ELSEVIER

Discrete Mathematics 242 (2002) 65–92

DISCRETE MATHEMATICS

www.elsevier.com/locate/disc

A branch-and-cut approach for minimum cost multi-level network design☆

Sunil Chopra[a], Chih-Yang Tsai[b,*]

[a] *J.L. Kellogg Graduate School of Management, Northwestern University, USA*
[b] *Department of Business Administration, The State University of New York at New Paltz, 75 South Manheim Boulevard, New Paltz, NY 12561-2443, USA*

Received 29 April 1998; revised 3 August 2000; accepted 28 August 2000

Abstract

Network design models with more than one facility type have many applications in communication and distribution problems. Due to their complexity, previous studies have focused on finding good heuristic solutions. In this study, we develop algorithms that solve the multi-level network design problem to optimality. In our approach, the problem is converted to a Steiner tree problem and is solved by a branch-and-cut approach. Our computational study shows that the approach outperforms a dual ascent approach in the literature (Mirchandani, INFORMS J. Comput. 8 (3) (1996) 202) not only on solution times but also on the quality of the solutions.

Keywords: Network design; Integer programming; Branch-and-cut; Dual ascent

1. Introduction

Network design problems have received a great deal of attention among researchers and practitioners in recent years due to applications in designing telecommunications, transportation, and distributed computer networks. Since the costs involved in setting up a new network or expanding an existing one is usually tremendously high, even a small percentage improvement in the cost can mean significant dollar savings [3,7]. In a distributed computer system, the problem is to find a minimum cost design that assigns and links each terminal to a backbone computer such that demands from all terminals are met and various network constraints are satisfied such as capacities on each node and link (see Gavish [18] for details). The network may have a centralized backbone computer [18] or several backbone computers whose locations either are specified [1] or result from the decision [20]. Routing algorithms that minimize

☆ Partially supported by SUNY-New Paltz Research and Creative Projects Grant.
* Corresponding author. Tel.: +1-845-257-2934; fax: +1-845-257-2947.
E-mail address: tsaic@matrix.newpaltz.edu (C. Tsai).

0012-365X/02/$ - see front matter
PII: S0012-365X(00)00404-0

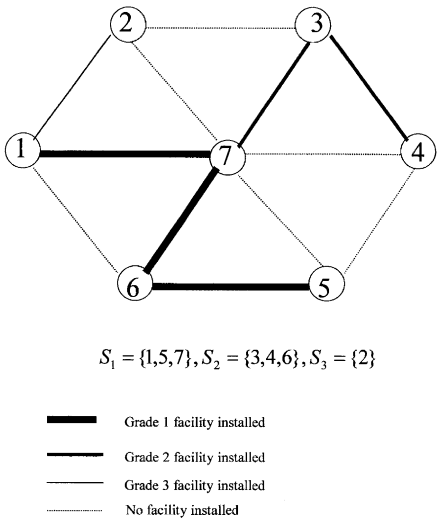

Fig. 1. $G = (N, E)$, 3 levels.

delays on a given network are addressed in [23,26]. Assigning primary routes and capacities simultaneously on the links of a given network is studied by Gavish and Altinkemer [19].

The Multi-Level Network Design (MLND) problem involves installing links of different grades between nodes depending on their demands and other requirements. A higher grade link has higher capacity but also incurs higher installation cost. This problem can be represented on an underlying undirected graph $G=(N,E)$, whose nodes N need to be linked while edges E correspond to potential links that may be established. Consider the graph in Fig. 1. Nodes $\{1,5,7\}$ require connections using grade 1 facilities (highest grade); nodes $\{3,4,6\}$ require connections using at least grade 2 facilities, while node 2 requires connections using at least grade 3 facilities. There are costs associated with installing each grade facility on each edge. The objective is to minimize total installation cost while ensuring that nodes are connected through appropriate grade facilities. Fig. 1 contains a potential solution to the problem.

The importance and potential applications of MLND model in power, transportation and communication systems are discussed in [4,5,13,15,22,25]. In telecommunications, grades may represent different types of cables ranging from fiber optic cables, DS3(T3 link), DS1(T1 link) to DS0 where a standard DS0 has the capacity of carrying one voice circuit; the capacity of a DS1 is equivalent to that of 24 DS0 while a DS3 holds 28 DS1; optical systems have nonstandard capacity and have a bandwidth between 90 Mbs to 1.1 Gbs [7]. In [7], a real application of a decision support system for designing multi-level networks developed and used by NYNEX (now Bell Atlantic) exemplifies the importance and complexity of the problem.

