Technical Report 01-EMIS-05

SONET/SDH Ring Assignment with Capacity Constraints

Olivier Goldschmidt
OPNET Technologies, Inc.
email: ogoldscmidt@opnet.com

Alexandre Laugier
Corporate Network Design Department, Network Architecture and Traffic Division
Centre National d'Etudes des Télécommunications/DSE/ISE
Sophia, France
email: Alexandre.Laugier@cnet.francetelecom.fr

Eli V.Olinick
Department of Engineering Management, Information, and Systems
Southern Methodist University
PO Box 750123
Dallas, TX 75275-0123
email: olinick@engr.smu.edu [1]

September 26, 2001

[1]Research supported in part by ONR contract N00014-91-J-1241, by NSF award No. DMI-9713482

Abstract

We consider the problem of interconnecting a set of customer sites using bidirectional SONET rings of equal capacity. Each site is assigned to exactly one ring and a special ring, called the federal ring, interconnects the other rings together. The objective is to minimize the total cost of the network subject to a ring capacity limit where the capacity of a ring is determined by the total bandwidth required between sites assigned to the same ring plus the total bandwidth request between these sites and sites assigned to other rings.

We present exact, integer-programming based solution techniques and fast heuristic algorithms for this problem. We compare the results from applying the heuristic algorithms with those produced by the exact methods for real-world as well as randomly-generated problem instances. We show that two of the heuristics find solutions that cost at most twice that of an optimal solution. Empirical evidence indicates that in practice the algorithms perform much better than their theoretical bound and often find optimal solutions.

1 Introduction

The use of fiber-optic technology in telecommunication has called for new network design concepts. One of the most popular designs is the Synchronous Digital Hierarchy (SDH) which is also known as Synchronous Optical NETwork (SONET) (the former name is more widely used in Europe and the latter in the United States). In a SONET ring, nodes (typically customer sites) are connected by a ring of fiber, each one sending, receiving, and relaying messages through a device called an add-drop-multiplexer (ADM). In bidirectional rings the traffic between two nodes can be sent clockwise or counterclockwise. The volume of traffic is limited by the ring capacity, which is equal in both directions. On the grounds of reliability, a major European telecommunication company has made the decision to send all traffic in one direction. A direct consequence of this decision is that the capacity of the unidirectional ring must accommodate the sum of bandwidth requests between all pairs of nodes connected by the ring.

The telecommunications provider mentioned above has decided to use a network topology that interconnects a set of customer sites using unidirectional SONET rings of equal capacity. Each site is assigned to exactly one ring and a special ring, called the *federal ring*, connects the other rings together. Note that the federal ring must have the same capacity as the other rings in the network.

The total request of bandwidth of sites assigned to the same ring is equal to the sum of bandwidth requests between the sites plus the total bandwidth request between these sites and sites assigned to other rings. If $d_{uv} \geq 0$ is the traffic demand between site u and site v (that is traffic from u to v plus traffic from v to u), the total demand on ring i is given by

$$D_i = \sum_{u \in i, v \in i, u < v} d_{uv} + \sum_{u \in i, v \notin i} d_{uv} \tag{1}$$

The first term of the sum computes the traffic between any two sites in the ring and the second part of the sum computes the traffic between the sites outside the ring and the sites in the ring. The total demand of the federal ring is equal to the sum of demands between sites in different rings. The common capacity B of the rings must accommodate the total demand of any ring including the federal ring.

The objective is to minimize the total cost of the network. There is a variable and a fixed cost associated with each ring. A site requires an ADM for each ring to which it is assigned. The ADM costs are considered a fixed in this design since every site is required to be connected to exactly one ring and all rings must use the size ADM.

The variable cost has two components. Each site that is connected to the federal ring requires an additional device called a digital cross-connect (DCS). The number of rings determines the number of DCS's required for the design and these cross-connects bear a major part of the total cost (they are significantly more expensive than the ADM's). The other part of the variable cost, is associated with fiber-optic cable in the rings. This cost is a linear function in the number of sites served by the ring and in the length of the ring. The length of a ring is the total length of a shortest tour through the sites of the ring where the distance between two consecutive sites in the tour is the maximum of the actual distance and one kilometer.

1.1 Some Related Literature

Optical networks consisting of a set of rings interconnected by a federal or *backbone* ring have been studied, albeit from a different view point, in the electrical engineering literature (see for example [CEFS99]). Sutter, Vanderbeck and Wolsey [SVW98] and Lee, Sherali, Han and Kim [LSHK98] consider an alternative design that also connects a set of customer sites using unidirectional SONET rings. In this design, a site may be assigned to more than one ring, but the traffic between two sites cannot be split between rings. It is assumed in [SVW98] and [LSHK98] that the rings are disjoint, i.e., there is are no DCS's, and hence no federal ring, to transfer the traffic from one ring to another. As before, a site requires a device called an ADM for each ring to which it is assigned and all rings in the network must use the same size ADM. The cost of an ADM is a function of the capacity of the ring it serves and the objective is to minimize the total cost of the ADMs. Exact methods, based on integer-programming formulations for this problem are presented in [SVW98] and [LSHK98]. Goldschmidt, Hochbaum and Olinick [GHO99] present linear-time approximation algorithms for the "uniform" version of this problem in which all the bandwidth requests are the same and the ring size is a given, fixed multiple of this common demand unit. The results in [GHO99] also apply to the general (i.e. non-uniform) case where traffic between two sites may be split among multiple rings.

Laguna [Lag94] studies a SONET ring design problem with the objective of finding a minimum cost assignment of ADMs to unidirectional rings of possibly varying capacities subject to a limit on the number of nodes allowed in a ring. This problem is similar to the one presented in this paper, but different in several important ways. A major difference is that in [Lag94] the traffic originating at a particular site u, including the point-to-point traffic between u and another site v, may be split among multiple rings. Another difference is that the network design in [Lag94] does not use a federal ring. Laguna gives a mixed-integer programming formulation for this problem and a heuristic solution procedure using tabu search.

SONET planning problems are complex and in practice are solved via a divide-and-conquer approach in which they are decomposed into a sequence of small, easier to manage subproblems. As described by Cosares, Deutsch, Saniee and Wasem [CDSW95] and in [Lag94], successive levels of the design process of SONET networks present different optimization problems. Some of these problems are "easy" to solve (e.g. determining whether a given network is 2-connected), but many turn out to be intractable. For example, Cosares and Saniee [CS94] show that the *SONET Ring Loading Problem* (SRLP) is $\mathcal{NP}$-hard.

The SONET Ring Loading Problem is that of partitioning the traffic on a single bidirectional symmetric SHR between clockwise and counterclockwise traffic so as to minimize the ring capacity required to satisfy all the pairwise demands. Dell'Amico, Labbé and F. Maffioli [DLMar] present lower and upper bound procedures and develop a branch-and-bound algorithm to find exact solutions for large instances of this problem within short computing times. Cosares and Saniee also give a polynomial-time 2-approximation algorithm for SRLP in addition to establishing the problem's complexity (see [CS94]). Schrijver, Seymour and Winkler [SSW98] devise a fast, simple algorithm which achieves a load that is guaranteed to exceed the optimum by at most 3/2 times the maximum demand. Building on this work, Khanna [Kha97] develops a polynomial-time approximation scheme for SRLP.

Myung, Kim and Tcha [MKT97] consider a variant of SRLP, that allows demand splitting (SRLPW). With demand splitting, traffic between a pair of nodes may flow in both directions on *both* rings of a bidirectional ring. They show that the demand-splitting version of SRLP is polynomial and present an efficient exact solution procedure for it. They also present a

polynomial-time 2-approximation algorithm for SRLP and show that optimal objective function value for SRLP is at least half that of SRLPW for the same problem instance.

SRLP is an example of a problem that occurs at a "lower" level in the design of the network than the problem we study in the sense that it would be addressed after resolving "higher" level problems such as forecasting the demand that will be placed on the network and determining which of the possible SONET rings will be built. The telecommuncations provider mentioned earlier has elected to decompose its SONET design problem by first attempting to assign sites to rings in such a way as to minimize the DCS costs. For this reason, we have chosen to single out the objective of minimizing the number of rings in this paper.

After solving the ring-assignment problem, the length of the rings can be minimized in a second phase using Traveling Salesman Problem (TSP) solution techniques: given the set of sites assigned to a ring, the problem is to interconnect these sites with a ring of minimum total distance. A simple heuristic such as 2-Opt is likely to yield good results for this part of the problem since the number of nodes in any given ring will be relatively small. The federal loop can be found using a simple heuristic for solving a variant of TSP called the generalized TSP, in which the nodes are partitioned into sets and the salesman has to visit at least one node from each set (see [FGT95]).

1.2 The SONET Ring Assignment Problem

The *SONET Ring Assignment Problem* (SRAP), which is the focus of this paper, can be described formally as a node-partitioning problem for a given graph G. The nodes of G represent the customer sites to be linked and the edge weights correspond to the traffic demands between sites. Note that edge (u, v) exists only if sites u and v communicate. The sets of a solution to our problem are sets of sites that will be placed on the same SONET ring. The rest of this paper is structured as follows. In Sections 2 and 3 we formulate SRAP and a variant of SRAP as integer programming problems with binary decision variables. SRAP is shown to be $\mathcal{NP}$-hard in Section 4. In Section 5 we show an upper bound on the number of rings found by all heuristics that share a certain simple property. Heuristic algorithms are presented in Section 6 and empirical results from using these algorithms on randomly-generated test problems are given and discussed in Section 7.

