

The Path Restoration Version of the Spare Capacity Allocation Problem with Modularity Restrictions: Models, Algorithms, and an Empirical Analysis

Jeffery L. Kennington • Mark W. Lewis

Department of Computer Science and Engineering Southern Methodist University,

Dallas, Texas 75275-0122, USA

jlk@seas.smu.edu • mlewis@seas.smu.edu

This investigation presents a strategy to construct a compact mathematical model of the path-restoration version of the spare capacity allocation problem. The strategy uses a node-arc formulation and combines constraints whenever multiple working paths affected by an edge failure have identical origins or destinations. Another unique feature of this model is the inclusion of modularity restrictions corresponding to the discrete capacities of the equipment used in telecommunication networks.

The new model can be solved using a classical branch-and-bound algorithm with a linear-programming relaxation. A preprocessing module is developed, which generates a set of cuts that strengthens this linear programming relaxation. The overhead associated with the cuts is offset by the improved bounds produced. A new branch-and-bound algorithm is developed that exploits the modularity restrictions. In an extensive empirical analysis, a software implementation of this algorithm was found to be substantially faster than CPLEX 6.5.3. For a test suite of 50 problems, each having 50 nodes and 200 demands from a uniform distribution with a small variance, our new software obtained solutions guaranteed to be within 4% of optimality in five minutes of CPU time on a DEC AlphaStation.

(Communications; Networks-Graphs; Programming Integer, Applications)

1. Introduction

The increased use of high bandwidth by the telecommunications industry has resulted in network-design problems that are amenable to operations-research techniques. One such problem is finding the minimum amount of spare capacity (bandwidth) to be allocated throughout a mesh network so that the network can survive the failure of an arc (variously referred to in the literature as a *link*, a *span*, or an *edge between nodes*). Since spare capacity is unused except during a failure, significant cost savings are achieved by minimizing spare capacity. This paper is concerned with mesh networks, which require the least amount of dedicated spare capacity.

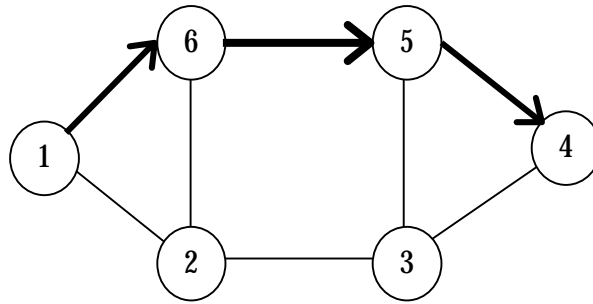
There are currently two approaches to modeling and solving the problem of assigning spare capacity and deriving restoration routes: these are *link restoration* and *path restoration*. Link restoration finds alternate paths between the nodes of the failed link. The optimal link-restoration solution may involve circuitous restoration routes, as illustrated in Figure 1. Path restoration avoids this problem by rerouting flow between the source and destination demand pairs affected by an arc failure. Path restoration yields a total spare capacity that is less than or equal to that obtained by link restoration.

The final implementation of any spare capacity solution requires values that are compatible with available technology. *Modularity* refers to the fact that the electrical-to-optical interface between the electrical signals and the fiber optic cable uses discrete capacities only, as illustrated in Figure 2. The upper bandwidth limits of circuit cards (number of multiplexed DS-3 circuits) are currently implemented in discrete steps of 3, 12, 48, and 192 circuits, known as OC-3, OC-12, OC-48, and OC-192. The number of slots available for these circuit cards is also limited. A solution that calls for implementing 50 spare circuits, when only OC-12 and OC-48 cards and two slots are available, would require the allocation of 60 circuits (one OC-48 and one OC-12, as illustrated in Figure 2). Thus, unless a solution happens to have exact modular values, the modularity requirements tend to add extra spare capacity to a network.

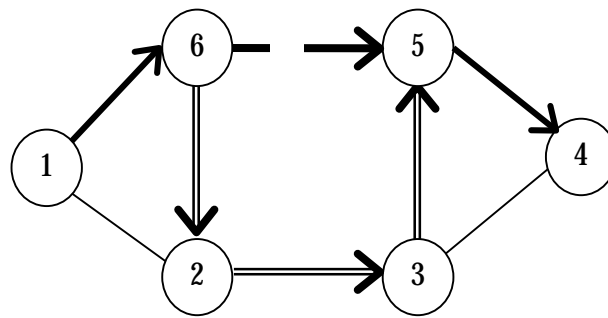
1.1 Mathematical Description of the Problem

Algorithms found in the literature use the arc-path (a.k.a. arc-chain) formulation for describing a network and network flows. An exact arc-path representation (Assad 1978, Kennington 1978) uses an arc-path incidence matrix that requires the enumeration of all paths between all origin-destination pairs in the network. This matrix and the corresponding integer program may be prohibitively large. An alternative formulation presented by Kennington and Whitley (1999) uses a node-arc incidence matrix to describe flow in the network. This description of the problem is based on conservation of flow into and out of a node. The restoration paths are constructed from flows on the individual arcs after a solution has been found. The node-arc formulation allows exact solutions or, for difficult problems, solutions that are within a known percentage of true optimality.