The simplest form of multi-level network design problems is the Hierarchical Network Design (HND) problem where all nodes are spanned by a tree with a path between

two given primary nodes on the network requiring higher grade links on every edge of the path [14,15,25]. Current et al. [14] propose a heuristic to obtain an upper bound to HND problem and Pirkul et al. [25] develop a Lagrangian relaxation based heuristic that produces a tighter bound. Duin and Volgenant [15] transform a HNDP with n nodes and m edges into a directed Steiner tree problem on a directed graph with $2n$ nodes and $4m+n$ directed arcs and introduce some graph reduction methods. Current and Pirkul [13] consider HND problem with transshipment facilities required on nodes connecting and converting traffics between two different grades of links. The instance of MLND problem with two levels (referred to as TLND problem by Balakrishnan et al. [4,5]) has been considered by Duin and Volgenant [17] and Balakrishnan et al. [4,5]. In [17], Duin and Volgenant develop a heuristic for TLND problem by combining heuristics proposed for the Steiner tree problem and HND problem. Balakrishnan et al. [4,5] discuss the MLND model that addresses topological design trade-offs in hierarchical networks requiring higher-grade interconnections for certain critical nodes. They define the problem on an undirected graph whose nodes are partitioned into L levels. Each edge of the network is allowed one of L different facility types, with higher-grade facilities requiring higher non-negative fixed costs.

In [5], Balakrishnan et al. develop a heuristic for TLND problem whose solution is guaranteed to be within $4/3$ of the optimal solution. In [4] they develop a dual ascent algorithm for TLND problem. This procedure is used by them to obtain solutions within 0.9% of optimality on average from instance sets with up to 500 nodes and 5000 edges. The general multi-level network design problem is studied by Mirchandani in [22] where more than two grades of connection are required. In [22] Mirchandani extends results in [4,5] from TLND problem to MLND problem. He considers instances with 400 and 800 nodes and up to 5 levels. Using a dual based algorithm he obtains gaps under 6% on average, even though in some instances gaps are as large as 13%.

In our notation a level r facility is a higher grade facility than a level $r+1$ facility. Thus level 1 nodes require interconnections with level 1 facilities, which are the highest grade. The objective is to select a minimum cost 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. Costs of installing transshipment or switching facility at the connecting node between two different grades are ignored (as in Balakrishnan et al. [4,5,22]), although it can be considered as an extension of our formulation. In [4,5], Balakrishnan et al. point out that HND problem is $\mathcal{NP}$-hard even in the case when all edges have the same primary to secondary cost ratio, or if the primary costs are 1 and the secondary costs are either 0 or 1. Thus, MLND problem is also $\mathcal{NP}$-hard.

In this study, we propose an extended formulation that exploits the fact that higher-grade facilities are at least as expensive as lower-grade facilities. We use Duin and Volgenant's idea [15] to transform a MLND problem on an undirected graph with n nodes, m edges and L levels into a directed Steiner tree problem on an extended directed graph with Ln nodes and $2Lm+(L-1)n$ arcs. We show that the directed Steiner cut formulation (see Chopra et al. [9]) applied to the extended graph is stronger than the

strongest formulations considered in [4], [5] or [22]. We then apply the branch-and-cut approach to the extended directed Steiner cut formulation. This approach allows us to solve most of the MLND problems tested (with up to 1200 nodes, 4200 edges and up to five levels) to optimality, in a reasonable amount of time using a Pentium III 500 MHz personal computer.

In Section 2, we describe the extended directed Steiner cut formulation for MLND problem and show it to provide a stronger LP-relaxation than the directed multicommodity formulation considered in [5,22]. Section 3 describes the branch-and-cut approach used by us. Section 4 discusses the computational study and summarizes the findings of this study.

2. Comparing formulations for MLND

In [5], Balakrishnan et al. show that the directed multicommodity (DM) formulation for TLND problem provides as strong an LP-relaxation as the undirected formulation with strengthening inequalities included. Mirchandani [22] uses the same argument to justify the use of the DM formulation to solve MLND problem. The DM formulation has a drawback in that the size of the LP to be solved becomes very large both in terms of the number of variables as well as the number of constraints. This makes it very difficult to even solve the LP-relaxation to optimality.

In this section we provide a directed Steiner cut (DSC) formulation for MLND problem and show that it provides an LP-relaxation that is as strong as the one given by the DM formulation. The DSC formulation typically has far fewer variables, though more constraints than the DM formulation. This makes it more amenable to a branch-and-cut approach. We then discuss the extended directed Steiner cut (EDSC) formulation for MLND problem and show that its LP-relaxation is strictly stronger than the LP-relaxation given by the DM and DSC formulations. The EDSC formulation has fewer variables, though potentially more constraints than the DSC formulation. This makes it even more suitable for a branch-and-cut approach.

