the concatenation of the routing nodes defined in its routing table to the destination node, with the states of the routing nodes. In order to evaluate the decision variable, two groups of anomalies can be distinguished:

- Anomalies that can be incorporated in the decision variable of the DP. They are called in this paper *Anomalies with Inheritance*. The main characteristic of this group is that if an anomaly of this group is shown for a certain sequence of nodes, any other sequence containing it will show the same anomaly. *Circular routing*, *routing tables not fully defined*, *maximum number of STPs in the routing exceeded*, are included in this group. Thus, the DP algorithm will provide information about the possibility of these kinds of anomalies.
- Anomalies that can not be incorporated in the decision variable of the DP. They are called in this paper *Anomalies of Symmetry*. Checking these anomalies requires comparing the signalling routes in both directions of the two signalling relations defined between two SPs. Lack of bidirectionality is defined like this. The analysis of this problem requires a later processing after the DP algorithm.

The algorithm can be described as a recursive procedure that develops the signalling routes from an origin node O to the destination node D for all the signalling relations. If needed, this procedure is recursively used for obtaining the signalling routes from the routing nodes in the routing tables of O to D, the destination node. At the same time, the procedure identifies and indicates anomalies with inheritance. Special attention is paid to loops, by means of a flag indicating the nodes that are used in the routing (loop flag). When the signalling routes are developed for the signalling relations O-D and D-O, they are compared for detecting lack of bidirectionality.

For each signalling relation defined in the network, between the nodes A and B:

```
Generate signalling routes between the nodes $A$ and $B$ and evaluate *Anomalies with Inheritance* in the routes to $B$.
Generate signalling routes between the nodes $B$ and $A$ and evaluate *Anomalies with Inheritance* in the routes to $A$.
Evaluate *Anomalies of Symmetry* between $A$ and $B$ based on routes between $A$ and $B$ and routes between $B$ and $A$.
```

Generate the signalling routes between nodes O and D and evaluate *Anomalies with Inheritance* in the routes to D:

```
If they had previously been calculated, the same results are valid and no more processing is needed here.
If the loop flag is activated, a **Circular Routing** problem is detected and no signalling route is calculated; else the loop flag is activated. (Some *ODO* loops are avoided by means of the TFP messages and this circumstance has to be checked before publishing the anomaly. Also, the priority of the "out" signalling link-set in the node should be equal or higher than the priority in the "in" link-set in the routing from $O$ to $D$).
Obtain the routing nodes in $O$ to $D$ from the routing table of $O$.
If no routing nodes are defined, a **Routing Tables Not Fully Defined** anomaly is detected.
For each routing node $R$:
    Generate the signalling routes between nodes $R$ and $D$ and evaluate *Anomalies with Inheritance* in the routes to $D$ (recursive procedure).
    Incorporate to the set of signalling routes from $O$ to $D$, the routes through $R$, adding $O$ at the beginning of the signalling routes from $R$ to $D$.
    If the signalling routes from $R$ to $D$ produce any *Anomalies with Inheritance*, these anomalies should be published also for the relation between $O$ and $D$.
    Evaluate the **Too Long Route** (detecting if the number of nodes in each signalling route is higher than a previously defined value).
Deactivate the loop flag.
```

Evaluate *Anomalies of Symmetry* between O and D by means of the routes between D and O and the routes between O and D:

```
It has to be checked that all the signalling routes between $O$ and $D$ are possible between $D$ and $O$, in the opposite direction.
```

4. Preventing routing problems in the updating process

Let suppose the planning process has produced a set of valid routing tables that have been checked using the previously defined algorithm. These routing tables are supposed to be free of routing problems in every network situation. They can be considered as the final objective. The next step is the modification of the current tables to these planned routing tables (updating process).

The changes needed for achieving the final configuration can be obtained as the differences between the elements of the current and the planned routing tables. Depending on the management capabilities of the network, it is not always possible to change all the tables remotely and instantaneously from a central node. In this case, it is needed the planning of the sequence of updating steps, in such a way that the network situation after each step is free of routing problems in any network situation.

