A DUAL-BASED ALGORITHM FOR MULTI-LEVEL NETWORK DESIGN

Anantaram Balakrishnan
Thomas L. Magnanti
Prakash Mirchandani

OR 261-91 December 1991

A Dual-based Algorithm for Multi-level Network Design

Anantaram Balakrishnan†
Thomas L. Magnanti

Sloan School of Management
M. I. T.
Cambridge, MA

Prakash Mirchandani

Katz Graduate School of Business
University of Pittsburgh
Pittsburgh, PA

December, 1991

† Supported in part by a grant from the AT&T Research Fund

Abstract

Given an undirected network with L possible facility types for each edge, and a partition of the nodes into L levels, the Multi-level Network Design (MLND) problem seeks a fixed cost minimizing design that spans all the nodes and connects the nodes at each level by facilities of the corresponding or higher type. This problem generalizes the well-known Steiner network problem and the hierarchical network design problem, and has applications in telecommunication, transportation, and electric power distribution network design. In a companion paper we introduced the problem, studied alternative model formulations, and analyzed the worst-case performance of heuristics based on Steiner network and spanning tree solutions. This paper develops and tests a dual-based algorithm for the Multi-level Network Design (MLND) problem. The method first performs problem preprocessing to fix certain design variables, and then applies a dual ascent procedure to generate upper and lower bounds on the optimal value. We report extensive computational results on large, random networks (containing up to 500 nodes, and 5000 edges) with varying cost structures. The integer programming formulation of the largest of these problems has 20,000 integer variables and over 5 million constraints. Our tests indicate that the dual-based algorithm is very effective, producing solutions guaranteed to be within 0 to 0.9% of optimality.

Keywords: Network design, integer programming, dual ascent algorithm

1. Introduction

Communication, transportation, and electric power distribution networks are often hierarchical, requiring higher grade interconnections for certain critical nodes. In a companion paper (Balakrishnan, Magnanti and Mirchandani [1991]), we proposed and formulated a **Multi-level Network Design (MLND)** model to address topological design tradeoffs in these hierarchical networks. The model is defined on an undirected graph whose nodes are partitioned into *L levels*. Each edge of the network can contain one of L different *facility types*, with higher grade facilities requiring higher fixed costs. In the MLND problem we must select a connected subset of edges, and choose a facility type for each edge so that all nodes at any level communicate via the corresponding or higher grade facilities. The objective is to minimize the total cost of the chosen facilities. We assume that all edge costs are nonnegative.

We refer to a special case of the MLND problem with only two node levels—*primary* and *secondary*—as the **Two-level Network Design (TLND)** problem. The TLND problem generalizes several well-known optimization models including the Steiner network problem (Dreyfus and Wagner [1972]), and the Hierarchical network design (HND) problem (Current, ReVelle, and Cohon [1986]). The Steiner network problem is a version of the problem with two levels: terminal nodes are primary nodes, and potential Steiner points are secondary nodes; in this case, every edge has zero secondary cost. The HND problem is a version with only two primary nodes; all other nodes of the network are secondary. Thus, the optimal solution to the HND problem is a spanning tree (assuming nonnegative costs) with primary facilities on all of the edges on the unique path in this tree connecting the two primary nodes.

Our previous paper (Balakrishnan et al. [1991]) examined modeling issues, and studied the worst-case performance of Steiner and spanning tree-based heuristics for the MLND problem. In this paper, we develop and test a dual-based solution strategy.

The MLND problem is NP-hard since it generalizes the Steiner network problem and the HND problem which are both known to be NP-hard even for certain special cost structures (Garey and Johnson [1979], Orlin [1991]). Consequently, our algorithmic strategy focuses on constructing a good heuristic solution using optimization-based techniques, and verifying the quality of this solution by generating lower bounds. Our proposed solution method for the MLND problem builds on the heritage of successful dual ascent procedures that researchers have previously developed for three related design problems—the Steiner network problem, the uncapacitated plant location problem, and the uncapacitated network design problems. Wong [1984] developed an efficient dual ascent algorithm to generate heuristic solutions (with associated performance guarantees) that were remarkably close to optimal. Erlenkotter [1978] and Balakrishnan, Magnanti, and Wong [1989] reported similar success using dual ascent algorithms for the uncapacitated plant location and (single-level) network design problems. We describe a closely related dual ascent method for the MLND problem. Furthermore, we augment this method by first performing a preprocessing step. This step applies certain problem reduction tests to identify included and excluded edges in the optimal solution. By reducing the problem prior to solving it, we reduce the computational effort.

Since dual ascent methods generate lower bounds by approximately solving the linear programming dual, the algorithm will be effective only if the linear programming formulation is a good (tight) approximation of the integer programming model of the problem. This requirement motivates our study of alternative model formulations. In Balakrishnan et al. [1991] we showed that an enhanced, undirected flow-based formulation, containing additional valid inequalities (called bidirectional, commodity-pair forcing constraints), is LP-equivalent to a more compact directed flow-based formulation (i.e., the linear programming relaxations for both formulations have the same optimal value). This result enables us, as in this paper, to use a simpler "directed" dual ascent algorithm (because the directed formulation has fewer constraints) without sacrificing the quality of the resulting solution. For the Steiner network problem, Wong [1984] and Chopra, Gorres, and Rao [1990] have successfully used a similar strategy of solving the compact, but strong, directed formulation.

Our previous paper also analyzed the worst-case performance of a combined heuristic method that considers Steiner tree and spanning tree-based solutions. We showed that if all the edges have the same primary-to-secondary cost ratio, whenever we can identify the optimal Steiner network, the combined heuristic has a worst-case performance ratio of 4/3. This situation applies, for example, to the HND problem, since in this case the Steiner network corresponds to a shortest path between the two primary nodes. When we solve the Steiner network problem approximately, the combined heuristic has the same worst-case performance ratio ρ as the Steiner network heuristic, and when the primary-to-secondary cost ratio varies by edge, the worst-case ratio of the combined heuristic becomes $\rho + 1$. Using worst-case examples, we demonstrated that these worst-case performance ratios are tight.

The dual ascent method does not ensure good worst-case performance (in an unpublished paper, Sastry [1987] constructed some Steiner tree problem instances for which the dual ascent method has arbitrarily bad performance ratios). Nevertheless, in extensive computational testing reported in this paper, the method has generated solutions guaranteed to be within 0 to 0.9% of optimality. In contrast, for a subset of our test problems, the combined (Steiner, spanning tree) heuristic method has generated solutions at least 20% more expensive. Our computational tests considered different network sizes (containing up to 500 nodes and 5000 edges), various primary-to-secondary cost ratios, and different cost structures (random, Euclidean, Manhattan, and L_∞ norms). The integer programming (directed) formulation for our largest test problem contains 10,000 binary variables, 5 million continuous variables, and over 5 million constraints.

Our discussion in this paper focuses on the TLND problem, even though the dual ascent methodology extends easily to problems with more than two levels. We begin by briefly reviewing the definition and a directed flow-based formulation of the TLND problem. In Section 2, we discuss properties of optimal TLND solutions, and develop preprocessing tests to fix certain design variables. Section 3 develops the dual ascent method and an associated

heuristic improvement strategy. Section 4 presents our computational results, and Section 5 offers concluding remarks.

