COMBINATORIAL OPTIMIZATION IN TELECOMMUNICATIONS

MAURICIO G. C. RESENDE

Abstract. Combinatorial optimization problems are abundant in the telecommunications industry. In this paper, we present four real-world telecommunications applications where combinatorial optimization plays a major role. The first problem concerns the optimal location of modem pools for an internet service provider. The second problem deals with the optimal routing of permanent virtual circuits for a frame relay service. In the third problem, one seeks to optimally design a SONET ring network. The last problem comes up when planning a global telecommunications network.

1. Introduction

Combinatorial optimization problems are abundant in the telecommunications industry. In this paper, we present four real-world telecommunications applications where combinatorial optimization plays a major role.

In Section 2, we consider the PoP (point-of-presence) placement problem, an optimization problem confronted by internet access providers. The most common, and potentially least expensive, way for a customer to access the internet is with a modem by making a phone call to a PoP of the provider. It has been conjectured that potential customers are more likely to subscribe to internet access service if they can make a local (free unmetered) phone call to access at least one of the internet provider's PoPs. Given that the number of PoPs that can be deployed is limited by a number of constraints, such as budget and office capacity, one would like to place (or locate) the PoPs in a configuration that maximizes the number of customers than can make local calls to at least one PoP. We call this number of customers the *coverage*. A greedy randomized adaptive search procedure (GRASP) is used to find solutions to this location problem that, in real-world situations, are shown to be near-optimal.

A Frame Relay (FR) service offers virtual private networks to customers by provisioning a set of permanent (long-term) virtual circuits (PVCs) between customer endpoints on a large backbone network. During the provisioning of a PVC, routing decisions are made either automatically by the FR switch or by the network designer, through the use of preferred routing assignments, without any knowledge of future requests. Over time, these decisions usually cause inefficiencies in the network and occasional rerouting of the PVCs is needed. The new PVC routing scheme is then implemented on the network through preferred routing assignments. Given a preferred routing assignment, the FR switch will move the PVC from its current route to the new preferred route as soon as that move becomes feasible.

Date: July 2001.

Key words and phrases. Telecommunications, combinatorial optimization, linear programming, GRASP, local search, multi-commodity flows.

Section 3, deals with a GRASP for optimal routing of permanent virtual circuits for a frame relay service.

Survivable telecommunications networks with fast service restauration capability are increasingly in demand by customers whose businesses depend heavily on continuous reliable communications. Businesses such as brokerage houses, banks, reservation systems, and credit card companies are willing to pay higher rates for guaranteed service availability. The introduction of the Synchronous Optical Network (SONET) standard has enabled the deployment of networks having a high level of service availability. In Section 4, we describe an approach for optimal design a SONET ring network.

With the worldwide market liberalization of telecommunications, the international telecommunications environment is changing from the traditional bilateral mode of operation, where each network between pairs of administrations (AT&T and British Telecom, AT&T and France Telecom, British Telecom and France Telecom, etc.) is planned separately, to a more global, alliance-based, environment, where the network needs of several administrations may be planned simultaneously. This allows network planning to be done more in the manner that a single national network is designed, as opposed to many individual networks [6, 3, 24]. In Section 5, we describe a problem that comes up when planning a global telecommunications network.

2. Pop placement for an internet service provider

In this section, we consider the PoP (point-of-presence) placement problem, an optimization problem confronted by internet access providers. The most common, and potentially least expensive, way for a customer to access the internet is with a modem by making a phone call to a PoP of the provider. It has been conjectured that potential customers are more likely to subscribe to internet access service if they can make a local (free unmetered) phone call to access at least one of the internet provider's PoPs. Given that the number of PoPs that can be deployed is limited by a number of constraints, such as budget and office capacity, one would like to place (or locate) the PoPs in a configuration that maximizes the number of customers than can make local calls to at least one PoP. We call this number of customers the *coverage*.

A formal statement of the problem is given next. Let $J = \{1, 2, \dots, n\}$ denote the set of n potential PoP locations. Define n finite sets $P_1, P_2, \dots, P_n$, each corresponding to a potential PoP location, such that $I = \cup_{j \in J} P_j = \{1, 2, \dots, m\}$ is the set of the m exchanges that can be covered by the n potential PoPs. With each exchange $i \in I$, we associate a weight $w_i \geq 0$, denoting for example, the number of lines served by exchange i. A *cover* $J^* \subseteq J$ covers the exchanges in set $I^* = \cup_{j \in J^*} P_j$ and has an associated weight $w(J^*) = \sum_{i \in I^*} w_i$. Given the number $p > 0$ of PoPs to be placed, we wish to find the set $J^* \subseteq J$ that maximizes $w(J^*)$, subject to the constraint that $|J^*| = p$.

This problem, also known as the maximum covering problem (MCP) [23], has been applied to numerous location problems, including rural health centers [4], emergency vehicles [11], and commercial bank branches [25], as well as other applications [7, 9, 10]. It has an compact integer programming formulation, first described by Church and ReVelle [8]. For $i = 1, \dots, m$ and $j = 1, \dots, n$, let x_j and

y_i be $(0,1)$ variables such that

$$x_j = \begin{cases} 1 & \text{if } j \in J^* \\ 0 & \text{otherwise} \end{cases}$$

and

$$y_i = \begin{cases} 1 & \text{if } i \in I^* \\ 0 & \text{otherwise.} \end{cases}$$

Define

$$a_{ij} = \begin{cases} 1 & \text{if } i \in P_j \\ 0 & \text{otherwise.} \end{cases}$$

The following is an integer programming formulation for the maximum covering problem:

$$\max \sum_{i=1}^{m} w_i y_i$$

subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \geq y_i, \quad i = 1, \ldots, m,$$

$$\sum_{j=1}^{n} x_j = p,$$

$$x_j = (0,1), \quad j = 1, \ldots, n$$

$$y_i = (0,1), \quad i = 1, \ldots, m.$$

The solution to the linear programming relaxation of the above integer program produces as its optimal objective function value, an upper bound on the maximum coverage. We shall call this bound, the LP upper bound, denoted by

$$\mathrm{UB} = \max\{w^\top y \mid Ax \geq y,\ e^\top x = p,\ 0 \leq x \leq 1,\ 0 \leq y \leq 1\},$$

where $w = (w_1, w_2, \ldots, w_m)$, $y = (y_1, y_2, \ldots, y_m)$, $A = [a_{\cdot 1}, a_{\cdot 2}, \ldots, a_{\cdot n}]$, $x = (x_1, x_2, \ldots, x_n)$, and $e = (1, 1, \ldots, 1)$ of dimension n.

In this subsection, we describe a greedy randomized adaptive search procedure (GRASP) for PoP placement that finds approximate, i.e. good though not necessarily optimum, placement configurations. GRASP [14] is a metaheuristic that has been applied to a wide range of combinatorial optimization problems, including set covering [15], maximum satisfiability [21], and p-hub location [19], all three of which have some similarities with the PoP placement problem. GRASP is an iterative process, with a feasible solution constructed at each independent GRASP iteration. Each GRASP iteration consists of two phases, a construction phase and a local search phase. The best overall solution is kept as the result.

In the construction phase, a feasible solution is iteratively constructed, one element at a time. At each construction iteration, the choice of the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes

```
procedure grasp($\alpha$,MaxIter,RandomSeed)
1    BestSolutionFound = $\emptyset$;
2    do $k = 1, \ldots,$ MaxIter →
3        ConstructGreedyRandomizedSoln($\alpha$,RandomSeed,$p$,$J^*$);
4        LocalSearch($J^*$);
5        if $w(J^*) > w$(BestSolutionFound) → BestSolutionFound = $J^*$;
6    od;
7    return(BestSolutionFound)
end grasp;
```

FIGURE 1. A generic GRASP pseudo-code

brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. This choice technique allows for different solutions to be obtained at each GRASP iteration, but does not necessarily compromise the power of the adaptive greedy component of the method.

As is the case for many deterministic methods, the solutions generated by a GRASP construction are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is usually beneficial to apply a local search to attempt to improve each constructed solution. While such local optimization procedures can require exponential time from an arbitrary starting point, empirically their efficiency significantly improves as the initial solutions improve. Through the use of customized data structures and careful implementation, an efficient construction phase can be created which produces good initial solutions for efficient local search. The result is that often many GRASP solutions are generated in the same amount of time required for the local optimization procedure to converge from a single random start. Furthermore, the best of these GRASP solutions is generally significantly better than the solution obtained from a random starting point.

An especially appealing characteristic of GRASP is the ease with which it can be implemented. Few parameters need to be set and tuned (candidate list size and number of GRASP iterations) and therefore development can focus on implementing efficient data structures to assure quick GRASP iterations. Finally, GRASP can be trivially implemented on a parallel processor in an MIMD environment. For example, each processor can be initialized with its own copy of the procedure, the instance data, and an independent random number sequence. The GRASP iterations are then performed in parallel with only a single global variable required to store the best solution found over all processors.

The TM is organized as follows. In Subection 2.1, we describe the GRASP. In Subsection 2.2, we show how the GRASP solution is better than the pure random or pure greedy alternatives. On a large instance arising from a real-world application, we show how the GRASP solution is near optimal. Parallelization of GRASP is also illustrated.

2.1. **GRASP for PoP placement.** As outlined in Section 2, a GRASP possesses four basic components: a greedy function, an adaptive search strategy, a probabilistic selection procedure, and a local search technique. These components are

```
procedure ConstructGreedyRandomizedSoln($\alpha$,RandomSeed,$p$,$J^*$)
1    $J^* = \emptyset$;
2    do $k = 1, \dots, p$ $\rightarrow$
3        RCL = MakeRCL($\alpha, J, J^*, \gamma$);
4        $s$ = SelectPoP(RCL,RandomSeed,$J^*$);
5        $J^* = J^* \cup \{s\}$;
6        AdaptGreedyFunction($s, J, J^*, \Gamma, \Gamma^{-1}, \gamma$);
7    od;
end ConstructGreedyRandomizedSoln;
```

FIGURE 2. GRASP construction phase pseudo-code

interlinked, forming an iterative method that, at each iteration, constructs a feasible solution, one element at a time, guided by an adaptive greedy function, and then searches the neighborhood of the constructed solution for a locally optimal solution. Figure 1 shows a GRASP in pseudo-code. The best solution found so far (`BestSolutionFound`) is initialized in line 1. The GRASP iterations are carried out in lines 2 through 6. Each GRASP iteration has a construction phase (line 3) and a local search phase (line 4). If necessary, the solution is updated in line 5. The GRASP returns the best solution found.

In the remainder of this subsection, we describe in detail the ingredients of the GRASP for the PoP placement problem, i.e. the GRASP construction and local search phases. To describe the construction phase, one needs to provide a candidate definition (for the restricted candidate list) and an adaptive greedy function, and specify the candidate restriction mechanism. For the local search phase, one must define the neighborhood and specify a local search algorithm.

2.1.1. *Construction phase.* The construction phase of a GRASP builds a solution, around whose neighborhood a local search is carried out in the local phase, producing a locally optimal solution. This construction phase solution is built, one element at a time, guided by a greedy function and randomization. Figure 2 describes in pseudo-code a GRASP construction phase. Since in the PoP placement problem there are p PoP locations to be chosen, each construction phase consists of p iterations, with one location chosen per iteration. In `MakeRCL` the restricted candidate list of PoP locations is set up. The index of the next PoP location to be chosen is determined in `SelectPoP`. The PoP location selected is added to the set J^* of chosen PoP locations in line 5 of the pseudo-code. In `AdaptGreedyFunction` the greedy function that guides the construction phase is changed to reflect the choice just made. As before, let $J = \{1, 2, \dots, n\}$ be set of indices of the sets of potential PoP locations. Solutions are constructed by selecting one PoP location at a time to be in the set J^* of chosen PoP locations. To define a restricted candidate list, we must rank the yet unchosen PoP locations according to an adaptive greedy function.

The greedy function used in this algorithm is the total weight of yet-uncovered exchanges that become covered after the selection in each construction phase iteration. Let J^* denote the set (initially empty) of chosen PoP locations being built in the construction phase. At any construction phase iteration, let Γ_j be the set of additional uncovered exchanges that would become covered if PoP location j (for

$j \in J \setminus J^*$) were to be added to J^*. Define the *greedy function*

$$\gamma_j = \sum_{i \in \Gamma_j} w_i$$

to be the incremental weight covered by the choice of PoP location $j \in J \setminus J^*$. The greedy choice is to select the PoP location k having the largest γ_k value. Note that with every selection made, the sets Γ_j, for all yet unchosen PoP location indices $j \in J \setminus J^*$, change to reflect the new selection. This consequently changes the values of the greedy function γ_j, characterizing the adaptive component of the heuristic.

We describe next the restriction mechanism for the restricted candidate list (RCL) used in this GRASP. The RCL is set up in `MakeRCL` of the pseudo-code of Figure 3. A value restriction mechanism is used. Value restriction imposes a parameter based *achievement level*, that a candidate has to satisfy to be included in the RCL. Let

$$\gamma^* = \max\{\gamma_j \mid \text{PoP location } j \text{ is yet unselected, i.e. } j \in J \setminus J^*\}$$

and α be the restricted candidate parameter ($0 \leq \alpha \leq 1$). We say a PoP location j is a *potential candidate*, and is added to the RCL, if $\gamma_j \geq \alpha \times \gamma^*$. `MakeRCL` returns the set `RCL` with the indices of all potential PoP locations that have greedy function values within $\alpha \times 100\%$ of the value of the greedy choice. Note that by varying the parameter α the heuristic can be made to construct a set of p random PoP locations ($\alpha = 0$) or act as a greedy algorithm ($\alpha = 1$).

Once the RCL is set up, a candidate from the list must be selected and made part of the solution being constructed. `SelectPoP` selects, at random, the PoP location index s from the RCL. In line 5 of `ConstructGreedyRandomizedSoln`, the choice made in `SelectPoP` is added to the set of PoP locations J^*.

The greedy function γ_j is changed in `AdaptGreedyFunction` to reflect the choice made in `SelectPoP`. This requires that some of the sets Γ_j as well as the values γ_j be updated. Let Γ_i^{-1} denote the set of PoP locations to which a caller in exchange i can make a local call to. Let s be the newly added PoP location. The potential PoP locations j whose elements Γ_j need to be updated are those not yet in the PoP location set J^* for which exchanges in P_s are covered by PoP location j.

2.1.2. *Local search phase.* Given a solution neighborhood structure $N(\cdot)$ and a weight function $w(\cdot)$, a local search algorithm takes an initial solution J^0 and seeks a locally optimal solution with respect to $N(\cdot)$. For a maximization problem, such as the PoP placement problem, a local optimum is a solution J^* having weight $w(J^*)$ greater than or equal to the weight $w(J^+)$ for any $J^+ \in N(J^*)$. The local search algorithm examines a sequence of solutions $J^0, J^1, \ldots, J^k = J^*$, where $J^{i+1} \in N(J^i)$, i.e. immediately after examining solution J^i, it can only examine a solution J^{i+1} that is a neighbor of J^i. Figure 5 illustrates a generic local search algorithm that finds a local maximum of the function $w(\cdot)$. If in line 2 there exists a solution J^+ in the neighborhood of the current solution J^* with a weight greater than that of the current solution, then in line 3 the improved solution is made the current solution. The loop from line 2 to 4 is repeated until no local improvement is possible.