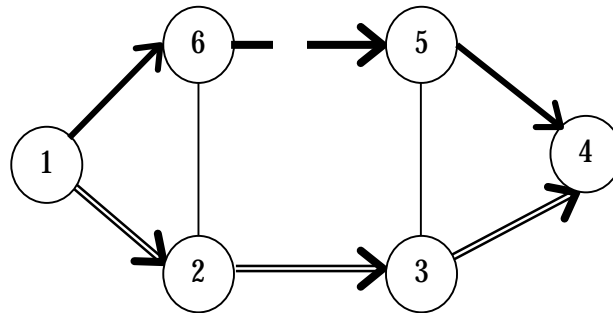
Let $[N,E]$ be a network with node set $N = \{1, \dots, \bar{n}\}$ and arc set $E = \{e_1, \dots, e_m\}$ of ordered pairs of nodes. That is, $e_m = (i, j)$ with $i \neq j$ and $i, j \in N$. Let A be a node-arc incidence matrix corresponding to $[N,E]$. Let d_{ij} denote the demand for circuits linking nodes i and j . We assume that the demands are defined only when $i < j$. A *path* in $[N,E]$ is defined as a sequence $\{i_1, e_{m_1}, i_2, e_{m_2}, \dots, i_p, e_{m_p}, i_{p+1}\}$ where arc $e_{m_q} \in \{(i_q, i_{q+1}), (i_{q+1}, i_q)\}$ and each arc and each node are distinct. For a path $P^k = \{i_1, e_{m_1}, \dots, i_p, e_{m_p}, i_{p+1}\}$ we say that the *origin* for P^k is node i_1 and the *destination* for P^k is node i_{p+1} . Let $P = \{P^1, \dots, P^{\bar{k}}\}$ denote the set of all paths in $[N,E]$. Let f_k, o_k and d_k denote the flow, origin, and destination for path P^k , respectively. Hence, for each demand $d_j > 0$, there will be a set of paths k_1, \dots, k_1 such that $o_{k_1} = o_{k_2} = \dots = o_{k_j} = i, d_{k_1} = d_{k_2} = \dots = d_{k_j} = j$ and $f_{k_1} + f_{k_2} + \dots + f_{k_j} = d_j$. If arc e fails, then service will be interrupted on all paths that contain e . For each such path P^k , we seek to reroute the f_k circuits from o_k to d_k using $[N, E \setminus \{e\}]$.



a. Network with Working Path 1-6-5-4



b. Link Restoration When (6, 5) Fails Yields the Restoration Path 1-6-2-3-5-4



c. Path Restoration When (6, 5) Fails Yields the Restoration Path 1-2-3-4

Figure 1: Sample Network Illustrating Link Restoration Versus Path Restoration

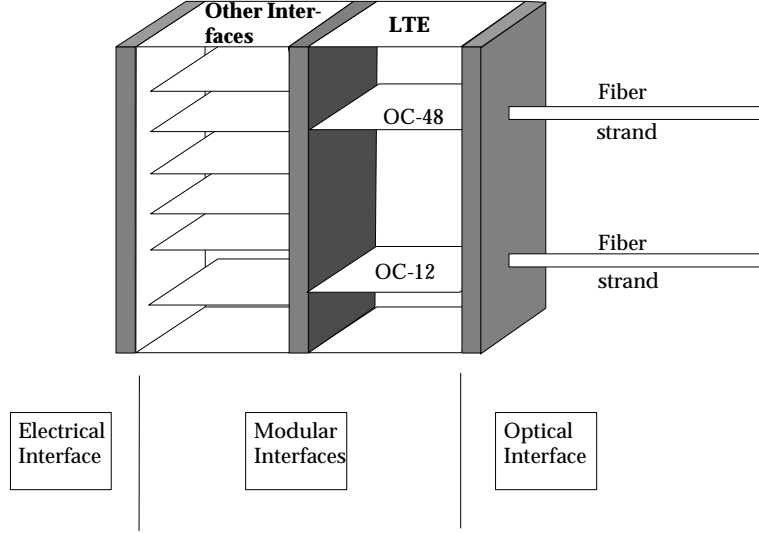


Figure 2: Example of Modularity at a Node

In a telecommunications network, the flow in arc $e = (i, j)$ can generally be in either direction, i.e. from i to j or from j to i . For arc $e = (i, j)$ we say that flow is in the *normal* direction when flow is from i to j and is in the *reverse* direction when flow is from j to i . The *working paths*, i.e. the paths used to meet demand in the network, and their flow, are known and compose the set $V = \{k: P^k \text{ is a working path}\}$. Let $I^e = \{k: e \in P^k, P^k \in V\}$ denote the index set of working paths that contain arc e . Suppose arc e which is used in working path P^k , $k \in I^e$, fails. Let x_m^{ek} denote the flow (number of circuits) in the normal direction on arc m used to restore the f_k circuits affected by this failure, with corresponding vector \mathbf{x}^{ek} . Let z_m^{ek} denote the flow in the reverse direction on arc m used to restore the f_k circuits affected by this failure, with corresponding vector \mathbf{z}^{ek} . Let

$$b_i^{ek} = \begin{cases} f_k & \text{if } i = o_k \\ -f_k & \text{if } i = d_k \\ 0 & \text{otherwise} \end{cases}$$

where e is the failed edge, k is the path index affected by the failure of edge e and i is the node number. The corresponding vector is \mathbf{b}^{ek} . The restoration flows for path P^k must be in the set

$\nabla^{ek} = \{(\mathbf{x}^{ek}, \mathbf{z}^{ek}) : A\mathbf{x}^{ek} - A\mathbf{z}^{ek} = \mathbf{b}^{ek}, \mathbf{x}^{ek} \geq \mathbf{0}, \mathbf{z}^{ek} \geq \mathbf{0}\}$, where $\mathbf{0}$ is a vector of zeroes and A is a node-arc incidence matrix for the directed version of $[N, E]$ (see Kennington and Helgason 1980). Letting an ek combination represent a commodity, the equations found in the definition of ∇^{ek} are the flow-conservation equations found in the well-known multicommodity flow problem (see Kennington 1978, Kennington and Helgason 1980).