1.1 TLND Problem Formulation

We are given an undirected graph G = (N,E) containing two types of nodes—P is the set of *primary* nodes, and S is the set of *secondary* nodes. We index the primary nodes from 1 to p (= |P|), and secondary nodes from (p+1) to n. On each edge $\{i,j\} \in E$, we can install either a primary facility at a fixed cost a_{ij} or a secondary facility at a fixed cost b_{ij}. We assume that $a_{ij} \geq b_{ij} \geq 0$ for all edges $\{i,j\} \in E$. The TLND problem seeks a minimum cost spanning tree that contains an embedded primary subtree connecting all the primary nodes (and optionally including secondary nodes). All of the edges of the primary subtree contain primary facilities, and all of the remaining edges of the spanning tree contain secondary facilities.

To formulate this problem as a mixed integer program, we introduce (n-1) commodities. All commodities have unit demand, and share a common primary node, say node 1, as their origin or root node. For k = 2, ..., n, commodity k has node k as its destination. We refer to commodities k = 2, ..., p as *primary commodities* and to commodities k = p+1, ..., n as *secondary commodities*. We let P and S denote, respectively, the set of primary and secondary commodities (for convenience, we use the same notation P and S to represent both node and commodity subsets; the usage will be clear from the context).

Our directed flow-based problem formulation transforms the undirected problem into a directed problem by replacing each undirected edge {i,j} of the original network with two *directed arcs* (i,j) and (j,i) oriented in opposite directions. Both these directed arcs have the same primary and secondary costs a_{ij} and b_{ij} as the original edge. Let A denote the set of all arcs in the resulting directed graph. The directed and undirected problems are equivalent since all commodities share a common origin, and all costs are nonnegative. (The directed problem, therefore, has an optimal solution that does not select both arc (i,j) and (j,i); therefore, replacing each arc in the

optimal directed solution with an undirected edge gives an undirected solution with the same total cost.)

The mixed integer formulation for the directed problem uses flow conservation equations to ensure connectedness of the design, and installs primary or secondary facilities as appropriate via forcing constraints. For this purpose, we define two sets of binary variables x_{ij} and y_{ij} for each arc $(i,j) \in A$, as well as directed, commodity routing variables f_{ij}^k for each arc $(i,j) \in A$ and every commodity $k \in P \cup S$. The *primary (secondary) arc selection variable* x_{ij} (y_{ij}) assumes a value of 1 if arc (i,j) contains a primary (secondary) facility, and is 0 otherwise. The continuous variable f_{ij}^k represents the fraction of commodity k's (unit) demand flowing from node i to node j. In terms of these decision variables, the TLND problem has the following Directed Flow-based formulation, which we denote as [DF]:

[DF]

$$\text{minimize} \quad \sum_{(i,j)\in A} a_{ij} x_{ij} + \sum_{(i,j)\in A} b_{ij} y_{ij} \tag{1.1}$$

subject to

Commodity flow conservation:

$$\sum_{i\in N} f_{ij}^k - \sum_{i\in N} f_{ji}^k = \begin{cases} -1 & \text{if } j = 1 \\ 1 & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases} \quad \text{for all } j\in N,\ k\in P\cup S, \tag{1.2}$$

Primary forcing constraints

$$f_{ij}^k \leq x_{ij} \quad \text{for all } (i,j) \in A,\ k\in P, \tag{1.3}$$

Secondary forcing constraints

$$f_{ij}^k \leq x_{ij} + y_{ij} \quad \text{for all } (i,j) \in A,\ k\in S, \tag{1.4}$$

Nonnegativity, integrality:

$$x_{ij}, y_{ij} = 0 \text{ or } 1 \quad \text{for all } (i,j) \in A, \text{ and} \tag{1.5a}$$

$$f_{ij}^k \geq 0 \quad \text{for all } (i,j) \in A,\ k\in P\cup S. \tag{1.5b}$$

Constraints (1.2) define flow paths for the primary and secondary commodities. The forcing constraint (1.3) ensures that arc (i,j) contains a primary facility if it carries a primary commodity; constraint (1.4) ensures that arc (i,j) contains a either a primary or secondary facility if it carries a secondary commodity. This directed, flow-based formulation extends easily to general MLND problems with more than two levels. The general formulation contains L different design variables for each arc, corresponding to the L facility types; a generalization of constraint (1.4) ensures that a level *l* commodity can flow on arc (i,j) only if this arc contains a level *l* or higher grade facility.

We next identify and exploit certain special characteristics of optimal solutions of the TLND problem.

2. Properties of the Optimal TLND solution

The optimal two-level network design solution enjoys certain structural properties that permit us to preprocess the problem by eliminating certain edges or by fixing certain edges as part of an optimal design. By reducing a problem prior to solving it, we can potentially improve algorithmic performance by reducing the gap between the lower and upper bounds, and by decreasing the total computational time. This section outlines these distinctive properties of the optimal two-level network design. These properties are analogous to previous characterizations of optimal solutions to the Steiner network problem (Balakrishnan and Patel [1987]). We state the properties in this section for the undirected version of the TLND problem. The results are valid for the directed case as well if we replace the undirected spanning trees and the undirected cuts in this discussion with (directed) spanning arborescences and directed cuts.

To begin this discussion, we introduce some notation. For any subset of nodes Q, let G(Q) denote the subgraph of the original graph G induced by the nodes in Q. For any subgraph G', let $T_a(G')$ and $T_b(G')$ denote the minimum spanning trees of G' using primary and secondary costs, respectively. We will often consider *condensed* graphs that aggregate (or contract) all the primary

nodes and possibly some secondary nodes into a single node, say node 0. When this node aggregation process creates parallel edges, we delete all but the cheapest parallel edge (using either primary or secondary costs, depending on the context). For any subset of secondary nodes S' not spanned by the primary subtree, let S'+ be the set S' $\cup$ {0}; the node 0 aggregates all of the (primary and secondary) nodes spanned by the primary subtree. In the following discussion, for convenience we assume that the primary cost of each edge strictly exceeds its secondary cost.

2.1 Optimal Primary Subtree and Secondary Completion

Let T^{opt} denote an optimal TLND solution. We refer to the graph defined by the subset of primary edges of T^{opt} as the *optimal primary subtree* T^P. (Because primary costs exceed secondary costs, all the primary edges in the optimal solution must form a single subtree.) Let S^{opt} be the subset of secondary nodes spanned by the optimal primary subtree; the set $\bar{S}^{opt}$ contains secondary nodes not belonging to T^P. Let $T^S = T^{opt} \backslash T^P$ denote the subnetwork (forest) containing all the secondary edges in T^{opt}. As the following observations indicate, finding the optimal design is easy if we know the secondary node subset S^{opt} spanned by the primary subtree.

Property 1:

$T^P = T_a(G(P \cup S^{opt}))$, i.e., the optimal primary subtree is the (primary) minimum spanning tree of the induced subgraph $G(P \cup S^{opt})$.

Property 2:

$T^S = T_b(G(\bar{S}^{opt}+))$, i.e., the edges in the optimal secondary forest coincide with the edges of the (secondary) minimum spanning tree of the condensed graph $G(\bar{S}^{opt}+)$.

We refer to the procedure for constructing the condensed secondary graph $G(\bar{S}'+)$ and finding its minimum spanning tree for any secondary node subset S', whether optimal or not, as the *Optimal Secondary Completion*

method. We will use this method to complete the primary solution in our heuristic algorithm.

