INFORMS Journal on Computing
Vol. 20, No. 2, Spring 2008, pp. 243–254
ISSN 1091-9856 | EISSN 1526-5528 | 08 | 2002 | 0243

DOI 10.1287/ijoc.1070.0237

Solving the Capacitated Local Access Network Design Problem

F. Sibel Salman
College of Engineering, Koç University, Istanbul 34450, Turkey, ssalman@ku.edu.tr

R. Ravi, John N. Hooker
Tepper School of Business, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213
{ravi@cmu.edu, john@hooker.tepper.cmu.edu}

We propose an exact solution method for a routing and capacity installation problem in networks. Given an input graph, the problem is to route traffic from a set of source nodes to a sink node and to install transmission facilities on the edges of the graph to accommodate the flow at minimum cost. We give a branch-and-bound algorithm that solves relaxations obtained by approximating the noncontinuous cost function by its lower convex envelope. The approximations are refined by branching on the flow ranges on selected edges. Our computational experiments indicate that this method is effective in solving moderate-size problems and provides very good candidate solutions early in the branch-and-bound tree.

Key words: network design; routing flow; capacity installation; branch and bound
History: Accepted by Prakash Mirchandani, former Area Editor for Telecommunications and Electronic Commerce; received August 2001; revised July 2004, February 2007; accepted June 2007. Published online in *Articles in Advance* January 25, 2008.

1. Introduction

Network design problems involve connecting a given set of locations that have traffic requirements among them by installing transmission facilities at minimum cost. These problems have applications in many areas such as the design of telecommunication, transportation, computer, and energy supply networks, as well as production-distribution systems. We focus on network design in telecommunications, an area that has attracted much interest. A common practice in telecommunication network design is to partition the network into several *local access networks* (LANs) and a *backbone network*. In each LAN, traffic is collected at a node specified as the switching center. The backbone network is used to route the high-volume traffic between the switching centers. In the hierarchical design approach, the set of nodes are first partitioned into LANs, and for each partition a node is chosen to locate the switching center. Next, the design problem for each LAN is solved. The local access network design is a single-sink multisource routing and link-capacity assignment problem. The solution specifies a network to route traffic to the switching center and the transmission facilities, such as fiber-optic cables, to be installed on the edges of the network. Then, the backbone network is designed by solving a multicommodity flow and capacity-assignment problem. A similar problem arises when a network has to be expanded to accommodate increased traffic. In the *LAN design problem*, we are given a graph with lengths on the edges, a set of source nodes with specified traffic demands, and a sink node. A set of cable types, each with a given cost per unit length and capacity (bandwidth), is available for purchase and installation on the edges. However, the cost per unit capacity of a thick (high-bandwidth) cable is considerably cheaper than that of a thin (low-bandwidth) cable, so that buying capacity in bulk becomes economical when traffic accumulates to large volumes. The problem is to find a minimum-cost installation of cables on the edges such that all the demand originating at the sources can be routed simultaneously to the sink node with the provided capacity. The *backbone network design problem* has the same characteristics except that traffic demands are between pairs of source-sink nodes, hence the traffic of each source-sink pair is considered as a separate commodity.

The problem of installing capacity on the edges of a network at minimum cost by purchasing from a set of available transmission facilities has also been called the *network-loading problem*. The study of the polyhedral structure of these network-design problems where capacity is bought in modular quantities was initiated by Magnanti et al. (1993), who introduced the residual capacity inequalities. Magnanti and Mirchandani (1993) considered a single-commodity problem and introduced the cut-set inequalities. The cut-set inequalities, which provide a lower bound on

the number of cables to be installed across a cut, have been the basis for subsequent work. Magnanti et al. (1995) considered the multicommodity network-loading problem with two cable types. They strengthened a natural multicommodity flow formulation with integer cable installation variables with cut-set, 3-partition, and arc-residual-capacity inequalities. Bienstock and Günlük (1996) generalized the cut-set inequalities to flow-cut-set inequalities that force the capacity on both edge sets of a bipartition of the cut edges to be integral. Later, Atamtürk (2002) generalized these flow-cut-set inequalities by expressing the coefficients with a subadditive function for the single-commodity problem. Atamtürk and Rajan (2002) studied continuous and 0-1 knapsack sets with a single integer variable to derive enhanced inequalities for network design problems and tested their effectiveness in a branch-and-cut algorithm for the multicommodity network-loading problem.

Several authors have taken a different approach and studied the polyhedron obtained by projecting out the flow variables (Bienstock et al. 1998, Barahona 1996, Mirchandani 2000). The resulting capacity formulation contains exponentially many metric inequalities, thus only a subset of them are generated. Bienstock and Günlük (1995) generated partition inequalities, and Barahona (1996) generated cut-set and multicut inequalities for the single-cable problem. For the many-cable problem, Mirchandani (2000) characterized several classes of facet-defining inequalities by analyzing the special topology of the demand network. Multicommodity network optimization problems with discontinuous step-increasing cost functions, which include the capacitated network-loading problems, were studied by Gabrel and Knippel (1999), and earlier by Stoer and Dahl (1994) in the context of survivable network design. Gabrel and Knippel (1999) used the Benders procedure to solve the capacity formulation and generated partition inequalities iteratively. Later, Gabrel et al. (2003) tested a heuristic implementation of the exact Benders-type cutting-plane-generation method and compared it to greedy-type algorithms on randomly generated instances. The authors concluded that this Benders-type approach is promising.

A related version of the network-loading problem was studied by Gavish and Altinkemer (1990), and later by Amiri and Pirkul (1997), where the cost function included queueing-delay costs in addition to connection costs. In both of these papers, Lagrangian relaxation and a subgradient-optimization approach were taken. In addition, Gavish and Altinkemer (1990) utilized cut-set inequalities to strengthen the lower bounds.

Chopra et al. (1998) considered the network-loading problem with a single source-sink pair and two cable types. They showed that even this special case is NP-hard and obtained a stronger formulation by characterizing the structure of optimal solutions. However, there is no direct generalization of their results for the single-sink multiple-source case arising in LAN design.

While Berger et al. (2000) proposed heuristics based on tabu search, there has been a flurry of work on designing approximation algorithms with provable worst-case performance ratios for the LAN design problem in the theoretical computer science literature. Salman et al. (1997, 2000) gave the first approximation results for the LAN design problem, which they dubbed the "single-sink buy-at-bulk network design problem." Consequently, Awerbuch and Azar (1997) proved a logarithmic approximation ratio for a broader class of this problem, including the backbone design problem. Gupta et al. (2003) formulated a mixed-integer program (MIP) for a closely related problem and rounded its LP relaxation solution to show an integrality gap of the order of the number of distinct cable types in the problem. An improved LP rounding algorithm proving a constant factor integrality gap for this closely related problem was given by Talwar (2002). Meanwhile, an alternate randomized algorithm also giving constant-factor approximation was presented by Guha et al. (2001).

We propose an exact solution method for the LAN design problem in §§2 and 3. In this method, the optimal choice of cables to accommodate a given flow value is first calculated for all possible flow values. This provides the optimal cost as a function of flow for every edge, which is a monotonically nondecreasing step function. For solution purposes, we replace this function with the convex hull of its epigraph, which is equivalent to projecting the standard 0-1 formulation onto the continuous variables. We then branch by dividing the interval of possible flow values on a particular edge into subintervals. As branching proceeds, the convex hull relaxation is successively refined. This strategy has led to improvements in both solution time and modeling convenience in other studies. Ottosson et al. (2002) obtained an order-of-magnitude speedup, relative to 0-1 modeling and specially ordered sets of type 2, in a production planning problem. Réfalo (1999) showed how to use the relaxation in "tight cooperation" with domain reduction to obtain maximum benefit. Hooker (2000) provides a general discussion on this subject. Our computational tests on solving the LAN design problem, which we present in §4, indicate that the method is effective in solving moderate-size instances. The proposed method can also be applied to solve the multicommodity case arising in the backbone design problem. Subsequent to the work presented in this paper, Raghavan and Stanojevic (2005) provided an

alternative way of interpreting our method as a stylized branch-and-bound approach on a particular MIP formulation of the problem. We elaborate on their results after we present these formulations in §2, and our algorithm in §3. The implementation details and computational results are given in §4. We conclude in §5.