2 Integer Programming Formulation

We now present a mathematical formulation of the SONET Ring Assignment Problem. Let $x_{ui} = 1$ if site u is assigned to ring i and $x_{ui} = 0$ otherwise. Let $y_i = 1$ if ring i is active (i.e. has sites assigned to it) and $y_i = 0$ otherwise. Using these decision variables and letting $d_{uv} = 0$ if $(u, v) \notin E$, a correct mathematical programming formulation of the problem is

$$\min \quad \sum_{i=1}^{n} y_i \tag{2}$$

s.t.

$$\sum_{u=1}^{n-1} \sum_{v=u+1}^{n} d_{uv} x_{ui} x_{vi} + \sum_{u=1}^{n} \sum_{v=1}^{n} d_{uv} x_{ui} (1 - x_{vi}) \leq B \qquad \forall i \tag{3}$$

$$\sum_{u=1}^{n} \sum_{v=1}^{n} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{uv} x_{ui} x_{vj} \leq B \tag{4}$$

$$\sum_{i=1}^{n} x_{ui} = 1 \qquad \forall u \tag{5}$$

$$x_{ui} \leq y_i \qquad \forall u, \forall i \tag{6}$$

$$x_{ui}, y_i \in \{0, 1\}$$

The objective function (2) minimizes the number of rings. Constraints (3) limit the total demand on a ring to the bandwidth B and (4) forces the capacity of the federal ring to be less than or equal to B. Recall that the federal ring needs to carry all the traffic between sites on different rings. Constraints (5) make sure that every site is placed on exactly one ring and (6) insures that a ring is active if a site is placed on it.

The above formulation has a major drawback in that constraints (3) and (4) are nonlinear. In order to linearize these constraints, we define the following sets of variables. Let $z_{uvi} = 1$ if site u belongs to ring i and site v does *not* belong to ring i, and $z_{uvi} = 0$ otherwise. Also let $p_{uvi} = 1$ if site u and site v are both in ring i and $p_{uvi} = 0$ otherwise. Note that if nodes u and v are both assigned to ring i then $p_{uvi} = p_{vui} = 1$. If neither, or only one of sites u and v is assigned to ring i, then $p_{uvi} = p_{vui} = 0$. Since we require all nodes to be assigned to exactly one ring, p_{uvi} and p_{vui} must have the same value in a feasible solution and it suffices to define p_{uvi} only for $u < v$. It is possible, however, z_{uvi} and z_{vui} may have different values in a feasible

solution; for example, if site v is assigned to ring i, but site u is not, then $z_{uvi} = 0$ and $z_{vui} = 1$.

Using the new variables, (4) is replaced by

$$\sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{n} d_{uv} z_{uvi} \leq B \tag{7}$$

$$x_{ui} - x_{vi} \leq z_{uvi} \qquad \forall u, v \in V, \forall i. \tag{8}$$

And constraints (3) are replaced by

$$\sum_{u=1}^{n-1} \sum_{v=u+1}^{n} d_{uv} p_{uvi} + \sum_{u=1}^{n} \sum_{v=1}^{n} d_{uv} z_{uvi} \leq B \qquad \forall i \tag{9}$$

$$x_{ui} + x_{vi} - p_{uvi} \leq 1 \qquad \forall \{u, v | u < v\}, \forall i. \tag{10}$$

This gives us the following integer linear programming (ILP) formulation of SRAP:

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} y_i \\
\text{s.t.} \quad & \\
& \sum_{u=1}^{n} \sum_{v=u+1}^{n} d_{uv} p_{uvi} + \sum_{u=1}^{n} \sum_{v=1}^{n} d_{uv} z_{uvi} \leq B \qquad \forall i \\
& \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{n} d_{uv} z_{uvi} \leq B \\
& \sum_{i=1}^{n} x_{ui} = 1 \qquad \forall u \\
& x_{ui} \leq y_i \qquad \forall u, \forall i \\
& x_{ui} + x_{vi} - p_{uvi} \leq 1 \qquad \forall \{u, v | u < v\}, \forall i \\
& x_{ui} - x_{vi} \leq z_{uvi} \qquad \forall u, v \in V, \forall i \\
& x_{ui}, y_i, p_{uvi}, z_{uvi} \in \{0, 1\}
\end{aligned}
$$

In any feasible solution, p_{uvi} must be one if $x_{ui} = x_{vi} = 1$. It is possible for a feasible solution to have $p_{uvi} = 1$ and x_{ui}, x_{vi}, or both equal to zero. In a case like this, however, the solution will still be feasible (and use the same number of rings) if we set $p_{uvi} = 0$. Likewise, z_{uvi} must be one when $x_{ui} = 1$ and $x_{vi} = 0$. Again, it is possible for a feasible solution to have $z_{uvi} = 1$ and $x_{ui} = 0$, but if this happens, we can always set $z_{uvi} = 0$ without violating constraints 9. Thus, the new constraints enforce the "correct" relationships between the new variables.

For a graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges, the formulation has variables $y_1, y_2, \ldots y_n$. There can be at most n rings, so the number of x_{ui} variables is n^2. Recall that $d_{uv} = 0$ if there is no edge between u and v, so we only use the variables z_{uvi}, z_{vui} and p_{uvi} if $(u, v) \in E$. Likewise, we can ignore constraints 8 and 10 when (u, v) is not in the graph. As described above, the formulation requires variables z_{uvi}, z_{vui} and p_{uvi} for each pair of nodes $\{u, v\}$ and ring i. Thus, there are $2mn$ of the z_{uvi} variables and mn of the p_{uvi} variables. There are a total of $n^2 + 3mn + n$ binary variables and $n^2 + 3mn + 2n + 1$ constraints.

3 The k SONET Ring Assignment Problem

In Section 7 we discuss our computational experience with the previously described ILP formulation for SRAP. Our results indicate that using this formulation in conjunction with a commercial ILP solver (i.e. CPLEX) is not a practical approach to solving SRAP. In this section we present an alternative approach to solving SRAP that is based on solving a series of instances of a related, but different, optimization problem, the k *SONET Ring Assignment Problem* (k-SRAP). As we describe in Section 7 this "indirect" approach is an effective, practical method for solving SRAP.

In the following discussion, we first describe k-SRAP as a graph optimization problem. We then describe it in terms of the network design application described in Section 1. Next, we describe k-SRAP as an ILP in terms of the SRAP ILP given in the previous section. Finally, we discuss how we can use solutions to a series of k-SRAP instances to solve a given SRAP instance.

The k SONET Ring Assignment Problem can be described formally as the following graph optimization problem: Given an undirected graph $G = (V, E)$ with nonnegative edge weight d_{uv} for all $e = (u, v) \in E$ and an integer $k \leq n = |V|$, partition the nodes of G into at most k clusters so that the the total weight of the edges incident to two different clusters is minimized and such that the total weight of the edges incident to any given cluster is less than or equal to a given value B.

In terms of the network design application, k-SRAP has the same input data as SRAP with the additional parameter k. As in SRAP, we are still assigning each node to exactly one SONET ring and connecting the rings with a federal ring. The network designs for k-SRAP and SRAP both require that each ring has a capacity requirement of at most B. There are two important differences between the problems, however. First, k-SRAP does not constrain the capacity of the federal ring to be B. Instead, the objective is to minimize the amount of traffic that would put on the federal ring. The second difference is that we can use at most k rings in a k-SRAP solution whereas we could use up to n rings in a SRAP solution.

Starting with SRAP ILP formulation given in Section 2, k-SRAP can be formulated as an ILP by swapping the roles of the objective function and the constraint on the capacity of the

federal ring. This gives us the ILP formulation for k-SRAP shown below.

$$\begin{aligned}
\min \quad & \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{n} d_{uv} z_{uvi} \\
\text{s.t.} \quad & \\
& \sum_{u=1}^{n} \sum_{v=u+1}^{n} d_{uv} p_{uvi} + \sum_{u=1}^{n} \sum_{v=1}^{n} d_{uv} z_{uvi} \leq B \qquad \forall i \\
& \sum_{i=1}^{n} y_i \leq k \\
& \sum_{i=1}^{n} x_{ui} = 1 \qquad \forall u \\
& x_{ui} \leq y_i \qquad \forall u, \forall i \\
& x_{ui} + x_{vi} - p_{uvi} \leq 1 \qquad \forall \{u, v | u < v\}, \forall i \\
& x_{ui} - x_{vi} \leq z_{uvi} \qquad \forall u, v \in V, \forall i \\
& x_{ui}, y_i, p_{uvi}, z_{uvi} \in \{0, 1\}
\end{aligned}$$

3.1 A Smaller ILP Formulation of k-SRAP

In this section we present a smaller formulation (i.e. one with fewer variables and constraints) of k-SRAP as an ILP. As before, let $p_{uvi} = 1$ if sites u and v are both on ring i and $p_{uvi} = 0$ otherwise. Also let $x_{ui} = 1$ if site u is assigned to ring i and $x_{ui} = 0$ otherwise. Define $W_u = \sum_{v \neq u} d_{uv}$ as the total weight of the edges incident to node u. Because the maximum number of rings is fixed, the objective is to minimize the traffic on the federal ring.

$$\begin{aligned}
& k\text{-SRAP} \\
& \max \quad \sum_{i=1}^{k} \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} p_{uvi} d_{uv} && (11) \\
& s.t. \\
& \qquad \sum_{u=1}^{n} x_{ui} W_u - \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} p_{uvi} d_{uv} \leq B \qquad \forall i && (12) \\
& \qquad \sum_{i=1}^{k} x_{ui} = 1 \qquad \forall u && (13) \\
& \qquad p_{uvi} \leq x_{ui} \qquad \forall i, \{u, v | u < v\} && (14)
\end{aligned}$$

$$p_{uvi} \leq x_{vi} \qquad \forall i, \{u, v | u < v\} \tag{15}$$

$$x_{ui}, p_{uvi} \in \{0, 1\}$$

The objective function (11) maximizes the traffic within the rings or minimizes the traffic on the federal ring, which is given by

$$\frac{\sum_{u=1}^{n} W_u}{2} - \sum_{i=1}^{k} \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} p_{uvi} d_{uv}.$$

Constraints (12) limit the total demand on a ring to its bandwidth and (13) assigns each site to exactly one ring. Constraints (14) and (15) force the p_{uvi} variables to be one only if x_{ui} and x_{vi} are both equal to one. Note that it is possible to have a feasible solution in which $x_{ui} = x_{vi} = 1$ for some u, v and i, but $p_{uvi} = 0$. Given such a solution, however, one could always find an improved solution by setting p_{uvi} equal to 1. This would increase the objective function value by d_{uv} without violating any of the constraints. Thus, $p_{uvi} = 1$ in an optimal solution if and only if nodes u and v are both assigned to ring i.