2.2 Excluded Secondary Edges

To identify secondary edges that do not belong to an optimal TLND solution, we first find the minimum spanning tree $T_b(G(S+))$ over a condensed secondary graph G(S+) that combines all the primary nodes into a single aggregate node 0.

Property 3:

The TLND problem has an optimal solution that contains only secondary edges belonging to $T_b(G(S+))$.

Proof:

We prove this property by contradiction. Suppose the optimal secondary completion with respect to the optimal primary subtree T^P selects a secondary edge e that does not belong to $T_b(G(S+))$. Let C(e) denote the cutset edges defined by eliminating edge e from this optimal tree. The minimum tree $T_b(G(S+))$ must have at least one edge, say e', belonging to this cutset; furthermore, since $T_b(G(S+))$ has minimum cost, the secondary cost of edge e' must be equal or lower than the secondary cost of edge e. Therefore, replacing edge e with edge e' in the given tree gives a solution with equal or lower total cost. ♦

Property 3 implies that, before solving the TLND problem, we can discard all secondary edges not belonging to $T_b(G(S+))$, leaving only $|S|$ secondary edges to consider. Equivalently, we can set $b_{ij} = a_{ij}$ for all edges {i,j} that do not belong to $T_b(G(S+))$. This preprocessing step reduces the computational effort to perform the optimal secondary completion in our heuristic algorithm.

2.3 Included Primary Edges

We now examine a converse property, namely, a sufficient condition that guarantees the inclusion of a primary edge in the optimal TLND solution. Consider the (primary) minimum spanning tree $T_a(G)$ of the original graph.

Property 4:

Every edge $\{i,j\} \in T_a(G)$ with $i,j \in P$ must also belong to an optimal TLND solution.

We can prove this property by a contradiction argument similar to the proof for Property 3. This property enables us to reduce the network by aggregating primary nodes that are directly connected in the minimum spanning tree $T_a(G)$.

2.4 Excluded Primary Edges

We now describe two properties that permit us to identify prohibited primary edges (i.e., edges that do not contain primary facilities in an optimal TLND solution) by solving minimum spanning tree problems over certain induced subgraphs.

Property 5:

The TLND problem has an optimal solution in which every edge $\{i,j\}$ with $i,j \in P$ also belongs to the minimum spanning tree $T_a(G(P))$.

The proof of this property, and of the related Property 6 to follow, is similar to the proof for Property 3.

Property 5 implies that we can eliminate from the network all the edges {i,j} connecting *primary* nodes i and j that do not belong to the minimum spanning tree $T_a(G(P))$ of the subgraph G(P) induced by the primary nodes. (Notice that the optimal TLND solution will never install a secondary facility on an edge connecting two primary nodes.) Effectively, of the $O(p^2)$ possible edges connecting pairs of primary nodes, we need to retain only the (p–1) edges belonging to $T_a(G(P))$. Next, we extend this property to edges

connecting a primary node i to a secondary node j. This property requires us to first solve the minimum spanning tree problem (using primary costs) over the subgraph $G(P \cup \{j\})$ induced by the node set $P \cup \{j\}$.

Property 6:

> *For every secondary node j, the TLND problem has an optimal solution containing a primary edge {i,j} with $i \in P$ only if {i,j} belongs to the minimum spanning tree $T_a(G(P \cup \{j\}))$.*

Property 6 requires us to first solve $|S|$ minimum spanning tree problems, one for each secondary node j. Then, for every edge {i,j} with $i \in P$ and $j \in S$, we can set the primary cost $a_{ij} = \infty$ if $\{i,j\} \notin T_a(G(P \cup \{j\}))$; notice, however, that the optimal TLND solution might still install secondary facilities on these edges.

These properties of the optimal TLND solution enable us to reduce the problem size prior to applying the dual ascent algorithm described in the next section. The implementation that we have used for computational testing did *not* incorporate the reduction test implied by Property 6. Also, we applied the preprocessing method to the given undirected problem, and then transformed the reduced, undirected network to a directed version for the dual ascent method (which requires a directed network). When the dual ascent method terminates, we construct an undirected heuristic design based on the dual solution to the directed network. A local improvement method then improves this dual-based network design. We next describe the dual ascent and heuristic solution methods.

3. Dual-based Algorithm for the TLND problem

The literature does not contain any solution methods for the general multi-level and two-level network design problems with more than two primary nodes, and positive secondary costs. However, researchers have studied several important special cases including the Steiner network problem and the HND problem. For the Steiner network problem, the

literature contains various optimal and optimization-based heuristic approaches (see Winter [1987] for a recent survey of Steiner network algorithms), including dynamic programming (Dreyfus and Wagner [1972]), Lagrangian relaxation (Beasley [1984], [1989]), dual ascent (Wong [1984]), polyhedral methods (Chopra et al. [1990]), and simulated annealing (Osborne and Gillett [1991]). Chopra et al. [1990] have solved to optimality Steiner network problems defined over complete networks containing up to 500 nodes, and sparse networks containing up to 2500 nodes, and 62500 edges. For the HND problem, Current et al. [1986] and Shier [1991] describe heuristic methods based on shortest path and spanning tree computations. Pirkul, Current, and Nagarajan [1991] recently developed a Lagrangian relaxation algorithm embedded in a branch-and-bound scheme. They report computational results for complete networks containing up to 85 nodes.

Since the TLND problem is NP-hard, our algorithmic development efforts focused on optimization-based heuristic methods that provide a posteriori performance guarantees (i.e., lower and upper bounds). This section describes a dual ascent method that generates a lower bound on the optimal TLND value, and also constructs a good heuristic solution that we improve using an Add–Drop method. As we mentioned in the introduction, the dual ascent technique has proven to be very effective for finding near-optimal solutions to several related discrete optimization problems including the Steiner network problem (Wong [1984]) and uncapacitated (single-level) network design problem (Balakrishnan et al. [1989]). Our TLND algorithm is closely related to these previous methods, but incorporates multiple commodity *types* (in contrast, both the Steiner network problem and the single-level network design problem consider only a single commodity type).

Since the directed TLND formulation [DF] is more compact than its LP-equivalent (enhanced) undirected version (see Balakrishnan et al. [1991]), we describe the dual ascent algorithm as it applies to the *directed* TLND problem. This dual ascent method extends easily to the general MLND problem as well. For problem contexts requiring multicommodity flows between multiple origins and multiple destinations (in this case, we cannot transform the undirected problem into a directed version), we can augment this method to directly solve the enhanced undirected formulation.

3.1 Dual Ascent Algorithm for Directed TLND Problems

The dual ascent method attempts to approximately solve the dual of the linear programming relaxation of the flow-based formulation [DF] using an iterative procedure that monotonically increases the dual objective function value at each iteration. The final value of the dual solution provides a lower bound on the optimal value of the TLND problem. The final dual solution also provides a heuristic solution that we improve using an Add–Drop procedure to obtain good upper bounds.