2. Formulations and Preliminaries

In this section, we contrast a natural MIP formulation with a flow formulation that has a discontinuous objective function. The latter motivates our solution method. We are given an underlying undirected graph $G = (V, E)$, $|V| = n$. The edges of G have lengths ℓ: $E \to \Re^+$. Without loss of generality, we assume that for every pair of nodes v, w, we can use the shortest-path distance between v and w as the length of the edge between these nodes. That is, we take the metric completion of the given graph and denote the length of edge e by l_e. Capacity can be installed on the edges of the graph by purchasing one or more copies from among a small set of cables, where each cable type $i \in \{1, \ldots, q\}$ has capacity u_i and cost c_i per unit length. The indexing of the cables is such that $u_1 \le u_2 \le \cdots \le u_q$ and $c_1 \le c_2 \le \cdots \le c_q$. Due to economies of scale, $c_1/u_1 > c_2/u_2 > \cdots > c_q/u_q$. Let S denote the index set of source nodes. For all $k \in S$, source node s_k has integer-valued demand d_k to be routed to the sink node t. Because demand originating at each source node shares the same destination, we can consider all of the traffic flow as a single commodity.

Let us first consider the natural formulation with integer cable installation variables on the edges, which was also used by Bienstock and Günlük (1996) and Magnanti et al. (1995) with multicommodity flows. As mentioned before, because all flow is headed to a single destination, we use a single-commodity formulation. For each edge $e = (i, j)$ of the input graph G, we have directed flow variables f_{ij} and f_{ji}, which denote the amount of flow through edge e in direction i to j, and j to i, respectively. The integer variables $y_{e,h}$ denote the number of cables of type h that need to be installed on edge e. In the IP formulation given below, constraints (1) are flow-balance constraints, and constraints (2) are capacity constraints.

$$\text{IP1:}\quad \min \sum_{e\in E} l_e \sum_{h=1}^{q} c_h y_{e,h}$$

$$\text{s.t.} \sum_{j\in V} f_{ij} - \sum_{j\in V} f_{ji} = \begin{cases} d_i & \text{if } i \in S \\ -\sum_{k\in S} d_k & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in V, \tag{1}$$

$$f_{ij} + f_{ji} \le \sum_{h=1}^{q} u_h y_{e,h} \quad \forall e = (i, j) \in E, \tag{2}$$

$$y, f \ge 0,$$

$$y \text{ integer.}$$

The LP relaxation of IP1 has a solution that uses only the cable type with the largest bandwidth, which is cable type q, because this provides the cheapest cost per capacity rate. Demand is routed through shortest source-sink paths. If flow on a path is small compared to the capacity of this high-bandwidth cable, only a small fraction of the cable is purchased. Therefore, this relaxation is known to give weak lower bounds.

Now let us consider the cable installation problem on a single edge. Let f_e indicate the total flow on edge $e = (i, j)$, that is, $f_e = f_{ij} + f_{ji}$. Then, the problem is to purchase cables at minimum cost (per unit length) to provide a total capacity of at least f_e. This is an *integer minimum knapsack problem* with formulation

$$\begin{aligned} C(f_e) = \min\ & \sum_{h=1}^{q} c_h y_{e,h} \\ \text{s.t.}\ & \sum_{h=1}^{q} u_h y_{e,h} \ge f_e, \\ & y_{e,h} \ge 0, \text{ integer.} \end{aligned} \tag{3}$$

This problem, which is common for all edges, can be solved in pseudo-polynomial time by a single run of a dynamic programming algorithm for all possible integer flow values, from zero to total demand. We calculate the optimal cable-cost function $C(f)$ for $f = 0, 1, 2, \ldots, \sum_{k\in S} d_k$ by the following recursion, starting with $C(0)$.

$$C(f) = \begin{cases} \min_{j=1,\ldots,q} \{c_j + C(f - u_j)\} & \text{if } f > u_q, \\ \min_{j=1,\ldots,k:\, u_{k-1} < f \le u_k} \{c_j + C(f - u_j)\} & \text{if } 0 < f \le u_q, \\ 0 & \text{if } f \le 0, \end{cases} \tag{4}$$

where $u_0 = 0$.

Proposition 1. *The function $C(f)$ for $0 \le f \le \sum_{k\in S} d_k$ has the following properties*:

(1) *It is a step function with finitely many points of discontinuity.*

(2) *It is a nondecreasing subadditive function.*

(3) *$C(f) \ge (c_q/u_q)f$ for all $0 \le f \le \sum_{k\in S} d_k$, and $C(f) = k(c_q)$ if $f = k(u_q)$ for some positive integer k.*

Proof. The first property follows from the recursion in (4). For the second property, it is easy to verify from (4) that $C(f_2) \ge C(f_1)$ for $f_2 \ge f_1$ as c_j take positive values and $C(f_1 + f_2) \le C(f_1) + C(f_2)$ for all f_2, f_1

due to economies of scale. The third property states that $C(f)$ lies above the line corresponding to the cheapest cable rate. Furthermore, $C(f)$ coincides with this line if $f = k(u_q)$ for some positive integer k, indicating that it is optimal to use k copies of cable q, the thickest one. □

The second and third properties imply that the optimal solution will have a tendency to aggregate flow to integer multiples of the cables with higher capacity. Because $C(f)$ can be generated with small computational effort, we consider using the function $C(f_e)$ for the cost of flow f_e on edge e in IP1, so that we can eliminate the integer variables and obtain the flow formulation FP:

$$\text{FP:} \quad \min \sum_{e\in E} C(f_e)\, l_e$$

$$\text{s.t.} \sum_{j\in V} f_{ij} - \sum_{j\in V} f_{ji} = \begin{cases} d_i & \text{if } i \in S \\ -\sum_{k\in S} d_k & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in V,$$

$$f_{ij} + f_{ji} = f_e \quad \forall e = (i,j) \in E,$$

$$f_{ij},\ f_{ji},\ f_e \geq 0 \quad \forall e = (i,j) \in E.$$

This formulation is equivalent to projecting IP1 into the space of the continuous flow variables. Now we have a convex solution set but the objective function is a step function with finitely many discontinuity points as implied by Proposition 1. We investigate two approaches to solve this problem. The first and more direct one is to represent the pieces of the step function by 0-1 variables and to solve the resulting MIP by branch and bound. In the second approach, we avoid introducing 0-1 variables, but instead give a stylized branch-and-bound algorithm that solves relaxations by convexifying the objective function. This algorithm is presented in §3.

We next give two formulations for the first direct approach, differing in their use of the binary variables. We use the following notation to represent $C(f)$, which is common for all edges. Let b_k be the flow values at which a jump occurs in $C(f)$, with $a_k = C(b_k)$ for $k = 1, \ldots, p$ defining p pieces. In addition, $b_0 = 0$ and $a_0 = 0$. Let $r_k = b_k - b_{k-1}$ be the range of flow and g_k be the incremental cost for the kth piece. Also, $r_0 = 0$ and $g_0 = 0$. Then, $a_k = \sum_{j=0}^{k} g_j$ is the optimal cost value for any flow in the range $(b_{k-1} = \sum_{j=0}^{k-1} r_j,\ b_k = \sum_{j=1}^{k} r_j]$ as illustrated in Figure 1.