This formulation requires $k + n + 2km$ constraints and $k(n + m)$ variables where k is the number of rings, n is the number of sites and m is the number of pair of sites which communicate. As with the SRAP formulation, it is possible to over specify the number of rings since the optimal solution may contain empty rings (all $x_{ui} = 0$ for some ring i).

3.2 Solving SRAP with k-SRAP

Consider an instance $\{G, B, k\}$ of k-SRAP. If the objective function value of the k-SRAP solution is less than or equal to B, then the k-SRAP solution is a feasible solution for the SRAP instance $\{G, B\}$. As we show in the following discussion, however, solving k-SRAP with $k = n$ is not necessarily equivalent to solving SRAP. This is because an SRAP solution that minimizes the number of rings, does not necessarily minimize the traffic on the federal ring. In other words, there be another feasible SRAP solution that uses more rings, yet puts less traffic on the federal ring; solving n-SRAP would give us this solution which is not optimal for SRAP.

Consider the SRAP problem instance in Figure 1. If we choose $b < \frac{3a}{4}$ and $B = 2b + 3a$, both solutions in the figure are feasible. Solution 1, an optimal SRAP solution, uses only two rings and has federal ring of capacity $3a$. Solution 2, an optimal n-SRAP solution, uses three

rings, but has a federal ring with a smaller capacity of $4b < 3a$. Thus, we cannot solve SRAP by simply setting $k = n$ and using the k-SRAP formulation. We can, however, solve SRAP by finding the smallest value of k for which the objective function value of the optimal k-SRAP solution is $\leq B$.

For example, solving n-SRAP for the graph in Figure 1 with $B = 2b + 3a$ gives a feasible SRAP solution with three rings. Thus, we know that the optimal SRAP solution uses at most three rings. Solving 2-SRAP for this problem instance, will give us Solution 1. Clearly, 1-SRAP is infeasible for the given value of B. Therefore, $k = 2$ is the smallest value of k for which the optimal k-SRAP solution is feasible for SRAP. Thus, the minimum number rings in any feasible SRAP solution is 2.

Instead of starting with $k = n$ and (possibly) solving n instances of k-SRAP to solve SRAP, we can use a binary search procedure to find the optimal value of k. This requires solving $O(\log n)$ instances of k-SRAP. The advantage of this approach, however, is that the k-SRAP formulation leads to more tractable integer programs than the SRAP formulation. We discuss this point in more detail in Section 7.

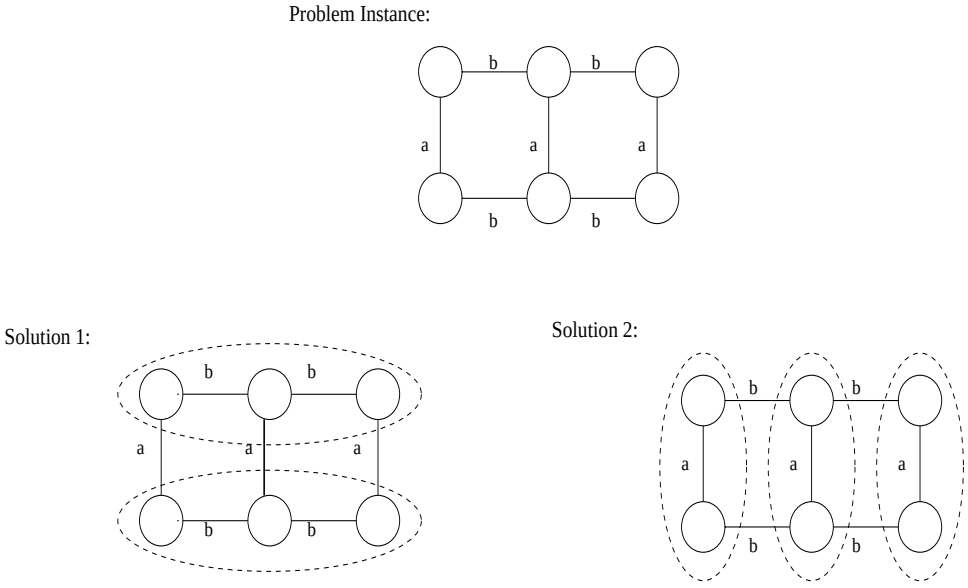

Figure 1: Minimizing the capacity of the federal ring vs. minimizing the number of rings.

4 Complexity of the SONET Ring Assignment Problem

In this section, we show that deciding whether or not the SONET Ring Assignment Problem has a solution is $\mathcal{NP}$-complete. The recognition version of SRAP can be stated as follows.

SONET Ring Assignment Problem
Instance: Undirected graph $G = (V, E)$, integers B and ℓ and a weight d_{uv} associated with each edge $(u, v) \in E$.
Question: Is there a partition of V into ℓ disjoint sets $V_1, V_2, \ldots, V_\ell$ such that

$$\sum_{u \in V_i, v \in V_i, u < v} d_{uv} + \sum_{u \in V_i, v \notin V_i} d_{uv} \leq B \text{ for } i = 1, 2, \ldots, \ell \text{ and,}$$

$$\sum_{i=1}^{\ell-1} \sum_{j=i+1}^{\ell} \sum_{u \in V_i} \sum_{v \in V_j} d_{uv} \leq B \ ?$$

We refer to the special case of SRAP where $\ell = 2$ and $d_{uv} = 1$ for all $(u, v) \in E$ as the *SONET Two-Ring Assignment Problem* (STRAP). We will show how to reduce the 3-*Regular Graph Bisection Problem*, which is defined below, to a special case of STRAP in polynomial time. The 3-Regular Graph Bisection Problem is known to be $\mathcal{NP}$-complete [WW93]. It follows that the SONET Ring Assignment Problem is $\mathcal{NP}$-hard.

Given a partition of the vertices of a graph $G = (V, E)$ into disjoint subsets W and $V \setminus W$, define $\Gamma(W)$ as the set of edges with one endpoint in W and the other in $V \setminus W$. The 3-Regular Graph Bisection Problem can now be stated as follows.

3-Regular Graph Bisection Problem
Instance: Undirected 3-regular graph $G = (V, E)$, $|V| = n$ (where n is an even number), and integer k.
Question: Is there a subset $W \subset V$, $|W| = n/2$ such that $|\Gamma(W)| \leq k$?

The main result of this section is that The SONET ring assignment problem is $\mathcal{NP}$-hard. More formally, we show that the following recognition is $\mathcal{NP}$-complete.

SONET Two-Ring Assignment Problem (STRAP) on a 3-Regular Graph
Instance: Undirected graph 3-regular graph $G = (V, E)$, $|V| = n$, and integer B.
Question: Is there a partition of V into sets W and $V \setminus W$, such that the number of edges incident to W ($V \setminus W$) is at most B ?

In order to prove this, we first prove the following lemma.

Lemma 4.1 *Let $G = (V, E)$ be a 3-regular graph and let $W \subset V$ be a subset of its vertices. Then the number of edges incident to W (with at least one endpoint in W) is $\frac{3|W|}{2} + \frac{|\Gamma(W)|}{2}$.*

Proof

Let the number of edges incident to W be $a + b$, where a is the number of edges with both endpoints in W and $b = |\Gamma(W)|$ is the number of edges with exactly one endpoint in W. The sum of degrees of vertices in W is $2a + b = 3|W|$. Therefore $a = \frac{3|W|}{2} - \frac{b}{2}$ and the number of edges incident to W is

$$a + b = \frac{3|W|}{2} - \frac{b}{2} + b = \frac{3|W|}{2} + \frac{b}{2} = \frac{3|W|}{2} + \frac{|\Gamma(W)|}{2}.$$

■

Theorem 4.1 *STRAP on a 3-regular graph is $\mathcal{NP}$-complete.*

Proof

The problem is clearly in $\mathcal{NP}$. Given a set W, the number of edges incident to W and to $V \setminus W$ can be checked in linear time.

We show that the 3-Regular Graph Bisection Problem can be reduced to STRAP on a 3-regular graph in in polynomial time. Let $\{G, k\}$ be an instance of the 3-regular graph bisection problem. The corresponding instance of STRAP is $\{G, B\}$, with $B = \frac{3n}{4} + \frac{k}{2}$. We claim that $\{G, k\}$ has a *yes* answer if and only if $\{G, B\}$ has a *yes* answer.

Let $W \subset V$ be a solution to $\{G, k\}$. By lemma 4.1, the number of edges incident to W $(V \setminus W)$ is

$$\frac{3n}{4} + \frac{|\Gamma(W)|}{2} \leq \frac{3n}{4} + \frac{k}{2} = B.$$

Hence, W is a solution to $\{G, B\}$.

Now let W be a solution to $\{G, B\}$. If $|W| = |V \setminus W| = \frac{n}{2}$, the solution is a bisection of V. From lemma 4.1, the number of edges incident to $W(V \setminus W)$ is

$$\frac{3n}{4} + \frac{|\Gamma(W)|}{2}.$$

Since the solution is feasible for $\{G, B\}$,

$$\frac{3n}{4} + \frac{|\Gamma(W)|}{2} \leq B = \frac{3n}{4} + \frac{k}{2}.$$

Thus, $|\Gamma(W)| \leq k$ and the solution is also feasible for $\{G, k\}$.

Now, suppose $|W| = \frac{n}{2} + c$ where $c > 0$. From lemma 4.1, the number of edges incident to W is

$$\frac{3n}{4} + \frac{3c}{2} + \frac{|\Gamma(W)|}{2}.$$

Since the solution is feasible for $\{G, B\}$,

$$\frac{3n}{4} + \frac{3c}{2} + \frac{|\Gamma(W)|}{2} \leq B = \frac{3n}{4} + \frac{k}{2}.$$

Thus, $|\Gamma(W)| \leq k - 3c$. If we move c nodes from W to $V \setminus W$, the number of edges with end points in both sets will be at most k (since G is a 3-regular graph). Therefore, the resulting solution is feasible for $\{G, k\}$. Therefore, $\{G, k\}$ has a *yes* answer if and only if $\{G, B\}$ has a *yes* answer. ■

Since SRAP contains STRAP on a 3-regular graph as a special case, the main result of this section follows immediately.