2.1. Directed multicommodity (DM) formulation

Mirchandani [22] provides the following Directed Multicommodity (DM) formulation for MLND problem. Each edge (i,j) in the undirected graph $G=(N,E)$ is replaced by two directed arcs (i,j) and (j,i). The cost of installing a level-r facility on arc (i,j) or (j,i) is equal to the cost of a level-r facility on edge (i,j) and is denoted by b^r_{ij}. Thus the undirected graph G is replaced by the corresponding directed graph $D=(N,A)$. The node set N is partitioned into $S_1,S_2,\ldots,S_L$ and node set S_i, $i\in\{1,\ldots,L\}$ contains nodes that need to be covered by edges with facility of level i or higher, i.e. levels $1,2,\ldots,i$. Node $1\in S_1$ is declared the root and we seek an arborescence rooted at node 1 that spans all nodes in N. The grade of facilities installed on each arc is such that all

arcs on the unique path from node 1 to a node k (in the arborescence) have facilities of a grade at least as high as that required by node k.

For each node $k \in N\backslash\{1\}$, we define a commodity k and require that one unit of flow of this commodity be sent from the root node to node k. The level of a node $k \in S_i$ is denoted by lev(k), i.e. lev(k) $= i$. Variables in the DM formulation are described as follows:

$$u_{ij}^r = \begin{cases} 1 & \text{if arc } (i,j) \text{ contains a level } r \text{ facility,} \\ 0 & \text{otherwise,} \end{cases}$$

$f_{ij}^k =$ flow from i to j on arc (i,j) of commodity k.

The problem MLND is formulated as follows:

$$\text{Min} \sum_{r=1}^{L} \sum_{(i,j)\in A} b_{ij}^r u_{ij}^r$$

s.t.

$$\sum_{i\in N} f_{ij}^k - \sum_{i\in N} f_{ji}^k = \begin{cases} -1 & \text{if } j = 1 \\ 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases} \quad \text{for } j \in N,\ k \in N\backslash\{1\},\ (i,j),(j,i) \in A \tag{1}$$

$$f_{ij}^k \leqslant \sum_{r=1}^{\text{lev}(k)} u_{ij}^r \quad \text{for } k \in N\backslash\{1\},\ (i,j) \in A \tag{2}$$

$$u_{ij}^r \in \{0,1\},\ f_{ij}^k \geqslant 0 \quad \text{for } k \in N\backslash\{1\},\ (i,j) \in A,\ r \in \{1,\ldots,L\}.$$

Eqs. (1) ensure that one unit of flow goes from the root node to every other node k. Inequalities (2) ensure that facilities of lev(k) or higher provide sufficient capacity for the flow. Define the polytope

$$\text{LP}_1 = \{f_{ij}^k \geqslant 0, 1 \geqslant u_{ij}^r \geqslant 0 \mid (f,u) \text{ satisfies (1) and (2)}\}.$$

LP_1 defines the LP-relaxation of the DM formulation.

The DM formulation has $2|E|L + 2|E||N|$ variables and $|N|^2 + 2|E||N|$ constraints in the LP-relaxation. For a problem on a graph with 600 nodes, 2100 edges and 5 levels this equals 2 541 000 variables and 2 880 000 constraints. The size is so large that even the LP-relaxation cannot be solved to optimality. Thus all approaches using this formulation have relied on Lagrangian relaxation to approximately solve the LP-relaxation.

2.2. Directed Steiner cut (DSC) formulation

To try and obtain an optimal solution to the LP-relaxation we need a formulation that is not as large as the DM formulation. The Directed Steiner Cut (DSC) formulation that we propose next has some characteristics that facilitate the solution of the LP-relaxation.

For the DSC formulation consider the directed graph $D=(N,A)$ defined earlier. The variables u^r_{ij} are exactly as defined for the DM formulation.