A combinatorial optimization problem can have many different neighborhood structures. For the PoP placement problem, a simple structure is 2-exchange. Two solutions (sets of PoP locations) J^1 and J^2 are said to be neighbors in the 2-exchange neighborhood if they differ by exactly one element, i.e. $\mid J^1 \cap \Delta J \mid \; =$

```
procedure MakeRCL($\alpha, J, J^*, \gamma$)
    RCL $= \emptyset$;
    $\gamma^* = \max\{\gamma_j \mid j \in J \setminus J^*\}$;
    do $s \in J \setminus J^* \rightarrow$
        if $\gamma_s \geq \alpha \times \gamma^* \rightarrow$
            RCL $=$ RCL $\cup \{s\}$;
        fi;
    od;
    return(RCL);
end MakeRCL;
```

FIGURE 3. MakeRCL pseudo-code

```
procedure AdaptGreedyFunction($s, J, J^*, \Gamma, \Gamma^{-1}, \gamma$)
    do $i \in \Gamma_s \rightarrow$
        do $j \in \Gamma_i^{-1} \cap \{J \setminus J^*\}$ $(j \neq i) \rightarrow$
            $\Gamma_j = \Gamma_j - \{i\}$;
            $\gamma_j = \gamma_j - w_i$;
        od;
    od;
end AdaptGreedyFunction;
```

FIGURE 4. AdaptGreedyFunction pseudo-code

```
procedure LocalSearch($J^0, N(\cdot), w(\cdot), J^*$)
    $J^* = J^0$;
    do $\exists\ J^+ \in N(J^*) \ni w(J^+) > w(J^*) \rightarrow$
        $J^* = J^+$;
    od;
end LocalSearch;
```

FIGURE 5. A generic local search algorithm

```
procedure LocalSearch($J^*$)
    do local maximum not found $\rightarrow$
        do $s \in J^* \rightarrow$
            do $t \in J \setminus J^* \rightarrow$
                if WeightGain($J^*, t$) $>$ WeightLoss($J^*, s$) $\rightarrow$
                    $J^* = J^* \cup \{t\} \setminus \{s\}$;
                fi;
            od;
        od;
    od;
end LocalSearch;
```

FIGURE 6. The local search procedure in pseudo-code

$| J^2 \cap \Delta J | = 1$, where $\Delta J = (J^1 \cup J^2) \setminus (J^1 \cap J^2)$. The local search starts with a set J^* of p PoP locations, and at each iteration attempts to find a pair of locations $s \in J^*$ and $t \in J \setminus J^*$ such that $w(J^* \setminus \{s\} \cup \{t\}) > w(J^*)$. If such a pair exists, then location s is replaced by location t in J^*. A solution is locally optimal with respect to this neighborhood if there exists no pairwise exchange that increases the total weight of J^*. This local search algorithm is described in the pseudo-code in Figure 6. Though it is not the objective of this TM to delve into implementation details, it is interesting to observe that the total weight of the neighborhood solutions need not be computed from scratch, Rather, in line 4 of the pseudo-code, procedures `WeightGain` and `WeightLoss` compute, respectively, the weight gained by J^* with the inclusion of PoP location j and the weight loss by J^* with the removal of PoP location i from J^*. The weight gained can be computed by adding the weights of all exchanges not covered by any PoP location in J^* that is covered by j, while the weight loss can be computed by adding up the weights of the exchanges covered by PoP location i and no other PoP location in J^*.

The GRASP construction phase described in Subsection 2.1.1 computes a feasible set of chosen PoP locations that is not necessarily locally optimal with respect the 2-exchange neighborhood structure. Consequently, local search can be applied with the objective of finding a locally optimal solution that may be better than the constructed solution. In fact, the main purpose of the construction phase is to produce a good initial solution for the local search. It is empirically known that simple local search techniques perform better if they start with a good initial solution. This will be illustrated in the computational results subsection, where experiments indicate that local search applied to a solution generated by the construction phase, rather than random generation, produces better overall solutions, and GRASP converges faster to an approximate solution.

2.2. **Computing PoP placements with GRASP.** In this subsection, we illustrate the use of GRASP on a large PoP placement problem. We consider a problem with $m = 18,419$ calling areas and $n = 27,521$ potential PoP location. The sum of the number of lines over the calling areas is 27,197,601. We compare an implementation of the GRASP described in Subsection 2.1 with implementations of an algorithm having a purely greedy construction phase and one having purely random construction. All three algorithms use the same local search procedure, described in Subsection 2.1.2. Furthermore, since pure greedy and pure random are special cases of GRASP construction, all three algorithms are implemented using the same code, simply by setting the RCL parameter value α to appropriate values. For GRASP, $\alpha = 0.85$, while for the purely greedy algorithm, $\alpha = 1$, and for the purely random algorithm, $\alpha = 0$. All runs were carried out on a Silicon Graphics Challenge computer (196MHz IPS R10000 processor). The GRASP code is written in Fortran and was compiled with the SGI Fortran compiler `f77` using compiler flags `-O3 -r4 -64`.

Two experiments are done. In the first, the number of PoPs to be place is fixed at $p = 146$ and the three implementations are compared. Each code is run on 10 processors, each using a different random number generator seed for 500 iterations of the build–local search cycle, thus each totaling 5000 iterations. Because of the long processing times associated with the random algorithm, the random algorithm processes were interrupted before completing the full 500 iterations on each processor. They did 422, 419, 418, 420, 415, 420, 420, 412, 411, and 410

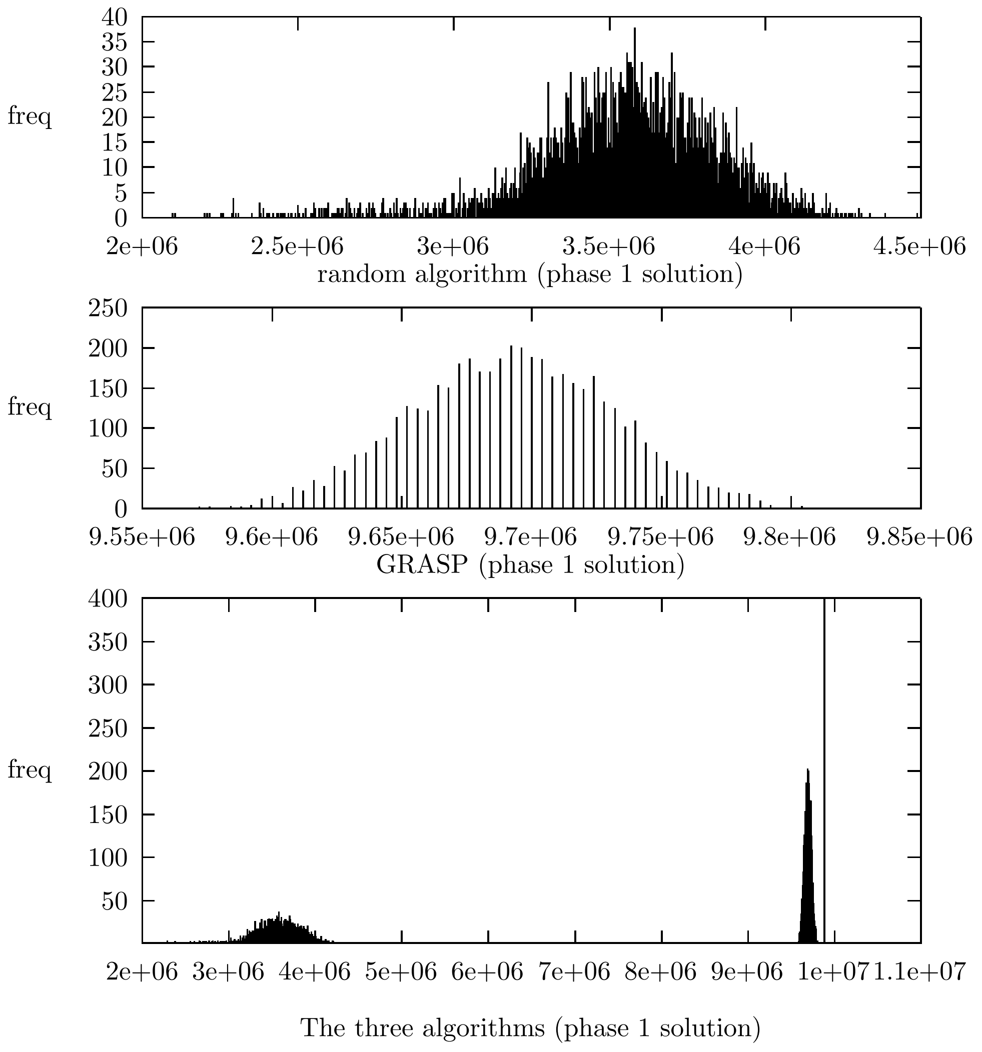

FIGURE 7. Phase 1 solution distribution for random algorithm (RCL parameter $\alpha = 0$), GRASP ($\alpha = 0.85$), and greedy algorithm ($\alpha = 1$)

iterations on each corresponding processor, totaling 4167 iterations. In the second experiment, GRASP was run 300 times on PoP placement problems defined by varying p, the number of PoPs, from 1 to 300 in increments of 1. Instead of running the algorithm for a fixed number of iterations, the LP upper bound was computed for each instance and the GRASP was run until it found a PoP placement within one percent of the LP upper bound.

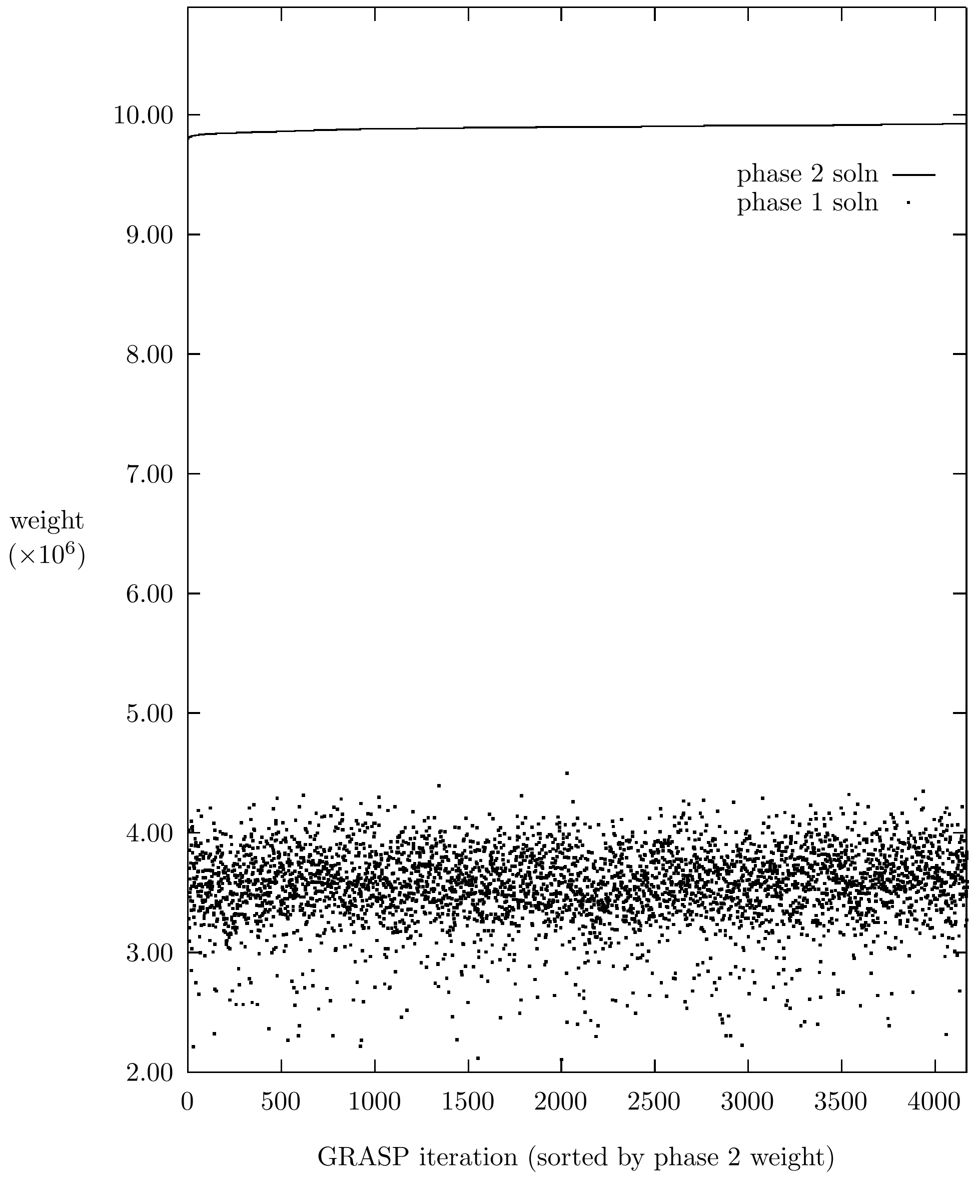

FIGURE 8. GRASP phase 1 and phase 2 solutions, sorted by phase 2, then phase 1 solutions. RCL parameter $\alpha = 0.0$ (purely random construction)

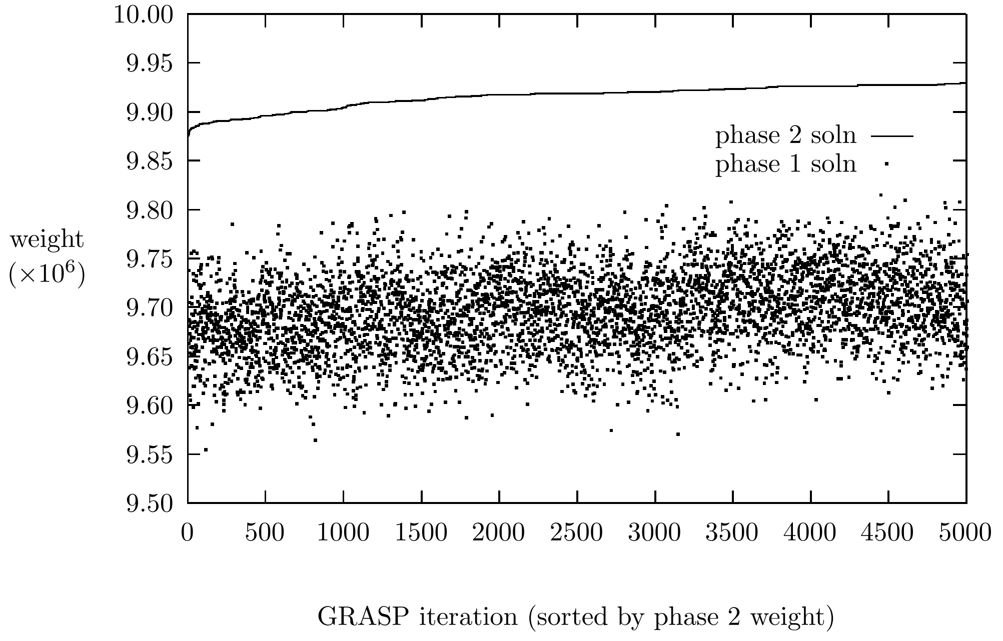

FIGURE 9. GRASP phase 1 and phase 2 solutions, sorted by phase 2, then phase 1 solutions. RCL parameter $\alpha = 0.85$

Figure 7 illustrates the relative behavior of the three algorithms. The top and middle plots in Figure 7 show the frequency of the solution values generated by the purely random construction and GRASP construction respectively. The plot on the bottom of Figure 7 compares the constructed solutions of the three algorithms. As can be observed, the purely greedy algorithm constructs the best quality solution, followed by the GRASP, and then by the purely random algorithm. On the other hand, the purely random algorithm produces the largest amount of variance in the constructed solutions, followed by the GRASP and then the purely greedy algorithm, which generated the same solution on all 5000 repetitions. High quality solutions as well as large variances are desirable characteristics of constructed solutions. Of the three algorithms, GRASP captures these two characteristics in its phase 1 solutions. As we will see next, the tradeoff between solution quality and variance plays an important role in designing a GRASP.