Also, it is possible that not all the network elements (nodes or link-sets) needed for the routing tables are yet available. Thus, firstly, the routing changes have to be "filtered" in such a way that only changes using available network elements are acceptable. The rest of changes should wait until the network elements they need are available. But of course it has to be guaranteed that the final routing tables obtained after the filtering can not produce network anomalies.

In this way, the problem of the updating process can be separated in two:

- First, an intermediate configuration has to be obtained. This intermediate configuration should be as near to the planned one as possible, taking into account that:
    - Only the available network elements can be used.
    - Routing problems should not be possible in any network situation.
- Finally, an updating sequence has to be calculated. This sequence is defined as an ordered set of updating steps, in such a way that no routing problems are possible after each step. Ideally, each step should incorporate only one routing table change, because of the difficulty of updating two or more routing tables at the same time. Nevertheless, depending on the sequence, it is possible that several changes should be developed at the same time, and in this way they build up only one updating step.

4.1. Calculation of the intermediate configuration

The next heuristic is proposed for this purpose:

1. The differences between the current and the planned routing tables are found and defined as the set of candidate changes.
2. The candidate changes using no yet available network elements are deleted. That is to say, the routing elements using link-sets or nodes not available are deleted of the list of candidate changes.
3. The network structure resulting after the updating of the set of candidates is checked. While this structure is not free of anomalies:

The signalling routes with anomalies with inheritance are detected. These signalling routes are avoided by the deactivation of the candidate change appearing in the route and nearest to the destination.

The signalling relations with symmetry anomalies are detected. The signalling route causing this anomaly is avoided by the deactivation of the candidate change appearing in the route and nearest to the origin.

4.2. Calculation of the sequence of changes

The direct approach of testing each sequence until a valid solution is found is not computationally possible. If N changes have to be done, $N!$ sequences have to be checked.

For each sequence, the basic algorithm for detecting problems have to be developed N times, in order to establish the groups of changes that should be developed at the same time in the same updating step. For the i change of a specific sequence, the analysed network is defined as the initial network configuration with the first i changes. If the algorithm indicates routing problems, the i change should be grouped with at least the next one in the same updating step.

The best sequence can be defined as the solution with the highest number of updating steps, because of the problem of developing several changes at the same time. So $N \cdot N!$ is clearly a problematic number, taking into account the big sizes of CCSS#7 networks. Two more realistic approaches have been tried at this point. The first one is based on the use of a Genetic Algorithm. The second one uses a specially developed heuristic.

4.2.1. Calculation of the sequence of changes: Genetic Algorithms approach.

Genetic Algorithms introduction. Genetic Algorithms (GA) can be considered a global optimisation method that emulate the process of natural selection in a parallel search for a good solution to an optimisation problem (Michalewicz, 1992). They can be included in the group of stochastic optimisation methods, near to Simulated Annealing, for instance. The main characteristic of GA differing from other stochastic methods, is the proposal of new candidates that have been created based on the merge of attributes of previously proposed solutions (crossover). Simultaneously, they try to escape from local minima by means of a random generation of new proposals (mutation).

The working scheme of GA is quite constant for all the applications. First, an initial set of individuals (population) is randomly proposed. A subset of this initial population is selected as "parents" and crossed over. In this way, a set of descendants is obtained. A mutation process is carried out over the descendants. The resulting individuals are inserted in the population, replacing some or all the individuals of the initial population. This process is iteratively developed over the new population until convergence in an individual is achieved. Different strategies have been studied for selecting parents, for the crossover and mutation processes and for replacing and inserting the descendants in the population.

A set of points has to be defined for each GA application:

- The encoding procedure.
- A function for the evaluation of the fitness of an individual.

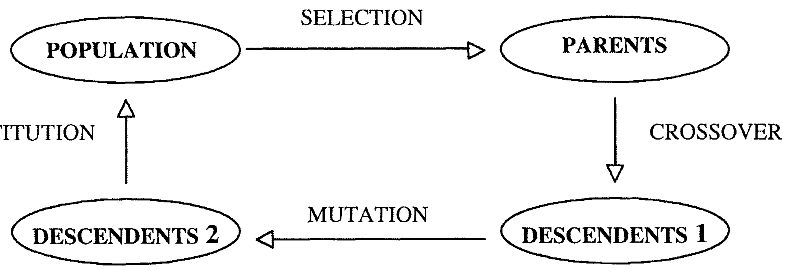


Figure 5. Main cycle in GA.