Restoration for a failure of arc e requires that we find a set of restoration flows $(\mathbf{x}^{ek}, \mathbf{z}^{ek}) \in \nabla^{ek}$ for each $k \in I^e$. Let y_m denote the spare capacity allocated to arc $e_m \in E$ with corresponding vector \mathbf{y} . The spare capacity needed to protect against a failure for arc e will be

$$\mathbf{y} \geq \sum_{k \in I^e} (\mathbf{x}^{ek} + \mathbf{z}^{ek}).$$

However, since spare capacity is idle most of the time and expensive, we seek to minimize the sum of spare capacity (i.e. minimize $\mathbf{1}'\mathbf{y}$). This objective function is being used as a surrogate for the true cost function, which can be difficult to determine. However, the algorithm and software used in this investigation can be easily modified to solve problems having $\mathbf{c}'\mathbf{y}$ as the objective function. Therefore, the mathematical model for the *path-restoration version of the spare capacity allocation problem* may be stated as follows:

$$\begin{aligned} & \text{minimize } \mathbf{1}'\mathbf{y} \\ & \text{s.t.} \\ & A\mathbf{x}^{ek} - A\mathbf{z}^{ek} = \mathbf{b}^{ek} \quad \text{for all } e \in E, \text{ all } k \in I^e \\ & \sum_{k \in I^e} (\mathbf{x}^{ek} + \mathbf{z}^{ek}) \leq \mathbf{y} \quad \text{for all } e \in E \quad (1) \\ & x_e^{ek} = 0, z_e^{ek} = 0 \quad \text{for all } e \in E, \text{ all } k \in I^e \\ & \mathbf{x}^{ek}, \mathbf{z}^{ek} \geq \mathbf{0} \quad \text{for all } e \in E, \text{ all } k \in I^e \\ & \mathbf{y} \geq \mathbf{0} \text{ and integer} \end{aligned}$$

where $\mathbf{1}'$ is a row vector of ones.

The above model can be reduced in size by combining the constraints represented by ∇^{ek} whenever at least two paths have the same origin or the same destination for a given e . This same strategy is commonly used for multicommodity network formulations. The choice of whether to combine constraints according to origin or destination depends on whether the

number of paths in I^e with the same origin is less than the number of paths in I^e with the same destination. In either case the formulation takes the same form as (1).

Recall that an edge is a fiber optic bundle composed of a number of fiber strands with circuit cards at each end of the strand, which determines the bandwidth for that strand. The bandwidth for the edge is the sum of the bandwidth's in the strand that compose the edge. Suppose bandwidth for a fiber strand on any edge e can only be implemented in fixed sizes from the set $\Pi^e = \{0 = M_0, M_1, \dots, M_n\}$. Currently, $\Pi^e = \{0, 3, 12, 48, 192\}$ for most e . Some edges may not have nodes that can accommodate an OC-192 at both ends. Alternatively, it is possible that a designer will deem that implementing small OC-3 links is not cost effective for a certain link. Let Ω_e^k denote the maximum number of circuit cards of size M_k that can be implemented on arc e . The number of circuit cards will typically be limited by the number of fiber strands available on an arc or the number of circuit card slots at a node. Let Ψ_e denote the minimum of the maximum number of circuit cards allowed at either end of edge e . Thus, Ψ_e is the upper limit on the total number of circuit cards that can be used to implement spare capacity for an edge. With these definitions, the set of possible modular spare capacities for an edge e is:

$$\Delta^e = \{M: M = \sum_k (M_k w_k), M_k \in \Pi^e, w_k \in \{0, 1, \dots, \Omega_e^k\}, w_k \leq \Psi_e\}.$$

Let the vector whose elements are lower bounds l_e on the spare capacity of an edge e be $\mathbf{l} = (l_1, l_2, \dots, l_{|E|})$ and let $\mathbf{u} = (u_1, u_2, \dots, u_{|E|})$ be the vector consisting of upper bounds on the spare capacities. In general, the lower bounds are all zero and the upper bounds are determined by the modular values and number of slots available (Π^e and Ψ_e). These two vectors are used to bound the set of possible modular spare capacities. Given an edge e let the set of spare capacity vectors \mathbf{y} that have modular components between lower and upper bounds \mathbf{l} and \mathbf{u} be denoted by

$$\mathcal{S}_e(\mathbf{l}, \mathbf{u}) = \{ \mathbf{y} : l_e \leq y_e \leq u_e, y_e \in \Delta^e \}.$$

Denote the mixed integer program for minimizing modular spare capacity as:

$$P(\mathbf{l}, \mathbf{u}) = \{ \text{minimize } \mathbf{1}' \mathbf{y} \quad (2)$$

s.t.