The solutions generated by the purely random algorithm and the GRASP are shown in Figures 8 and 9, respectively. The solution values on these plots are sorted according to local search phase solution value. As one can see, the differences between the values of the construction phase solutions and the local search phase solutions are much smaller for the GRASP than for the purely random algorithm. This suggests that the purely random algorithm requires greater effort in the local search phase than does GRASP. This indeed is observed and will be shown next. Figures 10 and 11 illustrate how the three algorithms compare in terms of best solution found so far, as a function of algorithms iteration and running time. Figure 10 shows local search phase solution for each algorithm, sorted by increasing value for each algorithm. The solution produced by applying local search to the solution constructed with the purely greedy algorithm is constant. Its value is only

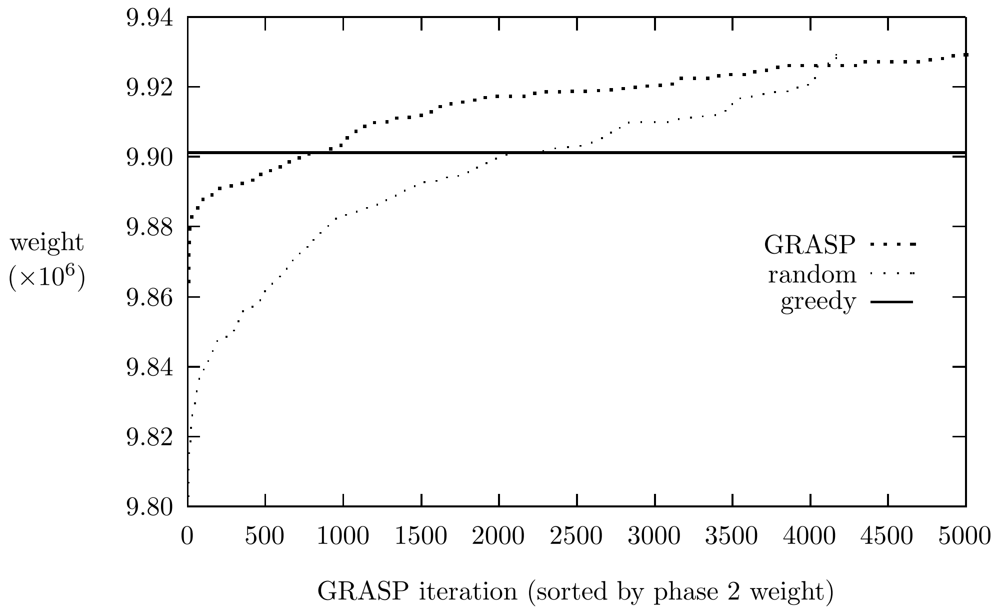

FIGURE 10. Phase 2 solutions, sorted by phase 2 for random, GRASP, and greedy algorithms

better than the worst 849 GRASP solutions and the worst 2086 purely random solutions. This figure illustrates well the effect of the tradeoff between greediness and randomness in terms of solution quality as a function of the number of iterations that the algorithm is repeated.

Figures 12 and 13 correspond to the second experiment, where GRASP was run until a solution within one percent of the LP upper bound was produced. For all 300 problems, the GRASP produced a design within 1% of the LP upper bound. Figure 12 shows the error of the GRASP solution as a percentage off of the LP upper bound when the algorithm was terminated. As can be observed, GRASP found tight solutions (GRASP solution equal to LP upper bound) for several instances and almost always produced a solution less than .5% off of the LP upper bound the first time it found a solution less than 1% of the upper bound. Figure 13 shows CPU times for each of the runs. CPU time grows with the complexity of the problem, as measured by the number of PoPs in the design. For up to about 50 PoPs (a number of PoPs found in practical incremental designs) the GRASP solution takes less than 30 seconds on a 196MHz Silicon Graphics Challenge. The longest runs took a little less than 3 minutes to conclude.

3. Rotuing permanent virtual circuits for frame relay service

A Frame Relay (FR) service offers virtual private networks to customers by provisioning a set of permanent (long-term) virtual circuits (PVCs) between customer endpoints on a large backbone network. During the provisioning of a PVC, routing decisions are made either automatically by the FR switch or by the network designer, through the use of preferred routing assignments, without any knowledge of future requests. Over time, these decisions usually cause inefficiencies in the network and occasional rerouting of the PVCs is needed. The new PVC routing

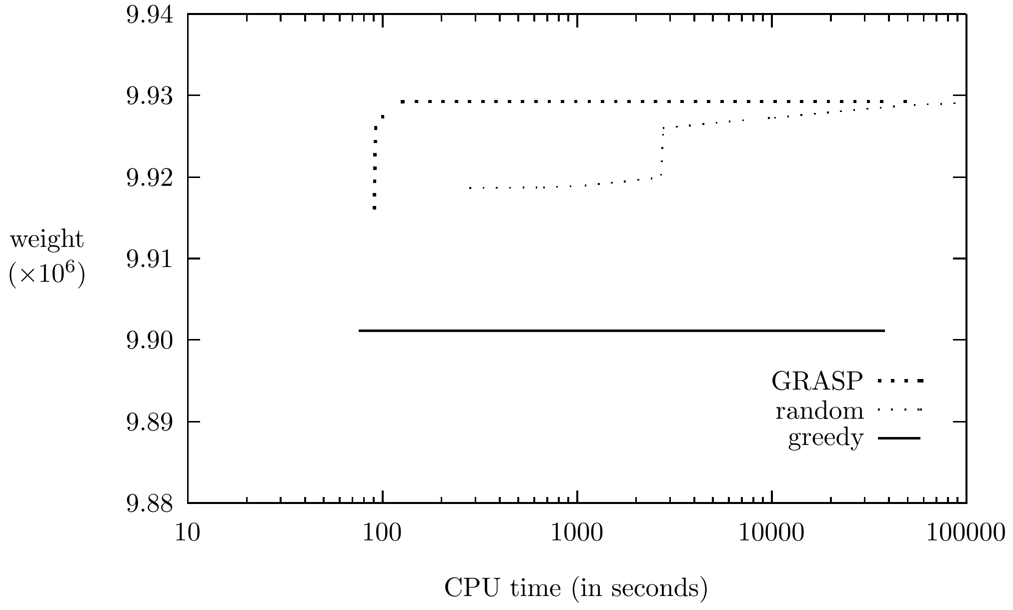

FIGURE 11. Incumbent phase 2 solution of random algorithm ($\alpha = 0$), GRASP ($\alpha = 0.85$), and greedy algorithm ($\alpha = 1$) as a function of CPU time (in seconds), running 10 processes in parallel.

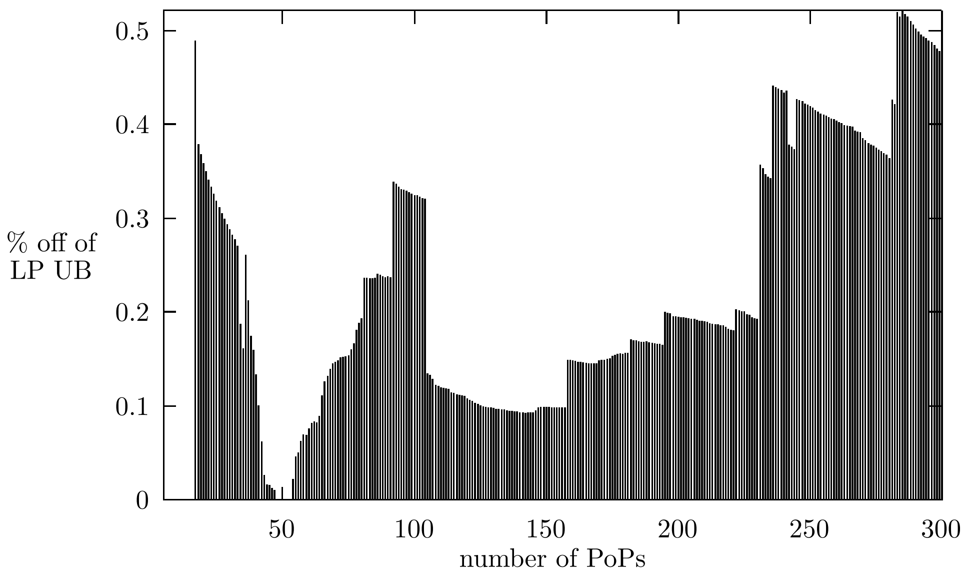

FIGURE 12. Percentage off of LP upper bound when stopping with a solution at most 1% off of bound

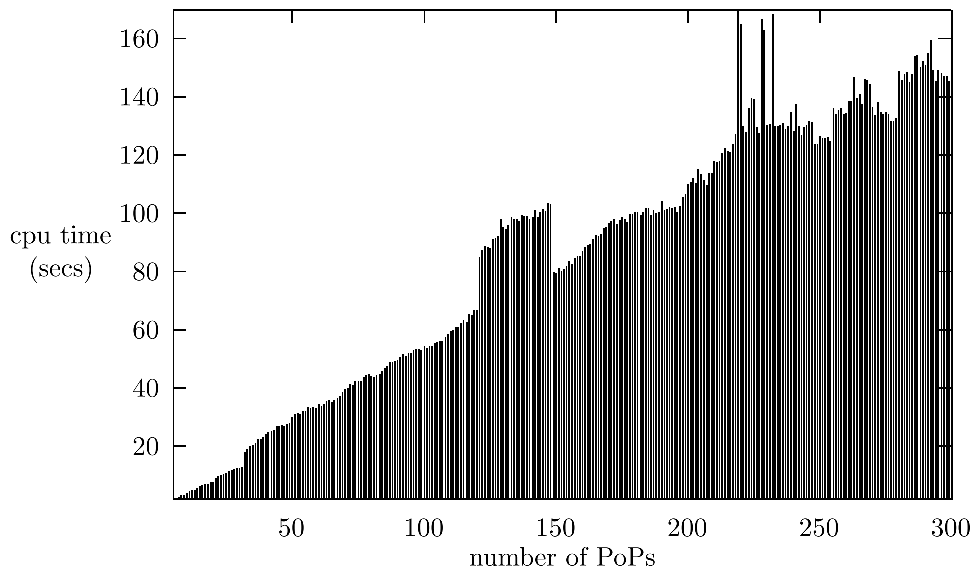

FIGURE 13. CPU time to stop when stopping with a solution at most 1% off of bound

scheme is then implemented on the network through preferred routing assignments. Given a preferred routing assignment, the FR switch will move the PVC from its current route to the new preferred route as soon as that move becomes feasible.

One way to create the preferred routing assignments is to appropriately order the set of PVCs currently in the network and apply an algorithm that mimics the routing algorithm used by the FR switch to each PVC in that order. However, more elaborate routing algorithms, that take into consideration factors not considered by the FR switch, could further improve the efficiency of network resource utilization.

Typically, the routing scheme used by the FR switch to automatically provision PVCs is also used to reroute PVCs in the case of trunk or card failures. Therefore, this routing algorithm should be efficient in terms of running time, a requirement that can be traded off for improved network resource utilization when building preferred routing assignments off-line.

This section describes a greedy randomized adaptive search procedure (GRASP) for the problem of routing off-line a set of PVC demands over a backbone network such that PVC delays are minimized and network load is balanced. We refer to this as the *PVC preferred routing problem*. The set of PVCs to be routed can include all or a subset of the PVCs currently in the network, and/or a set of forecast PVCs. The explicit handling of delays as opposed to just number of hops (as in the routing algorithm implemented in Cisco FR switches) is particularly important in international networks, where distances between backbone nodes vary considerably. Network load balancing is important for providing the maximum flexibility to handle the following three situations:

- Overbooking, which is typically used by network designers to account for non-coincidence of traffic;
- PVC rerouting due to a link or card failure;

- Bursting above the committed rate, which is not only allowed but sold to customers as one of the attractive features of FR.

The Technical Memorandum is organized as follows. In Section 3.1, we formulate the PVC preferred routing problem. In Section 3.2, the GRASP for PVC routing is described.

3.1. **Problem Formulation.** Let $G = (V, E)$ represent the FR network where routing takes place. Let V denote the set of n backbone nodes, where FR switches reside, and let E denote the set of m trunks that connect the backbone nodes. Parallel trunks are allowed. For each trunk $e \in E$, let c_e denote the trunk bandwidth (the maximum kbits/sec rate) allowed to be routed on the trunk, t_e denote the maximum number of PVCs that can be routed on the trunk, and d_e denote the delay associated with the trunk. The bound on the number of PVCs allowed on a trunk depends on the port card used to implement the trunk. The delay represents propagation delay, as well as the delay associated with hopping. The set $\mathcal{P}$ of PVCs to be routed is represented by a list of origin-destination (O-D) pairs,

$$\mathcal{P} = \{(o_1, d_1), (o_2, d_2), \ldots, (o_{|\mathcal{P}|}, d_{|\mathcal{P}|})\},$$

where we associate with each pair a bandwidth requirement, known as effective bandwidth, which takes into account the actual bandwidth required by the customer in the forward and reverse directions, as well as an overbooking factor. Let b_p denote the effective bandwidth of PVC p.

The ultimate objective is to minimize PVC delays while balancing trunk loads, subject to several technological constraints. Network load balancing is achieved by minimizing the load on the most utilized trunk. Routing assignments with minimum PVC delays may not achieve the best trunk load balance. Likewise, routing assignments having the best trunk load balance may not minimize PVC delays. A compromising objective is to route all PVCs in set $\mathcal{P}$ such that a desired point in the tradeoff curve between PVC delays and trunk load balancing is achieved.

A route for PVC (o, d) is a sequence of adjacent trunks, where the first trunk originates in node o and the last trunk terminates in node d. A set of routing assignments is feasible, if for all trunks $e \in E$, the total PVC effective bandwidth requirements routed on e does not exceed c_e and the number of PVCs routed on trunk e is not greater than t_e.

3.2. **A GRASP for PVC Routing.** In this section, we propose GRASP construction and local search procedures for the PVC Preferred Routing Problem, and comment on implementation issues.

3.2.1. *Construction Procedure.* To describe a GRASP construction procedure, we present greedy functions and RCL construction mechanisms. As stated in Section 3.1, the objective of the optimization is to minimize PVC delays while balancing trunk loads.