- The crossover and mutation procedures.
- The general GA procedure is illustrated in Figure 5.

GA approach. GA can be a good candidate for providing a valid solution for the updating problem. In the next paragraphs, the GA approach is described. As has just been indicated, the use of GA for the updating problem requires the definition of the fitness function and the encoding, crossover and mutation procedures. The tried solutions have been derived from solutions to the Travelling Salesman Problem (TSP).

This classical problem can be formulated as a travelling salesman having to visit a set of N cities; he knows the cost of travelling between all the cities and needs to find the cheapest path for visiting all the cities once and only once (Moscato).

In the updating problem, an optimal order for a set of tasks has also to be found. In this case, the purpose of the process should be to maximise the score of the sequence of changes, while trying to minimise the number of simultaneous updates. As has already been indicated, the score of a sequence of routing changes can be defined as the number of steps or groups of changes needed. Its upper bound is the number of routing changes, when each set is made up by just one routing change. In the worst case, the score is 1, when all the changes have to be done simultaneously. So, an asymmetric matrix of 0–1 costs is used, but this matrix can be different depending on the sequence.

Several encoding and crossover procedures have been described for the TSP (Goldberg, 1989). One of the most effective solutions to the encoding is named Path Representation. It is based on individuals defined as sets of n integers, if n is the number of updates to be done. So, the integers used in an individual should be equal or greater than 1 and equal or lower than n, and no integer can appear more than once in the same individual. Each change is identified by a number, and each individual/sequence of changes is a sequence of these integers. The integer i of a specific individual identifies the change i in the sequence of changes.

The classical crossover in GA, based on creating a new individual with the integers from the left side of the first parent and from the right side of the second parent, is not valid, because it will often produce no valid individuals. So, several specific crossover procedures have been defined for this encoding. OX procedure has been used in the trials. In this option, the left side of the descendent is taken from the left side of the first parent; the remaining positions are consecutively selected from the right side of the second parent, discarding the already appeared integers. If the end of the second parent is achieved and the descendent is not yet fully defined, the selection of the not yet appeared integers is continued by the

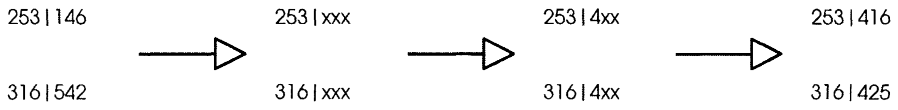


Figure 6. Example of OX crossover procedure.

beginning of the second parent. This crossover procedure maintains the relative order of the changes. It is possible to extend it for using a higher number of crossing points. Figure 6 illustrates the crossover procedure.

Finally, the mutation procedure that has been used is quite simple. It is developed by the swap of two integers in the mutated individual.

Fitness function used has been directly the number of steps needed. So, for the evaluation of each individual, the validation algorithm previously described has to be done *n* times: the network configuration resulting after each modification has to be evaluated, and the final score will be the number of valid situations. In the trials, the validation procedure of a network configuration has been relaxed, in such a way that only circular routing and not fully defined routing tables is checked in all the network situations. Other kinds of anomalies should be checked taking into account only a maximum number of network failures.

```
number of sets = 0
initialise network situation to the initial structure
for each change in the sequence:
    apply the change
    run the procedure of identification of routing problems
    if no routing problems are detected, increase the number of simultaneous groups
```

4.2.2. Calculation of the sequence of changes: a heuristic approach. This approach is based on the definition of a set of heuristic restrictions related to the order of the changes in order to prevent the possibility of anomalies. It analyses the routes forming every signalling relation of the intermediate configuration. These routes are easily obtained by means of the algorithm described before for the detection of anomalies in the routing tables. For every route, it creates a group of ordering relations with the different changes needed for the intermediate configuration.