$$A\mathbf{x}^{en} - A\mathbf{z}^{en} = \boldsymbol{\beta}^{en}, \quad \text{for all } e \in E, \text{ all } n \in \Theta^e \quad (3)$$

$$\sum_{n \in \Theta^e} (\mathbf{x}^{en} + \mathbf{z}^{en}) \leq \mathbf{y}, \quad \text{for all } e \in E \quad (4)$$

$$\mathbf{y} \in S_e(\mathbf{l}, \mathbf{u}), \quad \text{for all } e \in E \quad (5)$$

$$x_e^{en} = 0, z_e^{en} = 0, \quad \text{for all } e \in E, \text{ all } n \in \Theta^e \quad (6)$$

$$\mathbf{x}^{en}, \mathbf{z}^{en} \geq \mathbf{0}, \quad \text{for all } e \in E, \text{ all } n \in \Theta^e \quad (7)$$

}

where the set Θ^e results from combining either common sources or common destinations. This model is called the *path-restoration version of the spare capacity allocation problem with modularity restrictions*.

1.2 Literature Survey

The spare capacity allocation problem is related to the classical multicommodity network flow problem and is therefore a difficult problem to solve. The literature on multicommodity network flow problems is vast, and surveys may be found in Assad (1978) and Kennington (1978). One of the earlier papers dealing with the specific problem of spare capacity allocation in a mesh network is Chujo et al. (1991). They propose a network flooding algorithm that assumes that the first message received by the chooser node is the alternate path carrying the maximum bandwidth. They also define a metric called the restoration ratio that equals the number of restorable paths (if link i fails) divided by the number of failed paths (affected by the failure of link i). They attack the spare capacity allocation problem by using their improved flooding algorithm to get an initial solution, decreasing the spare capacity on a path, and recalculating the restoration ratio. This procedure is continued until no improvement is made to the restoration ratio. Although subtitled an ‘‘Optimum Spare-Capacity Assignment Algorithm’’, the proof of this is not presented.

Grover et al. (1991), proposed a three-phase heuristic approach. The first phase is *forward synthesis*, basically a greedy search looking for maximum restorability. They contrast this approach with optimization-based formulations that minimize cost subject to a large number of restorability constraints. The second phase is *design tightening* which attempts to maintain the achieved restorability while reducing redundancy on the spare link assignments. The third phase is called *design erosion for sensitivity analysis*, in which the effects of different types of failures and reprovisionings on the restorability of the network are analyzed. They conclude that modeling with modularity constraints using increments of 1 yields minimum redundancy required for restorability, while using modularity increments of 6, 12, 24, etc. will automatically increase the extra spare capacity in the network. After analyzing the effects on restorability of random single link failures, they discuss the use of the spare capacity in growing the network as demand increases and the effect this has on restorability. They call this conversion *erosion*. The running time of the algorithm varied from about five minutes for a 20 node network to ten hours for a 100 node network. They emphasize that this time is for a complete design and that updates can make use of previous work and should run in a fraction of the time, so iterative design changes are run more quickly.

The large size of integer-programming formulations for this problem has led to decomposition techniques. An interesting network-decomposition technique was proposed by Herzberg (1993). He proposed an algorithm that examines the network topology for certain subnetwork types that would automatically determine the spare capacity requirements for that subnetwork. He demonstrated his approach on the AT&T 23 edge test problem described by Grover et al. (1991).

The inclusion of hop count limits in the capacity allocation algorithm is discussed by Herzberg and Bye (1994) and later by Iraschako et al. (1998). Hop count limits reduce the total number of paths available in the network and thus reduce the number of columns in an arc-path integer-programming formulation. Network design engineers also prefer short restoration paths as opposed to long circuitous paths. The disadvantage is that more total spare capacity may be needed than required for the unrestricted case.

Herzberg and Bye (1994) present a new algorithm for minimizing total spare capacity that can include bandwidth modularity. Their algorithm is a two-stage approach, first solving an LP using an arc-path formulation where the number of path variables is considerably

reduced by allowing only those paths whose length is less than the hop limit. This solution is then rounded up to yield integer answers and a design-tightening approach is iteratively applied to all links by reducing their capacity and rechecking for restoration feasibility. Modularity constraints are easily incorporated in the rounding phase.

Murakami and Kim (1995) present the idea that solving the working and spare capacity allocation problems simultaneously will provide noticeable benefits, especially for large sparse networks. They report savings of 7% to 10% over solving the problems sequentially. Iraschko et al. (1998) reported gains (decreases in total capacity) of 4% to 27% from joint optimization, but noted that the working capacity often already exists (such as when upgrading or adding to a network). Abou-Sayed et al. (1997) also investigated the benefits of joint optimization and found only slight (~5%) improvements in the total capacity. They conclude that the added complexity of the joint optimization model limits the size of the problems that can be solved, so solving two easier problems sequentially yields relatively good results.