Corollary 4.1 *SRAP is $\mathcal{NP}$-hard.*

Corollary 4.2 *k-SRAP is $\mathcal{NP}$-hard.*

Proof

The recognition version of k-SRAP can be stated as follows.

k-SONET Ring Assignment Problem
Instance: Undirected graph $G = (V, E)$, integers B, C and k, and a weight d_{uv} associated with each edge $(u, v) \in E$.
Question: Is there a partition of V into k disjoint sets $V_1, V_2, \ldots, V_k$ such that

$$\sum_{u \in V_i, v \in V_i, u < v} d_{uv} + \sum_{u \in V_i, v \notin V_i} d_{uv} \leq B \text{ for } i = 1, 2, \ldots, k \text{ and,}$$

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{u \in V_i} \sum_{v \in V_j} d_{uv} \leq C \text{ ?}$$

We can restrict k-SRAP to STRAP on a 3-regular graph by allowing only instances where G is 3-regular, $C = B$, $k = 2$, and $d_{uv} = 1$ for all $(u, v) \in E$. ■

5 Upper Bounds

In this section we derive an upper bound on the number of rings found by all heuristic solutions that share a certain simple property. In the following section, we present polynomial-time heuristic algorithms that attempt to construct SRAP solutions that have this property. If we can find such a solution, then we know that it uses at most twice the optimal number of rings. If we then wish to use k-SRAP to find an optimal solution, we can use this upper bound to reduce the range of values we have to search for the optimal value of k.

Let k be the number of rings of some heuristic solution, say *Heus*. Define the operation of *merging* rings i and j as assigning all nodes in ring j to ring i. We say that a solution is *minimal* if no two of its rings can be merged to form a larger ring; i.e. the weight of the resulting ring would be strictly larger than B. A solution is said to be *feasible* if the weight of every ring, including the federal ring, is B or less.

Theorem 5.1 *If* Heus *is a minimal, feasible solution with k rings then $k \leq 2k^*$, where k^* is the number of rings in an optimal solution.*

Proof

For every ring of *Heus*, we partition the edges incident to it into two sets: the edges with both endpoints in the ring and the edges with exactly one endpoint in the ring. Let a_i be the sum of the weights of the edges with both endpoints in the ith ring, let b_i be the sum of the weights of the edges with exactly one endpoint in the ith ring, and let w_{ij} be the weight of the edges with one endpoint in the ith ring and one endpoint in the jth ring, $i \neq j$. Because no merging is possible,

$$a_i + b_i + a_j + b_j - w_{ij} > B \quad \forall i \neq j. \tag{16}$$

Indeed the left hand side of inequality (16) is the weight of the ring obtained by merging ring i and ring j and, since we have assumed that no such merge is possible within *Heus*, this weight must be strictly larger than B.

Adding these $\binom{k}{2}$ inequalities (there is an inequality for every pair of rings), we obtain

$$(k-1)\sum_{i=1}^{k} a_i + (k-1)\sum_{i=1}^{k} b_i - \sum_{i<j} w_{ij} > \binom{k}{2} B. \tag{17}$$

Note that $\sum\limits_{i<j} w_{ij} = \frac{1}{2}\sum\limits_{i=1}^{k} b_i$ is the weight of the federal ring. Replacing $\sum\limits_{i<j} w_{ij}$ by $\frac{1}{2}\sum\limits_{i=1}^{k} b_i$ and dividing both sides of (17) by $k-1$, we have

$$\sum_{i=1}^{k} a_i + \sum_{i=1}^{k} b_i - \frac{\sum\limits_{i=1}^{k} b_i}{2(k-1)} > \frac{k}{2} B. \tag{18}$$

On the other hand, the total weight of the edges in the graph is $\frac{1}{2}\ W = \sum\limits_{i=1}^{k} a_i + \sum\limits_{i=1}^{k} b_i$ and a lower bound on k^* is $\frac{W}{B}$. If k^* is the number of rings in an optimal solution, then

$$k^* B \geq \sum_{i=1}^{k} a_i + \frac{\sum\limits_{i=1}^{k} b_i}{2}. \tag{19}$$

From (18)

$$k^* B + \frac{\sum\limits_{i=1}^{k} b_i}{2} \geq \sum_{i=1}^{k} a_i + \sum_{i=1}^{k} b_i > \frac{k}{2} B + \frac{\sum\limits_{i=1}^{k} b_i}{2(k-1)}. \tag{20}$$

Thus,

$$k^* > \frac{k}{2} - \frac{\sum\limits_{i=1}^{k} b_i}{2B}\left(1 - \frac{1}{k-1}\right). \tag{21}$$

But, since the heuristic solution is feasible, $\sum\limits_{i=1}^{k} b_i \leq 2B$ and it follows that because k and k^* are integers,

$$k \leq 2k^* \tag{22}$$

■

6 Heuristic Algorithms for SRAP

In this section we present polynomial-time heuristic algorithms for the SONET Ring Assignment Problem. Since the problem of finding a feasible SRAP solution is itself $\mathcal{NP}$-complete, no heuristic with a polynomial worst-case running time can guarantee to always find a feasible

solution. As shown in Section 7, however, the algorithms presented in this section usually find good, feasible solutions.

Before describing the algorithms, we define some additional notation. Denote the ring to which node u belongs by $\text{ring}(u)$ and the nodes in that ring by $R_i = \{v|\text{ring}(v) = i\}$.

6.1 The Edge-Based Heuristic

The *edge-based* heuristic starts by assigning each node to a different ring. Note that if the problem is feasible, it follows that these initial rings must be feasible. Thus, the initial solution is only infeasible if the capacity of the federal ring is too high. The algorithm examines the edges one at a time in non-increasing order of d_{uv}. If $i = \text{ring}(u) \neq j = \text{ring}(v)$, the algorithm merges R_i and R_j if $R_i \cup R_j$ is a feasible ring. Notice that the capacity of the federal ring can only decrease at each iteration and that, with the exception of the federal ring, the algorithm never creates an infeasible ring. There is no guarantee, however, that the algorithm will return a feasible solution. Since the solution returned by this algorithm is always minimal, it follows from Theorem 5.1 that if it is feasible, then it uses at most twice the number of rings as necessary. The algorithm is described formally below.

Algorithm Edge-Based Heuristic
$F \leftarrow E$
$\forall v \in N$ $\text{ring}(v) \leftarrow v$
While $F \neq \emptyset$
  Choose a maximum capacity edge $(u, v) \in F$
  $i \leftarrow \text{ring}(u)$, $j \leftarrow \text{ring}(v)$
  If $R_i \cup R_j$ is a feasible ring **Then**
    $\forall v \in R_j$ $\text{ring}(v) \leftarrow i$.
    $F \leftarrow F \setminus \{(x, y)|\ \text{ring}(x) = i,\ \text{ring}(y) = j\}$
  Else
    $F \leftarrow F \setminus \{(u, v)\}$

Note that if (u, v) is not an edge of the original graph, we add an edge of weight zero between nodes u and v before implementing the edge-based heuristic.

6.2 The Cut-Based Heuristic

Like the edge-based heuristic, the *cut-based* heuristic first assigns each node to its own ring. At each iteration, it merges two rings with maximum edge weight between them (the total weight

of the edges with one endpoint in each of the two rings) that can be merged, i.e. the resulting capacity is B or less. This algorithm also returns a minimal solution, so it follows that if the solution is feasible, then it uses at most twice the necessary number of rings.

Algorithm Cut-Based Heuristic
$\mathcal{L} \leftarrow \emptyset$
$\forall v \in N$ ring$(v) \leftarrow v, \mathcal{L} \leftarrow \mathcal{L} \cup \{\text{ring}(v)\}$
Repeat
  max $\leftarrow 0, \mathcal{M} \leftarrow \emptyset$
  For $\{s, t | s \neq t\} \in \mathcal{L}$
    If $R_s \cup R_t$ is a feasible ring **Then**
      $E_{st} \leftarrow \{\{u, v\} | \text{ring}(u) = s, \text{ring}(v) = t\}$
      cut $\leftarrow \sum_{(u,v) \in E_{st}} w_{(uv)}$
      If cut $>$ max **Then**
        max $\leftarrow$ cut, $\mathcal{M} \leftarrow \{st\}$
  If $\mathcal{M} \neq \emptyset$ **Then**
    Merge $\mathcal{M}$
Until No rings can be merged ($\mathcal{M} = \emptyset$)

6.3 Node-Based Heuristic

In the node-based heuristic, the user specifies the number of rings, say k. The method first assigns a different site, picked randomly among the set of sites, to each of the k rings. The remaining sites are assigned one by one. At each iteration the heuristic considers a ring , say i, with the largest unused capacity. It selects an unassigned node v with the largest traffic with current nodes in ring i and adds v to ring i. Note that some of the rings may end up with a capacity that exceeds the limit B. Also the federal ring may have a capacity that exceeds B. If the k rings all have capacity B or less, we apply a post-processing procedure that attempts to merge the rings as is done in the cut-based heuristic.

Algorithm Node-Based Heuristic
$U \leftarrow V$
For $i = 1$ to k
    Choose $u \in U$, $R_i \leftarrow u$, $U \leftarrow U \setminus \{u\}$

While $U \neq \emptyset$
    Choose a minimum capacity ring R_i
    Choose $u \in U$ to maximize $\sum_{\{v \in R_i\}} d_{uv}$
    ring$(u) = R_i$, $U \leftarrow U \setminus \{u\}$

7 Empirical Results

In this section we present some empirical results of testing the heuristic algorithms and integer programming formulations for SRAP. We implemented the heuristic algorithms described in the previous section in MATLAB and tested them on two "real-world" problem instances given to us by the telecommunications provider mentioned in Section 1.2 as well as a series of randomly-generated test problems.

7.1 Real-World Problems

The data in the first problem instance come from a network interconnecting sixteen private telephone switches. The total demand is approximately 225 Mbs (Mega bits per second). We fixed $B = 155$ Mbs which corresponds to *STM-1*, the smallest line container for SONET. All heuristics found a solution with three rings. CPLEX took 2.60 seconds of CPU time to determine that the k-SRAP formulation for this problem with 2 rings has no feasible solution. Thus, the heuristic solutions for this problem are optimal.