The heuristic algorithm takes advantage of the characteristics of both kinds of CCSS#7 routing anomalies. In this way, the main idea is:

- The inheritance anomalies are characterised because each one is shared by a set of routes finishing at the same "core" anomaly. It is known that the objective routing tables are free of anomalies. So, if the changes needed for creating a route are done first the nearest to the destination node, it is guaranteed that no core anomaly is produced. In this way, the changes needed for a route are sequenced based on the distance to the destination (the nearest to the destination at the beginning of the sequence).
- The anomalies of symmetry considered (lack of bidirectionality) are related to the need that the same route is defined in both directions. In this way, the changes activating the same route in both directions should be done at the same time.

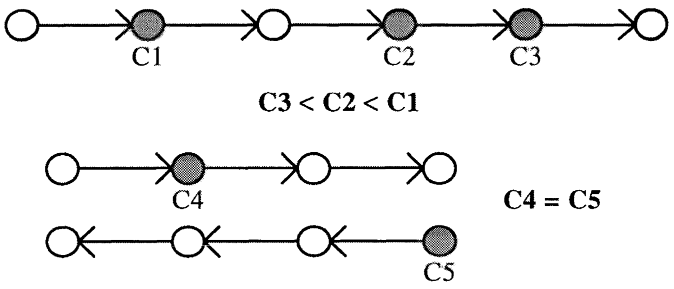


Figure 7. Ordering relations between atomic changes within an independent group of changes.

Let define O_i as the position in the sequence of updating for the change i. Thus, these restrictions can be formulated as:

- $O_i \leq O_j$ if the node affected by the change i is closer to the destination node than the node effected by the change j, within the same routes using the same changes.
- $O_i = O_j$ is established for the changes i and j activating equivalent routes in both directions (figure 7).

Some additional considerations have to be done to this main idea:

- The changes related with the deactivation of routing elements are not considered in the main idea. They should be done at the end of the process, in such a way that the initial routes going to disappear exist while the new routes are created and some path is always possible.
- Some "spurious" routes may appear in transitory network configurations. The simultaneousness requirement for the changes in the routes of the intermediate configuration should be extended to every possible route in these transitory network configurations. In this way, the segments of routes not presented in initial or intermediate routes should be detected and activated simultaneously in both directions.

In this way the ordering problem is delimited with a set of restrictions considering the previously defined relative orders. These relative order restrictions are enough for solving the problem. A modification of the topological classification algorithm described in De Wirth (1986) has been used, but also a constraint-programming package could be used.

4.2.3. Comparative example for the updating problem. In the following paragraphs an example exercise is proposed in order to test the intermediate configuration algorithm and to compare both sequencing algorithms. The described example consists of a small size network with the intention of making evident the behaviour of the algorithms. The example is described in figure 8.

The example consists of three pairs of signalling relations that entwine origins and destinations. Therefore, if a change can't be activated because lack of the needed network

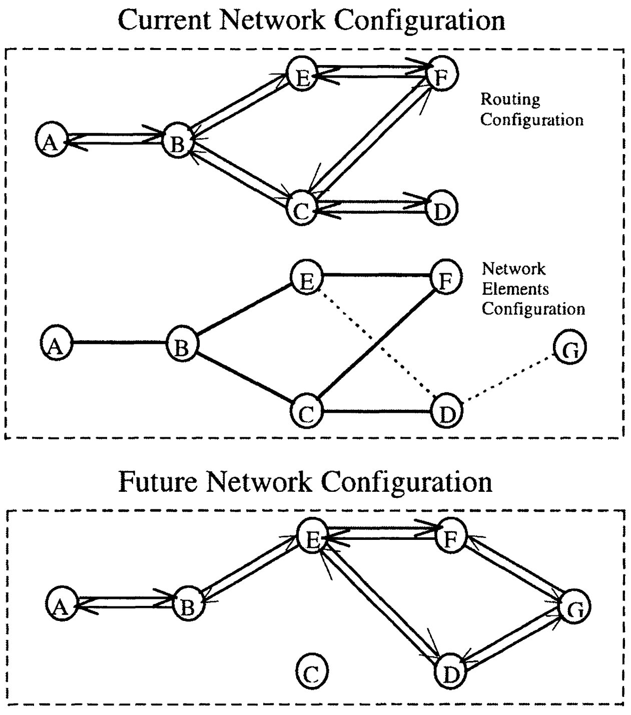

Current Network Configuration
A
B
E
F
C
D
Routing Configuration
A
B
E
F
C
D
G
Network Elements Configuration
Future Network Configuration
A
B
E
F
G
C
D

Figure 8. Comparison case.

elements, the changes that are parallel to them could not be activated either to avoid symmetry anomalies, leading to a situation in which very few changes could be activated. The routing tables in both configurations, current and future, are shown in Table 1.