We obtain the incremental cost formulation if we define a binary variable for each piece and each edge as follows. Let x_{ke} be a 0-1 variable such that if $f_e \in (b_{k-1}, b_k]$, then each of $x_{1e}, \ldots, x_{ke}$ equals one and each of $x_{k+1,e}, \ldots, x_{pe}$ equals zero. If $f_e = 0$, then all

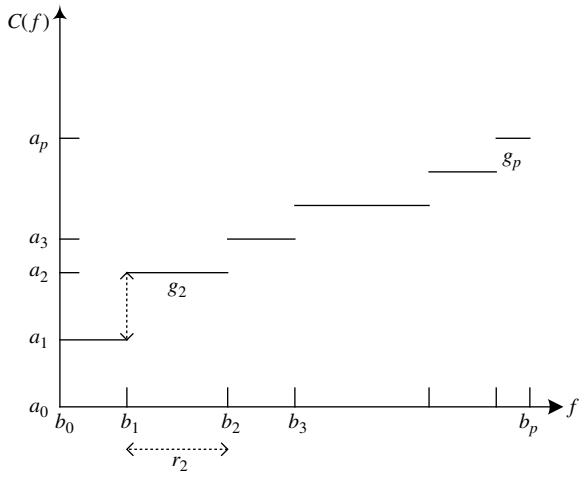

Figure 1 Representation of the Step Function $C(f)$

of $x_{1e}, \ldots, x_{pe}$ take the value zero. To ensure these conditions, we use constraints (5) and (6) given below, in the resulting formulation IP2:

$$\text{IP2:} \quad \min \sum_{e\in E} \left(l_e \sum_{k=1}^{p} g_k\, x_{ke} \right)$$

$$\text{s.t.} \sum_{j\in V} f_{ij} - \sum_{j\in V} f_{ji} = \begin{cases} d_i & \text{if } i \in S \\ -\sum_{k\in S} d_k & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in V,$$

$$f_{ij} + f_{ji} \leq \sum_{k=1}^{p} r_k\, x_{ke} \quad \forall e \in E, \tag{5}$$

$$x_{ke} \geq x_{(k+1)e} \quad k = 1, \ldots, p-1 \ \ \forall e \in E, \tag{6}$$

$$x_{ke} \in \{0,1\} \quad k = 1, \ldots, p \ \ \forall e \in E,$$

$$f_{ij},\ f_{ji} \geq 0 \quad \forall e = (i,j) \in E.$$

A slightly different formulation is obtained by defining the binary variable x_{ke} to represent whether piece k determines the optimal cable cost for edge e. In addition, x_{0e} is defined to represent the zero-flow case. Then, exactly one of the x_{ke}, $k = 0, \ldots, p$ has to be one. The resulting multiple-choice formulation IP3 is as follows:

$$\text{IP3:} \quad \min \sum_{e\in E} \left(l_e \sum_{k=1}^{p} a_k\, x_{ke} \right)$$

$$\text{s.t.} \sum_{j\in V} f_{ij} - \sum_{j\in V} f_{ji} = \begin{cases} d_i & \text{if } i \in S \\ -\sum_{k\in S} d_k & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in V,$$

$$f_{ij} + f_{ji} \le \sum_{k=1}^{p} b_k x_{ke} \quad \forall e \in E,$$

$$\sum_{k=0}^{p} x_{ke} = 1 \quad \forall e \in E, \tag{7}$$

$$x_{ke} \in \{0, 1\} \quad k = 0, \ldots, p \ \ \forall e \in E,$$

$$f_{ij}, f_{ji} \ge 0 \quad \forall e = (i, j) \in E.$$

IP3 has almost the same number of variables as IP2 but fewer constraints because (6) is replaced by (7). Because of this, the LP relaxation of IP3 has an optimal solution in which, for each edge e, at most two of the x_{ke} are nonzero. Due to this property, we may expect IP3 to perform better than IP2 in a branch-and-bound algorithm. Here we note that the LP relaxations of IP2 and IP3 have the same value as implied by the results of Keha et al. (2004) and Croxton et al. (2003) on piecewise-linear cost-minimization problems. Furthermore, they approximate the objective function of the flow problem FP with its lower convex envelope. This was also proven later on by Raghavan and Stanojevic (2005) for the LP relaxation of IP3 using duality and geometrical arguments. Raghavan and Stanojevic (2005) pointed out that if we consider each piece of $C(f)$ as a facility consisting of a combination of cables, this formulation corresponds to a network-loading problem where only one type of facility can be installed per edge.

3. Search by Objective Relaxation: A Branch-and-Bound Algorithm

In this section, we give a branch-and-bound algorithm to solve the flow formulation FP, in which the objective function is a linear combination of the discontinuous flow costs $C(f_e)$ defined in (3). We obtain a relaxation to FP by taking the lower envelope of $C(f_e)$ over its domain for all $e \in E$. This relaxation, which we refer to as the *convex hull relaxation*, is a multisource single-sink flow problem with a piecewise-linear convex objective function, which underestimates the actual costs. Its optimal objective-function value is equal to that of the LP relaxations of IP2 and IP3. However, the convex hull relaxation can be solved efficiently by utilizing a combinatorial minimum-cost network-flow algorithm. Furthermore, the obtained lower bound can be strengthened by branching on the range of flow on a selected edge. Each flow range defines a new domain for the cable-cost function of that edge. Taking the lower convex envelope over the new domain refines the function approximation and yields a new relaxation subproblem, which is again a flow problem with a different piecewise-linear convex objective function. Interestingly, the solution to any relaxation subproblem is a feasible flow for the LAN design problem. Therefore, the optimal cable cost corresponding to this flow solution is an upper bound to our problem. This leads to a branch-and-bound algorithm, in which the lower bounds are obtained by solving flow problems with approximate costs, and upper bounds are obtained at each branch-and-bound node with minimal effort from the relaxation solution f by replacing the approximate costs with the actual costs $C(f_e)$ for each edge e.

3.1. Approximating the Cost Function

Given an edge e and a flow range $[l, u]$, we approximate $C(f)$ by a function $h(f)$ with the following simple procedure, adapted from the well-known monotone-chain algorithm for two-dimensional convex hulls given points sorted along an axis (as we have here) due to Andrew (1979). Initially, set $h(f)$ to the linear function interpolating $C(l)$ and $C(u)$ so that $C(l) = h(l)$ and $C(u) = h(u)$. Recall that b_k denotes the flow values at which a jump occurs in $C(f)$ for $k = 1, \ldots, p$. For every breakpoint b_k in $[l, u]$, starting with the smallest one, check if $C(b_k) < h(b_k)$. If so, update $h(f)$ to the linear function interpolating $C(l)$ and $C(b_k)$ for all $f \in [l, u]$. Suppose that b_i is the last breakpoint at which h was updated. Fix $h(f)$ to the current linear function for $f \in [l, b_i]$. Recursively repeat the procedure for $[b_i, u]$. We illustrate the approximation procedure by an example in Figure 2. When the procedure