Let [LDF] denote the linear programming relaxation of formulation [DF], obtained by replacing the integrality conditions (1.5a) on the binary variables x_{ij} and y_{ij} with nonnegativity constraints. Let π_j^k, α_{ij}^k, and β_{ij}^k denote, respectively, the dual variables corresponding to constraints (1.2), (1.3), and (1.4). We refer to π_j^k as the *node price* for commodity k at node j and the variables α_{ij}^k and β_{ij}^k as the *allocated arc costs* for primary and secondary commodities on arc (i,j). The motivation for this terminology will become apparent as we describe the dual ascent method. Since one of the flow conservation equations in (1.2) is redundant for each commodity k, we can arbitrarily select the value π_j^k for one node j. Accordingly, we set $\pi_1^k = 0$ for every commodity k = 2, 3, ... , n. (Recall that node 1 is the common origin for all commodities.) Then, the dual of [LDF], denoted [DDF], has the following formulation:

[DDF]

$$\text{minimize} \quad \sum_{k=2}^{n} \pi_k^k \tag{3.1}$$

subject to

$$\pi_j^k - \pi_i^k \le \alpha_{ij}^k \qquad \text{for all } (i,j) \in A,\ k \in P, \tag{3.2}$$

$$\pi_j^k - \pi_i^k \le \beta_{ij}^k \qquad \text{for all } (i,j) \in A,\ k \in S, \tag{3.3}$$

$$\sum_{k \in P} \alpha_{ij}^k + \sum_{k \in S} \beta_{ij}^k \le a_{ij} \qquad \text{for all } (i,j) \in A, \tag{3.4}$$

$$\sum_{k \in S} \beta_{ij}^k \quad \leq \quad b_{ij} \qquad \text{for all } (i,j) \in A, \text{ and} \tag{3.5}$$

$$\alpha_{ij}^k,\ \beta_{ij}^k \quad \geq \quad 0 \qquad \text{for all } (i,j) \in A,\ k \in P \cup S. \tag{3.6}$$

Formulation [DDF] has the following shortest path interpretation. Suppose we are given values of α_{ij}^k and β_{ij}^k that satisfy constraints (3.4) and (3.5). Then, by dropping constraints (3.4) and (3.5), we decompose [DDF] into (n–1) independent subproblems, one for each commodity k; the subproblem for commodity k corresponds exactly to the dual of a (directed) shortest path problem from node 1 (the origin) to node k (the destination) using the allocated costs α_{ij}^k and β_{ij}^k as arc lengths for the primary and secondary commodities k. In particular, the objective function value π_k^k for the k^{th} subproblem is the shortest distance from node 1 to node k using the primary or secondary allocated arc costs for $k \in P$ or $k \in S$. The dual ascent method exploits this observation by iteratively increasing the allocated arc costs for selected arcs and commodities in order to monotonically increase the shortest path length(s). To ensure that the allocated arc costs satisfy constraints (3.4) and (3.5), the method maintains and updates the slack in these two constraints for every arc (i,j). For given values of α_{ij}^k and β_{ij}^k, let p_{ij} and s_{ij} denote the *slack* in constraints (3.4) and (3.5), respectively. We refer to p_{ij} as the *primary slack* or *p-slack* for arc (i,j); s_{ij} is the *secondary slack* or *s-slack*.

Let us now examine how the method increases the allocated arc costs while maintaining dual feasibility. It initializes all of the allocated arc costs and node potentials to value zero, and sets $p_{ij} = a_{ij}$, and $s_{ij} = b_{ij}$. Consider a *primary commodity* $k \in P$. To increase the shortest path length π_k^k from node 1 to node k for this commodity, we must increase values of α_{ij}^k. However, increasing α_{ij}^k for an arc (i,j) that does not lie on the current (directed) shortest O–D (origin-to-destination) path does not increase π_k^k; furthermore, we cannot increase the α_{ij}^k values for arcs (i,j) that have zero p-slack p_{ij}. To easily identify arcs that belong to current shortest O–D path(s) for commodity k, the dual ascent method maintains and iteratively updates a (directed) cutset of arcs, say A(k), separating the commodity's origin and destination. Let N(k) denote the node subset that defines this cutset and contains the destination

node k (i.e., all the arcs in the cutset A(k) are directed into the nodes in N(k)); we refer to nodes in N(k) as the *labeled nodes for commodity k*. By construction, all the arcs directed from the unlabeled to the labeled nodes for commodity k must belong to some shortest O–D path for this commodity. At every step, the method:

(i) determines the minimum p-slack, say, δ among all arcs in the cutset,

(ii) reduces the p-slacks p_{ij} for all the cutset arcs (i,j) by δ,

(iii) increases α_{ij}^k for all the cutset arcs (i,j) by δ,

(iv) increases the node potentials π_j^k of all the labeled nodes j by δ,

(v) increases the dual objective function value by δ, and

(vi) labels a currently unlabeled node i whose incident cutset arc, say (i,j), is *critical*, i.e., the p-slack for arc (i,j) defined the minimum value δ.

We refer to these six operations collectively as one *labeling* step. The method stops increasing the allocated arc costs α_{ij}^k for commodity k when it has labeled the origin (node 1) for this commodity. For *secondary commodities* k, the procedure is very similar, except that the method increases β_{ij}^k (instead of α_{ij}^k, in operation (iii)) for cutset arcs (i,j), and defines the amount of increase δ (in operation (i)) as the minimum of the p-slack and the s-slack among all cutset arcs. (Recall that the β_{ij}^k values must satisfy both constraints (3.4) and (3.5)).

Our implementation differs from this conceptual description of the algorithm in two ways. First, instead of allocating costs for a single commodity in consecutive labeling steps, we cycle through the commodity list, performing one labeling step for each commodity in every cycle. Second, to calculate δ at each labeling step, the procedure does not explicitly require the current values of α_{ij}^k, β_{ij}^k and π_j^k. Therefore, we maintain and update only the *p-slack* p_{ij} and *s-slack* s_{ij} for each arc (i,j), the *dual objective function value* Z_D, and the set of *labeled nodes N(k)* for every commodity k. Consequently, the dual ascent procedure is deceptively simple to state, extremely easy to implement, and imposes only very modest data storage requirements.

The Dual Ascent Labeling Method

Step 0: *Initialization*

Set	$N(k) \leftarrow \{ k \}$	for all $k = 2, 3, \ldots, n$,	**labeled nodes**
	$p_{ij} = a_{ij}$	for all $(i,j) \in A$,	**p-slack**
	$s_{ij} = b_{ij}$	for all $(i,j) \in A$, and	**s-slack**
	$Z_D = 0$		**dual objective value**

Step 1: *Labeling cycle*

For $k = 2, 3, \ldots, n$ that satisfy the property that the root node $1 \notin N(k)$,

Step 1a: Labeling Step for Primary Commodities

if $k \in P$,

$f_i = \min \{p_{ij}: (i,j) \in A, j \in N(k)\}$ for all $i \notin N(k)$;

$\delta = \min \{f_i: i \notin N(k)\}$;

$i^* = \text{argmin } f_i$;

$p_{ij} \leftarrow p_{ij} - \delta$ for all $(i,j) \in A$ with $i \notin N(k), j \in N(k)$;

$Z_D \leftarrow Z_D + \delta$; and

$N(k) \leftarrow N(k) \cup \{ i^* \}$;

Step 1b: Labeling Step for Secondary Commodities

if $k \in S$,

$f_i = \min \{p_{ij}: (i,j) \in A, j \in N(k)\}$ for all $i \notin N(k)$;

$\delta = \min \{f_i: i \notin N(k)\}$;

$i^* = \text{argmin } f_i$;

$p_{ij} \leftarrow p_{ij} - \delta$ for all $(i,j) \in A$ with $i \notin N(k), j \in N(k)$;

$s_{ij} \leftarrow s_{ij} - \delta$ for all $(i,j) \in A$ with $i \notin N(k), j \in N(k)$;

$Z_D \leftarrow Z_D + \delta$; and

$N(k) \leftarrow N(k) \cup \{ i^* \}$;

next k;

If the root node $1 \in N(k)$ for all $k = 2, 3, \ldots, n$, **STOP**; **else** repeat Step 1.