Let $\mathcal{P}_1, \ldots, \mathcal{P}_m$ be the sets of PVCs routed on trunks $1, \ldots, m$, respectively. The delay component $D(\mathcal{P}_1, \ldots, \mathcal{P}_m)$ of the objective function is defined to be the sum of delays over all trunks, i.e.

$$D(\mathcal{P}_1, \ldots, \mathcal{P}_m) = \sum_{e \in E} z(e),$$

where $z(e)$ is a trunk delay function that needs to be appropriately defined. Let d_e be the delay associated with trunk e. Two plausible delay functions are:

```
procedure ConstructGRSolution($\mathcal{P}, \mathcal{R}_1, \ldots, \mathcal{R}_{|\mathcal{P}|}$){
    $\tilde{\mathcal{P}} = \mathcal{P}$;
    for $e \in E$ {
        $\mathcal{P}_e = \emptyset$;
    }
    while $\tilde{\mathcal{P}} \neq \emptyset$ {
        for $e \in E$ {
            Compute edge length $\mathcal{L}_e^{\mathcal{P} \setminus \tilde{\mathcal{P}}}$;
        }
        for $p \in \tilde{\mathcal{P}}$ {
            $\mathcal{R}_p = \mathtt{sp}(o_p, d_p, \mathcal{L}_1^{\mathcal{P} \setminus \tilde{\mathcal{P}}}, \ldots, \mathcal{L}_m^{\mathcal{P} \setminus \tilde{\mathcal{P}}})$;
            for $e \in E$ {
                if $e \in \mathcal{R}_p$ {
                    $\tilde{\mathcal{P}}_e = \mathcal{P}_e \cup \{p\}$;
                }
                else {
                    $\tilde{\mathcal{P}}_e = \mathcal{P}_e$;
                }
            }
            $g(p) = \mathcal{G}(\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_m)$;
        }
        $\underline{g} = \min\{g(p) \mid p \in \tilde{\mathcal{P}}\}$;
        $\overline{g} = \max\{g(p) \mid p \in \tilde{\mathcal{P}}\}$;
        RCL = $\{p \in \tilde{\mathcal{P}} \mid \underline{g} \leq g(p) \leq \underline{g} + \alpha(\overline{g} - \underline{g})\}$;
        Pick $p^*$ at random from the RCL;
        for $e \in \mathcal{R}_{p^*}$ {
            $\mathcal{P}_e = \mathcal{P}_e \cup \{p^*\}$;
        }
        $\tilde{\mathcal{P}} = \tilde{\mathcal{P}} \setminus \{p^*\}$;
    }
}
```

FIGURE 14. GRASP construction phase pseudo code

- $z(e) = n_e d_e$, where n_e is the number of PVCs routed on trunk e;
- $z(e) = d_e \sum_{p \in \mathcal{P}_e} b_p$, where b_p is the effective bandwidth of PVC p.

The former delay is an unweighted delay, while the latter is weighted by the amount of bandwidth used in the trunk.

The trunk load balance component of the objective function is

$$B(\mathcal{P}_1, \ldots, \mathcal{P}_m) = \max_{e \in E}\{c_e - \sum_{p \in \mathcal{P}_e} b_p\}.$$

Given subsets of PVCs $\mathcal{P}_1, \ldots, \mathcal{P}_m$, the objective function is defined to be a convex combination of the two above components, i.e.

$$\mathcal{G}(\mathcal{P}_1, \ldots, \mathcal{P}_m) = \delta D(\mathcal{P}_1, \ldots, \mathcal{P}_m) + (1 - \delta) B(\mathcal{P}_1, \ldots, \mathcal{P}_m),$$

where δ $(0 \leq \delta \leq 1)$ can be set to indicate preference for delay minimization (δ close to 1) or load balancing (δ close to 0).

We next describe the GRASP construction phase which is presented in pseudo code in Figure 14. The idea in this construction procedure is to build the PVC routing solution by routing one PVC at a time until all PVCs ($p \in \mathcal{P}$) have been routed. At anytime, the greedy choice selects from the set $\tilde{\mathcal{P}}$ of PVCs not yet routed, the one that, when routed, minimizes the delay while balancing the trunk loads. To implement such a greedy selection procedure, circuits are always routed from their origination node to their destination node on a shortest path route. The length $\mathcal{L}_e^{\mathcal{P}\setminus\tilde{\mathcal{P}}}$ of trunk e in the shortest path computation can be for example,

$$\mathcal{L}_e^{\mathcal{P}\setminus\tilde{\mathcal{P}}} = \left(\frac{c_e}{c_e - x_e}\right)^{\beta} d_e^{1-\beta} \text{ or } \mathcal{L}_e^{\mathcal{P}\setminus\tilde{\mathcal{P}}} = \beta\frac{c_e}{c_e - x_e} + (1-\beta)d_e,$$

where c_e is the bandwidth of trunk e, d_e is the delay associated with trunk e, β ($0 \leq \beta \leq 1$) is a parameter that determines whether delay or load balancing is emphasized in the routing, and

$$x_e = \sum_{p \in \mathcal{P}_e} b_p$$

is the effective bandwidth of the traffic routed on trunk e, where here $\mathcal{P}_e$ is the set of PVCs routed so far on trunk e.

The route chosen for a PVC depends on the routing decisions previously made, since $\mathcal{L}_e^{\mathcal{P}\setminus\tilde{\mathcal{P}}}$ depends on which PVCs were previously routed and how they were routed. Since PVCs cannot be routed on trunks that cannot accommodate the PVC's bandwidth requirement or that have reached their maximum number of PVCs, the shortest path computation disregards any such trunk.

The GRASP construction procedure takes as input the set $\mathcal{P}$ of PVCs to be routed and returns the p routes $\mathcal{R}_1, \ldots, \mathcal{R}_{|\mathcal{P}|}$. The set $\tilde{\mathcal{P}}$ of PVCs to be processed is initialized with $\mathcal{P}$ in line 1 and the sets of PVCs routed so far on each trunk are initialized empty in lines 2-4. The construction loop (lines 5–29) is repeated until all PVCs have been routed. To route the next PVC, as well as to compute the incremental cost associated with each PVC routing, in lines 6–8 the trunk cost functions $\mathcal{L}_e^{\mathcal{P}\setminus\tilde{\mathcal{P}}}$ ($e = 1, \ldots, m$) are computed. The incremental cost $g(p)$ incurred by routing each PVC $p \in \tilde{\mathcal{P}}$ on its corresponding shortest path from o_p to d_p, is computed in lines 9–20. The smallest and largest incremental costs are computed in lines 21 and 22, respectively, and the restricted candidate list is set up in line 23. In line 24, a PVC p^* is selected, at random, from the RCL. In lines 25–28, the sets of PVCs routed in each trunk are updated, as well as is the set of PVCs yet to be routed.

3.3. **Local Search Procedure.**

Once all PVCs have been routed in the construction phase of GRASP, an improvement is attempted in the local search phase. The local search proposed here reroutes each PVC, one at a time, checking each time if the new route taken together with the other $|\mathcal{P}| - 1$ fixed routes improves the objective function. If it does, the new route is kept and the local search procedure is called recursively. If no individual rerouting improves the total cost, the local search procedure terminates with a locally optimal routing scheme.

Figure 15 describes the local search procedure in pseudo-code. The procedure takes the current solution $(\mathcal{P}, \mathcal{R}_1, \ldots, \mathcal{R}_{|\mathcal{P}|})$ as input. The sets of PVCs routed on each trunk are computed in lines 1–8 and the cost of the current solution is computed in line 9. The tentative rerouting of the PVCs takes place in the loop in lines 9–33. For each PVC p, the search procedure tentatively reroutes p (lines 11–26) and computes the cost of the new rerouting scheme (line 27). To reroute PVC p, in lines 11–14 the edge lengths are computed and the rerouting of the PVC through the shortest path computation takes place in line 15. To compute the cost of the tentative routing scheme, the sets $\hat{\mathcal{P}}_1, \hat{\mathcal{P}}_2, \dots, \hat{\mathcal{P}}_m$, of PVCs riding on the different trunks need to be revised (lines 16–26).

If the rerouting scheme is better than the original routing, the rerouting is made the current solution and the local search procedure is called recursively (lines 28–32).

4. SONET ring network design

Survivable telecommunications networks with fast service restauration capability are increasingly in demand by customers whose businesses depend heavily on continuous reliable communications. Businesses such as brokerage houses, banks, reservation systems, and credit card companies are willing to pay higher rates for guaranteed service availability. The introduction of the Synchronous Optical Network (SONET) standard has enabled the deployment of networks having a high level of service availability. SONET is generally configured as a network of self-healing rings, offering at least two paths between each pair of demand points. SONET compatible equipment is capable of detecting problems with the signal and quickly react to reestablish communications, often in milliseconds. Arslan, Loose, and Strand [2] identify unit cost reduction, increased reliability, higher bandwidth, SONET/SDH hands-off, and SONET services as the needs of AT&T business units that can be impacted by SONET ring deployment.

The design of cost-effective SONET ring designs is a crucial step to enable AT&T to make good use of SONET technology. In this section, we describe several linear programming based models for designing low-cost SONET ring networks.

The section is organized as follows. In Subsection 4.1, we define the network design problem. An integer programming model of a basic network design problem is described in Subsection 4.2. In Subsection 4.3, we present a heuristic approach used to find approximate solutions of the integer programming model. The lower bound produced by solving the linear programming relaxation of the integer program provides an indication of the quality of the approximate solution found. A small network design problem is used to illustrate the solution procedure in Subsection 4.4. In Subsection 4.5, extensions to the basic model, dealing with dual-ring interworking, are described.

4.1. **The SONET ring network design problem.** A *telecommunications network* can be viewed as a graph $G = (V, E)$ having a set V of vertices or nodes, each representing a large customer or a remote terminal (where low bandwidth traffic from a group of customers is aggregated) or a central office where switching takes place, and a set E of edges or links, each representing fiber cable connecting two nodes.

Demand between pairs of nodes (not necessarily all pairs of nodes have demand between themselves) in the network is an estimate of the number of circuits needed to provide communications between that pair of nodes. Demands are expressed in units of DS3 (51.84 Mbits/sec).

To ensure rapid restauration, SONET equipment is configured on logical rings. A ring is simply a cycle in the graph induced by a subset of vertices of V. The

SONET ring network is a set of rings that covers the nodes of G and that allows the demand to be satisfied. A demand between a pair of nodes is said to be satisfied if bandwidth equal to the number of required DS3s is reserved on one or more paths between the pair of nodes, where each path traverses only nodes with SONET equipment.

SONET equipment are associated with rings and interring nodes (nodes where traffic moves from one node to the next). Every node on a ring has one *add-drop multiplexer* (ADM) per OC48 (48 units of DS3). An ADM is capable of adding or dropping signals going through it without having to multiplex or demultiplex the entire digital hierarchy. Each link on a ring has two *dense wave division multiplexer* (DWDM) per each eight units of OC48 that traverse that link. Furthermore, every multiple of 75 miles on a link requires an *optical amplifier* (OA) per OC48 and every multiple of 225 miles on a link requires a signal regenerator (REGEN) per OC48. Finally, digital cross-connect systems (DCS) or low-speed terminators are used to add, drop, or interconnect signals between ADMs within the central offices. Each three units of DS3 at demand points and interring crossconnect points require one DS3 circuit pack.

In the network design, the allowable rings are usually restricted to be from a set of predetermined rings. We call this set the set of candidate rings. One ring generation scheme developed at AT&T is described by Rosenwein and Wong [22].

The *SONET ring design problem* we consider in this technical memorandum can be stated as follows:

> Given a network of nodes and links, a set of point-to-point demands, and a set of candidate rings covering the nodes, find a minimum cost SONET ring network using only rings from the candidate ring set such that the resulting equipment and fiber links have sufficient capacity to satisfy the demands. Technical constraints can further restrict the configuration of a network design.

4.2. **An integer programming formulation.** In this subsection, we describe an integer programming formulation for the first, and simplest, network design problem considered in this memorandum, the single node interworking ring network design problem. In this problem, demand is loaded and unloaded from nodes and demands flow on ring links within a ring and can crossconnect between two rings having a common node at any of their common nodes. Table 1 defines sets needed to describe the integer programming formulation.

The model described in this technical memorandum is based on multicommodity flows on a network. Point-to-point demands are commodities that flow on the network, sharing link and node resources. Ring size is a function of the maximum capacity over all link capacities on a ring. Costs are linear functions of ring, link, and node capacities. The objective is to move demand between demand pairs only on links that are part of at least one ring, satisfying flow conservation and demand requirements while minimizing the total cost. The model seeks the optimal values for flows, as well as ring and link capacities. The integer program uses several integer variables described in Table 2.

In the multicommodity flow model, a commodity could be defined as a unique point-to-point traffic. Point-to-point demand is given an arbitrary direction (one point is the source, the other is the sink) and we move demands from sources to sinks. This definition, however, can result in a large number of commodities if there

Table 1. Set definitions for integer programming model

$\mathcal{N}$	$\longleftarrow$	set of nodes in network
$\mathcal{L}$	$\longleftarrow$	set of undirected links (arcs) in network
$\mathcal{R}$	$\longleftarrow$	set of rings
$\mathcal{R}_n^{\mathcal{N}}$	$\longleftarrow$	set of rings containing node $n \in \mathcal{N}$
$\mathcal{R}_l^{\mathcal{L}}$	$\longleftarrow$	set of rings containing link $l \in \mathcal{L}$
$\mathcal{K}$	$\longleftarrow$	set of commodities
$\mathcal{N}_r^{\mathcal{R}}$	$\longleftarrow$	set of nodes on ring $r \in \mathcal{R}$
$\mathcal{A}_r^{\mathcal{R}}$	$\longleftarrow$	set of undirected links on ring $r \in \mathcal{R}$
$\vec{\mathcal{A}}_r^{\mathcal{R}}$	$\longleftarrow$	set of directed links on ring $r \in \mathcal{R}$

Table 2. Variable definitions for integer programming model. All variables are integer valued.

y_k^r	$\longleftarrow$	demand of commodity k unloaded at sink node k from ring r
$z_{i,k}^r$	$\longleftarrow$	demand of commodity k loaded on ring r at node i
$f_{i,j}^{r,k}$	$\longleftarrow$	flow of commodity k on ring r directed from node i to node j
$x_{n,k}^{r,s}$	$\longleftarrow$	crossover flow at node n of commodity k from ring r to ring s
u_r	$\longleftarrow$	size of ring r
w_l	$\longleftarrow$	size of link l

is a large number of pont-to-point demands, generating large hard to solve models. In our model, we use the concept of aggregate flow as commodities. A commodity k is defined to be the aggregate flow having node k as sink. Nodes are picked as sinks, one by one, and all unassigned demand terminating at that node is aggregated into a commodity. At this point, the flows of demand are given a direction, i.e. a source node and a sink node. Nodes are picked until no further point-to-point demand is unassigned to a commodity. In this fashion, $|\mathcal{K}| \leq |\mathcal{N}|$ commodities are produced.

The problem of generating the smallest number of commodities is a node covering problem on the graph $H = (V, D)$ where D is a set of edges such that $(i, j) \in D$ if and only if nodes i and j have positive point-to-point demand. Vertex covering is an NP-complete [17] problem, thus computing the guaranteed smallest number of commodities in reasonable time is probably only possible for small instances [5]. A quick way to get an approximate assignment of demands to commodities is to apply a greedy algorithm. A greedy algorithm for this problem selects as the next sink the node in H with the greatest degree. The graph H is redefined as the original H with the all sink nodes and all edges incident to them removed. The process is repeated until no node is available for picking. An easy way to improve upon greedy algorithms is by using GRASP (greedy randomized adaptive search procedures) [14]. If the instance is small enough, an exact algorithm for minimum set covering or maximum independent set or maximum clique could be used to find the best set of commodities [18].

The individual point-to-point flows corresponding to a given commodity can be recovered from the aggregate flows of that commodity by solving a series of maximum flow problems on auxiliary networks. Define the set of nodes of the auxiliary network to be all node-ring pairs n, r, such that node n is in ring r, i.e. $n \in \mathcal{N}_r^{\mathcal{R}}$. For each commodity, we define a different set of directed arcs in each auxiliary network. Consider the auxiliary network associated with commodity k. For all i, j, r such that $f_{i,j}^{r,k} > 0$, define a directed arc from node i on ring r (we call this node n-r) to node j on ring r (node j-r) with capacity $f_{i,j}^{r,k}$. Likewise, for all r_1, r_2, n such that $x_{n,k}^{r_1,r_2} > 0$, define a directed arc from node n-r_1 to node n-r_2 having capacity $x_{n,k}^{r_1,r_2}$. Define sink node t and uncapacitated arcs from all nodes k-r such that $r \in \mathcal{R}_k^{\mathcal{N}}$, to node t. Consider demand pair s, k. Define a source node s^* and an auxiliary node a_s and define an arc from s^* to a_s having capacity $d_{s,k} > 0$, the point-to-point demand between s and k. Finally define uncapacitated arcs from a_s to all s-r such that $r \in \mathcal{R}_s^{\mathcal{N}}$. The maximum flow from s^* to t is the point-to-point flow between demand points s and k. To proceed with another demand pair of the same commodity, subtract that maximum flow from all arc capacities of arcs used in that flow, and repeat.