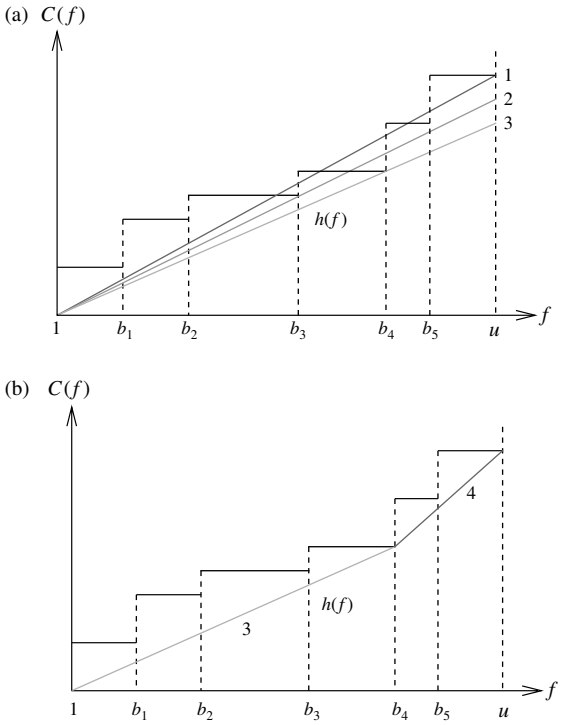

Figure 2 **Approximation of the Step Function $C(f)$ by Its Convex Envelope $h(f)$**

Notes. Initially, $h(f)$ is set to linear function 1. Then it is replaced by 2 at b_3, and 3 at b_4. After checking b_5, 3 is fixed for $[l, b_4]$. For $[b_4, u]$, it is set to the linear piece 4 shown in (b).

stops, the function $h(f)$ is the lower convex envelope of $C(f)$ over $f \in [l, u]$ by construction. It is piecewise linear and the slope of every piece increases as f increases.

3.2. Obtaining Lower and Upper Bounds

At a given node of the branch-and-bound tree, for every edge e, a range $(L_e, U_e]$ has been specified on flow f_e, such that flow on this edge should be at least L_e and at most U_e. Let h_e be the approximate cost function for edge e over the given range. A lower bound on the problem is obtained by solving a minimum-cost-flow problem defined by the approximate cost functions h_e. Due to the special structure of the h_e, the relaxation can be solved as follows. For each linear piece $1, \ldots, p$ in h_e, we divide f_e into variables $f_{e,1}, \ldots, f_{e,p}$ and add the constraint $f_e = f_{e,1} + \cdots + f_{e,p}$. Let piece k have range r_k and slope δ_k as shown in Figure 3. Then, $f_{e,k}$ takes values between zero and r_k, and has objective function coefficient δ_k. The relaxation has the form

$$\text{RP:} \quad \min \sum_{e \in E} l_e \left(\sum_{k=1}^{p} \delta_k f_{e,k} + C(L_e) \right)$$

$$\text{s.t.} \sum_{j \in V} f_{ij} - \sum_{j \in V} f_{ji} = \begin{cases} d_i & \text{if } i \in S \\ -\sum_{k \in S} d_k & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in V,$$

$$\sum_{k=1}^{p} f_{e,k} + L_e = f_{ij} + f_{ji} \quad \forall e = (i, j) \in E,$$

$$0 \le f_{e,k} \le r_k \quad k = 1, \ldots, p \quad \forall e \in E,$$

$$f_{ij},\ f_{ji} \ge 0 \quad \forall e = (i, j) \in E.$$

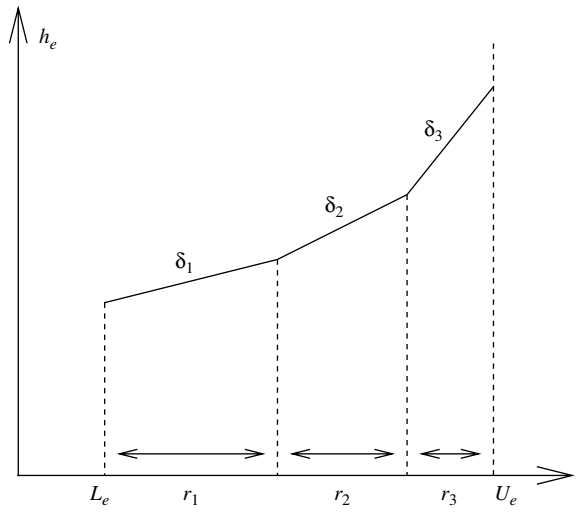

Figure 3 **Convex Hull Approximation h_e**

Note that $C(L_e)$ in the objective function is constant. For any edge e, the approximate function h_e has linear pieces with increasing slopes. Therefore, an optimal solution will assign flow starting from the leftmost piece so that $f_{e,k}$ will be positive only if $f_{e,k-1} = r_{k-1}$. As a result, the optimal objective-function value of RP gives a lower bound to the subproblem at the given node of the branch-and-bound tree. At the root node of the branch-and-bound tree, RP is defined by $L_e = 0$, $U_e = \sum_{k \in S} d_k$ for each edge e. Also, note that the lower bound obtained by solving RP at the root node has the same value as that of the LP relaxations of IP2 or IP3.

At a given node of the branch-and-bound tree, let $\bar{f}$ be an optimal solution to the relaxation RP. Because $\bar{f}$ is a feasible flow solution, when it is augmented with the optimal cable choice for each edge, we obtain the upper bound $\sum_{e \in E} l_e C(\bar{f}_e)$ for our problem. We keep the best upper bound over all nodes of the branch-and-bound tree.

3.3. Fathoming

There are three fathoming rules. The first is by comparing the lower bound at a node with the global upper bound UB. If the lower bound at a node is at least UB, then this node is fathomed. The second is by comparing the lower bound and upper bound obtained at a node. After the relaxation solution $\bar{f}$ is obtained, for each edge e, we calculate $C(\bar{f}_e) - h(\bar{f}_e)$. If the difference is zero (or smaller than a precision parameter) for all edges, then we fathom this node. Third, if the relaxation is infeasible at a node, then the node is fathomed.

3.4. Branching

We pick an edge according to one of three rules, and branch on the flow range for this edge. In the first rule, we pick the edge that has the most potential improvement in the cost approximation. That is, we pick the edge with maximum cost deviation $(C(\bar{f}_e) - h(\bar{f}_e))l_e$. In the second rule, we aim at improving the cost approximation for a large flow range. We pick the edge whose flow $\bar{f}_e$ has the largest range in C, i.e., the range of the piece with value $C(\bar{f}_e)$. In the third rule, we simply pick an edge with maximum flow.

Suppose that edge e^* is picked. Consider the piece of C with value $C(\bar{f}_{e^*})$, where $b_k \le \bar{f}_{e^*} \le b_{k+1}$. Branch into three problems by forcing three ranges on f_{e^*}: (1) $b_k < f_{e^*} \le b_{k+1}$, (2) $L_e \le f_{e^*} \le b_k$, and (3) $b_{k+1} < f_{e^*} \le U_e$. In the first problem, the cost of f_{e^*} is taken as the constant $C(\bar{f}_{e^*})$. For the other two problems, the cost function needs to be approximated again with the new ranges. This will lead to better approximations and strengthen the lower bound. Note that if $b_k = f_{e^*}$, then we have a degenerate case and branch into two. Alternatively, one can also