The final value of Z_D obtained using this procedure is a valid lower bound on the optimal value of the TLND problem since it corresponds to a feasible solution to the dual (3.1)–(3.6) of the linear programming relaxation of the problem. Our previous description of the method's underlying principle intuitively justifies the validity of this procedure. A formal proof of validity based on induction would exploit the observations that, at each stage, (i) π_j^k is 0 for all nodes $j \notin N(k)$, (ii) for every node $j \in N(k)$, π_j^k increases by the same amount, thus preserving feasibility with respect to constraints (3.2) and (3.3) for arcs (i,j) within N(k), and (iii) the nonnegativity of the p-slacks and s-slacks guarantees the feasibility of the allocated costs with respect to constraints (3.4) and (3.5). For each commodity, the procedure labels one node at each step, until it has labeled the origin. Since the flow-based formulation uses (n–1) commodities, the dual ascent labeling algorithm terminates in $O(n^2)$ steps.

The labeling method extends very easily to the general MLND problem. In particular, the labeling step for a l-level commodity will define δ as the minimum l'-level slack (in the constraints analogous to (3.5)) for $l' = 1, 2, \ldots, l$, over all the current cutset arcs for that commodity. The step will reduce all these l slacks by the value δ for every arc in the cutset. Thus, for a problem with L levels, the method requires $O(Ln^2)$ basic operations.

The dual ascent labeling method has several variants. For instance, instead of cycling through all the commodities, we could divide the procedure into two stages. The first stage would examine all primary (or secondary) commodities in sequence, and terminate when it has labeled the source node for all primary (secondary) commodities. The second stage then repeats this procedure for secondary (primary) commodities. Even in the single stage procedure, we can sequence the commodities in various alternative ways, e.g., first examine the secondary commodities, and then the primary commodities within each cycle. Some preliminary computational results suggested that our choice of commodity sequence (single stage, primary first, and secondary next) often generates the best lower bounds.

We could also consider alternative multiplier initialization schemes prior to performing dual ascent. Our current procedure initializes all allocated arc costs and node potentials to zero value. Alternatively, we might use an "advanced start" procedure that begins with, say, equal allocations of the secondary costs b_{ij} over all commodities, and equal allocations of the incremental cost e_{ij} over all primary commodities; we can then use a shortest path routine to initialize the node potentials. We found our simple (zero) initialization scheme to be quite effective in solving the test problems.

3.2 Dual-based Heuristic

Besides providing a lower bound Z_D on the optimal value of the TLND problem, the dual solution can also generate a feasible design. We use this design as a starting point for local improvement by an Add–Drop heuristic. To construct an undirected heuristic design for the original problem from the dual solution to the directed problem, we install (undirected) *edges* corresponding to certain (directed) *arcs* have zero dual slack. We next describe two alternative ways to construct the dual-based heuristic design.

Method 1: *(Dual-based primary design + secondary completion)*

> Select as primary edges all the arcs with zero primary slack in the final dual solution, and apply the optimal secondary completion procedure (defined in Section 2) to obtain a feasible TLND solution.

This method exploits the following property of the labeling algorithm: at termination, the root node is connected to every primary node along at least one path containing only zero p-slack arcs. Thus, when the labeling algorithm terminates, the set of zero p-slack edges contains a subtree connecting all the primary nodes (we discard all zero p-slack edges that are incident to secondary leaf nodes). Method 1 completes this subtree by finding the (secondary) minimum spanning tree for the residual condensed graph (see Section 2).

Method 2: *(Dual-based primary + secondary design)*

> Select as primary edges all the arcs with zero primary slack, and install secondary facilities on all the arcs with zero secondary slack in the final dual solution.

Instead of applying the optimal secondary completion procedure, Method 2 directly constructs a feasible TLND solution by choosing all zero s-slack arcs as secondary edges. Again, we can prove that, at termination, the selected edges form a connected, feasible design.

Our implementation uses Method 1 to construct the starting solution for subsequent local improvement. To improve the dual-based starting solution, we apply an *Add–Drop heuristic*. The Add–Drop procedure attempts to decrease the cost of the heuristic solution by iteratively adding secondary nodes to the primary subtree or deleting secondary nodes from this subtree.

Let S_P denote the subset of secondary nodes that the primary subtree spans in the current heuristic solution (the incumbent). For each secondary node $j \notin S_P$, the *Add* procedure evaluates the potential cost saving if we introduce node j into the primary subgraph. To evaluate this saving, the procedure:

(i) finds the minimum cost tree (using primary costs) spanning all the nodes of $P \cup S_P \cup \{j\}$; and,

(ii) applies the optimal secondary completion procedure with respect to this tree.

The resulting design has the least cost among all designs that contain the secondary nodes $S_P \cup \{j\}$ in the primary subtree. The cost differential between this solution and the current incumbent gives the cost saving for including node j in the primary subtree. The Add procedure evaluates this savings for all nodes $j \notin S_P$, and introduces the best node, say j^* (i.e., the node that produces the maximum savings) into the primary subgraph by updating $S_P \leftarrow S_P \cup \{j^*\}$. The procedure terminates if none of the secondary nodes generate savings.

The *Drop* procedure evaluates the savings obtained by dropping secondary nodes belonging to the current primary subtree. Again, we evaluate the

savings by finding, for each node $j \in S_P$, the minimum primary subtree spanning all nodes of $P \cup S_P \setminus \{j\}$, followed by the optimal secondary completion. At each step, we drop the secondary node that produces the maximum savings from the primary subtree. The method terminates when none of the secondary nodes produce savings when dropped from the current primary subtree. Our implementation alternately applies the Add and Drop procedures until no additional improvement is possible.

4. Implementation and Computational Results

Since the general TLND problem is new, the literature does not contain any standard test problems. We, therefore, tested the dual ascent algorithm on numerous randomly generated problem instances. Our computational experiments were designed to assess the sensitivity of the dual ascent method's performance (quality of solutions and computation time) to variations in (i) problem size and structure (number of nodes and edges, and the relative proportion of secondary nodes), and (ii) cost structure (various norms for the primary costs, and different primary-to-secondary cost ratios).

Our test problems varied in size from 100 nodes (50 primary nodes) and 500 edges, to 500 nodes (200 primary nodes) and 5000 edges. The directed flow-based formulation [DF] for our largest test problem contains 20,000 integer variables, 5 million continuous variables, and more than 5 million constraints. We used four different cost structures to generate the primary costs: (i) Euclidean costs, (ii) random (uniform) costs, (iii) manhattan (or L_1) costs, and (iv) L_∞ costs. For each primary cost structure, we considered two methods for computing secondary costs:

(i) the *proportional cost method*: using a constant (but randomly chosen) primary-to-secondary cost ratio for all edges; and

(ii) the *nonproportional cost method*: using different (randomly chosen) primary-to-secondary cost ratios for different edges.