The three pairs of signalling relations are A-D/D-A, A-F/F-A and D-F/F-D. As can be seen, the three signalling relations have changes, and several of these changes have to be activated in a co-ordinated way with the intention of avoiding anomalies of symmetry. For example, the changes in node B and in node D (before the path was through C and now it is through E) have to be activated at the same time to avoid lack of bidirectionality in the $A \rightarrow D$ and $D \rightarrow A$ relations. In the same way, the changes in nodes F and D (before the path was through C and now it is through G) have to be activated at the same time. That is why, if some of those changes (B, F or D) could not be activated because of a lack of network elements (in the example the link doesn't exist between F and G), none of them could be activated. So, a good algorithm to find the intermediate configuration must avoid the obvious solution of a void set of changes, and must get as many changes in this configuration as possible.

4.2.4. Results. The intermediate configuration algorithm has been tested with the example exercise proposed in the last section, and later, its solution has been used as an entry to compare the two sequencing algorithms described before, the genetic and the heuristic algorithm.

Table 1. Routing table.

Current configuration			Future configuration			
Owner node	Destination node	Routing nodes	Owner node	Destination node	Routing nodes	Change
A	D	B	A	D	B	
A	F	B	A	F	B	
B	A	A	B	A	A	
B	D	C	B	D	E	C1
B	F	E	B	F	E	
C	A	B				C2
C	D	D				C3
C	F	1 F				C4
		2 B				
D	A	C	D	A	E	C5
D	F	C	D	F	1 G	C6
					2 E	
E	A	B	E	A	B	
E	D	B	E	D	D	C7
E	F	F	E	F	F	
F	A	E-C	F	A	1 E	C8
					2 G	
F	D	C	F	D	G	C9
			G	A	D	C10
			G	D	D	C11
			G	F	F	C12

Table 2. Results of intermediate configuration search.

	Intermediate algorithm	Configuration	Searching
Atomic changes allowed		c1, c5, c7, c10 and c11	

In a first step, the intermediate configuration obtained is shown in Table 2. As can be seen, this algorithm includes several changes from the group of entwined changes explained in the section before, avoiding in this manner the problem of local minima (that is, non optimal solutions that include a number of changes lower than the actually possible number of changes).

The second step is to compare the solution obtained by both sequencing algorithms. In Table 3 can be seen the logical expression for the possible sequences of changes obtained by the heuristic algorithm. The GA obtains results not formed by tree of sequences but formed by a particular sequence every time it gives a solution. It is remarkable that most of the times the sequence obtained by the GA is contained by the tree provided by the heuristic algorithm. In some occasions it is not contained because the ordering criteria adopted by

Table 3. Results of sequencing.

```
Order( 1st. Independent( Order( 1st. c7
                                2nd. Simultaneous( c1, c5 ) )
                         c11 )
       2nd. c10 )
```

the heuristic algorithm are too inflexible for some situations, reducing the richness of the obtained tree.

Nevertheless, the most importance difference between the working of both approaches for the sequencing problem is fixed by the response times. Some trial tests have been carried out with networks of 22 and 76 nodes. The number of signalling relations is, in each case, 20 and 1980, and the number of changes is 248 and 1770. For the genetic approach, each trial cycle (figure 5) needs about half a minute for the 22 SPs trial, and some hours for the 76 SPs example, and usually a high number of cycles are needed. However, the heuristic solution requires one or two seconds in the first case and about one minute in the second.