choose to branch into two problems instead of three: (1) $L_e \le f_{e^*} \le \bar{f}_{e^*}$, and (2) $\bar{f}_{e^*} < f_{e^*} \le U_e$. Then, for both problems the cost function is reapproximated within the new ranges. We refer to this branch-and-bound method as *search by objective relaxation* (SOR). Achim et al. (1975) gave a branch-and-bound algorithm for solving general concave-cost network flow problems that is also based on relaxing the objective function. Their algorithm uses a linear cost approximation to obtain lower bounds and bisects the flow interval for branching, whereas SOR uses the convex envelope for lower bounds. As mentioned in §1, Raghavan and Stanojevic (2005) showed that the SOR method can be reinterpreted as a stylized branch-and-bound solution strategy on IP3. While SOR calculates the lower convex envelope and solves the network-flow problem RP at each node of the branch-and-bound tree, their procedure solves the LP relaxation of IP3 with additional constraints to impose the branching. In addition to bounds on flow variables specifying the flow range for that node, they add constraints, setting the x_{ke} variables to zero for pieces of $C(f_e)$ that are outside the flow range. In this way, they avoid explicitly calculating the lower convex envelope, and it becomes possible to utilize the branch-and-bound algorithm of a commercial MIP solver with callback functions for branching. On the contrary, our SOR implementation manages the branch-and-bound tree itself as explained in the next section. Raghavan and Stanojevic (2005) independently proved (this also follows from earlier results of Croxton et al. 2003 and Keha et al. 2004) that the LP relaxation of the restricted problems on which they branch have exactly the same value as calculated by our lower convex relaxation RP. They used this to argue that their branching strategy directly mimics the SOR method.

4. Computational Study

To test the effectiveness of the SOR method, we performed computational tests. We compared the performance of SOR to the three alternate formulations, IP1, IP2, and IP3, solved by a general-purpose MIP solver (`CPLEX 9.1`). We used a test bed consisting of LAN design instances that are either randomly generated according to the characteristics of real-life instances or selected from instances available in the literature. In the following, we first describe the data generation and implementation. Next, we present the computational results.

4.1. Input Data

From our interactions with a telecommunication company, we learned that (1) real-life instances have input graphs that are very close to Euclidean graphs with low edge density, and (2) the link between two nodes cannot be very long because the quality of transmission degrades by distance. Based on this information and similar work in the literature (Berger et al. 2000, Magnanti et al. 1995), we generated random instances with varying size and parameters as follows.

We generated problems with $n = 20$, 30, and 40 nodes. We generated the coordinates of the nodes uniformly on an $R \times R$ region in the Euclidean plane, where $R = 30$, 40, and 50, respectively, for $n = 20$, 30, and 40. We randomly chose a target degree for each node. We picked a target degree of 1 or 2 with probability 0.1 each, and one of 3, 4, 5, and 6 with probability 0.2 each. We set the upper bound on the length of an edge to be 15. We connected each node to the node closest to it until its degree requirement was satisfied, or there were no more nodes within distance 15 of it. If in the end, the total number of edges was less than $2(n-1)$ or greater than $n(n-1)/4$, we rejected the graph due to the extreme density levels and repeated the process. Finally, if the graph was not connected, we discarded it and constructed a new one. We had to repeat the procedure only a few times to obtain each instance.

The choice of the sink node was made in two ways. In the first case, the sink was chosen uniformly among all nodes. In the second case, we picked the center node, i.e., a node that has the minimum value of the maximum distance to any node. We chose each of the remaining nodes independently to be a source node with probability 0.5. We generated two levels of demand data with integer demands. In the low-demand case, we chose the demand of each source node uniformly between 1 and 30. In the high-demand case, demand was chosen between 1 and 60.

We named these randomly generated problems e$(n)(c/r)(l/h)$, where n is the number of nodes, c refers to the sink node being the "center" node, and r refers to the sink node being chosen randomly; l refers to a low-demand case and h to a high-demand case. There are a total of 12 types of problems and we generated five instances of each type. We report the average performance of the methods over these instances. We selected four more instances based on the graphs named *ARPA*, *OCT*, *USA*, and *RING* used by Gavish and Altinkemer (1990), which were also used later by Amiri and Pirkul (1997), with random demand data. The problem size for these instances and the rounded average sizes of the 12 types are given in Table 1.

We used the same cable data as Berger et al. (2000), which is based on the cable types available in the market. The data consist of nine cable types, with costs and capacities given in Table 2. We used different subsets of the cable types to test the performance of SOR under different cost structures. We solved each problem twice, once using all the available cable types,

Table 1 Problem Size of the Instances

Problem	Nodes (n)	Edges (m)	Sources (k)
e20ch	20	40	9
e20cl	20	40	9
e20rh	20	40	9
e20rl	20	40	10
e30ch	30	58	16
e30cl	30	58	16
e30rh	30	59	12
e30rl	30	59	14
e40ch	40	80	19
e40cl	40	80	19
e40rh	40	81	18
e40rl	40	81	21
ARPA	21	26	12
OCT	25	29	14
USA	26	39	16
RING	32	60	17

Table 2 Capacity and Per-Unit Length Cost of Available Cable Types

Cable type	Capacity	Cost/length	Cost/cap. length
1	6	0.55	0.092
2	12	0.73	0.061
3	24	1.03	0.043
4	36	1.39	0.039
5	48	1.67	0.035
6	72	2.31	0.032
7	96	3.03	0.031
8	144	4.37	0.030
9	216	6.33	0.029

and once using cable types 1, 3, 5, and 7. The optimal cost as a function of flow on an edge, $C(f)$, is plotted for both cases in Figure 4. The cost function for cable types $1, \ldots, 9$ has more pieces, but the jumps between the steps are larger in the cost function for cable types 1, 3, 5, and 7.

4.2. Implementation of SOR

We implemented SOR with Microsoft Visual C++ and represented some of our data structures using the library LEDA (Library of Efficient Data types and Algorithms). We note that the implementation could be enhanced with better handling of data structures and better memory management. To obtain a lower bound at each branch-and-bound node, we need to solve a relaxation in the form of a minimum-cost network flow problem. In our implementation, we solved the relaxations RP by calling the minimum-cost flow function of LEDA 3.7, which is based on the Edmonds and Karp (1972) capacity-scaling and successive shortest-path algorithm, and has running time $O(m \log D(m + n \log n))$, where D is the total demand, m is the number of edges, and n is the number of nodes in the input graph. To use this function, we created parallel edges for each piece of the approximate function $h(f)$ in the relaxed problem RP. As mentioned earlier, the convex cost structure on these parallel edges ensures that their utilization is in the intended order in the optimal solution.

4.3. Branching and Search Rules

The effectiveness of a branch-and-bound procedure depends largely on the strategies used for branching and searching the tree. We implemented several branching and search rules and compared them with tests to select suitable rules for all experiments. We first compared the three branching rules given in §3 in terms of choosing the edge on which to branch. The branching rule that selects the edge with maximum difference of approximate and actual costs outperformed the others by a large margin. Therefore, we used this rule throughout the experiments. We repeated all of the SOR runs by branching into two and three subproblems as defined in §3. We use SORb2 and SORb3 to indicate the corresponding versions of SOR. We compared various rules to pick the next node to be processed, including depth-first search and breadth-first search, and decided to select the node with the minimum lower bound. This rule minimizes the number of branch-and-bound nodes processed and makes good progress in improving the global lower bound. However, the number of active branch-and-bound nodes grows rapidly by iterations, hence increasing the memory allocated to the tree.