The data in the second instance come from a network of seventeen public administration centers interconnected by a regional network of fifty-five links. The total demand is approximately 845.3 Mbs. No solution was found with $B = 155$ Mbs (*STM-1*). Using CPLEX we verified the problem is infeasible with $B = 155$ Mbs. This took 409.17 seconds of CPU time. With $B = 622$ Mbs, which corresponds to *STM-2*, the edge-based and cut-based heuristics found a solution with three rings while the node-based heuristic found one with two rings which, given the total demand, is the minimum number of rings with $B = 622$ Mbs. The CPLEX runs for these two

data sets were performed with version 4.0.8 running on a Sun Microsystems SPARCstation 20 Model 151 workstation with a 150 Mhz hyperSPARC processor with 288 megabytes of memory. The heuristics were implemented on the same machine with MATLAB version 4.2c.

7.2 Random Test Problems

We generated two types of random test data[1]. To create *geometric* demand graphs, we randomly selected n points in the unit square to be the nodes. We then included the edge (i, j) if the Euclidean distance between points i and j was less than or equal to a given distance d. We also generated a set of *random* demand graphs by starting with a complete graph and retaining edge (i, j) with probability p. In other words, each edge of the complete graph exists in the random graph independently with probability p.

The rationale behind the choice of geometric demand graphs is that natural clusters are created. Sites tend to communicate more with their close neighbors than with their distant ones. This is usually what happens in the real world and SONET networks are especially well suited for this type of traffic.

We generated two sets of each type of graph (geometric or random). The first sets were designed to represent *low-demand* cases where 155 Mbs ADMs are being considered and second were *high-demand* cases where the ring capacity is 622 Mbs. ADMs come in a fixed set of standard sizes and 622 Mbs is the next larger capacity after 155 Mbs. The demand on edge (i, j) was determined by a discrete, uniform random variable indicating the equivalent number of T1 lines required for the anticipated volume of traffic between sites i and j. A T1 line has a capacity of approximately 1.5 Mbs and is a fundamental "building block" for small to medium-sized corporate telecommunications networks. The number of T1 lines for an edge was selected from the range $[3, 7]$ for low-demand cases and from the range $[11, 17]$ for high-demand cases. Thus, the average demand pair was 7.5 Mbs in low-demand cases and 21 Mbs in high-demand cases.

The values of d and p for the geometric and random graphs are given in Table 1. These values were determined experimentally. For a given graph, the total demand (i.e. the sum of the edge weights) D divided by the ADM size B gives a lower bound of $k_{\ell b} = \lceil \frac{D}{B} \rceil$ on the

[1] These randomly-generated problem instances are available at www.seas.smu.edu/~olinick/srap/GRAPHS.tar.

Graph Type	$n = 15$	$n = 25$	$n = 30$	$n = 50$
Geometric Low-Demand	$d = 0.45$	$d = 0.28$	$d = 0.24$	$d = 0.16$
Geometric High-Demand	$d = 0.45$	$d = 0.3$	$d = 0.27$	$d = 0.19$
Random Low-Demand	$p = 0.35$	$p = 0.16$	$p = 0.12$	$p = 0.055$
Random High-Demand	$p = 0.45$	$p = 0.2$	$p = 0.14$	$p = 0.06$

Table 1: Parameters for Geometric and Random Graphs.

number of rings in a feasible SRAP solution. We wanted to choose values for d and p that were large enough to avoid generating trivial cases where the problem could be solved with one or two rings. As we increased the size of the graphs, however, we wanted to avoid generating cases where $k_{\ell b}$ was too large. If too many rings are required, it is likely that the capacity of the federal ring will exceed B. Thus, we decreased the values of d and p for larger graphs.

The heuristic algorithms described in Section 6 do not specify how to break ties. For example, at any given iteration of the edge-based heuristic, there may be more than one maximum-weight edge to choose for the merge operation. Therefore, we used implementations of the heuristics which use a random number generator to break ties, and ran each algorithm ten times on each graph.

We started the node-based heuristic with $k = k_h$ rings, where k_h is the number of rings in the best solution found by the edge and cut-based heuristics. If the node-based heuristic found a solution with k rings, we then decremented k to $k - 1$; if not, we tried again with k. This process was repeated until the algorithm had been tried ten times. In cases where neither of the other two heuristics found a feasible solution, we ran the node-based heuristic, with the post-processing step described in Section 6.3, ten times and each time started the algorithm with a random value for k selected from the range $[k_{\ell b}, n]$. We present and discuss the results from these experiments in Section 8 and Appendix A.

The CPLEX runs for these problems were performed with version 6.0 running on a Sun Microsystems Ultra 10 workstation with a 300 Mhz UltraSPARC-IIi processor and 128 megabytes of memory.

8 Results for Low-Demand Geometric Graphs

In this section we present and discuss the results of running the heuristics, and CPLEX with the SRAP and k-SRAP formulations, on the low-demand geometric graphs. The results for high-demand geometric and random graphs are summarized at the end of this section and presented in detail in Appendix A.

The results of running the three heuristics on a set of graphs can be classified into three categories. If the number of rings k_h in the best heuristic solution is equal to the lower bound $k_{\ell b}$, then we know that the solution is optimal. We refer to these cases as optimal or *type 1* results. In cases where the best heuristic solution uses some number of rings $k_h > k_{\ell b}$, the solution is feasible (and minimal), but not necessarily optimal. We refer to these cases as *type 2* results. Clearly, type 1 results are the best possible. Type 2 results can be viewed as partial successes of the heuristics since they give us a bound on the minimum number rings in an optimal solution. Cases where all three heuristics fail to find a feasible solution (and therefore give us no useful information about the minimum number of rings) are referred to as *type 3* results.

8.1 Type 1 Results

For the 40 low-demand geometric graphs, the heuristics gave type 1 results for 7 graphs, type 2 results for 16 and type 3 results for the remaining 17. Tables 2 and 3 give statistics for these graphs and their corresponding heuristic solutions. The number of nodes and edges for a given graph are given in columns 2 and 3, respectively. The value of D is given in Mbs in column 4 and the lower bound $k_{\ell b}$ on the number of rings is given in column 5. The numbers of rings in the best solution found by the edge-based, cut-based and node-based heuristics are given in the columns marked EB, CB and NB, respectively. The number of rings in the best heuristic solution is given in column 6. Note that a blank entry in one of these columns indicates that the particular algorithm failed to find a feasible solution for the given graph. The last three columns give the total time in CPU seconds for the six runs of each algorithm.

From the results reported in Tables 2 and 3, it appears that the cut-based heuristic consistently dominates the other two algorithms. With the exception of the last graph listed in

					Number of Rings				CPU Seconds		
Graph	n	m	Tot. Mbs.	$k_{\ell b}$	k_h	EB	CB	NB	EB	CB	NB
GL.15.1	15	41	313.5	3	3	3	3	3	0.65	1.65	2.18
GL.15.7	15	44	313.5	3	3	3	3	3	0.58	1.76	2.04
GL.25.3	25	49	354	3	3	4	3		1.78	6.8	7.22
GL.25.7	25	49	325.5	3	3	3	3	3	1.76	6.17	9.31
GL.30.5	30	58	391.5	3	3	4	3		2.58	10.1	12.29
GL.50.8	50	86	646.5	5	5	6	5		11.19	49.15	56.8
GL.50.9	50	63	499.5	4	4	4	4	4	9.21	49.24	73.64

Table 2: Type 1 Results for Low-Demand Geometric Graphs ($B = 155$ Mbs).

Table 3, the node-based heuristic always returned a solution with k_h rings. The edge-based heuristic ran significantly faster than the other two, but the running times for all heuristics are reasonable; most of the 50-node problems were solved in less than a minute of CPU time.

8.2 Type 2 Results

Notice that in all 16 of the type 2 results that $k_h = k_{\ell b} + 1$. For each graph that gave a type 2 result, we used the k-SRAP formulation described in Section 3 to determine whether or not there was a feasible SRAP solution with exactly $k_{\ell b}$ rings. Since the objective function in k-SRAP minimizes the required capacity of the federal ring, we cut off nodes of the branch-and-bound tree with objective function values greater than B. Rather than searching for an optimal solution, we terminated the branch-and-bound procedure once a feasible integer solution was found.

In all 16 cases, it turned out the best heuristic solution was optimal. The column labeled $k = k_{\ell b}$ in Table 4 shows the time in CPU seconds for CPLEX to verify that the associated k-SRAP formulation for each graph in Table 3 was infeasible for $k_{\ell b}$. In the next column, labeled "Cut-Based and k-SRAP", we add this time to the running time of the cut-based heuristic to calculate total time to find an optimal solution. Note that for the graph GL.50.7 the best solution returned by the cut-based heuristic used six rings. So in this case, the entry in the last column is the time for the cut-based heuristic plus k-SRAP with $k = 5$ and $k = 4$. The last column of Table 4 shows the time in CPU sections required to find the optimal number of rings k^* using the binary search procedure with the k-SRAP formulation as described in Section 3.

In most cases, the binary search procedure with the k-SRAP formulation was fairly quick and in some cases actually took less time than the combination of the cut-based heuristic followed by k-SRAP. We found that the k-SRAP IPs were generally easy to solve for small values of k; however, there are two points in the binary search where we may run into trouble with this approach. The first is that the initial value of $k = \lceil \frac{k_{lb}+n}{2} \rceil$ can be be fairly large. For example, the initial value for k for the three 50-node graphs in Table 4 was 27 for GL.50.1 and GL.50.7 and 28 for GL.50.4. CPLEX took 33, 60 and 210 seconds of CPU time to solve the initial k-SRAP IPs for GL.50.1, GL.50.7 and GL.50.4, respectively. These running times are reasonable; however, the initial k-SRAP IPs for many of the other randomly-generated graphs took on the order of 25 seconds of CPU time to solve. For one problem, RH.25.5 (see Table 19), CPLEX took a total of 46,000 CPU seconds (more than 12 hours !) to solve the first two IPs in the binary search.