5. Conclusions

Reliability is a basic requirement in CCSS#7 networks, due to the strong dependence of PSTN, IN and GSM network on it. In this way, the routing problems should be avoided in any network situation, that is to say, in every combination of network failures. The planning department has to guarantee the validity of planned routing tables, previously to any network management procedure. Taking into account the current needs for flexibility and quick responses, an algorithm for the validation of any network structure is required and has been presented. This algorithm has been carefully designed because of the size of the networks involved and its high use in the updating process. The anomalies that have to be avoided are described and catalogued taking into account the possibility of a quicker recognition of its chance in the algorithm. The algorithm does not correct badly defined routing tables but just identifies the problems. Usually, it is not interesting to incorporate in this kind of planning systems automatic decisions, because of the not systematic considerations related with the planning process. Any way, a pruning algorithm can be easily derived and indeed is included in the algorithm proposed for obtaining an intermediate solution.

The planned routing tables have to be implemented in the signalling nodes. A methodology is proposed, based on obtaining a valid intermediate objective configuration, and sequencing the changes for achieving this configuration.

Because of the possibility of routing problems in transitory network configurations, the sequencing process has to be properly planned. The complexity of the procedure is so increased by the need of checking different sequences of changes, looking for a valid one. Two algorithms have been proposed for this purpose.

First one is based on a Genetic Algorithms approach. The trials with this approach usually have obtained several solutions with the same score. Thus, other considerations can be incorporated in the fitness function, but then a poor performance has been obtained. Also, GA option does not guarantee the quality of the solution.

A heuristic approach based on the characteristics of the unwanted anomalies having to be avoided is also explained in the paper. Heuristic requirements in the order of updating the tables based on its distance to the destination node are used for some kinds of anomalies, and the need for simultaneous activation of routes are used in other cases. The main disadvantage of this option is related to its dependence of the specific kinds of anomalies considered. In this way, the GA is more flexible and easily could incorporate new kinds of anomalies or planning considerations. But the performance of the heuristic algorithm has been much more suitable. Evolutions of networks with 300 nodes and 8000 signalling relations require about half an hour for the full process (intermediate configuration + sequencing), a correct value for this process not requiring a real time performance. The kind of solutions provided by this approach is indeed much more interesting that the format of solutions provided by the GA. In the heuristic approach, the solution can be formed by a set of independent trees. Each tree resumes a set of different evolution paths, and independent trees identify completely independent sequences of changes. GA produces one or a set of valid paths, and so a later processing is needed if the format of independent trees is required.

The analysis of real situations shows that the symmetry considerations reduce too much the possibility of evolutions, and the solutions obtained considering these problems are characterised by too many groups of simultaneous changes. Taking into account the usual very low possibility that the lack of bidirectionality produces problems in the network, the planner may not consider this problem in the updating process.

The algorithms described in this paper are being used in a tool included in the network management system for the CCSS#7 network in the Spanish Telefonica de España S.A. network, SERES. Current work is related with including traffic load considerations in the updating process, in normal and failure situations, as well as practical aspects related to additional criteria of the provisioning department.

References

Niklaus, De Wirth. (1986). *Algorithms & Data Structures*. Prentice Hall.

Glitho, Roch H. (1996). “Isolating Faulty Routing Tables in SS7 Networks: Present and Future,” *IEEE Communications Magazine*.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Houck, D.J., K.S. Meier-Hellstern, F. Saheban, and R.A. Skoog. (1994). “Failure and Congestion Propagation Through Signaling Controls,” ITC 14.

ITU-Q700. (1993). “Introduction to CCITT Signalling System No. 7.”

ITU-Q709. (1993). “Hypothetical Signalling Reference Connection.”

ITU-Q753. (1993). “Signalling System No. 7 Management Functions MRVT, SRVT and CVT and Definition of the OMASE-User.”

ITU-Q704. (1996). “Signalling Network Functions and Messages.”

Jabbari, Bijjan. (1992). “Routing and Congestion Control in Common Channel Signaling System No. 7,” *Proceedings of the IEEE* 80(4).

Zbigniew, Michalewicz. (1992). Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag.

Moscato, Pablo. http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html.

Nemhauser, George L. and Laurence, A. Wolsey. (1988). *Integer and Combinatorial Optimization*. John Wiley & Sons.