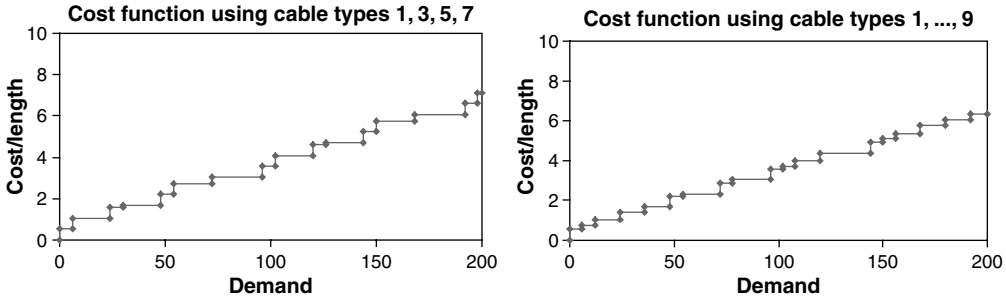

Figure 4 Optimal Cable Cost Per Unit Length as a Function of Flow Using Cables 1, 3, 5, 7 and 1,...,9

Table 3 Closeness of the Initial Upper Bound to the Optimal Solution on the Selected Instances

	Cable types 1, 3, 5, 7			Cable types 1, . . . , 9		
Problem	UB	LB	%Difference	UB	LB	%Difference
ARPA	2,714.33	2,714.33	0.00	2,604.85	2,571.36	1.30
OCT	2,684.72	2,639.75	1.70	2,473.19	2,432.32	1.68
USA	2,444.84	2,273.80	7.52	2,239.80	2,125.57	5.37
RING	1,511.10	1,379.07	9.57	1,428.19	1,314.90	8.62

Note. Column UB gives the upper bound at the first node of the branch-and-bound tree. LB is the global lower bound when SOR terminates. %Difference equals (UB – LB)/LB in percentage.

4.4. Progress of the SOR Algorithm

We observed that the SOR procedure rapidly generates high-quality upper bounds for the problem. Much of the progress in closing the gap between the lower and upper bounds is made to improve the lower bound. The improvement in the lower bound gets considerably smaller as the iterations progress. In our experiments, the upper bound found by the SOR method at the *first* node of the branch-and-bound tree was already fairly close to the optimal solution. In Table 3, we report the gap between the *initial* upper bound and the global lower bound found when the SORb2 algorithm terminates, as a percentage of the global lower bound for the selected instances. The largest gap in the table is 9.57% for problem RING with cable types 1, 3, 5, and 7. The average of all gaps reported is 4.47%.

4.5. Computational Comparison of SOR and the MIP Formulations

We solved the three MIP formulations, IP1, IP2, and IP3 using ILOG CPLEX 9.1.0 with the default settings, allowing it to generate cuts as needed. We ran both CPLEX and our SOR code on a PC with an Intel Pentium D 3.0 GHz processor and 2 GB of RAM operating under Windows XP. We set a time limit of 5,400 seconds for each instance. For all the instances that could not be solved to optimality because this limit was reached, we report the gap between the best solution and the best lower bound obtained, and the number of branch-and-bound nodes processed so far. For instances solved to optimality, the number of processed branch-and-bound nodes is a better indicator than is the CPU time elapsed. As a commercial code, CPLEX can process nodes much faster and manage the search tree much more efficiently. In fact, on average, CPLEX processes nodes around 10 times faster than does our code.

We solved each randomly generated and selected instance with two different cost functions; once with cable types 1, 3, 5, and 7 (four cable types) and once with all of the nine cable types available. Tables 4 and 5 show the size of the various formulations for these two types of problems. Here, NV denotes the total number of variables in the model, NIV denotes the number of integer/binary variables in the model, NC denotes the number of constraints in the model, and NCN denotes the number of constraint nonzeros in the model.

Table 4 Average Formulation Sizes on Randomly Generated Problems

		Cable types 1, 3, 5, 7				Cable types 1, . . . , 9			
Problem	Method	NV	NIV	NC	NCN	NV	NIV	NC	NCN
e20h	IP1	241	160	61	561	441	360	61	961
	IP2	927	846	866	2,697	1,099	1,018	1,038	3,209
	IP3	967	886	101	2,778	1,139	1,058	101	3,294
e20l	IP1	241	160	61	561	441	360	61	961
	IP2	497	419	437	1,409	594	513	533	1,698
	IP3	537	456	101	1,490	634	553	101	1,779
e30h	IP1	353	235	90	823	648	530	90	1,413
	IP2	2,084	1,966	1,896	6,131	2,489	2,370	2,400	7,346
	IP3	2,143	2,024	149	6,250	2,536	2,441	149	7,465
e30l	IP1	353	235	90	823	648	530	90	1,413
	IP2	1,236	1,117	1,147	3,587	1,483	1,364	1,394	4,327
	IP3	1,295	1,176	149	3,705	1,541	1,423	149	4,446
e40h	IP1	483	321	122	1,125	885	723	122	1,928
	IP2	3,336	3,174	3,214	9,843	4,002	3,841	3,881	11,843
	IP3	3,416	3,254	202	10,005	4,083	3,921	202	12,022
e40l	IP1	483	321	122	1,125	885	723	122	1,928
	IP2	2,105	1,944	1,984	6,152	2,522	2,361	2,401	7,404
	IP3	2,186	2,024	202	6,314	2,603	2,441	202	7,565

We summarize the performance of the various methods (including the two- and three-way branching versions of SOR indicated by SORb2 and SORb3) on randomly generated and specific instances in Tables 6–9.

Overall, in terms of the gap closed, IP1 has the overall best performance, while IP2 and IP3 still perform better than does the SOR method. The overhead of the LEDA calls and managing the search tree typically make the running time per call to compute each node of the SOR method about 10 times more than that for the IP solver, as can be seen from the number of nodes explored by both types of methods. The superiority of IP1 might stem from the higher effectiveness of the

Table 5 Formulation Sizes on Selected Instances

		Cable types 1, 3, 5, 7				Cable types 1, . . . , 9			
Problem	Method	NV	NIV	NC	NCN	NV	NIV	NC	NCN
ARPA	IP1	156	104	48	365	286	234	48	625
	IP2	780	728	749	2,288	962	910	931	2,834
	IP3	806	754	74	2,341	988	936	74	2,887
OCT	IP1	174	116	55	407	319	261	55	697
	IP2	1,015	957	982	2,987	1,189	1,131	1,156	3,509
	IP3	1,044	986	84	3,046	1,218	1,160	84	3,568
USA	IP1	234	156	66	547	429	351	66	937
	IP2	1,482	1,404	1,430	4,368	1,794	1,716	1,742	5,304
	IP3	1,521	1,443	105	4,447	1,833	1,755	105	5,383
RING	IP1	360	240	93	841	660	540	93	1441
	IP2	2,460	2,341	2,372	7,260	2,940	2,820	2,852	8,700
	IP3	2,520	2,400	153	7,381	3,000	2,880	153	8,821

Table 6 Number of Nodes Explored and Gap Closed for Random Instances on 20 Nodes

		Cable types 1, 3, 5, 7		Cable types 1, . . . , 9	
Problem	Method	Gap	BB nodes	Gap	BB nodes
e20ch	IP1	0.00	24,291	0.00	171,490
	IP2	0.00	62,771	0.00	68,874
	IP3	0.00	2,145,070	0.00	522,628
	SORb2	0.86	208,026	0.70	280,439
	SORb3	1.36	205,636	1.90	258,755
e20cl	IP1	0.00	4,108	0.00	13,576
	IP2	0.00	5,261	0.00	7,024
	IP3	0.00	10,103	0.00	7,524
	SORb2	0.23	126,833	0.00	89,333
	SORb3	0.46	116,805	0.83	154,770
e20rh	IP1	0.00	35,745	0.00	1,994,178
	IP2	0.00	114,746	0.00	23,865
	IP3	0.01	769,040	0.00	302,764
	SORb2	1.47	284,113	0.97	264,907
	SORb3	1.48	255,787	2.05	283,576
e20rl	IP1	0.00	188,158	0.18	1,565,700
	IP2	0.00	595,501	0.60	945,680
	IP3	0.00	1,553,857	0.26	1,981,946
	SORb2	3.84	338,819	1.91	353,827
	SORb3	4.19	378,604	4.36	375,517