Unless $k^* = k_{lb}$, the binary search will eventually attempt to solve an infeasible k-SRAP IP. This is the other point in the binary search where we may see long running times for solving k-SRAP. With the exception of graph GL.50.4, CPLEX was able to quickly prove that the last k-SRAP instance was infeasible for the graphs in Table 4. By way of comparison, we ran CPLEX on a five-ring SRAP formulation for GL.50.4 (i.e. there were only five y_i variables). CPLEX ran for 86,000 CPU seconds on this problem, but was unable to prove that the problem was infeasible before stopping due to a preset 400-megabyte limit on the size of the branch-and-bound tree. Proving the infeasibility of the last k-SRAP IP in the binary search was generally not a time-consuming procedure for the other types of randomly-generated graphs, however.

8.2.1 SRAP Results for Type 2 Problems

In order to compare the effectiveness of the SRAP and k-SRAP formulations, we ran CPLEX with the SRAP formulation on each of the graphs listed in Table 4. In order to reduce the size of the SRAP integer programs, we preassigned a node u with maximum total demand (W_u) to ring 1. We also added a set of constraints that set the first k_{lb} of the y_i variables to one. CPLEX found optimal solutions for three of the 15-node graphs and three of 30-node graphs and found a feasible solution with five rings for graph GL.25.4. CPLEX failed to find any solution within a preset three-hour limit on CPU time for the remaining nine graphs. In the cases where it

did find solutions, it found them fairly early on in the branch-and-bound process and spent most of its time attempting to prove the solutions were optimal. CPLEX was not able to prove optimality for any of these cases within the time limit; if we had not previously solved these problems with k-SRAP, we would not know whether the 7 solutions in question were optimal.

8.3 Type 3 Results

The heuristics all failed to find feasible solutions for the remaining 17 low-demand geometric graphs. Table 5 shows the statistics from running the algorithms on these graphs. As mentioned in Section 5, we cannot guarantee that any of the heuristics will find a solution for a feasible problem instance since they are polynomial-time algorithms and SRAP is $\mathcal{NP}$-hard. Thus, it is possible that the problems listed in Table 4 are feasible. So, we ran the heuristics an additional 1000 times each for each of these graphs. These additional runs did not produce solutions to any of these problems.

Next, we attempted to find feasible solutions for these problems with the k-SRAP formulation. As indicated in Table 4, the k-SRAP formulation worked well for the type 2 problems. In these cases, we were able to find feasible SRAP solutions (i.e. ones where the objective function value was less than or equal to B) fairly quickly even for the largest value of k in the binary search. Therefore, we gave CPLEX a time limit of 3 hours of CPU time to find a feasible SRAP solution using the k-SRAP formulation with $k = n$ for each graph giving a type 3 result. CPLEX was able to prove that five of the 15-node problems were infeasible in an average of $2,494$ seconds of CPU time, but was unable to find feasible solutions or to prove infeasibility for the remaining graphs in Table 14 within the three-hour time limit. Finally, we gave CPLEX three hours of CPU time to find a feasible solution using the SRAP formulation. CPLEX was unable to find feasible solutions or to prove infeasibility for any of these problems within a preset three-hour limit per problem. Since none of the methods outlined above found feasible solutions for any of the graphs in Table 14, we conjecture that these problems are infeasible.

					Number of Rings				CPU Seconds		
Graph	n	m	Tot. Mbs.	$k_{\ell b}$	k_h	EB	CB	NB	EB	CB	NB
GL.15.4	15	41	295.5	2	3	3	3	3	0.63	1.73	3.05
GL.15.9	15	34	247.5	2	3	3	3	3	0.61	1.73	1.83
GL.25.1	25	54	403.5	3	4	4	4		1.91	7.05	7.23
GL.25.4	25	59	429	3	4	4	4	4	1.61	6.72	14.5
GL.25.8	25	57	445.5	3	4		4		2.02	6.31	7.12
GL.30.1	30	57	424.5	3	4	4	4		2.76	11.01	12.26
GL.30.2	30	57	436.5	3	4	4	4	4	2.69	10.06	20.93
GL.30.3	30	57	408	3	4	4	4	4	2.6	11.11	17.25
GL.30.4	30	54	397.5	3	4	4	4	4	2.61	10.67	18.47
GL.30.6	30	54	405	3	4	4	4	4	2.72	10.32	13.56
GL.30.7	30	57	412.5	3	4	4	4	4	2.73	11.35	18.52
GL.30.8	30	60	430.5	3	4	4	4	4	2.7	11.07	19.8
GL.30.9	30	57	418.5	3	4	4	4	4	2.47	10.98	17.22
GL.50.1	50	85	616.5	4	5	5	5		9.75	46.66	57.03
GL.50.4	50	84	649.5	5	6	6	6		10.9	50.56	56.21
GL.50.7	50	85	601.5	4	5	5	6	5	10.33	48.56	96.64

Table 3: Type 2 Results for Low-Demand Geometric Graphs ($B = 155$ Mbs).

		CPU Seconds using k-SRAP		
Graph	k^*	Binary Search	$k = k_{\ell b}$	Cut-Based and k-SRAP
GL.15.4	3	2.43	0.05	1.78
GL.15.9	3	1.92	0.12	1.85
GL.25.1	4	16.71	1.1	8.15
GL.25.4	4	9.01	1.3	8.02
GL.25.8	4	74.36	0.26	6.57
GL.30.1	4	18.13	0.33	11.34
GL.30.2	4	18.11	0.18	10.24
GL.30.3	4	11.03	1.3	12.41
GL.30.4	4	13.34	3.7	14.37
GL.30.6	4	13.56	2.2	12.52
GL.30.7	4	14.14	1.3	12.65
GL.30.8	4	13.33	0.99	12.06
GL.30.9	4	11.06	4	14.98
GL.50.1	5	85.27	0.67	47.33
GL.50.4	6	12292.3	12000	12050.56
GL.50.7	5	87.4	1.4	57.56

Table 4: Finding Optimal Solutions for Type 2 Low-Demand Geometric Graphs ($B = 155$ Mbs).

					CPU Seconds		
Graph	n	m	Tot. Mbs.	LB	ET	CT	NT
GL.15.2	15	52	378	3	0.82	1.87	0.15
GL.15.3	15	39	330	3	0.74	1.73	0.15
GL.15.5	15	60	454.5	3	1	1.89	0.16
GL.15.6	15	46	343.5	3	0.77	1.72	0.16
GL.15.8	15	55	403.5	3	0.82	1.86	0.15
GL.15.10	15	62	474	4	1.08	1.8	0.16
GL.25.2	25	61	480	4	2.5	7.35	0.39
GL.25.5	25	61	484.5	4	2.39	7.38	0.4
GL.25.6	25	74	561	4	2.77	7.12	0.42
GL.25.9	25	63	471	4	2.55	6.56	0.4
GL.25.10	25	60	489	4	1.9	6.69	0.4
GL.30.10	30	76	567	4	3.92	11.3	0.56
GL.50.2	50	92	729	5	13.12	50.84	1.52
GL.50.3	50	98	757.5	5	12.64	49.84	1.52
GL.50.5	50	90	708	5	12.98	50.32	1.5
GL.50.6	50	92	694.5	5	11.43	50.15	1.51
GL.50.10	50	87	652.5	5	11.92	51.04	1.51

Table 5: Type 3 Results for Low-Demand Geometric Graphs ($B = 155$ Mbs).

8.4 Summary of Empirical Results

Section A gives analogous tables to Tables 2, 3, 4 and 5 for the high-demand geometric graphs and the random graphs. Table 6 shows the number of type 1, 2 and 3 results for each type of data set.

		Result Type		
Graph Type	Demand	Type 1	Type 2	Type 3
Geometric	Low	7	16	17
Geometric	High	12	15	13
Random	Low	7	21	12
Random	High	10	23	7

Table 6: Summary of Empirical Results.

As with the low-demand geometric graphs, the type 2 results turned out to be optimal for the three other sets of graphs. Again, the binary search procedure with the k-SRAP formulation was far more effective in finding solutions for this problem than the SRAP formulation. Using the k-SRAP formulation, the average time for finding optimal solutions for the problems giving type 2 results was 15.06, 2550.42, 215.42 and 1871.11 CPU seconds for the 15, 25, 30 and 50-node problems, respectively. The average time for the 25-node problems is skewed by graph RH.25.5 which took 46,000 CPU seconds to solve. Without this graph, the average time for 25-node problems drops to 215.42 seconds of CPU time. Using the SRAP formulation in the manner described in Section 8.3, CPLEX was unable to find feasible solutions for, or to prove the infeasiblitity of, all but six the problems giving type 2 results. These six problems were all low-demand random graphs with 15 or 25 nodes. Although five of these solutions obtained with SRAP turned out to be optimal, CPLEX was not able to verify optimality within the three-hour time limit.

We were able to find feasible solutions for a small number of problems giving type 3 results by running the heuristics an additional 1000 times each. Table 7 lists the values for $k_{\ell b}$, k_h, k^* and the time in CPU seconds to run the binary search procedure with k-SRAP on the range $[k_{\ell b}, k_h - 1]$ for these cases.

Using the SRAP formulation in the manner described in Section 8.3, CPLEX was unable

Graph	$k_{\ell b}$	k_h	k^*	CPU Seconds for k-SRAP
RL.25.1	3	4	4	2.9
RL.30.3	3	5	4	1.64
RL.30.6	3	5	4	110.56
RH.15.3	2	4	3	10.13
RH.50.7	4	5	5	16000

Table 7: Feasible Problems Giving Type 3 Results.

to find feasible solutions for, or to prove the infeasiblitity of, any of the problems giving type 3 results. Using the k-SRAP formulation with $k = n$, however, we found feasible solutions for all of the graphs listed in Table 7 except graph RH.50.7. We were also able to prove that no feasible solutions exist for the two geometric and three random 15-node low-demand graphs giving type 3 results.

The running times for the heuristics on the other three types of graph were consistent with the running times for the low-demand geometric graphs and the cut-based heuristic dominated the other two algorithms in terms of solution quality for the type 1 and 2 cases. However, the combination of the cut-based heuristic followed by k-SRAP was significantly faster than the binary-search procedure with k-SRAP for the other three sets of type 2 results. This difference in running time becomes more pronounced for larger graphs.

9 Conclusions

We have designed and implemented an exact solution procedure as well as fast, simple heuristic algorithms for the SONET ring assignment problem (SRAP). The algorithms have been tested on data from real-world problems and randomly-generated problem instances. The results suggest that an effective approach to solving SRAP is to use the heuristic algorithms to find an upper bound on the minimum number of rings and then use the k-SRAP formulation with a binary search procedure to find an optimal solution.