As stated in the problem definition in Subsection 4.1, point-to-point demands are given between pairs of nodes in the network. We denote by D^k the demand sink node $n = k$ has for commodity k and by S_n^k the demand source node $n \neq k$ has for commodity k.

We next describe the set of constraints that defines the set of feasible solutions. At every sink node k (for which there corresponds a commodity k), the total demand for commodity k unloaded at that node, off of all rings, must equal the demand for commodity k at node k, i.e.

$$\sum_{r \in \mathcal{R}_k^{\mathcal{N}}} y_k^r = D^k, \ k \in \mathcal{K}.$$

Likewise, the amount of commodity k loaded at source node n, onto all rings, must equal the demand for commodity k at node n, i.e.

$$\sum_{r \in \mathcal{R}_n^{\mathcal{N}}} z_{n,k}^r = S_n^k, \ k \in \mathcal{K}, \ n \in \mathcal{N}.$$

Flow conservation of each commodity at each node is done at the ring level. For every ring $r \in \mathcal{R}$, ring node $n \in \mathcal{N}_r^{\mathcal{R}}$, and commodity $k \in \mathcal{K}$, the sum of flows of commodity k into node n carried on ring r together with the amount of commodity k loaded onto ring r at node n and the amount of commodity k crossconnected onto ring r at node n from all rings sharing node n must equal the sum of flows of commodity k out of node n carried on ring r together with the amount of commodity k loaded off of ring r at node n and the amount of commodity k crossconnected

Table 3. Cost coefficients for integer programming model.

$c_r^{\mathcal{R}}$	$\longleftarrow$	unit cost of ring r
$c_l^{\mathcal{L}}$	$\longleftarrow$	unit cost of link l
$c^{\mathcal{X}}$	$\longleftarrow$	unit crossconnect cost
$c^{\mathcal{I}}$	$\longleftarrow$	unit load / unload cost

from ring r at node n to all rings sharing node n, i.e. for $r \in \mathcal{R},\ n \in \mathcal{N}_r^{\mathcal{R}},\ k \in \mathcal{K}$,

$$\sum_{i \ni (i,n) \in \vec{\mathcal{A}}_r^{\mathcal{R}}} f_{i,n}^{r,k} + z_{n,k}^{r} + \sum_{s \in \mathcal{R}_n^{\mathcal{N}} \setminus \{r\}} x_{n,k}^{s,r} - \sum_{i \ni (n,i) \in \vec{\mathcal{A}}_r^{\mathcal{R}}} f_{n,i}^{r,k} - \sum_{s \in \mathcal{R}_n^{\mathcal{N}} \setminus \{r\}} x_{n,k}^{r,s} = \begin{cases} y_r^k & \text{if } n = k, \\ 0 & \text{if } n \neq k \end{cases}. \tag{1}$$

Ring size is determined by the maximum size of the links that make up the ring. For all rings $r \in \mathcal{R}$ and every undirected link $(i,j) \in \mathcal{A}_r$, the sum of flows of all commodities moving on ring r from node i to node j together with the sum of flows of all commodities moving on ring r from node j to node i must be no greater than 48 times the size of the ring (rings sizes are expressed in OC48, while flows are given in units of DS3 and thus the factor of 48), i.e. for $r \in \mathcal{R},\ \ (i,j) \in \mathcal{A}_r^{\mathcal{R}}$,

$$\sum_{k \in \mathcal{K}} f_{i,j}^{r,k} + \sum_{k \in \mathcal{K}} f_{j,i}^{r,k} \leq 48 u_r.$$

Ring sizes are limited by link sizes. Since each unit of link can carry 8 OC48 signals, the sum of all sizes of rings that contain link $l \in \mathcal{L}$ (this set of rings is denoted by $\mathcal{R}_l^{\mathcal{L}}$) can be at most 8 times the size of that link, i.e.,

$$\sum_{r \in \mathcal{R}_l^{\mathcal{L}}} u_r \leq 8 w_l, \ \ l \in \mathcal{L}.$$

In the heuristic used to approximately solve the integer program, upper bounds on the ring sizes are needs, i.e.

$$0 \leq u_r \leq \overline{u}_r, \ \ r \in \mathcal{R}.$$

The objective of the integer programming problem is to minimize the total cost. The total cost consists of five major components, ring cost, link cost, crossconnect cost, loading cost, and unloading cost. We use the cost coefficients defined in Table 3. The total ring cost is defined to be

$$\sum_{r \in \mathcal{R}} c_r^{\mathcal{R}} u_r.$$

The total link cost is defined as

$$\sum_{l \in \mathcal{L}} c_l^{\mathcal{L}} w_l.$$

The total crossconnect cost is

$$\sum_{r \in \mathcal{R}, s \in \mathcal{R} (r \neq s)} \left\{ \sum_{n \in \mathcal{N}_r^{\mathcal{R}} \cap \mathcal{N}_s^{\mathcal{R}}} \left(\sum_{k \in \mathcal{K}} c^{\mathcal{X}} x_{n,k}^{r,s} \right) \right\}.$$

The total loading cost is defined to be

$$\sum_{r\in\mathcal{R}}\left\{\sum_{n\in\mathcal{N}_r^{\mathcal{R}}}\left(\sum_{k\in\mathcal{K}\ni S_n^k>0} c^{\mathcal{I}} z_{n,k}^r\right)\right\}.$$

The total unloading cost defined as

$$\sum_{r\in\mathcal{R}}\left\{\sum_{n\in\mathcal{N}_r^{\mathcal{R}}\cap\mathcal{K}} c^{\mathcal{I}} y_n^r\right\}.$$

The cost coefficients are computed as follows. Some costs are associated with rings. A link $l \in \mathcal{A}_r^{\mathcal{R}}$ has associated with it a link distance $d_l > 0$. Every 225 miles, a distance requires a signal regenerator (REGEN) per ring containing that link per DS3 of capacity. Therefore, link l requires $\lfloor d_l/225\rfloor$ regenerators per ring. Furthermore, each ring link requires two add drop multiplexer (ADM), hence ring r has $2|\mathcal{A}_r^{\mathcal{R}}|$ ADMs. To account for the backup ring, these numbers are doubled. Hence we have

$$c_r^{\mathcal{R}} = 2\cdot\lfloor d_l/225\rfloor\cdot\texttt{REGENCOST} + 4\cdot|\mathcal{A}_r^{\mathcal{R}}|\cdot\texttt{ADMCOST},$$

where `REGENCOST` is the unit cost of a regenerator and `ADMCOST` is the unit cost of an add drop multiplexer.

Some costs are associated with links. Link l requires one optical amplifier (OA) every 75 miles, i.e. $\lfloor d_l/75\rfloor$ OAs are needed on link l. Two dense wave division multiplexer (DWDM) are required per link, with an additional two per REGEN site on that link. Hence, $2\cdot(1+\lfloor d_l/225\rfloor)$ DWDMs are needed on link l. Finally, each link has, per DS3 of capacity, a low-speed terminator used on the add drop multiplexer (ADM) associated with the link. To account for the backup links, these numbers must be doubled. Therefore, the link cost

$$c_l^{\mathcal{L}} = 2\cdot\lfloor d_l/75\rfloor\cdot\texttt{OACOST} + 4\cdot(1+\lfloor d_l/225\rfloor)\cdot\texttt{DWDMCOST} + 2\cdot\texttt{LOWCOST},$$

where `OACOST` is the unit cost of an optical amplifier, `DWDMCOST` is the unit cost of a dense wave division multiplexer, and `LOWCOST` is the unit cost of a low-speed terminator.

At each crossconnect point two DACSs and 17/24 low-speed terminators per DS3 are needed. With the doubling to account for backup, we have

$$c^{\mathcal{X}} = 4\cdot\texttt{DACSCOST} + 17/12\cdot\texttt{LOWCOST},$$

where `DACSCOST` is the cost of a DACS terminal.

Finally, each point of loading and unloading demand requires one DACS and 17/96 low-speed terminators per DS3. Therefore,

$$c^{\mathcal{I}} = 2\cdot\texttt{DACSCOST} + 17/48\cdot\texttt{LOWCOST}.$$

The integer programming problem for designing SONET ring networks with single ring interworking is

$$\begin{aligned}\min\qquad & \sum_{r\in\mathcal{R}} c_r^{\mathcal{R}} u_r + \sum_{l\in\mathcal{L}} c_l^{\mathcal{L}} w_l + \sum_{r\in\mathcal{R},s\in\mathcal{R}(r\neq s)}\left\{\sum_{n\in\mathcal{N}_r^{\mathcal{R}}\cap\mathcal{N}_s^{\mathcal{R}}}\left(\sum_{k\in\mathcal{K}} c^{\mathcal{X}} x_{n,k}^{r,s}\right)\right\} + \\ & \sum_{r\in\mathcal{R}}\left\{\sum_{n\in\mathcal{N}_r^{\mathcal{R}}}\left(\sum_{k\in\mathcal{K}\ni S_n^k>0} c^{\mathcal{I}} z_{n,k}^r\right)\right\} + \sum_{r\in\mathcal{R}}\left\{\sum_{n\in\mathcal{N}_r^{\mathcal{R}}\cap\mathcal{K}} c^{\mathcal{I}} y_n^r\right\}\end{aligned}$$

subject to

$$\sum_{r\in\mathcal{R}_k^{\mathcal{N}}} y_k^r = D^k,\ k\in\mathcal{K},$$

$$\sum_{r\in\mathcal{R}_n^{\mathcal{N}}} z_{n,k}^r = S_n^k,\ k\in\mathcal{K},\ n\in\mathcal{N},$$

$$(2)\quad \sum_{i\ni(i,n)\in\vec{\mathcal{A}}_r^{\mathcal{R}}} f_{i,n}^{r,k} + z_{n,k}^r + \sum_{s\in\mathcal{R}_n^{\mathcal{N}}\setminus\{r\}} x_{n,k}^{s,r} -$$

$$\sum_{i\ni(n,i)\in\vec{\mathcal{A}}_r^{\mathcal{R}}} f_{n,i}^{r,k} - \sum_{s\in\mathcal{R}_n^{\mathcal{N}}\setminus\{r\}} x_{n,k}^{r,s} = \begin{cases} y_r^k & \text{if } n=k, \\ 0 & \text{if } n\neq k \end{cases} ,r\in\mathcal{R},\ n\in\mathcal{N}_r^{\mathcal{R}},\ k\in\mathcal{K},$$

$$\sum_{k\in\mathcal{K}} f_{i,j}^{r,k} + \sum_{k\in\mathcal{K}} f_{j,i}^{r,k} \le 48u_r, r\in\mathcal{R},\ \ (i,j)\in\mathcal{A}_r^{\mathcal{R}},$$

$$\sum_{r\in\mathcal{R}_l^{\mathcal{L}}} u_r \le 8w_l,\ \ l\in\mathcal{L},$$

$$y_k^r \ge 0, r\in\mathcal{R}, k\in\mathcal{K}, \text{integer},$$

$$z_{i,k}^r \ge 0, r\in\mathcal{R}, i\in\mathcal{N}_r^{\mathcal{R}}, k\in\mathcal{K}, \text{integer},$$

$$f_{i,j}^{r,k} \ge 0, r\in\mathcal{R}, k\in\mathcal{K}, (i,j)=l\in\mathcal{A}_r^{\mathcal{R}}, \text{integer},$$

$$x_{n,k}^{r,s} \ge 0, r,s\in\mathcal{R}\ni\mathcal{N}_r^{\mathcal{R}}\cap\mathcal{N}_s^{\mathcal{R}}\neq\emptyset, n\in\mathcal{N}_r^{\mathcal{R}}\cap\mathcal{N}_s^{\mathcal{R}}, k\in\mathcal{K}, \text{integer},$$

$$w_l \ge 0,\ \ l\in\mathcal{L}, \text{integer},$$

$$0\le u_r\le \overline{u}_r,\ \ r\in\mathcal{R}, \text{integer}.$$

4.3. **Heuristic solution of the integer program.** The integer program presented in Subsection 4.2 is, for practical purposes, far larger than current integer programming software can handle. Therefore we will will settle for a heuristic that produces an approximate solution to the integer program, i.e. an integer solution not guaranteed to be optimal. In this subsection we describe one such heuristic solution method. Since the value of the objective function of the linear program provides us with a lower bound on the value of the optimal integer solution, we can in some sense measure the quality of the integer solution produced.

The procedure for producing an integer solution starts with the optimal solution of the linear programming relaxation of the integer program. Since ring sizes may be fractional in the optimal linear programming solution, an easy way to produce a feasible integer ring solution is to round up each fractional ring size. Rounding all ring sizes up by one, may produce enough slack to allow one or more ring sizes to be reduced by one or more units. This is the central scheme of the heuristic described in this subsection.

The heuristic begins by setting the upper bounds $\overline{u}_r$ on each ring size are set to $\lceil u_r \rceil$ and proceeds by attempting to find a ring for which the upper bound can be reduced by a unit and still produce a feasible solution. This is done, greedily and with randomization, in the manner prescribed by the construction phase of a GRASP [14]. The procedure repeats itself until no ring can be reduced in size without causing infeasibility, thus producing a feasible integer solution. Since randomization is present in the construction phase, this construction phase can itself be repeated several times, each one using a different random number generator seed, thus producing possibly different integer solutions, the best of which is kept as the heuristic solution to the integer program.

We next describe in detail this heuristic procedure. In a GRASP construction phase one builds a solution one element at a time. In this context, a solution is built by setting one ring size variable u_r at a time to an integer value. The variable to be set is selected in a greedy randomized fashion. A greedy choice is to pick a variable that has a high cost associated with it, or that was only slightly above an integer value in the linear programming solution but had to be rounded up because it was fractional. For example a variable that was rounded up from 1.02 to 2.0 would be preferred to a variable that was rounded up from 1.95 to 2.0, given that their costs were identical. To take both greedy components into account, define the parameter

$$\xi_r = 1 + u_r - \lceil u_r \rceil,$$

to be a measure of how much a ring size was rounded up. Rings with a small ξ_r value were rounded up by a large amount, while rings with a large ξ_r value were rounded up slightly. Ignoring costs, a greedy choice would pick rings having small ξ_r values to attempt size reduction.

To take into consideration costs, define the parameter

$$\gamma_r = \frac{c_r^R}{\xi_r}.$$

This parameter is larger for rings with a high cost and that have be rounded up by a large amount than for rings with small cost that have been rounded up slightly. A greedy choice picks the ring having the largest γ_r value as the ring to attempt size reduction.

Picking the greedy choice limits the number of solutions that can be generated. If a deterministic rule breaks ties, then this algorithm would produce a single solution. Even if ties are broken at random, the number of solutions is still somewhat limited. A GRASP construction increases the number of greedy-like solutions generated by defining a set of well ranked choices (with respect to the greedy parameter γ_r) and picks one candidate from this set at random. This set is called the restricted candidate list (RCL). To restrict the candidates, let

$$\underline{\gamma} = \min \{\gamma_r \mid \text{ring } r \text{ has not yet been selected}\},$$

and

$$\overline{\gamma} = \max \{\gamma_r \mid \text{ring } r \text{ has not yet been selected}\}.$$

A ring r is made a member of the RCL if

$$\gamma_r \geq \underline{\gamma} + \alpha(\overline{\gamma} - \underline{\gamma}),$$

where α $(0 \leq \alpha \leq 1)$ is a parameter that controls the greediness or randomness of the choice. A value of $\alpha = 1$ produces a greedy algorithm, while a value $\alpha = 0$ generates a random algorithm. Good quality diverse solutions are generated using a value of α between 0 and 1.