Most integer-programming formulations are hampered by the explosive growth in the size of the model, particularly the number of variables (columns). Vanderbeck and Wolsey (1996) presented a general branch-and-bound algorithm using column generation for solving large LP relaxations at nodes of the branch-and-bound tree. They claim that no satisfactory general branch-and-bound scheme is available for large IPs where the coefficient matrix is not composed of only zeros and ones and where the right-hand-side can be other than 1. In a more recent technical report, Barnhart et al. (1992) discuss formulations of integer programs with a huge number of variables and present their solution via branch-and-price using column-generation methods to solve the LP relaxations at the nodes. Their paper gives details on the branch-and-price approach, which is similar to branch-and-cut except the former focuses on column generation and the latter on row generation. They also emphasize the similarities between column generation and Lagrangian duality, stating that column generation has been out of favor because of the complexity in solving the subproblems needed in column generation. They state that this has been mitigated by improvements in LP solving packages such as CPLEX and that there will be a resurgence in the use of column generation for solving large, difficult problems.

Kennington and Whitley (1999) propose a decomposition technique for quickly solving the spare capacity allocation problem for link-based failures on large (70 node) network

problems. Their method utilizes a node-arc formulation and an algorithm based on improvements to Bender's decomposition. Their software obtains integer solutions approximately eight times faster than CPLEX can solve the corresponding linear-programming relaxation. Their algorithm quickly generates additional constraints for the master problem via a new network cut procedure. They solve a large selection of problems to within 2% of optimality. Their paper demonstrates the validity of decomposition techniques and the value of including cuts in the formulation. Their investigation differs from the present investigation in two aspects: (i) it is for link restoration and the present investigation is for path restoration, and (ii) it does not account for modularity.

Iraschko et al. (1998) present a study comparing their path-restoration formulation to their link-restoration formulation. They investigate six network cases involving combinations of path and link restoration and jointly optimized working and spare capacity. They use an arc-path IP formulation where the paths to be included in the formulation are hop-limited and supplemented by the k-shortest paths (see MacGregor and Grover 1994). Five network topologies are analyzed and it is concluded that mesh-restorable networks using path restoration are the most capacity efficient. The work by Iraschko et al. (1998) differs from our investigation in several respects. They use an arc-path formulation using only a subset of possible paths while we use a node-arc formulation. The latter model yields valid upper and lower bounds while the former produces only an upper bound. In addition, their study does not account for modularity. They only solve one problem having at least 50 nodes and it requires many hours of CPU time depending upon the number of eligible routes used in the model. Our study presents computational results for a test set of fifty problems of this size that are each solved in only a few minutes of CPU time.

All the studies cited above treat the design problem as a deterministic problem. There are also related models that explicitly consider the stochastic nature of such problems. Sen et al. (1994) use stochastic linear programming to install working capacity needed to handle stochastic demands. Bai et al. (1997) use robust optimization to determine the best demand routing in response to random node failures. Both use an arc-path formulation while our investigation uses a node-arc formulation.

2. Modular Branch and Bound

A branch-and-bound algorithm has been developed for $P(\mathbf{l}, \mathbf{u})$. We use the continuous relaxation $P_r(\mathbf{l}, \mathbf{u}) = \{\text{minimize } \mathbf{1}'\mathbf{y} : (3), (4), (6), (7), \mathbf{l} \leq \mathbf{y} \leq \mathbf{u}\}$ and nodes in the branch-and-bound tree are composed of solved problems of the form $P_r(\mathbf{l}, \mathbf{u})$. If a node produces an incumbent, then this node is fathomed; otherwise, the spare capacity is rounded up to the nearest modular value in an attempt to obtain a new incumbent. Branching involves changing the bounds of some edge to an appropriate modular value. Suppose that for some edge e the available modular values are $\{0, 3, 12, 24\}$ and the capacity (y_e) obtained by solving the LP is 10. Then two nodes are created, one with $y_e \leq 3$ and the other with $y_e \geq 12$. The hope is that this specialized branching strategy will quickly lead to a modular solution.

Our candidate-problem selection procedure uses a mixture of a depth-first search and a best-candidate problem strategy. The input parameter D is used to determine the number of nodes to investigate in a depth-first search before returning to the best-bound strategy. The minimum modularity t , which is the smallest nonzero value in Δ^e taken over all $e \in E$, is also used for fathoming. For our test cases, t is usually 3 corresponding to an OC-3. Since the spare capacity for link e must be an element of Δ^e , t can be used to fathom solutions that are not at least t better than the current incumbent.

Our modular branch-and-bound code uses the CPLEX callable library to solve the LP relaxations. During the depth-first search, we use a branch-right rule corresponding to increasing some links capacity. The idea is that this rule will quickly lead to a new incumbent. Since the LP relaxations of both the left and right nodes are similar, both are solved before branching right. Hence, the candidate list is composed of problems in which the LP relaxation has been solved. A detailed description of the procedure may be found in the appendix.

3. Cut Generation

The performance of the software implementation of the branch-and-bound algorithm is improved when constraints generated by network cuts are included in the IP formulation of the node-arc model. The constraints are generated by describing the flow requirements through the cut, then making the total flow an allowable modularity value. Which cuts to generate and which constraints to include from these cuts is an open question. After some

preliminary investigation, a system was constructed that produced all minimum cuts between all node pairs (using edge capacities of one).