Our initial experimentation indicated that problems with approximately equal numbers of primary and secondary nodes were the most difficult to solve. Therefore, except for the HND special case (which contains only two

primary nodes), all other problem categories have a comparable number of primary and secondary nodes. For each category, we tested 3 problem instances (generated using different random number seeds); all the performance measures that we report represent average values over these 3 instances. We also tested the corresponding Steiner network version of each problem instance, obtained by setting all secondary costs to value zero.

To solve the problems optimally, we could incorporate the dual ascent method in a branch-and-bound scheme; at each node of the branch-and-bound tree, the dual ascent method generates lower and upper bounds for the current partition. We did not implement this branch-and-bound algorithm; we applied the dual ascent labeling method and dual-based heuristic only once to each test problem. For almost all problem instances, the gap between the dual ascent lower bound and heuristic solution cost was less than 1% (even for the Steiner network versions), suggesting that this approximate method is very effective.

4.1 Random problem generator

We used a random network generator to construct the test problems. The user first specifies the required number of nodes (n), number of primary nodes (p), number of edges (m), and the cost structure—Euclidean, random, manhattan, or L_∞ primary costs, with either constant or varying primary-to-secondary cost ratios.

The problem generator randomly locates the required number of nodes on a 1000 x 1000 grid on the plane, and designates the first p nodes as primary nodes. To ensure problem feasibility, the method first constructs a random spanning tree, and then randomly adds the required number of additional edges. The generator sets the *primary* cost a_{ij} of each edge {i,j} equal to the integer part of the user-specified metric: *Euclidean, manhattan,* or L_∞ distance between nodes i and j, or a *random* integer (uniformly distributed) between 0 and 1000. The generator sets the secondary cost b_{ij} equal to a_{ij}/r_{ij} for each edge {i,j}. We choose the reciprocal of the primary-to-secondary cost ratio r_{ij}, which is either constant for all edges (the proportional costs case) or varies by edge (nonproportional costs), from a uniform distribution between two user-

specified limits ρ_l and ρ_u, with $0 \leq \rho_l \leq \rho_u \leq 1$. For our test problems, we used $\rho_l = 0$ and $\rho_u = 1$.

4.2 Implementation of the Dual Ascent Method

We implemented the dual ascent labeling method and the dual-based heuristic for the TLND problem in FORTRAN on an IBM 4381 computer (compiled at optimization level 3 using a VSFORTRAN compiler). The program also incorporates an optional preprocessing module that performs the following functions (see Section 2):

(i) identifies included primary edges, and aggregates the corresponding primary nodes (Section 2.3);
(ii) identifies excluded secondary edges (Section 2.2); and,
(iii) identifies excluded primary edges (Section 2.4).

To gauge the effectiveness of preprocessing, we tested a few problems both with and without preprocessing. As the results of Section 4.4 indicate, preprocessing is very effective in reducing the computational time for the dual ascent labeling algorithm as well as heuristic local improvement.

Our preliminary experiments indicated that the choice of root node impacts the quality of both the upper and lower bounds. (Recall that, to formulate the problem as a multicommodity flow model, we are free to choose any one of the p primary nodes as the common root node for the (n–1) commodities.) To identify good bounds, we can apply the dual ascent method p times, once for each primary node as the root node. Since this repeated application is very time consuming for large problems, our final implementation uses the following stopping criterion to terminate this procedure early: starting with node 1 as the root node, we sequentially select each primary node as the root, and repeat the dual ascent computations until the % gap between the best upper and lower bounds drops below 2% (or until the algorithm exhausts all p choices of the root node). Also, for a given root node, we apply the local improvement procedure to a dual-based starting solution only when this solution has a lower total cost than the previous best starting solution. For almost all of our test problems, the dual ascent method

terminated after the first application, i.e., the upper and lower bounds obtained with node 1 as the root node differed by less than 2%.

4.3 Data Structures

Identifying all the current cutset arcs for each commodity is the main computationally intensive operation in the dual ascent procedure. Therefore, refining the data structures to speed up this dual ascent step can have a significant impact on total computational time, especially for large problems. Our implementation incorporates the following data structures.

Recall that we transformed the undirected problem containing m edges to a directed version with 2m directed arcs. Our implementation employs a forward star pointer representation to easily identify all the arcs emanating from a given node. That is, we maintain an array of size 2m x 2 containing, for each node i and every emanating arc (i,j), two pieces of information: the arc index and the node j. We maintain the arcs emanating from node i in consecutive rows in this array. For each node i, we also maintain two pointers identifying the starting and ending rows for its incident arc list. For each commodity, we maintain a n-vector indicating the labeled nodes for that commodity; element i of this vector has value 1 if node i is labeled, and 0 otherwise. Other data structures include two 2m-vectors to store the current primary and secondary slacks for each arc.

These lists and vectors enable us to quickly identify the cutset arcs at each labeling step without imposing extensive storage requirements. Consider a dual ascent step for commodity k. Using the label vector, we sequentially examine each unlabeled node for this commodity. For an unlabeled node i, we examine each of its incident arcs; every arc (i,j) whose other end j is labeled (for commodity k) belongs to the current cutset for commodity k. Having identified the cutset arcs, we can easily find the critical arc (the arc that limits the amount of increase δ in the dual value), and perform the updating steps (label a node, and reduce the slacks). Observe that we can further reduce computation time by maintaining a n x 2m matrix that directly stores the indices of all arcs belonging to the current cutset for each commodity. However, this scheme increases the storage requirements by an order of

magnitude (e.g., 5 million integers for a 500 node, 5000 edge problem), which might possibly even increase the total CPU time. Finally, we use only a naive implementation of Kruskal's minimum spanning tree algorithm for the optimal secondary completion step in the heuristic improvement subroutine. For large problems, our experience suggests that heuristic improvement requires considerably more computational time than dual ascent; therefore, improving the implementation of the heuristic method (e.g., using more efficient sorting methods) might significantly improve performance and increase the problem sizes that we can solve using this algorithm. Furthermore, for problems defined on the plane, we could potentially exploit ideas from computational geometry (see, for example, Shamos [1978]) to efficiently solve the minimum spanning tree subproblems.

4.4 Computational Results

To evaluate the effectiveness and robustness of the dual-based method, we focus on two algorithmic performance measures:

(i) algorithmic effectiveness, measured by the *% gap*, defined as the difference between the best upper and lower bounds as a % of the best lower bound; and,

(ii) computational time, measured by *CPU time* (in seconds, on the IBM 4381), classified into 3 components:
   (a) *Set-up time*: to read the input data, to initialize the data structures, and to perform preprocessing;
   (b) *Ascent time*: to perform the dual ascent; and
   (c) *Add-drop time*: to construct and improve the dual-based heuristic solution.

   Note that the IBM 4381 is a mini-computer that is considerably slower than, say, the Cray X-MP supercomputer (Beasley [1989]) or the VAX 8700 computer (Chopra et al. [1990]).

To isolate the effects of different factors—problem size, cost structure, and primary-to-secondary cost ratio—we performed various sets of experiments, for each set varying one dimension while keeping other parameters fixed.

For one of these experiments, we also compared the performance of the dual-based heuristic solution with the Minimum Spanning tree and Approximate Steiner tree methods described in Balakrishnan et al. [1991].