TABLE 4. Point-to-point demand

	a	b	c	d	e	f
a	·	·	10	10	10	10
b	·	·	10	10	10	10
c	10	10	·	·	10	10
d	10	10	·	·	10	10
e	10	10	10	10	·	·
f	10	10	10	10	·	·

Once a ring is selected to be reduced, its upper bound is reduced by a unit, i.e. $\overline{u}_r = \overline{u}_r - 1$, and the linear programming relaxation is reoptimized. If a feasible (linear programming) solution is produced, the choice is accepted. If the upper bound $\overline{u}_r = 0$, then ring r is no longer considered as a candidate for reduction. If the resulting linear program is infeasible, the choice is rejected and the upper bound is reset to its previous value, i.e. $\overline{u}_r = \overline{u}_r + 1$. This ring is no longer a potential candidate to enter the RCL. The construction procedure is repeated until no further candidate exists.

Figure 16 illustrates the heuristic solution method with pseudo-code. The linear programming relaxation is solved in line 2 of the pseudo-code using unlimited ring sizes (upper bounds are set to ∞ in line 1). The possibly fractional ring sizes produced by the linear program is array u^*. In line 3, ring size upper bounds are saved in $\overline{u}'$ as the fractional ring size values rounded up and `BestCost` is initialized in line 4. The GRASP iterations are carried out in the loop in lines 5 to 26. In line 6 of each iteration, the set of possible candidate rings $\tilde{\mathcal{R}}$ is initially set to the set of all rings and the ring size upper bounds are initialized to the original rounded up bounds $\overline{u}'$. The construction of an integer ring solution is done in lines 7 to 26. The greedy function is computed in lines 8 and 9 and the restricted candidate list is built in lines 10 and 11. A ring r^* is picked at random from the RCL in line 12 and the upper bound of that ring size is decreased by a unit (line 13) and the linear programming relaxation using the newly set upper bounds is solved in line 14 (u^* is the vector of optimal ring sizes generated by the linear program, if the linear program is feasible). However, the linear program may be infeasible for this set of ring size upper bounds. If that is the case, the upper bound of ring r^* is reset to its previous value (line 16) and ring r^* is removed from further consideration as a candidate for size reduction in line 17. If the linear program is feasible, then if ring r^* has been removed from the network ($\overline{u}_{r^*} = 0$), then ring r^* is removed from the set of possible candidate rings $\tilde{\mathcal{R}}$ in line 20. If the design cost using the current settings of upper bounds is the best so far, it is saved in lines 22 and 23.

4.4. **A small example of SONET ring network design.**

In this subsection, we illustrate, on a small network, a possible application of the SONET ring network design tool.

Assume we are given the network shown in Figure 17. This network has 12 nodes (labeled "a," "b," ..., "f") and 15 links (labeled "1," "2," ..., "15"). Only nodes "a," "b," "c," "d," "e," and "f" have point-to-point demands (shown in Table 4) associated with them.

The task we seek to accomplish is design a SONET ring network using rings from a set of seven candidate rings, identified by their links in Table 5 and shown

```
procedure LocalSearch($\mathcal{P}, \mathcal{R}_1, \ldots, \mathcal{R}_{|\mathcal{P}|}$){
1    for $e \in E$ {
2        $\mathcal{P}_e = \emptyset$;
3        for $p \in \mathcal{P}$ {
4            if $e \in \mathcal{R}_p$ {
5                $\mathcal{P}_e = \mathcal{P}_e \cup \{p\}$;
6            }
7        }
8    }
9    Compute routing cost: cost $= \mathcal{G}(\mathcal{P}_1, \ldots, \mathcal{P}_m)$;
10   for $p \in \mathcal{P}$ {
11       $\tilde{\mathcal{P}} = \mathcal{P} \setminus \{p\}$;
12       for $e \in E$ {
13           Compute edge length $\mathcal{L}_e^{\tilde{\mathcal{P}}}$;
14       }
15       Reroute $p$: $\mathcal{R}_p^{rr} = \mathtt{sp}(o_p, d_p, \mathcal{L}_1^{\tilde{\mathcal{P}}}, \ldots, \mathcal{L}_m^{\tilde{\mathcal{P}}})$;
16       for $e \in E$ {
17           $\hat{\mathcal{P}}_e = \mathcal{P}_e$;
18       }
19       for $e \in E$ {
20           if $e \in \mathcal{R}_p$ {
21               $\hat{\mathcal{P}}_e = \hat{\mathcal{P}}_e \setminus \{p\}$;
22           }
23           if $e \in \mathcal{R}_p^{rr}$ {
24               $\hat{\mathcal{P}}_e = \hat{\mathcal{P}}_e \cup \{p\}$;
25           }
26       }
27       Compute rerouting cost: rrcost $= \mathcal{G}(\hat{\mathcal{P}}_1, \ldots, \hat{\mathcal{P}}_m)$;
28       if rrcost $<$ cost {
29           $\mathcal{R}_p = \mathcal{R}_p^{rr}$;
30           LocalSearch($\mathcal{P}, \mathcal{R}_1, \ldots, \mathcal{R}_{|\mathcal{P}|}$);
31           return;
32       }
33   }
}
```

FIGURE 15. GRASP local search phase pseudo code

TABLE 5. Definition of rings by link set

ring	links							
1	1	2	3	4				
2	5	6	7	8				
3	9	10	11	12				
4	3	7	11	13	14	15		
5	1	2	13	7	14	11	15	4
6	5	6	14	11	15	3	13	8
7	9	10	15	3	13	7	14	12

```
procedure mkSonet
1    $\overline{u} = \infty$;
2    $u^* = \texttt{SolveLP}(\overline{u})$;
3    do $r \in \mathcal{R} \rightarrow \overline{u}'_r = \lceil u^*_r \rceil$ od;
4    $\texttt{BestCost} = \infty$;
5    do $\texttt{Gitr} = 1, \ldots, \texttt{maxGitr} \rightarrow$
6        $\tilde{\mathcal{R}} = \mathcal{R}$;  do $r \in \mathcal{R} \rightarrow \overline{u}_r = \overline{u}'_r$ od;
7        do $\tilde{\mathcal{R}} \neq \emptyset \rightarrow$
8            do $r \in \tilde{\mathcal{R}} \rightarrow \xi_r = 1 + u^*_r - \overline{u}_r$ od;
9            do $r \in \tilde{\mathcal{R}} \rightarrow \gamma_r = c^{\mathcal{R}}_r / \xi_r$ od;
10           $\underline{\gamma} = \min\{\gamma_r \mid r \in \tilde{\mathcal{R}}\}$;   $\overline{\gamma} = \max\{\gamma_r \mid r \in \tilde{\mathcal{R}}\}$;
11           $\texttt{RCL} = \{r \in \tilde{\mathcal{R}} \mid \gamma_r \geq \underline{\gamma} + \alpha(\overline{\gamma} - \underline{\gamma})\}$;
12           $r^* = \texttt{Pick@Random}(\texttt{RCL})$;
13           $\overline{u}_{r^*} = \overline{u}_{r^*} - 1$;
14           $u^* = \texttt{SolveLP}(\overline{u})$;
15           if LP is infeasible $\rightarrow$
16               $\overline{u}_{r^*} = \overline{u}_{r^*} + 1$;
17               $\tilde{\mathcal{R}} = \tilde{\mathcal{R}} \setminus \{r^*\}$
18           fi;
19           if LP is feasible $\rightarrow$
20               if $\overline{u}_{r^*} = 0 \rightarrow \tilde{\mathcal{R}} = \tilde{\mathcal{R}} \setminus \{r^*\}$ fi;
21               if $\texttt{LPcost}(\overline{u}) < \texttt{BestCost} \rightarrow$
22                   $\overline{u}^* = \overline{u}$;
23                   $\texttt{BestCost} = \texttt{LPcost}(\overline{u}^*)$
24               fi
25           fi
26       od
27   od
end mkSonet;
```

FIGURE 16. Pseudo-code for a SONET ring design tool heuristic

TABLE 6. Cost parameters used in small example

parameter	description	value
`REGENCOST`	regenerator	\$165,000.00
`ADMCOST`	add drop multiplexer	\$92,500.00
`OACOST`	optical amplifier	\$251,000.00
`DWDMCOST`	dense wave division multiplexer	\$266,000.00
`LOWCOST`	low-speed terminator	\$4,500.00
`DACSCOST`	DACS terminator	\$900.00

in Figure 18. For costing purposes, we assume that all links are of length 50 miles and use the cost parameters shown in Table 6.

To apply the multicommodity flow model, we require the definition of the set of commodities. Applying the greedy algorithm, described in Subsection 4.2, results

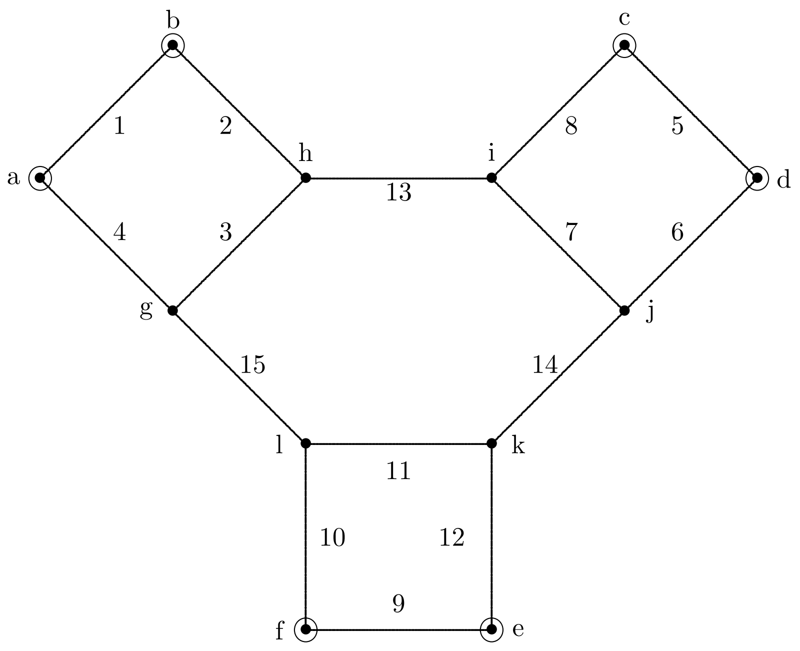

FIGURE 17. Example network.

TABLE 7. Point-to-commodity demand

	a	b	c	d
c	10	10	·	·
d	10	10	·	·
e	10	10	10	10
f	10	10	10	10

in commodities "a," "b," "c," and "d" and point-to-commodity demands shown in Table 7.

Note that, as presented, this problem is symmetric with respect to nodes, links, demand requirements, and ring sets. As such, one should expect several designs of equal cost. Using the arbitrary set of commodities reduces the symmetry by fixing sinks and sources, making sink nodes "a" and "b" each demand 40 DS3s, sink nodes "c" and "d" each demand 20 DS3s, while source nodes "c" and "d" each supply 10 DS3s, and source nodes "e" and "f" each supply 40 DS3s.

The heuristic was run 200 iterations, with each GRASP iteration using RCL parameter $\alpha = 0.5$. Table 8 shows the ring configurations produced by the heuristic. In each solution, each ring was dimensioned with one OC48 of capacity. Seven ring configurations were produced. In 72% of the iterations, a ring configuration having cost \$23.3 million was produced. In 27.5%, the cost of the configuration was \$25.5 million, while in 1 out of the 200 iterations a ring configuration of cost \$27.0 million was generated. Note that the \$23.3 million designs each have two small rings and

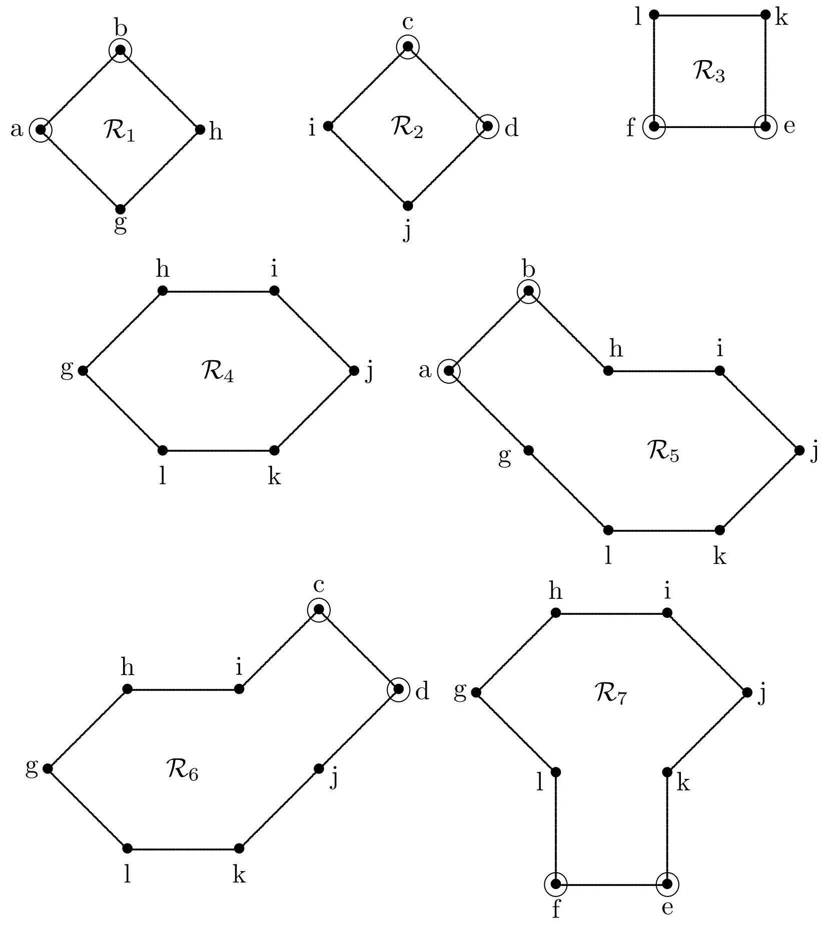

FIGURE 18. Seven candidate rings for network design

TABLE 8. Ring configurations produced by SONET ring design model

occurrences	rings	cost
37	1,2,7	\$23,356,437.50
52	1,3,6	\$23,356,437.50
55	2,3,5	\$23,356,437.50
18	1,6,7	\$25,510,718.75
19	2,5,7	\$25,510,718.75
18	3,5,6	\$25,510,718.75
1	5,6,7	\$26,990,718.75

one large ring. The \$25.5 million designs each have two large rings and one small ring, while the \$27.0 million configuration has three large rings and no small ring. The model generated all ring configurations having one large and two small rings,

Table 9. Aggregate link flow solution produced by SONET ring model

		orientation			
ring	link	from	to	commodity	flow
1	2	h	b	b	40
1	4	g	a	a	40
2	5	c	d	b	10
2	5	d	c	a	10
2	6	d	j	b	20
2	6	j	d	d	20
2	7	j	i	b	20
2	7	j	i	c	20
2	8	c	i	a	20
2	8	i	c	c	20
7	3	g	h	b	20
7	3	h	g	a	20
7	9	e	f	a	10
7	9	e	f	b	10
7	9	f	e	c	10
7	9	f	e	d	10
7	10	f	l	a	20
7	10	f	l	b	20
7	12	e	k	c	20
7	12	e	k	d	20
7	13	i	h	a	20
7	13	i	h	b	20
7	14	k	j	c	20
7	14	k	j	d	20
7	15	l	g	a	20
7	15	l	g	b	20

Table 10. Aggregate crossconnect solution produced by SONET ring model

ring 1	ring 2	node	commodity	flow
2	7	i	a	20
2	7	i	b	20
7	1	g	a	40
7	1	h	b	40
7	2	j	c	20
7	2	j	d	20

two large and one small ring and three large rings. Intermediate size ring $\mathcal{R}_4$ was, as expected, not used in any solution, since it is superfluous.