A cut is a partition of the nodes from the network $[N, E]$ into two parts N_1 and $N_2 = N - N_1$. The arcs in the cut are defined to have one endpoint in N_1 and the other in N_2 . Let \mathbb{A} define the set of arcs in the cut. Let $e \in \mathbb{A}$ and K_e denote the set of indices k such that o_k is in one set of the cut, d_k is in the other, and e is contained in path P^k . The arcs in $\mathbb{A} \setminus \{e\}$ must have at least $\min_{k \in K_e} f_k$ spare capacity. If $\min_{k \in K_e} f_k$ is not an allowable modular value, then it can be increased to the next allowable modular value. Following this strategy, $|\mathbb{A}|$ constraints can be appended to the node-arc formulation.

4. Empirical Analysis

Ten small test problems (A through J) having from 13 to 16 nodes and from 19 to 24 arcs, were randomly generated. Each problem had randomly generated demands of 1 to 10 DS-3s between every demand pair. A shortest-path algorithm was used to obtain the working paths between the demand pairs. Every node in the network was allocated five slots for provisioning any mix of OC-12, OC-48, and OC-192.

By using integer variables for each modularity value, constraints of type (5) can be modeled using linear inequalities so that CPLEX can be applied to a version of $P(\mathbf{l}, \mathbf{u})$. The model sizes for the input to CPLEX are slightly larger than those required for our software, which treats (5) implicitly. For problem A, the number of additional variables is $3 \times 19 = 57$, corresponding to the 3 modular values and the 19 edges. Each of these integer variables has bounds of 0 and 5 corresponding to the number of slots. The number of additional constraints is twice the number of edges.

The modular branch-and-bound algorithm has been implemented in software using ANSI C. The CPLEX-callable library is used to obtain solutions to $P_R(\mathbf{l}, \mathbf{u})$ and the complete package is called MODCAP, in reference to its ability to solve modular capacity problems. These ten problems were solved using CPLEX 6.5.3 and MODCAP and the solution times appear in Table 1. Except for problem F, MODCAP solved the problems much faster than CPLEX. Summing the time it takes to solve these ten problems to optimality shows an

approximately four-fold improvement of MODCAP over CPLEX. The average decrease in time by using MODCAP is a factor of 56. Removing extreme cases B and F yields average times of 847 seconds for CPLEX and 72 seconds for MODCAP and an average speedup factor of 12 when using MODCAP.

The effect of including constraints generated by cuts is also illustrated in Table 1. The same cut constraints were added to both the CPLEX and MODCAP formulations. These cuts were generated using a max-flow min-cut algorithm between all node pairs. Improvements in speed of about 42 times for MODCAP and 38 times for CPLEX were obtained. Comparing the total times to solve the ten problems, using cuts with MODCAP resulted in about a four-fold improvement over using cuts with CPLEX. The improvement over the original CPLEX formulation without cuts is a factor of about 171.

Table 1: Comparison of CPLEX and MODCAP on Small Test Problems

Problem ID	Base Model (2) - (7)		Number of Additional Constraints	Base Model Plus Cuts	
	CPLEX Time (sec)	MODCAP Time (sec)		CPLEX Time (sec)	MODCAP Time (sec)
A	960	25	164	0.6	0.3
B	738	2	152	0.4	0.2
C	741	66	164	34.7	2.3
D	649	19	197	132.6	2.5
E	1089	53	197	0.6	0.3
F	1135	1534	224	42.2	36.9
G	1026	228	198	5.3	2.1
H	639	21	173	4.1	1.1
I	840	144	278	3.3	2.0
J	834	17	278	5.3	2.9
Totals	8651	2109	-	229.1	50.6
Scaled Time	171.0	41.7	-	4.5	1.0

Iraschko et al. (1998) solved several problems using an arc-path formulation. Problem 1 has 10 nodes, 22 edges, and 45 O-D pairs with total demand of 90. Case 2 of this problem is the path-restoration version and has 7549 columns in the integer-programming formulation. The node-arc formulation for input to MODCAP uses only 1301 columns. MODCAP obtained optimal solutions using a variety of modularities in under two seconds average, compared to the 24 minutes reported in the above reference. These results are summarized in Table 2. Note that the mix of modularities available affects the total spare capacity allocation.

Table 2: Optimal Solutions for Network 1, Case 2 (Iraschko et al. 1998)

Mods used	Slots	# BAB Nodes	Obj. Val	Time (sec)
12	1	80	120	2
12	3	72	120	1
12	5	70	120	1
48	1	72	480	2
48	3	80	480	1
48	5	80	480	2
192	1	62	1920	1
192	3	78	1920	2
192	5	78	1920	1
12 48	1	84	120	2
12 48	3	86	120	2
12 48	5	86	120	2
12 192	1	82	120	3
12 192	3	78	120	1
12 192	5	98	120	3
48 192	1	80	480	1
48 192	3	78	480	2
48 192	5	82	480	2
12 48 192	1	82	120	2
12 48 192	3	78	120	2
12 48 192	5	98	120	2

Using combinations of OC-n, $n \in \{3, 12, 48\}$ and with 5 slots available, 50 randomly generated 50-node networks with average node degree of 2.5 were solved by MODCAP. The

networks had from 61 to 87 links and each network had 200 demand pairs with a demand range from 1 to 3. The results generated using a five-minute termination criteria are tabulated in Table 3. The IPs generated were large, generally having over 17,000 constraints and over 26,000 variables. MODCAP obtained solutions for all 50 problems with an average optimality gap of 1.4%. These gaps are computed using the LP lower bounds and the true deviation from optimality is less than or equal to the gaps reported.