Table 8 Number of Nodes Explored and Gap Closed for Random Instances on 40 Nodes

		Cable types 1, 3, 5, 7		Cable types 1, . . . , 9	
Problem	Method	Gap	BB nodes	Gap	BB nodes
e40ch	IP1	1.92	4,701,213	2.55	3,341,192
	IP2	4.32	847,753	3.79	891,703
	IP3	6.65	4,014,209	6.65	3,727,009
	SORb2	7.97	394,058	7.87	386,911
	SORb3	8.77	402,400	8.79	402,003
e40cl	IP1	2.55	3,599,122	2.59	3,577,142
	IP2	5.79	1,392,361	6.24	1,497,733
	IP3	6.16	4,418,148	8.33	4,521,243
	SORb2	14.14	424,275	14.65	426,575
	SORb3	15.02	418,073	15.58	412,895
e40rh	IP1	1.67	3,647,909	1.63	2,819,516
	IP2	3.47	1,063,011	3.25	967,777
	IP3	5.54	4,024,210	4.86	3,549,717
	SORb2	6.54	393,132	6.32	386,114
	SORb3	6.91	398,989	7.18	384,165
e40rl	IP1	1.87	4,191,998	3.05	3,205,997
	IP2	4.29	1,325,119	4.77	1,204,428
	IP3	6.20	4,567,211	6.30	4,340,000
	SORb2	10.17	414,606	10.38	343,215
	SORb3	10.68	412,810	11.57	330,611

default cut-generation methods on this formulation compared to the others. We observed that CPLEX 9.1 utilizes many mixed-integer rounding cuts in solving IP1, in addition to fractional Gomory cuts. On the other hand, flow cuts, flow-path cuts, and fractional Gomory cuts are generated to solve IP2, while only flow cuts are utilized to solve IP3. The effectiveness of the latter two formulations may, however, improve if we make use of the special constraints on the ordered set of choice variables using the SOS options in the IP solver, which we did not investigate.

As the problem size scales, SOR is able to find solutions with reasonable gaps with substantially fewer nodes, suggesting more promise for this method on

Table 7 Number of Nodes Explored and Gap Closed for Random Instances on 30 Nodes

		Cable types 1, 3, 5, 7		Cable types 1, . . . , 9	
Problem	Method	Gap	BB nodes	Gap	BB nodes
e30ch	IP1	0.43	2,018,486	1.47	6,141,812
	IP2	1.28	979,855	1.70	1,194,326
	IP3	3.54	5,417,557	3.76	5,351,024
	SORb2	5.10	411,795	6.15	398,974
	SORb3	5.36	417,486	8.21	407,280
e30cl	IP1	0.80	2,193,000	1.54	2,720,303
	IP2	1.88	1,332,672	1.68	1,311,462
	IP3	2.59	3903886	3.21	3,735,251
	SORb2	8.74	435,870	7.07	436,302
	SORb3	8.82	437,850	8.38	424,108
e30rh	IP1	0.52	1,719,622	1.46	3,907,437
	IP2	1.17	1,274,260	1.24	1,142,406
	IP3	2.34	4,936,039	2.91	5,750,762
	SORb2	4.70	421,013	4.53	422,675
	SORb3	5.16	413,102	5.39	417,002
e30rl	IP1	1.25	2,290,804	1.41	2,308,493
	IP2	1.86	1,292,462	0.86	1,619,925
	IP3	3.43	4,334,304	1.77	3,180,826
	SORb2	8.07	430,614	7.59	436,131
	SORb3	9.11	427,236	9.20	420,581

Table 9 Number of Nodes Explored and Gap Closed for Selected Instances

		Cable types 1, 3, 5, 7		Cable types 1, . . . , 9	
Problem	Method	Gap	BB nodes	Gap	BB nodes
ARPA	IP1	0.00	30,230	0.00	342,171
	IP2	0.00	4,461	0.00	11,183
	IP3	0.00	84,807	0.00	401,857
	SORb2	0.00	31,139	0.00	52,511
	SORb3	0.00	16,254	0.00	63,716
OCT	IP1	0.00	129,569	0.00	10,881
	IP2	0.00	65,898	0.00	9,284
	IP3	0.00	193,385	0.00	29,607
	SORb2	0.00	406,611	0.00	102,573
	SORb3	0.00	312,357	0.00	155,471
USA	IP1	0.00	523,272	1.72	6,191,421
	IP2	0.00	213,098	0.00	1,347,575
	IP3	2.29	7,221,023	1.38	6,701,171
	SORb2	3.84	424,109	4.82	410,393
	SORb3	4.05	420,709	6.28	403,522
RING	IP1	0.00	892,771	1.64	3,720,450
	IP2	0.95	1,008,977	3.27	892,675
	IP3	3.90	5,019,263	3.27	4,714,931
	SORb2	5.88	425,887	6.49	407,253
	SORb3	6.00	423,775	6.55	404,389

Table 10 Number of Nodes Explored Until the Gap Was Reduced to 2% for ARPA and OCT, and 7% for USA and RING

		Cable types 1, 3, 5, 7		Cable types 1, . . . , 9	
Problem	Method	Target nodes	Total nodes	Target nodes	Total nodes
ARPA	IP1	3,662	30,230	26,105	342,171
	IP2	2,933	4,461	6,257	11,183
	IP3	50,410	84,807	147,278	401,857
	SORb2	12,660	31,139	22,318	52,511
	SORb3	12,003	16,254	29,973	63,716
OCT	IP1	35,302	129,569	6,955	10,881
	IP2	32,520	65,898	1,395	9,284
	IP3	73,544	193,385	15,868	29,607
	SORb2	171,530	406,611	47,190	102,573
	SORb3	161,379	312,357	82,753	155,471
USA	IP1	6,718	523,272	3,917	6,191,421
	IP2	1,042	213,098	2,875	1,347,575
	IP3	38,840	7,221,023	30,539	6,701,171
	SORb2	17,908	424,109	47,052	410,393
	SORb3	23,602	420,709	184,292	403,522
RING	IP1	9,639	892,771	6,435	3,720,450
	IP2	8,350	1,008,977	27,269	892,675
	IP3	468,606	5,019,263	539,976	4,714,931
	SORb2	153,756	425,887	237,686	407,253
	SORb3	156,078	423,775	243,000	404,389

even larger instances. A similar pattern is detected by examining its performance on the larger selected instances in Table 9.

Among the two branching strategies for SOR, our experiments also indicate that the two-way branching strategy is somewhat more effective in closing the gap owing to its ability to explore more nodes and also more diverse parts of the solution space compared to its three-way branching counterpart. We also examined the number of nodes required by each of the methods in the selected instances to reach a certain gap, to make a comparison that is not affected by the speed of solving the nodes. The results given in Table 10 suggest that SOR as a strategy is more effective than only IP3 solved by default settings of CPLEX in this dimension. When we examine the performance of SOR on the different types of randomly generated instances in Tables 6–8, we see that SOR is more effective when the demand range is higher and more cable types are available. Naturally, instances with nine cable types are harder for IP1 compared to ones with four cable types as the number of integer variables increases (Table 4). On the other hand, IP2, IP3, and SOR all use the pregenerated cost function $C(f)$, and thus the performance of these methods is not sensitive to the number of cables. This suggests that generating the optimal cable-cost function first is advantageous when the number of cables is large. Note that SOR avoids defining binary variables, as opposed to IP2 and IP3. This seems to be an advantage when the demand range is high because the number of binary variables in IP2 and IP3 almost doubles in high demand instances as seen in Table 4. We have also observed that instances in which the sink node is in the center are typically harder than their counterparts.