TABLE 11. Point-to-point demand routes

demand from	to	flow	route: sequence of node-rings							
c	a	10	c-2	i-2	i-7	h-7	g-7	g-1	a-1	
c	b	10	c-2	d-2	j-2	i-2	i-7	h-7	h-1	b-1
d	a	10	d-2	c-2	i-2	i-7	h-7	g-7	g-1	a-1
d	b	10	d-2	j-2	i-2	i-7	h-7	h-1	b-1	
e	a	10	e-7	f-7	l-7	g-7	g-1	a-1		
e	b	10	e-7	f-7	l-7	g-7	h-7	h-1	b-1	
e	c	10	e-7	k-7	j-7	j-2	i-2	c-2		
e	d	10	e-7	k-7	j-7	j-2	d-2			
f	a	10	f-7	l-7	g-7	g-1	a-1			
f	b	10	f-7	l-7	g-7	h-7	h-1	b-1		
f	c	10	f-7	e-7	k-7	j-7	j-2	i-2	c-2	
f	d	10	f-7	e-7	k-7	j-7	j-2	d-2		

Consider now one of the lowest-cost solutions, with ring configuration "1,2,7." For that design, the model produced aggregate link flow ($f_{i,j}^{r,k}$) shown in Table 9 and aggregate crossconnect flows ($x_{n,k}^{r_1,r_2}$) shown in Table 10.

To extract the individual point-to-point flows from the aggregate solution, four auxiliary networks are set up, one for each commodity, as prescribed in our discussion in Subsection 4.2. These networks are shown in Figures 19, 20, 21, and 22, respectively, for commodities "a," "b," "c," and "d." Consider the case of commodity "a" in Figure 19. Four point-to-point demand pairs ("c"-"a," "d"-"a," "e"-"a," and "f"-"a") have "a" as commodity (or sink node), each with a demand requirement of 10 DS3s. Solving the four maximum flow problems (from "c" to "a," from "d" to "a," from "e" to "a," and from "f" to "a") on the auxiliary network, yields the individual point-to-point flows. Flow from "c" to "a" begins on ring $\mathcal{R}_2$, and is routed through node "i," where it switches to ring $\mathcal{R}_7$. From node "i" it goes to node "h" and node "g," where it switches to ring $\mathcal{R}_1$ and arrives in node "a." Table 11 lists routes for all node pairs. Note that here, all demand for a particular node pair is routed on a single route, though this need not be the case. Furthermore, the aggregate flows produced by the model were all integer. This also need not always occur since the linear programming coefficient matrix is not necessarily unimodular and thus the linear program can have fractional optimal solutions. However, since the extraction of the individual flows is done with a maximum flow algorithm, if the aggregate flows are all integer, then there exist integer flows for the individual demand pairs and most maximum flow algorithms produce integer flows.

Adding up the directed flows in the network produces the link capacities shown in Figure 23, where one can see the result of arbitrarily selecting commodities "a," "b," "c," and "d," in that order.

4.5. **Extensions to the basic model.** In this subsection, we consider some extensions of the basic model. We consider changes needed to allow for dual ring interworking, discuss the use of mesh in a hybrid routing scheme, and consider different levels of routing to reduce the number of linear programming variables. We show how to modify the basic model to disallow U-turning and identify two known problems with the currently proposed method.

4.5.1. *Dual Ring Interworking.* The basic model allows inter-ring traffic to cross between rings r_1 and r_2 if r_1 and r_2 have a node n in common (see Figure ??). But this does not make the SONET network survivable for a node failure at n. In order to achieve survivability in the SONET network a number of variations of Dual Ring Interworking (DRI) have been standardized. In this subsection we will describe how to extend the basic model to handle all the currently proposed methods of doing DRI.

Each interworking method is determined by a start ring, a start node on the start ring, an end ring, an end node on the end ring, and a set of links used (both primary and secondary) by the interworking method. For example, in same side interworking if ring r_1 and r_2 share a link $l = (a, b)$ then the same side routing interworking method has start ring r_1, start node a, end ring r_2, end node a, and links (r_1, l) and (r_2, l) (we describe each link with a (ring,link) pair).

For any interworking method, c, we let $start_c = (r_1, n_1) \in \mathcal{R} \times \mathcal{N}$, where r_1 is the start ring and n_1 the start node, and $end_c = (r_2, n_2) \in \mathcal{R} \times \mathcal{N}$, where r_2 is the end ring and n_2 the end node, and $links_c$ is the subset of $\mathcal{R} \times \mathcal{L}$ used. We assume that we are given as input an enumeration $\mathcal{C}$ of the possible methods of interworking, and for each $c \in \mathcal{C}$, $start_c$, end_c, and $links_c$.

Note that this input formulation does not assume any specific variant of DRI. A given problem instance may include some interworking methods and not others. We have software that, given the candidate ring set, will construct the enumerations for three scenarios: SRI (single node interworking, i.e., as described in the basic model), basic DRI (using same and opposite side interworking), extended DRI (also using common node DRI). Other scenarios are very easy to construct.

In the extended linear programming formulation we have the following changes of the basic model. The crossover variables $x^{r,s}_{n,k}$ are replaced by variables x^c_k, where $k \in \mathcal{K}$ and $c \in \mathcal{C}$, changing constraints

$$(3)\qquad \sum_{i \in \mathcal{N}^{\mathcal{R}}_r \setminus \{n\}} f^{r,k}_{i,n} + z^r_{n,k} + \sum_{\substack{c \in \mathcal{C} \\ (r,n) = end_c}} x^c_k = \sum_{i \in \mathcal{N}^{\mathcal{R}}_r \setminus \{n\}} f^{r,k}_{n,i} + y^r_k + \sum_{\substack{c \in \mathcal{C} \\ (n,r) = start_c}} x^c_k, \quad r \in \mathcal{R},\ n \in \mathcal{N}^{\mathcal{R}}_r,\ k \in \mathcal{K},$$

and

$$\sum_{k \in \mathcal{K}} f^{r,k}_{i,j} + \sum_{k \in \mathcal{K}} f^{r,k}_{j,i} + \sum_{\substack{c \in \mathcal{C} \\ (r,l) \in links_c}} x^c_k \le 48 u_r, r \in \mathcal{R},\ \ l = (i,j) \in \mathcal{A}^{\mathcal{R}}_r.$$

4.5.2. *Mesh.* In the transition from using a mesh based network to a fully SONET ring based network there will be a need for hybrid solutions where some demands are routed on the mesh and some are routed on the rings. It is quite straightforward to extended the basic model to incorporate the mesh network.

4.5.3. *Different Levels of Routing Detail.* As a way of speeding up the linear program by reducing the number of variables and constraints, it is useful to have two levels of routing detail. The high detail routing is the one described in the basic model. If a demand k is to be routed at low detail on a ring r then all we require is that half of the demand be routed one way on the ring and the other half the other way. This ensures that the demand incurs the same load on every link on the ring. For a given ring we may have some demand being routed at high detail and some at low detail.

For example, given an office in Rochester, NY, there may be no demand for traffic to Rochester from anywhere in California. Thus there is not much chance that any traffic to Rochester will use any ring located in California. Thus we can route the Rochester traffic at low detail on the rings in California, whereas we would route them at high detail on rings closer to Rochester.

This trick reduces the size of the linear program by an order of magnitude. The decision of which level to route the traffic at is done iteratively. In the first iteration all demands are routed at low detail on all rings. In the following iteration we route at high detail a demand k on ring r if, in the solution of the previous iteration, ring r was actually used to route demand k.

This framework allows us to solve instances covering the whole AT&T network, using large sets of candidate rings.

4.5.4. *Disallowing U-turns.* In a SONET ring, a demand cannot "make a u-turn" at a node. Our basic model, however, has no constraint imposing this. In order to take care of this problem we have to keep track of the direction that the flow is taking on each ring.

The solution is to split each flow conservation constraint (for (r, n)) into two constraints; one for each direction. This roughly doubles the number of constraints and adds a small number of new variables (we need to splits the y's and the z's, too). One might expect the time to solve the LP would increase, but in fact the time decreases since the solver takes advantage of the more constrained problem.

This splitting also allows us to avoid other routing problems: e.g., a route that only touches a node on a ring but never uses an edge.

4.5.5. *Currently Known Problems.*

4.5.6. *Slotting.* The model does not assign time slots to the demands, and we know of examples where the solutions generated by our system cannot be slotted in the given number of rings. We have found this not to be a problem for SRI, but for DRI roughly 2% more rings are required than our system suggests in order to assign time slots.

If one were willing and able to perform a small number (about 1%) of time slot interchanges, it would be possible to do the time slotting using the number of rings given by our solution. It is also possible that rerouting a small number of demands after performing slotting would solve the problem. Otherwise, a few extra rings are be required.

4.5.7. *Integrality of Flows.* In the LP solution there is no constraint that says that the flows have to be integral. In practice almost all are integral, eg. of 18890 demands all except 103 (.5%) could be integrally routed. Of those 103 many (maybe all) could be rerouted within the rings provided by our system.

5. Global network planning

With the worldwide market liberalization of telecommunications, the international telecommunications environment is changing from the traditional bilateral mode of operation, where each network between pairs of administrations (AT&T and British Telecom, AT&T and France Telecom, British Telecom and France Telecom, etc.) is planned separately, to a more global, alliance-based, environment, where the network needs of several administrations may be planned simultaneously. This allows network planning to be done more in the manner that a single national network is designed, as opposed to many individual networks [6, 3, 24].

In this section, we describe a methodology for analyzing global facility capacity requirements which includes, in its core, a linear programming (LP) model for multicommodity flows. Techniques are applied to reduce the problems to manageable sizes. In particular, we use the concept of some aggregate flow as commodities, as opposed to the traditional point-to-point traffic [20], which results in drastic reduction in the size of the derived LP problem.

5.1. Problem Description.

The minimum cost capacity problem that we address assumes the following data is provided:

- a set of nodes from which traffic originates and terminates,
- an existing facility network from which capacity may be acquired,
- demand for both switched and dedicated services between the nodes.

It is also assumed that any aggregation of low bandwidth traffic from a group of locations and its homing into pre-specified locations, called backbone nodes, have already occurred. In this problem, we seek to determine the amount of capacity on the different facilities of the existing underlying network, so that the demand forecast is satisfied at minimum cost. For the problems of interest to us, the costs in the objective function will be formulated as yearly depreciation or leased costs plus operations and maintenance costs. Depending on the financial arrangements under which the network will operate, the costs associated with facilities already owned may, or may not, be different from the costs of facilities that must be acquired to satisfy the demand.

The heuristic approach we propose to analyze the minimum cost capacity problem described, encompasses two main steps, circuit demand forecast determination, and facility capacity requirements determination, each to be described next.

5.2. Circuit Demand Determination.

Demand between a pair of backbone nodes in the network is an estimate of the number of circuits needed to provide communications between that pair of nodes. Switched demand forecast, however, is usually initially developed as an estimate of the number of minutes needed between the node pairs. Determining the number of circuits required to satisfy a given number of minutes under a given blocking assumption is a well known procedure [6, 13]. It is also well known that when dealing with this problem in a worldwide basis, non-coincidence of demand plays an important role and should be taken into account so that overestimation of circuit requirements does not occur [6, 12].

Our heuristic for determining the facility capacity requirements involves the computation of the circuit demand for each of the 24 hours of the day, based on the CCITT load profiles [6], for each demand pair for which minute demand forecast exists. The circuit demand requirements for each demand pair is computed independently of each other. Non-coincidence of demand is considered during the facility capacity requirements determination step.

To finalize the determination of the circuit demand forecast to be considered in the facility capacity analysis step, the dedicated demand forecast is added to the switched circuit requirements of each of the twenty four periods.

5.3. Facility Capacity Analysis: A Mathematical Programming Formulation.

The problem addressed in the next step of the heuristic is the determination of the minimum cost set of facility requirements that satisfy the circuit demand forecast for all twenty four time periods. Depending on the scenario one wants to analyze, an embedded network representing facility capacity already owned, may or may not be considered.

Our heuristic approach involves first determining the minimum cost set of facility requirements for each of the twenty four periods separately, and then computing the overall capacity requirement for a given facility as the maximum requirement for that facility obtained from all the twenty four individual minimization problems.

The model used for computing the minimum set of facility requirements for a given time period is based on multicommodity flows on a network, a well known mathematical programming problem. Point-to-point demands are commodities that flow on the network, sharing link and node resources. Costs are linear functions of link capacities. The objective is to move demand between demand pairs satisfying flow conservation and demand requirements while minimizing the total cost.

5.3.1. *The Multicommodity Flow Problem.*

A telecommunications network can be viewed as a graph having a set of vertices, each representing a backbone node or a cable landing point housing cable heads of major cable systems, and a set of edges or arcs, each representing cable connecting two vertices. Let $G = (V, A)$ be a directed network consisting of a set V of vertices and a set A of arcs whose elements are ordered pairs of distinct vertices with a cost c_{ij} and a capacity u_{ij} associated with every arc $(i, j) \in A$. Let $N \subset V$ be the set of pre-defined backbone nodes of the network. We associate with each backbone node pair $i \in N$, $j \in N$, an integer d_{ij} representing the demand forecast between backbone nodes i and j. In the multicommodity flow model, a commodity could be defined as a unique point-to-point traffic. Point-to-point demand is given an arbitrary direction (one point is the source, the other is the sink) and we move demands from sources to sinks. This definition, however, can result in a large number of commodities if there is a large number of point-to-point demands, generating large hard to solve models. In our model, we use the concept of aggregate flow as commodities. The definition of commodities as an aggregate flow has the disadvantage of not explicitly providing the facility routing used in the solution found by the optimization model to satisfy the demand. However, if desired, the actual routes may be determined by a maximum flow type of algorithm as described in [1]. In our model, a commodity k is defined to be the aggregate flow having node k as sink. For that, the demand flows are given a direction, i.e. a source node and a sink node, and all demand terminating at each sink is aggregated into a commodity. Therefore, $|K| \leq |N|$, where K is the set of commodities.