Given any subset $X \subseteq N$, $1 \in X$, $X \neq N$, define $\delta(X)$ to be

$$\delta(X) = \{a = (p,q) \in A \mid p \in X, q \in N \setminus X\}.$$

Consider any node $k \in N \setminus X$. Each feasible solution to a MLND problem satisfies the *directed multilevel Steiner cut inequality* defined for node k,

$$\sum_{r=1}^{\mathrm{lev}(k)} \sum_{(i,j)\in\delta(X)} u^r_{ij} \geqslant 1. \tag{3}$$

Inequality (3) is a generalization of the directed Steiner cut inequality in [10] and ensures that there is a directed path with $\mathrm{lev}(k)$ or higher facilities from the root to the node k. Notice that there is a distinct inequality for each set X and node $k \in N \setminus X$. The problem MLND can be formulated as follows:

$$\mathrm{Min}\left\{ \sum_{r=1}^{L} \sum_{(i,j)\in A} b^r_{ij} u^r_{ij} \mid u \in \{0,1\}, u \text{ satisfies all inequalities (3)} \right\}.$$

This is referred to as the DSC formulation. Define the polytope

$$\mathrm{LP}_2 = \{1 \geqslant u^r_{ij} \geqslant 0 \mid u \text{ satisfies all inequalities (3)}\}.$$

LP_2 is the LP-relaxation of the DSC formulation.

The DSC formulation has only $2|E|L$ variables but an exponential number of constraints. In particular, for the problem with 600 nodes, 2100 edges and 5 levels, we end up with only 21 000 variables. Even though we have an exponential number of constraints, a cutting plane approach has a chance to give us an optimal solution to the LP-relaxation.

Solving the LP-relaxation of DSC to optimality would not have much value if it provides a weaker bound for the integer optimum than the DM formulation. However next we prove that the LP-relaxations LP_1 and LP_2 are equally strong in terms of the bounds they provide.

Proposition 1. *If* $(\bar{u}, \bar{f}) \in \mathrm{LP}_1$ *then* $\bar{u} \in \mathrm{LP}_2$. *Conversely, for any* $\hat{u} \in \mathrm{LP}_2$, *there exists* $\hat{f}$ *such that* $(\hat{u}, \hat{f}) \in \mathrm{LP}_1$.

Proof. Consider any vector $(\bar{u}, \bar{f}) \in \mathrm{LP}_1$. Clearly $0 \leqslant \bar{u}^r_{ij} \leqslant 1$. Thus we need to show that $\bar{u}$ satisfies all inequalities (3). Given $X \subseteq N$, $1 \in X$, $k \in N \setminus X$, we have

$$\begin{aligned}
\sum_{(i,j)\in\delta(X)} \sum_{r=1}^{\mathrm{lev}(k)} \bar{u}^r_{ij} &\geqslant \sum_{(i,j)\in\delta(X)} \bar{f}^k_{ij} \quad \text{by (2)} \\
&\geqslant \sum_{(i,j)\in\delta(X)} \bar{f}^k_{ij} - \sum_{(i,j)\in\delta(N\setminus X)} \bar{f}^k_{ij} = \sum_{j\in X} \left\{ \sum_{i\in N} \bar{f}^k_{ji} - \sum_{i\in N} \bar{f}^k_{ij} \right\} \\
&= 1 \quad \text{by (1) since } 1 \in X,\ k \in N \setminus X.
\end{aligned}$$

Thus $\bar{u}$ satisfies all inequalities (3).

Now consider any solution $\hat{u} \in \mathrm{LP}_2$. Consider any node $k \in N\backslash\{1\}$. On the graph D, assign a capacity $\sum_{r=1}^{\mathrm{lev}(k)} \hat{u}_{ij}^r$ to each arc (i,j). Since $\hat{u}$ satisfies all inequalities (3), the minimum cut separating nodes 1 and k has capacity at least 1 with arc capacities as defined above. Thus it is possible to send one unit of flow from node 1 to node k using the capacities defined above. Repeating the above procedure for each of the nodes $k \in N\backslash\{1\}$ gives us the required flow $\hat{f}$ such that $(\hat{u}, \hat{f}) \in \mathrm{LP}_1$. The result thus follows. □

Proposition 1 shows that the LP-relaxations of the DM and DSC formulations are theoretically identical in terms of the bounds they provide. Thus if we are able to solve the LP-relaxation to the DSC formulation to optimality, it would provide a better bound than the LP-relaxation to the DM formulation which we are only able to solve approximately and obtain a lower bound.

2.3. Extended directed Steiner cut (EDSC) formulation

We exploit the fact that a higher-grade facility can cost no less than a lower-grade facility to come up with an extended formulation for MLND problem. This cost structure allows us to assume that in any optimal solution, each arc will contain the lowest grade facility that is feasible. Our formulation is similar in spirit to a transformation suggested by Duin and Volgenant [15].