The binary-search procedure with the k-SRAP formulation is also effective, but more time-consuming. Practitioners who design SONET networks often conduct scenario analyses. They estimate the cost of building a new network or expanding an existing one under a variety of conditions involving different demand patterns and equipment costs. Such studies are short-

term in nature and may involve solving a series of related SRAP problems. Thus, it is desirable to solve the SRAP instances as quickly as possible. In this study, the heuristics rarely failed to find solutions for feasible problems. Given this and the fact that the first k-SRAP instances solved by the binary search procedure may take a long time to solve (recall our discussion of graph RH.25.5 in Section 8), it makes sense to first reduce the search range for k^* by running one or more of the heuristics several times before starting the binary search procedure.

We have also shown that feasible solutions returned by the cut-based and edge-based heuristics use at most twice the minimum number of rings. An interesting question for future research is whether one can make any useful conclusions about the optimal number rings when these algorithms fail to produce a feasible solution. As a practical matter, one can always determine this via the binary-search procedure with the k-SRAP formulation assuming sufficient computing resources and time are available. Alternatively, one can attempt to solve the problem with the next larger size ADM. In fact, a network planner may with do this even if the heuristics find a feasible solution as a larger ADM capacity may reduce the number of rings required which could, depending on the ADM costs, actually lead to a less expensive solution.

Another direction for future research on SRAP is to consider approaching it as a nonlinear programming problem with continuous variables. Recall that in Section 2, we introduced binary integer variables in order to remove the nonlinearities in two sets of constraints in our initial formulation of SRAP as a mathematical program. This allows us to approach the problem with standard methods for solving integer linear programs. An alternative approach would be to use well know formulation techniques to replace the binary integer variables in the initial SRAP formulation with continuous variables and attempt to solve the problem with nonlinear programming techniques.

A Additional Empirical Results for SRAP

					Number of Rings				CPU Seconds		
Graph	n	m	Tot. Mbs.	k_{lb}	k_h	EB	CB	NB	EB	CB	NB
GH.15.3	15	42	888	2	2	2	2	2	0.43	1.22	1.16
GH.15.6	15	41	871.5	2	2	2	2	2	0.38	1.29	1.16
GH.15.7	15	36	739.5	2	2	2	2	2	0.37	1.32	1.07
GH.15.10	15	39	790.5	2	2	2	2	2	0.38	1.25	1.07
GH.25.2	25	52	1069.5	2	2	2	2	2	0.9	4.79	4.37
GH.25.3	25	50	1032	2	2	2	2		1	4.74	4.21
GH.25.4	25	64	1323	3	3	4	3	3	1.07	4.94	4.67
GH.25.8	25	63	1326	3	3	3	3	3	0.98	5.01	5.06
GH.25.9	25	63	1309.5	3	3	3	3	3	1.06	5.38	6.31
GH.30.1	30	64	1347	3	3	3	3	3	1.48	7.73	7.87
GH.30.4	30	72	1504.5	3	3	3	3	3	1.42	7.52	7.91
GH.50.4	50	102	2140.5	4	4	5	4		5.1	34.04	32.98

Table 8: Type 1 Results for High-Demand Geometric Graphs ($B = 622$ Mbs).

					Number of Rings				CPU Seconds		
Graph	n	m	Tot. Mbs.	k_{lb}	k_h	EB	CB	NB	EB	CB	NB
GH.15.1	15	43	921	2	3	3	3	3	0.5	1.34	1.1
GH.15.2	15	50	1032	2	3	3	3	3	0.45	1.3	1.1
GH.15.8	15	52	1110	2	3		3		0.5	1.38	0.97
GH.15.9	15	48	1023	2	3	3	3	3	0.45	1.33	1.09
GH.25.1	25	57	1191	2	3	3	3	3	0.97	4.83	5.47
GH.25.7	25	71	1515	3	4		4		1.32	5.16	4.15
GH.30.2	30	75	1636.5	3	4	4	4	4	1.57	8.4	8.63
GH.30.3	30	78	1665	3	4	4	4		1.64	8.29	7.14
GH.30.5	30	73	1546.5	3	4	4	4	4	1.61	8.39	9.36
GH.30.9	30	75	1585.5	3	4	4	4	4	1.43	8.03	9.36
GH.30.10	30	57	1207.5	2	3	3	3	3	1.33	7.8	7.9
GH.50.1	50	110	2310	4	5	5	5		5.05	34.7	32.68
GH.50.5	50	107	2226	4	5	5	5		4.93	37.01	32.84
GH.50.6	50	116	2427	4	5	6	5		5.16	35.23	32.75
GH.50.7	50	108	2275.5	4	5	6	5	5	5.05	33.75	52.4

Table 9: Type 2 Results for High-Demand Geometric Graphs ($B = 622$ Mbs).

					CPU Seconds		
Graph	n	m	Tot. Mbs.	LB	ET	CT	NT
GH.15.4	15	75	1590	3	0.62	1.55	0.09
GH.15.5	15	76	1603.5	3	0.67	1.52	0.09
GH.25.5	25	86	1830	3	1.35	5.29	0.22
GH.25.6	25	91	1878	4	1.31	5.37	0.23
GH.25.10	25	88	1824	3	1.31	5.38	0.22
GH.30.6	30	87	1824	3	1.99	8.95	0.32
GH.30.7	30	94	1995	4	2.15	8.63	0.32
GH.30.8	30	92	1959	4	2.12	8.84	0.32
GH.50.2	50	130	2716.5	5	6.09	33.87	0.86
GH.50.3	50	143	3007.5	5	7.6	34.56	0.86
GH.50.8	50	126	2697	5	5.95	36.58	0.86
GH.50.9	50	142	3046.5	5	7.54	36.7	0.86
GH.50.10	50	112	2428.5	4	5.69	34.02	0.86

Table 10: Type 3 Results for High-Demand Geometric Graphs ($B = 622$ Mbs).

		CPU Seconds using k-SRAP		
Graph	k*	Binary Search with k-SRAP	k-SRAP with $k = k_{lb}$	Cut-Based and k-SRAP
GH.15.1	3	9.4	1	5.83
GH.15.2	3	11.1	0.6	5.76
GH.15.8	3	66.05	0.55	8.95
GH.15.9	3	13.82	0.52	8.81
GH.25.1	3	22.83	0.53	8.92
GH.25.7	4	737	14	22.03
GH.30.2	4	181.3	12	19.8
GH.30.3	4	378.9	4.8	39.5
GH.30.5	4	260.8	15	52.01
GH.30.9	4	177.1	36	71.23
GH.30.10	3	30.58	0.48	34.23
GH.50.1	5	3458	34	35.22
GH.50.5	5	1184	630	631.29
GH.50.6	5	973.8	5.8	7.12
GH.50.7	5	1702	63	64.25

Table 11: Finding Optimal Solutions for Type 2 High-Demand Geometric Graphs (B = 622 Mbs).

					Number of Rings				CPU Seconds		
Graph	n	m	Tot. Mbs.	k_{lb}	k_h	EB	CB	NB	EB	CB	NB
RL.15.3	15	28	208.5	2	2	2	2	2	0.38	1.46	1.06
RL.25.2	25	42	325.5	3	3	3	3	3	1.22	5.19	5.08
RL.30.1	30	44	336	3	3	3	3	3	1.61	8.13	7.91
RL.30.7	30	47	354	3	3	4	3		1.56	8.18	7.22
RL.50.3	50	65	489	4	4	4	4	4	5.5	35.04	65.85
RL.50.4	50	66	484.5	4	4	4	4		5.2	36.11	32.85
RL.50.5	50	61	471	4	4	4	4	4	5.25	34.36	42.78

Table 12: Type 1 Results for Low-Demand Random Graphs ($B = 155$ Mbs).

					Number of Rings				CPU Seconds		
Graph	n	m	Tot. Mbs.	k_{lb}	k_h	EB	CB	NB	EB	CB	NB
RL.15.1	15	32	250.5	2	3	3	3	3	0.52	1.41	1.11
RL.15.4	15	38	288	2	3	3	3	3	0.49	1.46	1.29
RL.15.6	15	34	249	2	3	3	3	3	0.45	1.25	1.1
RL.15.8	15	36	265.5	2	3	3	3	3	0.46	1.31	1.09
RL.15.9	15	32	252	2	3	3	3	3	0.47	1.35	1.1
RL.15.10	15	40	298.5	2	3		3		0.45	1.28	0.97
RL.25.4	25	50	358.5	3	4		4		1.27	4.97	4.15
RL.25.5	25	44	345	3	4	4	4	4	1.2	5.07	4.65
RL.25.6	25	51	364.5	3	4	4	4		1.2	4.95	4.15
RL.25.7	25	49	373.5	3	4		4	4	1.21	5.09	7.55
RL.25.8	25	40	303	2	3	3	3	3	1.11	5.1	4.64
RL.25.9	25	47	372	3	4	4	4		1.19	5.1	4.16
RL.25.10	25	48	361.5	3	4	4	4		1.26	5.15	4.14
RL.30.4	30	55	432	3	4	4	4		1.63	8.09	7.16
RL.30.5	30	47	361.5	3	4	4	4	4	1.64	8.98	7.97
RL.30.8	30	54	373.5	3	4	4	4	4	1.61	8.11	10.12
RL.30.10	30	53	405	3	4	4	4		1.72	8.88	7.13
RL.50.1	50	70	513	4	5		5		5.84	34.78	32.84
RL.50.6	50	62	462	3	4	4	4	4	5.03	37.66	49.31
RL.50.7	50	71	529.5	4	5	5	5		5.84	34.46	32.84
RL.50.10	50	62	459	3	4	4	4	4	5.48	34.31	39.52

Table 13: Type 2 Results for Low-Demand Random Graphs ($B = 155$ Mbs).