4.6. Conclusions from Computational Tests

The computational results indicate that the SOR method is promising in solving the local access network design problem. The SOR method, while unable to solve many instances with 30 or more nodes to optimality, was able to reduce the integrality gap by exploring a small number of nodes. It also shows good scaling properties in large instances, as compared to the IP formulations. The two-way branching variant seems to be more effective than the three-way branching strategy in almost all instances examined.

5. Summary and Conclusions

We presented an exact solution method, namely, the SOR branch-and-bound algorithm, for the single-source multiple-sink routing and link-capacity installation problem with multiple facilities with applications in network design.

The SOR algorithm solves relaxations obtained by approximating the noncontinuous cost function by its convex envelope. The approximations are refined by branching on the flow ranges on selected edges. We compared the performance of this method to three MIP formulations on generated and widely available LAN design instances. In our experiments, SOR managed to approximate the cable-cost function closely by the piecewise-linear convex hull and outperformed the three formulations solved by CPLEX, especially in larger instances.

An investigation of the progress of the SOR method indicated that the method finds very strong upper bounds early in the branch-and-bound procedure. This suggests that the early termination of the SOR method could serve as an effective heuristic.

Finally, we note that the SOR method is applicable to the backbone network design problem where flow is sent from multiple sources to multiple sinks. When the backbone network design problem is solved using the SOR approach, the objective function and its representation is the same, hence the branching will be the same. The only difference will be that the relaxation problems are in the form of multicommodity-flow problems as opposed to single-commodity minimum-cost flow problems. Hence, the relaxation solution is likely to take more computation time. This method remains to be investigated in future work.

Acknowledgments

The authors thank three anonymous referees and Prakash Mirchandani for valuable comments and suggestions, in

particular for suggesting the use of IP3. The second author was supported in part by National Science Foundation (NSF) Career Grant CCR-9625297.

References

Achim, C., M. Florian, P. Robillard. 1975. *Experiments in Solving Concave Cost Network Flow Problems*. Department d'informatique, Université de Montréal, Montréal.

Amiri, A., H. Pirkul. 1997. Routing and capacity assignment in backbone communication networks. *Comput. Oper. Res.* **24** 175–287.

Andrew, A. M. 1979. Another efficient algorithm for convex hulls in two dimensions. *Inform. Processing Lett.* **9** 216–219.

Atamtürk, A. 2002. On capacitated network design cut-set polyhedra. *Math. Programming* **92** 425–437.

Atamtürk, A., D. Rajan. 2002. On splittable and unsplittable flow capacitated network design arc-set polyhedra. *Math. Programming* **92** 315–333.

Awerbuch, B., Y. Azar. 1997. Buy-at-bulk network design. *Proc. 38th Annual Sympos. Foundations Comput. Society* (*FOCS*), *Miami Beach, FL*, 542–547.

Barahona, F. 1996. Network design using cut inequalities. *SIAM J. Optim.* **6** 823–837.

Berger, D., B. Gendron, J. Potvin, S. Raghavan, P. Soriano. 2000. Tabu search for a network loading problem with multiple facilities. *J. Heuristics* **6** 253–267.

Bienstock, D., O. Günlük. 1995. Computational experience with a difficult multicommodity flow problem. *Math. Programming* **11** 213–237.

Bienstock, D., O. Günlük. 1996. Capacitated network design—Polyhedral structure and computation. *INFORMS J. Comput.* **8** 243–259.

Bienstock, D., S. Chopra, O. Günlük, C.-Y. Tsai. 1998. Minimum cost capacity installation for multicommodity network flows. *Math. Programming* **81** 177–199.

Chopra, S., I. Gilboa, S. T. Sastry. 1998. Source sink flows with capacity installation in batches. *Discrete Appl. Math.* **85** 165–192.

Croxton, K. L., B. Gendron, T. L. Magnanti. 2003. A comparison of mixed-integer programming models for non-convex piecewise linear cost minimization problems. *Management Sci.* **49** 1268–1273.

Edmonds, J., R. M. Karp. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* **19**(2) 248–264.

Gabrel, V., A. Knippel. 1999. Exact solution of multicommodity network optimization problems with general step cost functions. *Oper. Res. Lett.* **25** 15–23.

Gabrel, V., A. Knippel, M. Minoux. 2003. A comparison of heuristics for the discrete cost multicommodity network optimization problem. *J. Heuristics* **9**(5) 429–445.

Gavish, B., K. Altinkemer. 1990. Backbone network design tools with economic tradeoffs. *ORSA J. Comput.* **2** 236–252.

Guha, S., A. Meyerson, K. Munagala. 2001. A constant factor approximation for the single sink edge installation problems. *Proc. ACM Sympos. Theory Comput.*, Heraklion, Crete, Greece, 383–388.

Gupta, A., A. Kumar, T. Roughgarden. 2003. Simpler and better approximation algorithms for network design. *Proc. ACM Sympos. Theory Comput.*, San Diego, 365–372.

Hooker, J. N. 2000. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction.* John Wiley & Sons, New York.

Keha, A. B., I. R. de Farias Jr., G. L. Nemhauser. 2004. Models for representing piecewise linear cost functions. *Oper. Res. Lett.* **32** 44–48.

Magnanti, T. L., P. Mirchandani. 1993. Shortest paths, single origin-destination network design and associated polyhedra. *Networks* **23** 103–121.

Magnanti, T. L., P. Mirchandani, R. Vachani. 1993. The convex hull of two core capacitated network design problems. *Math. Programming: Ser. A* **60** 233–250.

Magnanti, T. L., P. Mirchandani, R. Vachani. 1995. Modeling and solving the two-facility capacitated network loading problem. *Oper. Res.* **43** 142–157.

Mirchandani, P. 2000. Projections of the capacitated network loading problem. *Eur. J. Oper. Res.* **122** 534–560.

Ottosson, G., E. Thorsteinsson, J. N. Hooker. 2002. Mixed global constraints and inference in hybrid CLP-IP solvers. *Ann. Math. AI* **34** 271–290.

Raghavan, S., D. Stanojevic. 2005. A note on search by objective relaxation. S. Raghavan, G. Anandalingam, eds. *Telecommunications Planning: Innovations in Pricing, Network Design and Management*. Springer, New York, 181–202.

Réfalo, P. 1999. Tight cooperation and its application in piecewise linear optimization. J. Jaffar, ed. *Principles and Practice of Constraint Programming. Lecture Notes in Computer Science*, Vol. 1713. Springer, Berlin, 373–389.

Salman, F. S., J. Cheriyan, R. Ravi, S. Subramanian. 1997. Buy-at-bulk network design: Approximating the single-sink edge installation problem. *Proc. ACM-SIAM Sympos. Discrete Algorithms*, New Orleans, 619–628.

Salman, F. S., J. Cheriyan, R. Ravi, S. Subramanian. 2000. Approximating the single-sink link-installation problem in network design. *SIAM J. Optim.* **11** 595–610.

Stoer, M., G. Dahl. 1994. A polyhedral approach to multicommodity survivable network design. *Numerische Mathematik* **68** 149–167.

Talwar, K. 2002. The single-sink buy-at-bulk LP has constant integrality gap. *Proc. Conf. Integer Programming and Combin. Optim., Lecture Notes in Computer Science*, Vol. 2337. Cambridge, MA, 475–486.