Table 3: Comparison of CPLEX and MODCAP on 50 Node Networks Using a 5 Minute Time Limit

Problem Sets	CPLEX		MODCAP	
	Max Number of Variables	Max Optimality Gap	Max Number of Variables	Max Optimality Gap
1 - 10	31,288	10.3%	31,048	3.2%
11 - 20	29,588	8.4%	29,327	3.7%
21 - 30	31,308	10.0%	31,047	2.6%
31 - 40	29,492	18.0%	29,261	2.3%
41 - 50	30,342	22.2%	30,108	1.7%
Max	31,308	22.2%	31,048	3.7%

For comparison, these same networks were also formulated using integer variables for each modular value, then solved using the CPLEX mixed-integer branch-and-bound solver. The optimality gaps found range from 0.3% to 22.2% versus MODCAP's range from 0.6% to 3.7%. The average optimality gap for CPLEX is 5.9% versus 1.4% for MODCAP.

5. Summary and Conclusions

Traditionally, models for finding spare capacity allocation in a network use an arc-path formulation. This is a natural approach that provides the restoration paths as part of the

solution. However, an exact solution to the problem requires complete enumeration of all possible restoration paths in the network or a complex column-generation approach. Real-world problems are typically large (> 30 nodes) and difficult to solve. Therefore, at the cost of a guaranteed optimal solution, a subset of all possible paths is used to make the problem manageable. The node-arc formulation used in this investigation implicitly includes all possible restoration paths.

A unique feature of this investigation is the incorporation of modularity into the formulation. The permissible modular values are usually a subset of the positive integers and hence, the feasible region is smaller than the pure integer version. This smaller set of feasible solutions has been exploited in the development of special techniques to solve the path-restoration version of the spare capacity allocation problem. An additional technique that provided good results was the inclusion of modular cuts into the formulation. Including all constraints generated from all the cuts found by examining all node pairs provided an improvement in speed to optimality by a factor of about 42 for the set of small problems tested.

Solving a large modular spare capacity path-restoration MIP is a very complex problem. Without including cuts, MODCAP was about four times faster than CPLEX 6.5.3 for the ten small test problems solved to optimality. When the modular cuts are included in formulations for both the CPLEX and the MODCAP models, MODCAP is again about four times faster than CPLEX. The improvement from the original CPLEX formulation without cuts was a factor of about 171 when using MODCAP with modular cuts. MODCAP solved a small ten-node problem seen in the literature in about 2 seconds, versus 24 minutes. On 50 larger networks, no problems were solved to optimality, but using a five-minute termination limit, MODCAP achieved optimality gaps that were about four times better than those found by CPLEX. The largest optimality gap using MODCAP was about six times smaller than that obtained using CPLEX. Note that the problems in our test suite are fairly similar and have a small variance in the demand pattern. Although this characteristic has not been exploited in the development of our software, demand variance could affect the performance of the specialized software. Computational issues related to problem characteristics have not been investigated in this study. The issue of obtaining restoration paths corresponding to a particular spare capacity allocation may be found in Kennington and Lewis (1999).

Appendix: Modular Branch and Bound Algorithm

Begin Modular_Branch_and_Bound

```

let  $(\tilde{\mathbf{y}}, \mathbf{x}, \mathbf{z})$  be a solution to  $P_R(\mathbf{l}, \mathbf{u})$ ;
if ( $\tilde{\mathbf{y}} \in S(\mathbf{l}, \mathbf{u})$ ) return ( $\tilde{\mathbf{y}}$ ); endif
if ( $P_R(\mathbf{l}, \mathbf{u})$  is infeasible) return (INFEASIBLE); endif
select  $e$  such that  $\tilde{y}_e \notin \Delta^e$ ;
 $V \leftarrow \mathbf{1}' \tilde{\mathbf{y}}$ ;  $CL \leftarrow \{[ P(\mathbf{l}, \mathbf{u}), V, e, \tilde{y}_e ]\}$ ;
 $V^* \leftarrow \min \{ V : [ P(\mathbf{l}, \mathbf{u}), V, C, \tilde{y}_e ] \in CL \}$ 
 $v^* \leftarrow \infty$ ;
while ( $CL \neq \emptyset$  and  $V^* \leq v^* - t$ )
  choose  $[ P(\mathbf{l}, \mathbf{u}), V, e, \tilde{y}_e ] \in CL \ni V = V^*$ ;
   $CL \leftarrow CL \setminus \{[ P(\mathbf{l}, \mathbf{u}), V, e, \tilde{y}_e ]\}$ ;  $d \leftarrow 0$ ;
  while ( $d \leq D$ )
     $\mathbf{u}' \leftarrow \mathbf{u}$ ;  $\mathbf{u}'_e \leftarrow U(\tilde{y}_e)$ ;  $\mathbf{y}' \leftarrow \tilde{\mathbf{y}}$ ;
    let  $(\tilde{\mathbf{y}}, \mathbf{x}, \mathbf{z})$  be a solution to  $P_R(\mathbf{l}, \mathbf{u}')$ ;  $W \leftarrow \mathbf{1}' \tilde{\mathbf{y}}$ ;
    if ( $W \leq v^* - t$  and  $P_R(\mathbf{l}, \mathbf{u}')$  is feasible)
      if ( $\tilde{\mathbf{y}} \in S(\mathbf{l}, \mathbf{u}')$ )  $v^* \leftarrow W$ ;  $\mathbf{y}^* \leftarrow \tilde{\mathbf{y}}$ ;
      else
        select  $f$  such that  $\tilde{y}_f \notin \Delta^e$ ;
         $CL \leftarrow CL \cup \{[ P(\mathbf{l}, \mathbf{u}), W, f, \tilde{y}_f ]\}$ ;
         $y_i^M \leftarrow L(\tilde{y}_i) \forall i \in E$ ;
        if ( $\mathbf{1}' \mathbf{y}^M \leq v^* - t$ )  $v^* \leftarrow \mathbf{1}' \mathbf{y}^M$ ;  $\mathbf{y}^* \leftarrow \mathbf{y}^M$ ; endif
      endif
    endif
     $l_e \leftarrow L(\mathbf{y}')$ ;
    let  $(\tilde{\mathbf{y}}, \mathbf{x}, \mathbf{z})$  be a solution to  $P_R(\mathbf{l}, \mathbf{u})$ ;  $W \leftarrow \mathbf{1}' \tilde{\mathbf{y}}$ ;
    if ( $W > v^* - t$  and  $P_R(\mathbf{l}, \mathbf{u})$  is feasible)
      if ( $\tilde{\mathbf{y}} \in S(\mathbf{l}, \mathbf{u})$ )  $v^* \leftarrow W$ ;  $\mathbf{y}^* \leftarrow \tilde{\mathbf{y}}$ ; break;
      else
         $y_i^M \leftarrow L(\tilde{y}_i) \forall i \in E$ ;
        if ( $\mathbf{1}' \mathbf{y}^M \leq v^* - t$ )  $v^* \leftarrow \mathbf{1}' \mathbf{y}^M$ ;  $\mathbf{y}^* \leftarrow \mathbf{y}^M$ ; endif
      endif
     $d \leftarrow d + 1$ ; select  $e$  such that  $\tilde{y}_e \notin \Delta^e$ ;
  else break;
  endif
  if ( $d > D$ )  $CL \leftarrow CL \cup \{[ P(\mathbf{l}, \mathbf{u}), W, e, \tilde{y}_e ]\}$ ;
  endif
endwhile
endwhile
return ( $\mathbf{y}^*$ );
End