We identify each test problem with a label *(p/n, m, cost structure, cost ratio)* that distinguishes its parameters. In this notation, p is the number of primary nodes, n is the total number of nodes in the network, and m is the number of (undirected) *edges* of the original graph. The four *cost structures* we tested are Euclidean (E), random (R), manhattan (M), or L_∞ (I). The primary-to-secondary *cost ratio* is either proportional (P) (i.e., a constant ratio for all edges) or nonproportional (N). The following subsections summarize our results and inferences concerning the robustness of the dual-based algorithm.

Effectiveness of Preprocessing and Impact of Problem Size

Table I summarizes the impact of preprocessing, and problem size (including variations in the proportion of primary nodes) on the effectiveness of the algorithm. For this part of the computational study, we used only Euclidean (E) and random (R) cost structures, and considered both proportional (P) and nonproportional (N) secondary costs. The results of Table I confirm the effectiveness of the dual-based algorithm: the average % gap ranges from 0% to only 0.89%. Although preprocessing does not affect this % gap, it decreases the total computation time considerably for larger networks; for the (300/400, 2000, EP) problem category, preprocessing reduced the total CPU time by 78%. Observe that the savings in dual ascent time accounts for most of the reduction in total computation time due to preprocessing.

Table I also reports a more direct measure of preprocessing effectiveness, namely, the proportion of primary nodes that preprocessing was able to aggregate (using the included primary edges test described in Section 2.3). On average, preprocessing aggregates 54% of the primary nodes, with higher percentage aggregation for problems containing a larger proportion of primary nodes. All of our subsequent computational tests include preprocessing.

Finally, we note that the heuristic improvement (add-drop) algorithm consumes less time than the dual ascent procedure for nearly all the test problems. As a general trend, the add-drop time, as a percentage of total time, appears to decrease as the problem size increases. The add-drop algorithm reduced the cost of the dual-based starting design from 0 to 18%. Also, we observed that the method often dropped many more nodes from the primary subtree than it added. For instance, for the 500 node, 5000 edge problems that we discuss next, the algorithm typically added fewer than 10 nodes to the primary subtree in the dual-based starting solution, but dropped more than 50 nodes from this subtree. This phenomenon suggests that we might be able to reduce the heuristic improvement time by using a different heurisitic initialization scheme, e.g., by selecting as primary edges only a subset of the arcs with zero p-slack in the final dual solution (see Section 3.2).

Impact of Cost Structure:

Table II summarizes the effect of different cost structures. For this comparison, we selected a fixed network size of 500 nodes (200 primary nodes) and 5000 edges. The average gap between the upper and lower bounds is less than 0.83% for all problem categories, and the results suggest no discernable difference between the % gaps or computation times across the different primary cost structures. However, for a given primary cost structure, the gaps are higher when the primary-to-secondary cost ratio varies across edges. This behavior is consistent with the heuristic worst-case analysis in Balakrishnan et al. [1991] (in that analysis, we observed that the worst-case bound for the nonproportional case is larger than the bound for the proportional case).

Table II also reports computational statistics for the Steiner network versions (with all the secondary costs set to zero) of our 500 node, 5000 edge test problems. Since the Steiner problem has zero secondary costs, we need not cycle through the list of secondary commodities during the dual ascent labeling iterations. Consequently, as we observe in Table II, Steiner problems require considerably less ascent time than their TLND counterparts. The % gaps for the Steiner problems range from 0.75% to 1.22%, and are uniformly higher than the gaps for the corresponding TLND problems. This increase in % gap results mainly from the lower value of the denominator, i.e., since the Steiner problem has zero secondary costs, its optimal value and hence its

lower bound is smaller than the lower bound for the corresponding TLND problem with positive secondary costs. In summary, our results for the Steiner problem confirm the findings by Wong [1984]: namely, for this class of problems, dual ascent is a very effective solution strategy.

Impact of Primary-to-Secondary cost ratio:

Table III summarizes the effect of varying the primary-to-secondary cost ratio on the % gap. Again, we fixed the problem size at 500 nodes (200 primary nodes) and 5000 edges, and we considered only the proportional cost case. We considered three values of the primary-to-secondary cost ratio: r = 2, 4, and 10. The results reported in Table III suggest that the average % gap, which is less than 0.91% for all cases, does not depend on the cost ratio. However, the average total time seems to decrease as this ratio increases.

Comparative performance of AST and MST heuristics:

Table III also contains the solution values of the Minimum Spanning Tree (MST) and the Primary Node Completion (PNC) heuristics analyzed in Balakrishnan et al. [1991]. The MST heuristic installs primary facilities on all edges of the minimum spanning tree of the given network. The PNC heuristic, a special type of approximate Steiner tree heuristic, uses the primary spanning tree (the minimum tree spanning only the primary nodes) to approximate the Steiner tree. The method installs primary facilities on the edges of this primary spanning tree, and applies optimal secondary completion to determine the configuration of secondary facilities. The composite heuristic chooses the better among the MST and PNC solutions. As Table III indicates, the composite heuristic performs substantially worse than our dual-based heuristic. For all three primary-to-secondary cost ratios, the average cost of the composite heuristic (as a % of the best lower bound) exceeds 20%. Furthermore, for all our test problems, the PNC heuristic gave better heuristic solutions than the MST heuristic.

Results for HND problems:

Table IV studies the special class of TLND problems containing only 2 primary nodes. The first two HND problems correspond to Current et al.'s (1986) test networks containing, respectively, 15 nodes and 33 edges, and 21 nodes and 39 edges. For both problems, our algorithm terminates almost

instantaneously; it finds the optimal solution for the first problem, and proves that the heuristic solution for the second problem is at most 0.75% more expensive than the optimal solution. The remaining problem categories reported in Table IV correspond to random problems with Euclidean primary costs and varying network sizes. For these problems, the average % gaps are less than 0.1% suggesting that the dual-based algorithm is also very effective for solving HND problems. Note that the number of edges in the largest problem in this test group is more than two orders of magnitute larger than those considered by Current et al. in their original study of the HND problem.

5. Conclusions

This paper has developed an algorithmic approach for solving a new class of multi-level network design problems. We have identified some structural properties of optimal solutions that enable preprocessing, developed a dual ascent method for generating lower and upper bounds, and performed extensive computational testing. The dual ascent method is a very simple and efficient technique, and does not require solving expensive linear programs using general purpose LP solvers (see, for example, Chopra et al. [1990]). Our computational results demonstrate that this method is quite robust and generates good upper and lower bounds: for a variety of cost structures, the average gap was less than 1% for problems ranging in size from 100 nodes and 500 edges to 500 nodes and 5000 edges. The algorithm performed equally well for the Steiner network and hierarchical network design special cases.

Future research might further explore the polyhedral structure of the TLND (and more generally the MLND) problem and its variants, and test the dual ascent method for multi-level problems with variable (flow-dependent) edge costs. One of the MLND model's main shortcomings is that it does not incorporate more stringent connectivity requirements (e.g., all primary nodes must be 2-connected). With the emerging emphasis on topological robustness and reliability, studying enhanced models that incorporate connectivity constraints is a very promising area for further investigation.

The successful polyhedral and computational investigations of Groetschel, Monma, and Stoer [1990a, 1990b], who have studied a single hierarchy (in edges) network design problem with varying connectivity requirements for primary and secondary nodes, might prove to be useful in pursuing these extensions. So might the work by Goemans and Talluri [1991] and Talluri [1991] on k-connected network design problems. The results in this paper suggest that on average the relative disparity between the optimal objective function values of the TLND problem and its linear programming relaxation are quite small. Another potentially fruitful avenue of investigation would be a probabilistic analysis that might analytically confirm these findings.