Cable capacity around the world can be wholly owned by a telecommunication carrier or jointly owned by two or more telecommunication carriers. If we want to consider already owned capacity and want to differentiate between the two different types of capacity already owned (e.g. AT&T whole owned and AT&T joint ownership with a foreign TA) so that one type of capacity is used before the other, then based on the earlier definitions, the minimum cost network flow problem can be formulated as a multicommodity minimum cost flow problem as follows:

$$\min \sum_{(i,j)\in A} (c^1_{ij} y_{ij} + c^2_{ij} z_{ij}) + \sum_{(i,j)\in A} c^3_{ij} (\sum_{k=1}^{|K|} x_{ijk} - y_{ij} - z_{ij}) \tag{4}$$

subject to

$$\sum_{k=1}^{|K|} x_{ijk} \leq u^1_{ij} + y_{ij} + z_{ij}, \text{ for all } (i,j) \in A \text{ such that } i > j, \tag{5}$$

$$\sum_{i:(i,n)\in A} x_{ink} - \sum_{j:(n,j)\in A} x_{njk} = \sum_{i\in N} d_{in}, \text{ for all } n \in N \text{ and } k \in K \text{ such that } n = k, \tag{6}$$

$$\sum_{i:(i,n)\in A} x_{ink} - \sum_{j:(n,j)\in A} x_{njk} = -d_{nk}, \text{ for all } n \in N \text{ and } k \in K \text{ such that } n \neq k, \tag{7}$$

$$x_{ijk} \geq 0,\ y_{ij} \geq 0,\ z_{ij} \leq u^2_{ij}, \text{ for all } (i,j) \in A \text{ and all } k \in K.$$

In this formulation, commodity k is defined as the demand going to backbone node k from all other backbone nodes of the network, x_{ijk} represents the amount of flow (load) for commodity k on arc (i,j), i.e. flow going from node i to node j, while z_{ij} and y_{ij} represent the amount of jointly owned capacity and the amount of extra capacity on arc (i,j) required to satisfy the requirements for all commodities, respectively. Associated with z_{ij} are its upper bound u^2_{ij} (available amount of jointly owned capacity) and its utilization cost c^2_{ij}. Parameter c^1_{ij} is the annualized cost of acquiring extra capacity on arc (i,j). Associated with wholly owned capacity are its utilization cost c^3_{ij} and its available amount u^1_{ij}. The *bundle constraints* (5) indicate that the total flow on any arc cannot exceed the available wholly-owned plus jointly-owned capacities plus any acquired capacity. Each constraint (6) and (7) implies that the total flow out of a node minus the flow into that node must equal the net supply/demand of the node. These constraints are usually referred to as *mass balance constraints.* Since $d_{ij} > 0$, then constraints (7) imply that node n is a supplier of commodity k, and constraints (6) imply that node n is the demand node for commodity k, which by definition is the aggregate flow having node k as sink. With the presence of the bundle constraints, the essential problem is to determine where to acquire extra capacity and to distribute the capacity (wholly owned, jointly owned, or to be acquired) of each arc to individual commodities in a way that minimizes overall flow costs.

The generic multicommodity model described allows for changes in the types of scenarios to be run by setting some of its parameters to zero. Some of the changes may be as follow:

- If $u^1_{ij} = 0$, $u^2_{ij} = 0$, $c^2_{ij} = 0$, and $c^3_{ij} = 0$, then no embedded network (already owned capacity) is considered. In this case the problem is to acquire capacity from scratch to satisfy the demand forecast.
- If $c^2_{ij} = 0$ and $c^3_{ij} = 0$, then the already owned capacity will be used at zero cost before any extra capacity is acquired.

5.3.2. *Overall Capacity Requirements Determination.* After obtaining the hourly facility capacity requirements by solving the described multicommodity flow problem for each of the twenty four hour periods, the overall facility capacity requirement is obtained by finding the maximum capacity requirement over all periods. This approach may produce some overestimation of the capacity requirements, since it does not take full advantage of non-coincidence of demand.

5.4. **Commodities Determination.** If a non-zero forecast exists between every backbone node pair, then the minimum number of commodities when using the aggregate flow definition is $|N|-1$. However, if only a subset of backbone node pairs have non-zero demand forecast, then the number of commodities can be significantly reduced by the right choice of nodes as sinks. As described in [1], the problem of generating the smallest number of commodities, when using the aggregate flow commodity definition, is a node covering problem on the graph $H = (V, D)$ where D is a set of edges such that $(i, j) \in D$ if and only if nodes i and j have positive point-to-point demand. Given the inherent intractability of node covering, approximate heuristics, such as GRASP (greedy randomized adaptive search procedures) [16], can be used if the problem instance is not small enough so that an exact algorithm could be applied. We used GRASP on a problem containing 36 backbone nodes and found that only 18 needed to be defined as sinks, therefore leading to 18 commodities. This resulted in 39% reduction in the number of variables and 26% reduction in the number of constraints, yielding in a 22% reduction in the solution time, as shown in Table 12.

5.5. **Problem Size and Computational Results.** A few global network scenarios were used to test this methodology. The scenarios were run on a Silicon Graphics Challenge computer (250MHz MIPS R10000) using AMPL as the model generator and the parallel CPLEX 4.0 barrier (interior point) solver to solve the linear programs. We run the twenty four periods in four simultaneous batches containing six periods each. Therefore, the overall solution time for the minimum capacity problem is four times what appears in Table 12. The solution times reported include the model generation by AMPL as well as the time spent by the solver. All scenarios involved an underlying facility network, from which capacity could be acquired, containing over 1500 facilities and over 300 vertices. The desert model implies no initial capacity is owned by the partners in the global network being planned, while the embedded model takes into consideration the existence of capacity already owned by the partnership. The difference in the number of commodities presented in the last two rows in Table 1, is due to the difference between choosing the commodities using the node covering approach described in section 5.4 as opposed to randomly choosing them.

Acknowledgement

The author would like to acknowledge the following people who worked with him on the projects described in this paper. David Johnson, Anja Feldman, and Chris

Table 12. Computational Results

Backbone Nodes	Demand Pairs	Comm.	Var.	Const.	Model Characteristics	Sol. Time per Period
16	120	15	49905	6210	Embedded	2 1/2 hr
16	120	15	40068	5928	Desert	2 hr
36	56	25	80887	9306	Embedded	4 1/2 hr
36	56	18	49230	6858	Embedded	3 1/2 hr

Sorensen suggested local search strategies and provided data for th PoP placement study. Lucia Resende introduced the author to the private virtual circuit routing problem and modeled and implemented the routing tool. David Applegate, Carsten Lund, David Johnson, Steven Phillips, Nick Reingold, and Peter Winkler proposed, designed, and implemented the SONET network design tool. Larry Fosset, Dah Nain Lee, and Lucia Resende proposed, modeled, and implemented the tool for global network planning.

References

[1] D. Applegate, C. Lund, D. S. Johnson, S. Phillips, N. Reingold, M. G. C. Resende, and P. Winkler. A sonet ring design tool, 1997. Manuscript.

[2] A.V. Arslan, R.W. Loose, and J.L. Strand. Transport Plan of Record, Issue 2.0. Technical report, AT&T Network Services Division, Holmdel, NJ, January 1996.

[3] G. R. Ash, R. H. Cardwell, and R. P. Murray. Design and optimization of networks with dynamic routing. *The Bell System Technical Journal*, 60:1787–1820, 1981.

[4] V. L. Bennett, D. J. Eaton, and R. L. Church. Selecting sites for rural health workers. *Social Science Medicine*, 16:63–72, 1982.

[5] R. Carraghan and P.M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

[6] *CCITT Blue Book, Fascicle II.3, Telephone Network and ISDN Quality of Service, Network Management and Traffic Engineering Recommendations.*

[7] C. H. Chung. Recent applications of the maximal covering location planning (M. C. L. P.) model. *Journal of the European Operational Research Society*, 37:735–746, 1986.

[8] R. Church and C. ReVelle. The maximal covering location problem. *Papers of the Regional Science Association*, 32:101–118, 1974.

[9] M. S. Daskin, P. C. Jones, and T. J. Lowe. Rationalizing tool selection in a flexble manufacturing system for sheet metal products. *Operations Research*, 38:1104–1115, 1990.

[10] F. P. Dwyer and J. R. Evans. A branch and bound algorithm for the list selection problem in direct mail advertising. *Management Science*, 27:658–667, 1981.

[11] D. J. Eaton, M. S. Daskin, D. Simmons, B. Bulloch, and G. Jansma. Determining medical service vehicle deployment in Austin, Texas. *Interfaces*, 15:96–108, 1985.

[12] M. Eisenberg. Engineering traffic networks for more than one busy hour. *B.S.T.J.*, 546:1–20, 1977.

[13] A. K. Erlang. Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. *P. O. Elect. Engrs. J.*, 10:189, 1917.

[14] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[15] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[16] Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[17] M.R. Garey and D.S. Johnson. *Computers and intractability - A guide to the theory of NP-completeness.* W.H. Freeman and Company, 1979.

[18] David S. Johnson and Michael A. Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. DIMACS Series on Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.

[19] J. G. Klincewicz. Avoiding local optima in the p-hub location problem using tabu search and GRASP. *Annals of Operations Research*, 40:283–302, 1992.

[20] M. Minoux. Network synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, 19:313–360, 1989.

[21] M. G. C. Resende, L. S. Pitsoulis, and P. M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In D.-Z. Du, J. Gu, and P. M. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, Providence, R.I., 1996.

[22] M.B. Rosenwein and R.T. Wong. Physical ring generation for SONET ring networks. Technical memorandum AK0313500-960328-01TM, AT&T Laboratories, Holmdel, NJ, March 1996.

[23] D. Schilling, V. Jayaraman, and R. Barkhi. A review of covering problems in facility location. *Location Science*, 1:25–55, 1993.

[24] R. R. Stacey and D. Songhurst. Dynamic alternative routing in the british telecom trunk network. In *Innovations in Switching Technology, Proceeding of the International Switching Symposium*, New York, NY, 1987. IEEE Communications Society.

[25] D. J. Sweeney, R. L. Mairose, and R. K. Martin. Strategic planning in bank location. In *AIDS Proceedings*, November 1979.

(M.G.C. Resende) Information Sciences Research, AT&T Labs Research, Florham Park, NJ 07932 USA.

E-mail address, M.G.C. Resende: `mgcr@research.att.com`

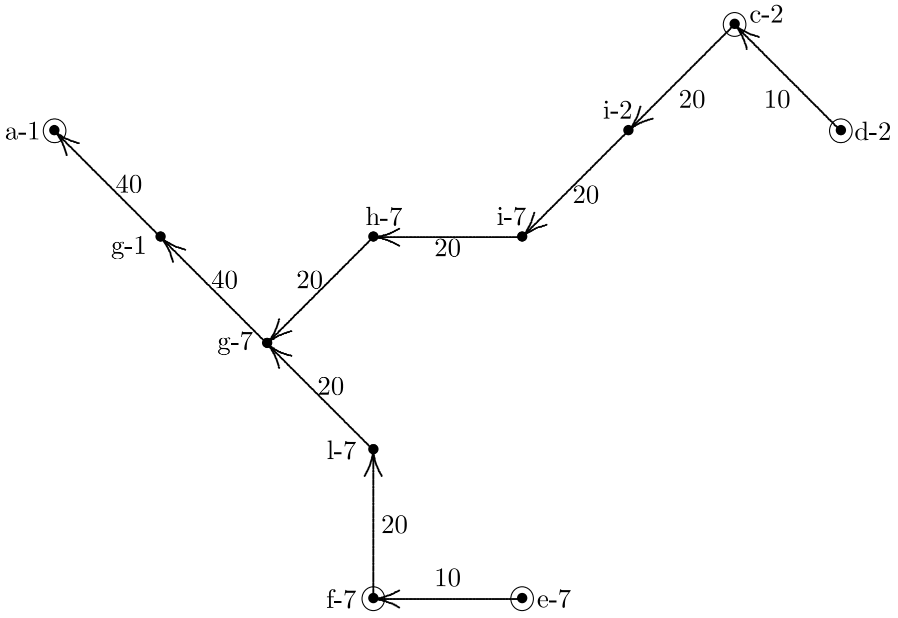

Figure 19. Flows and crossconnect solution for commodity a.

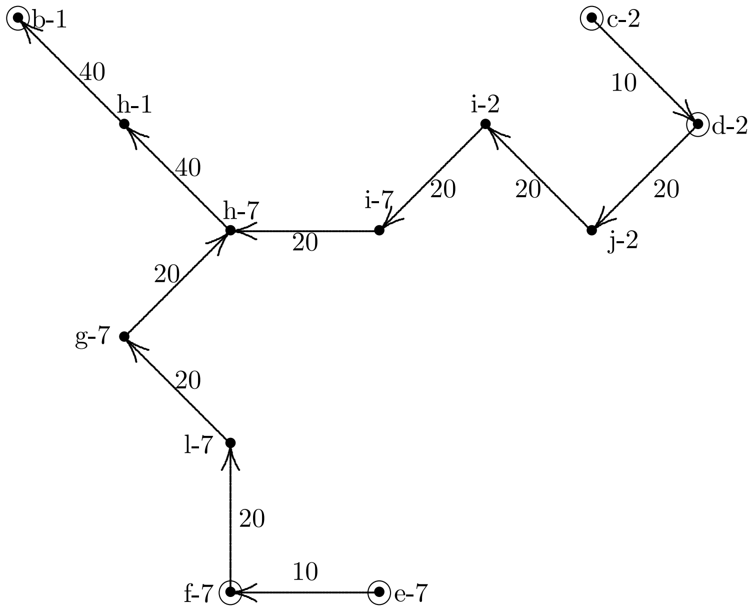

Figure 20. Flows and crossconnect solution for commodity b.

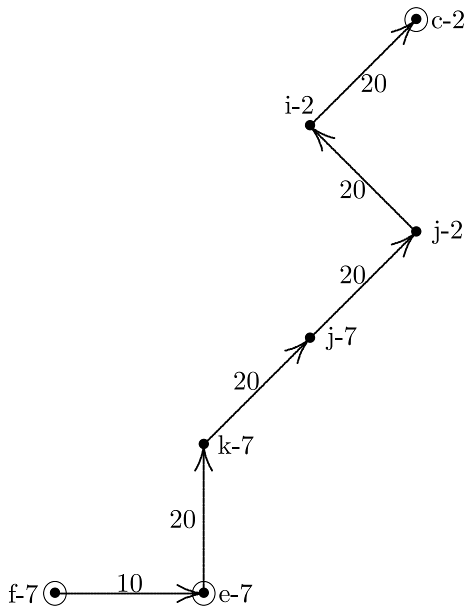

Figure 21. Flows and crossconnect solution for commodity c.

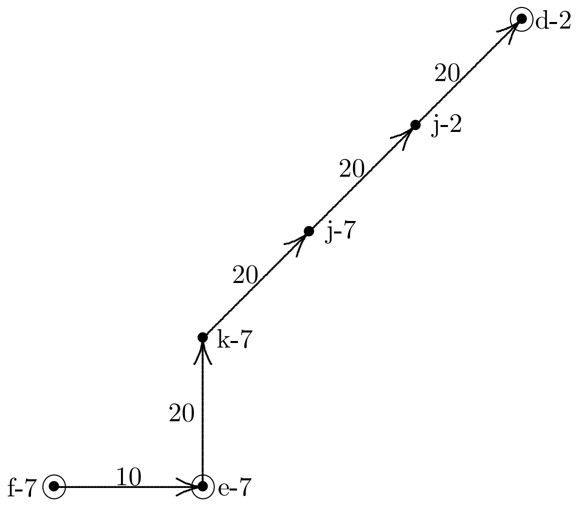

Figure 22. Flows and crossconnect solution for commodity d.

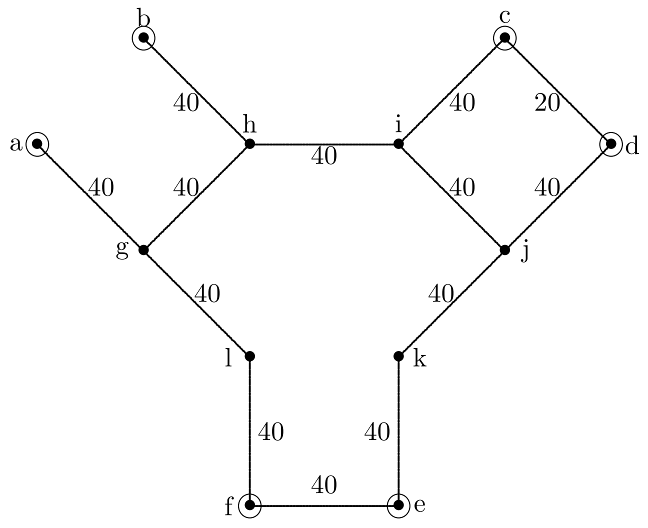

FIGURE 23. Link usage on small example (in DS3's)