```

Acknowledgment

This research was supported in part by the Texas Higher Education Coordinating Board Grant Number 003613-002.

References

- Abou-Sayed, M., J. Kennington, S. Nair. 1997. Joint working and spare capacity assignment in a link restorable mesh network. Technical Report 96-CSE-16, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX.
- Assad, A. 1978. Multicommodity network flows - a survey. *Networks* **8** 31-91.
- Bai, D., T. Carpenter, J. Mulvey. 1997. Making a case for robust optimization models. *Management Science* **43** 895 - 907.
- Barnhart, C., E. Johnson, G. Nemhauser, M. Savelsbergh, P. Vance. 1992. Branch-and-price: column generation for solving huge integer programs. Technical Report, Computer Optimization Center, Georgia Institute of Technology, Atlanta, GA..
- Chujo, T., H. Komine, K. Miyazaki, T. Ogura, T. Soejima. 1991. Distributed self-healing network and its optimum spare-capacity assignment algorithm. *Electronics and Communication in Japan* **74**, 1 - 9.
- Grover, W., T. Bilodeau, B. Venables. 1991. Near optimal synthesis of a mesh restorable network. *GLOBECOM 1991* 2007-2012.
- Herzberg, M. 1993. A decomposition approach to assign spare channels in self healing networks. *GLOBECOM 1993* 1601-1605.
- Herzberg M., S. Bye. 1994. An optimal spare capacity assignment model for survivable networks with hop limits. *GLOBECOM 1994* **3** 1601-1606.
- Iraschko, R., M. MacGregor, W. Grover. 1998. Optimal capacity placement for path restoration in STM or ATM mesh survivable networks. *IEEE/ACM Transactions on Networking* **6**, 325-336.
- Kennington, J. 1978. A survey of multicommodity network flows. *Operations Research* **26** 209 - 236.
- Kennington J., R. Helgason. 1980. *Algorithms for Network Programming* John Wiley and Sons, New York, NY.

- Kennington J., M. Lewis. 1999. Models and algorithms for creating restoration paths in survivable mesh networks. Technical Report 99-CSE-5, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX. www.seas.smu.edu/~jlk/publications/99-CSE-5.doc.
- Kennington J., J. Whitler. 1999. An efficient decomposition algorithm to optimize spare capacity in a telecommunications network. *INFORMS Journal on Computing* **11** 149 - 160.
- MacGregor M., W. Grover. 1994. Optimized k-shortest paths algorithm for facility restoration. *Software - Practice and Experience* **24** 823-834.
- Murakami K., H. Kim. 1995. Joint optimization of capacity and flow assignment for self-healing ATM networks. *Proceedings of 1995 IEEE International Conference on Communications* 216-220.
- Sen, S., R. Doverspike, S. Cosares. 1994. Network planning with random demand. *Telecommunication Systems* **3** 11 - 30.
- Vanderbeck F., L. Wolsey. 1996. An exact algorithm for IP column generation. *Operations Research Letters* **19** 151-159.

Accepted by Anant Balakrishnan; received January 1999; revised May 2000, November 2000; accepted December 2000.