					CPU Seconds		
Graph	n	m	Tot. Mbs.	LB	ET	CT	NT
RL.15.2	15	43	340.5	3	0.57	1.38	0.09
RL.15.5	15	40	310.5	3	0.47	1.46	0.09
RL.15.7	15	51	405	3	0.59	1.5	0.09
RL.25.1	25	51	364.5	3	1.28	5.43	0.22
RL.25.3	25	54	438	3	1.48	5.32	0.22
RL.30.2	30	66	487.5	4	1.88	8.58	0.31
RL.30.3	30	58	415.5	3	1.74	8.56	0.32
RL.30.6	30	56	429	3	1.73	8.31	0.32
RL.30.9	30	63	487.5	4	1.95	8.33	0.32
RL.50.2	50	85	643.5	5	5.9	37.77	0.85
RL.50.8	50	81	609	4	5.87	35.03	0.85
RL.50.9	50	68	540	4	5.91	37.93	0.85

Table 14: Type 3 Results for Low-Demand Random Graphs ($B = 155$ Mbs).

		CPU Seconds using k-SRAP		
GRAPH	k^*	Binary Search with k-SRAP	k-SRAP with $k = k_{lb}$	Cut-Based and k-SRAP
RL.15.1	3	4.85	0.35	1.66
RL.15.4	3	30.17	0.17	1.52
RL.15.6	3	6.4	0.48	1.76
RL.15.8	3	9.93	0.3	5.27
RL.15.9	3	5.72	0.37	5.44
RL.15.10	3	55.43	0.33	5.28
RL.25.4	4	594	10	15.09
RL.25.5	4	101.3	47	52.1
RL.25.6	4	394.8	8.8	13.9
RL.25.7	4	206.8	4.7	9.85
RL.25.8	3	27.55	0.25	8.34
RL.25.9	4	194.8	7.6	16.58
RL.25.10	4	139.5	6.5	14.61
RL.30.4	4	338.98	0.98	9.86
RL.30.5	4	111.2	39	73.78
RL.30.8	4	147.8	31	68.66
RL.30.10	4	475.5	4.5	38.96
RL.50.1	5	1701	1.30E+02	164.31
RL.50.6	4	188.35	0.95	2.41
RL.50.7	5	3149	69	74.19
RL.50.10	4	311.1	1.1	9.23

Table 15: Finding Optimal Solutions for Type 2 Low-Demand Random Graphs ($B = 155$ Mbs).

					Number of Rings				CPU Seconds		
Graph	n	m	Tot. Mbs.	k_{lb}	k_h	EB	CB	NB	EB	CB	NB
RH.15.4	15	38	778.5	2	2	2	2	2	0.43	1.52	1.22
RH.15.7	15	39	804	2	2	2	2	2	0.45	1.4	1.26
RH.15.8	15	41	891	2	2	3	2	2	0.44	1.37	1.32
RH.25.9	25	41	831	2	2	2	2	2	1.07	5.07	5.08
RH.30.8	30	62	1332	3	3	4	3	3	1.79	8.91	11.76
RH.30.9	30	60	1288.5	3	3	3	3	3	1.69	8.57	9.2
RH.50.2	50	68	1413	3	3	3	3	3	5.14	38.18	49.72
RH.50.5	50	66	1380	3	3	3	3	3	5.05	36.27	41.98
RH.50.6	50	67	1384.5	3	3	3	3	3	5.25	36.98	41.36
RH.50.10	50	86	1870.5	4	4		4		6.31	37.99	38.2

Table 16: Type 1 Results for High-Demand Random Graphs ($B = 622$ Mbs).

					Number of Rings				CPU Seconds		
Graph	n	m	Tot. Mbs.	k_{lb}	k_h	EB	CB	NB	EB	CB	NB
RH.15.1	15	48	990	2	3	3	3	3	0.48	1.61	1.37
RH.15.5	15	47	955.5	2	3	3	3	3	0.43	1.48	1.29
RH.15.6	15	45	930	2	3	3	3	3	0.46	1.53	1.25
RH.15.9	15	43	909	2	3	3	3	3	0.43	1.41	1.25
RH.25.1	25	62	1297.5	3	4		4		1.39	5.57	4.83
RH.25.2	25	53	1069.5	2	3	3	3	3	1.06	5.04	5.49
RH.25.3	25	52	1084.5	2	3	3	3	3	1.17	5.18	5.49
RH.25.4	25	61	1314	3	4	4	4		1.36	5.94	4.97
RH.25.5	25	67	1402.5	3	4		4		1.22	5.49	4.88
RH.25.6	25	62	1296	3	4	4	4		1.26	5.54	4.82
RH.25.7	25	51	1072.5	2	3	3	3	3	1.17	5.06	5.33
RH.25.8	25	66	1353	3	4		4		1.34	5.32	4.88
RH.25.10	25	49	997.5	2	3	3	3	3	1.11	5.36	5.39
RH.30.1	30	50	1056	2	3	3	3	3	1.67	8.93	9.15
RH.30.2	30	67	1402.5	3	4		4		1.87	9.14	8.23
RH.30.3	30	52	1096.5	2	3	3	3	3	1.59	8.73	9.02
RH.30.4	30	56	1185	2	3	3	3	3	1.61	8.69	9.14
RH.30.6	30	70	1482	3	4		4		1.97	9.6	8.33
RH.30.7	30	67	1404	3	4		4		1.78	9.06	8.31
RH.30.10	30	64	1368	3	4		4		1.88	11.25	8.27
RH.50.1	50	81	1738.5	3	4	4	4	4	6.03	38.33	42.21
RH.50.8	50	80	1689	3	4	4	4	4	6.07	37.78	61.22
RH.50.9	50	73	1573.5	3	4	4	4	4	5.44	38.33	41.3

Table 17: Type 2 Results for High-Demand Random Graphs ($B = 622$ Mbs).

					CPU Seconds		
Graph	n	m	Tot. Mbs.	LB	ET	CT	NT
RH.15.2	15	58	1203	2	0.57	1.67	0.11
RH.15.3	15	53	1113	2	0.56	1.52	0.1
RH.15.10	15	58	1197	2	0.49	1.47	0.1
RH.30.5	30	82	1656	3	2.01	9.68	0.37
RH.50.3	50	89	1872	4	5.99	40.64	0.99
RH.50.4	50	89	1848	3	6.57	38.65	1.05
RH.50.7	50	90	1876.5	4	6.21	38.2	1.02

Table 18: Type 3 Results for High-Demand Random Graphs ($B = 622$ Mbs).

		CPU Seconds using k-SRAP		
graph	$k*$	Binary Search with k-SRAP	k-SRAP with $k = k_{\ell b}$	Cut-Based and k-SRAP
RH.15.1	3	5.92	0.32	1.48
RH.15.5	3	6.82	0.46	1.94
RH.15.6	3	6.19	0.68	2.21
RH.15.9	3	4.84	0.43	1.57
RH.25.1	4	572	74	79.57
RH.25.2	3	26.99	0.34	5.38
RH.25.3	3	27.53	0.31	5.49
RH.25.4	4	205.6	78	83.94
RH.25.5	4	49137	17	22.49
RH.25.6	4	96.8	3.9	9.44
RH.25.7	3	16.07	0.19	5.25
RH.25.8	4	947	47	52.32
RH.25.10	3	11.14	0.43	5.79
RH.30.1	3	20.53	0.27	9.2
RH.30.2	4	424	59	68.14
RH.30.3	3	20.04	0.21	8.94
RH.30.4	3	24.77	0.11	8.8
RH.30.6	4	1842.1	7.1	16.7
RH.30.7	4	347.7	43	52.06
RH.30.10	4	276	3.7	14.95
RH.50.1	4	359.5	1.9	40.23
RH.50.8	4	442.8	3.4	41.18
RH.50.9	4	261	22	60.33

Table 19: Finding Optimal Solutions for Type 2 High-Demand Random Graphs ($B = 622$ Mbs).

Acknowledgments

The authors would like to thank Elizabeth C. Junqueira for contributing Figure 1. The authors would also like to thank Ilan Adler, Michael and Judy Olinick and three anonymous referees for their helpful suggestions which have greatly improved the clarity and presentation of the material in this paper.

References

[CDSW95] S. Cosares, D.N. Deutsch, I. Saniee, and O.J. Wasem. SONET Toolkit: A decision support system for designing robust and cost-effective fiber-optic networks. *Interfaces*, 25(1):20–40, January-February 1995.

[CEFS99] I. Chlamtac, V. Elek, A. Fumagalli, and C. Szabo. Scalable wdm access network architecture based on photonic slot routing. *IEEE/ACM Transactions on Networking*, 7(1):1–9, February 1999.

[CS94] S. Cosares and I. Saniee. An optimization problem related to balancing loads on SONET rings. *Telecommunication Systems*, 3:165–181, 1994.

[DLMar] M. Dell'Amico, M. Labbé, and F. Maffioli. Exact solution of the SONET ring loading problem. To Appear.

[FGT95] M. Fischetti, J.J. Salazar Gonzalez, and P. Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26(2):113–123, 1995.

[GHO99] O. Goldschmidt, D. Hochbaum, and E. Olinick. The k-edge-partitioning problem. Working Paper, 1999.

[Kha97] S. Khanna. A polynomial time approximation scheme for the sonet ring loading problem. *Bell Labs Technical Journal*, 2(2):36–41, Spring 1997.

[Lag94] M. Laguna. Clustering for the design of SONET rings in interoffice telecommunications. *Management Science*, 40(11):1533–1541, November 1994.

[LSHK98] Y. Lee, H. Sherali, J. Han, and S. Kim. A branch-and-cut algorithm for solving an intra-ring synchronous optical network design problem. Working Paper, 1998.

[MKT97] Y-S. Myung, H-G. Kim, and D-W. Tcha. Optimal load balancing on SONET bidirectional rings. *Operations Research*, 45(1):148–152, January-February 1997.

[SSW98] A. Schrijver, P. Seymour, and P. Winkler. The ring loading problem. *SIAM Journal on Discrete Mathematics*, 11(1):1–14, February 1998.

[SVW98] A. Sutter, F. Vanderbeck, and L. Wolsey. Optimal placement of add/drop multiplexers: Heuristic and exact algorithms. *Operations Research*, 46(5):719–728, September/October 1998.

[WW93] D. Wagner and F. Wagner. Between min cut and graph bisection. *Lecture Notes in Computer Science*, 711:744–750, 1993.