Table I. Effect of Preprocessing and Problem Size (Euclidean and random costs)
(All computational times in seconds on IBM 4381)

Problem category †	Without preprocessing			With preprocessing				
	Average % gap††	Average Ascent time	Average add-drop time	Avg % of primary nodes aggregated	Average % gap††	Average set-up time	Average Ascent time	Average add-drop time
(50/100, 500, EP)	0.21	11	4	40	0.21	2	8	3
(50/100, 500, EN)	0.15	6	3	53	0.13	2	4	2
(80/200, 1000, EP)	0.08	56	59	39	0.08	7	48	57
(80/200, 1000, EN)	0.48	66	75	35	0.48	7	53	71
(300/400, 2000, EP)	0.01	842	189	77	0.02	25	115	82
(300/400, 2000, EN)	0.00	685	234	74	0.00	25	188	212
(50/100, 500, RP)	0.06	6	2	53	0.07	2	4	2
(50/100, 500, RN)	0.09	5	4	53	0.07	2	3	3
(80/200, 1000, RP)	0.89	2710	42	40	0.89	7	1900	40
(80/200, 1000, RN)	0.68	274	150	43	0.65	7	124	94
(300/400, 2000, RP)	0.02	307	72	71	0.02	26	131	57
(300/400, 2000, RN)	0.01	1286	311	73	0.01	25	351	223

† E denotes Euclidean cost structure
R denotes Random cost structure
P denotes Proportional costs (same primary-to-secondary cost ratio for all edges)
N denotes Nonproportional costs

†† % gap = (best upper bound – best lower bound) ÷ best lower bound

Table II. Effect of Cost Structure (network size: **500 nodes, 200 primary nodes, 5000 edges**)
(All computational times in seconds on IBM 4381)

Problem category †	TLND problem			Steiner problem		
	Average % gap††	Average Ascent time	Average add-drop time	Average % gap††	Average Ascent time	Average add-drop time
(EP)	0.18	3358	1197	1.17	884	2394
(EN)	0.76	2072	2373	0.75	283	1764
(RP)	0.34	2919	1212	0.76	588	2368
(RN)	0.56	1937	1489	0.66	561	1191
(MP)	0.16	4066	1094	1.08	1075	2738
(MN)	0.64	1341	2439	1.05	166	1856
(IP)	0.10	3936	1148	0.90	920	2284
(IN)	0.83	1820	2448	1.22	212	1665

† E denotes Euclidean cost structure
R denotes Random cost structure
M denotes Manhattan cost structure
I denotes L_∞ cost structure
P denotes Proportional costs
N denotes Nonproportional costs

†† % gap = (best upper bound – best lower bound) ÷ best lower bound

Table III. Effect of Primary-to-Secondary Cost Ratio
(network size: 500 nodes, 200 primary nodes, 5000 edges)
(All computational times in seconds on IBM 4381)

Problem Category	Average % gap††	Average Ascent time	Average add-drop time	MST Heuristic % gap	PNC Heuristic % gap
Euclidean cost structure, Cost Ratio r = 2	0.59	2719	2405	27.37	21.56
Euclidean cost structure, Cost Ratio r = 4	0.91	1303	2953	47.94	26.53
Euclidean cost structure, Cost Ratio r = 10	0.66	1272	2677	72.17	23.40

†† % gap = (best upper bound – best lower bound) ÷ best lower bound

Table IV. Hierarchical Network Design (HND) Problems
(All computational times in seconds on IBM 4381)

Problem Category	Average % gap††	Average Ascent time	Average add-drop time
Current et al., Problem # 1	0.00	0	0
Current et al., Problem # 2	0.75	0	0
Euclidean cost structure 100 nodes,500 edges	0	8	0
Euclidean cost structure 200 nodes, 1000 edges	0.03	125	3
Euclidean cost structure 400 nodes, 2000 edges	0.02	754	28
Euclidean cost structure 500 nodes, 5000 edges	0.01	3265	79

†† % gap = (best upper bound – best lower bound) ÷ best lower bound

References

BALAKRISHNAN, A., T. L. MAGNANTI, and P. MIRCHANDANI. 1991. The Multi-level Network Design Problem. Working Paper, Operations Research Center, Massachusetts Institute of Technology, Cambridge.

BALAKRISHNAN, A., T. L. MAGNANTI, and R. T. WONG. 1989. A Dual-Ascent Procedure for Large-Scale Uncapacitated Network Design. *Opns. Res.* **37**, 716–740.

BALAKRISHNAN, A., and N. R. PATEL. 1987. Problem Reduction Methods and a Tree Generation Algorithm for the Steiner Network Problem. *Networks* **17**, 65–85.

BEASLEY, J. E. 1984. An Algorithm for the Steiner Problem in Graphs. *Networks* **14**, 147–159.

BEASLEY, J. E. 1989. An SST-based Algorithm for the Steiner Problem in Graphs. *Networks* **19**, 1–16.

CHOPRA, S., E. GORRES, and M. R. RAO. 1990. Solving the Steiner Tree Problem on a Graph using Branch and Cut. Working Paper, Northwestern University, Evanston.

CURRENT, J. R., C. S. REVELLE, and J. L. COHON. 1986. The Hierarchical Network Design Problem. *Eur. J. Oper. Res.* **27**, 57-66.

DREYFUS, S. E., and R. A. WAGNER. 1972. The Steiner Problem in Graphs. *Networks* **1**, 195–207.

ERLENKOTTER, D. 1978. A Dual Based Procedure for Uncapacitated Facility Location. *Opns. Res.* **26**, 992-1009.

GAREY, M. R., and D. S. JOHNSON. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco.

GOEMANS, M. X., and TALLURI, K. T. 1991. 2-change for k-connected Networks. *Operations Research Letters*.

ORLIN, J. B. 1991. Personal communication.

OSBORNE, L. J., and B. E. GILLETT. 1991. A Comparison of Two Simulated Annealing Algorithms applied to the Directed Steiner Problem in Networks. *ORSA J. Comput.* **3**, 213–225.

PIRKUL, H., J. CURRENT, and V. NAGARAJAN. 1991. The Hierarchical Network Design Problem: A New Formulation and Solution Procedures. *Trans. Sci.* **25**, 175-182.

SASTRY, T. 1987. Some Bounds on the Gaps Between the Dual Ascent, Linear Programming and the Integer Programming Solutions to the Steiner Problem. Unpublished manuscript, Sloan School of Management, Massachusetts Institute of Technology, Cambridge.

SHAMOS, M. I. 1978. *Computational Geometry*. Unpublished Ph.D. thesis, Yale University, New Haven.

SHIER, D. 1991. A Heuristic for the Two-level Network Design Problem. Presented at the TIMS/ORSA Annual Meeting, Nashville.

TALLURI, K. T. 1991. *Issues in the Design and Analysis of Survivable Networks*. Unpublished PhD dissertation, Operations Research Center, Massachusetts Institute of Technology, Cambridge.

WINTER, P. 1987. Steiner Problem in Networks: A Survey. *Networks* **17**, 129–167.

WONG, R. T. 1984. Dual Ascent Approach for Steiner Tree Problems on a Directed Graph. *Math. Prog.* **28**, 271–287.