Consider an optimal solution $\hat{u}$ to an MLND problem on the directed graph D. Define $T = (N, AT)$ to be the arborescence defined by the arcs in the optimal solution, i.e. an arc $a \in AT$, if and only if, $\sum_{r=1}^{L} \hat{u}_a^r = 1$. For any node $k \in N\backslash\{1\}$, there exists a directed path $P_T(1,k) = \{a_1, \ldots, a_s\}$ from the root 1 to node k in T. Consider any two arcs a_g and a_h where a_g comes before a_h on the path $P_T(1,k)$. Let $r_g(r_h)$ be the level of facility installed on arc $a_g(a_h)$ in the solution $\hat{u}$, i.e. $\hat{u}_{a_g}^{r_g} = \hat{u}_{a_h}^{r_h} = 1$. If arc a_h has a higher-grade facility than arc a_g, i.e. $r_g > r_h$, define a new solution $\bar{u}$ where

$$\bar{u}_a^r = \begin{cases} 1 & \text{for } r = r_g,\ a = a_h, \\ 0 & \text{for } r = r_h,\ a = a_h, \\ \hat{u}_a^r & \text{otherwise.} \end{cases}$$

The vector $\bar{u}$ is a feasible solution to the MLND problem since the arc a_g is on every path in T that contains a_h, and the grade of facility on a_h is no lower than that on a_g in this solution. Since the cost of a level r_g facility on arc a_h is no more than the cost of a level r_h facility, the total cost of solution $\bar{u}$ is no more than that of $\hat{u}$. This allows us to restrict attention to optimal solutions as stated in the next result.

Proposition 2. *Since higher-grade facilities cost no less than lower-grade facilities, there exist optimal solutions to MLND problem such that on the directed path from the root* 1 *to any node k, the level of facility installed is nondecreasing as we move from* 1 *to k, i.e. no arc that occurs later on the path from* 1 *to k has a higher-grade facility than an arc that precedes it.*

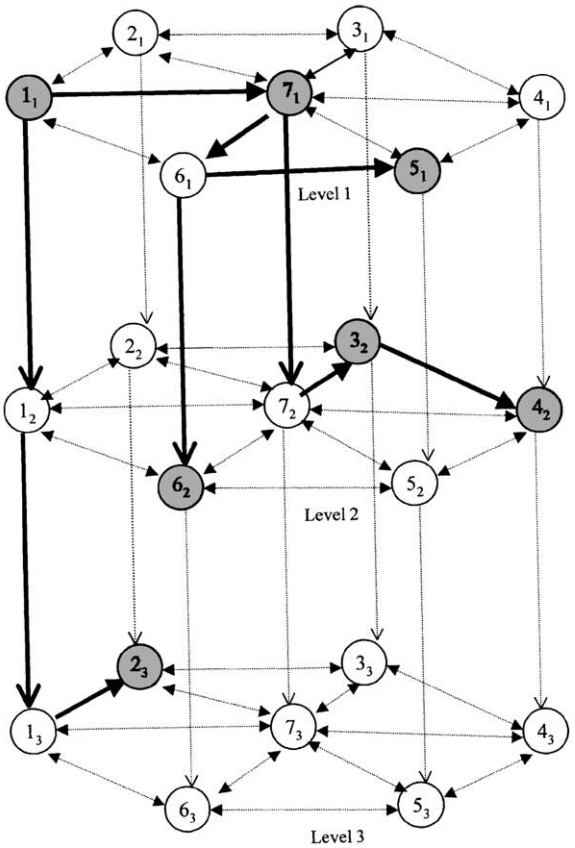

Fig. 2. $ED = (V, EA)$ converted from Fig. 1 with the corresponding solution.

This characterization allows us to redefine MLND problem on an extended graph as follows. Given the directed graph $D = (N, A)$ we construct an extended directed graph $ED = (V, EA)$ where $V = N_1 \cup N_2 \cdots \cup N_L$ and $EA = A_1 \cup A_2 \cdots \cup A_L \cup A_C$. For each node i in N, construct L copies $i_1, i_2, \ldots, i_L$ where $i_r \in N_r$. For $r \in \{1, \ldots, L\}$, define $\phi_r(i) = i_r$. Given $S \subseteq N$, define $\phi_r(S) = \{\phi_r(i): i \in S\}$. Thus $|V| = L|N|$. Similarly for each arc $a = (i, j)$ in A, we construct the arcs (i_r, j_r), $r \in \{1, \ldots, L\}$, where $(i_r, j_r) \in A_r$. For each node $i \in N$, there are cross-level arcs $(i_r, i_{r+1}) \in A_C$, $r \in \{1, \ldots, L-1\}$. Thus $|A_1| = |A_2| = \cdots = |A_L| = |A|$, $|A_C| = (L-1)|N|$ and $|EA| = L|A| + (L-1)|N|$. Given the costs b_{ij}^r for each arc $(i, j) \in A$, define arc costs $c_{i_r, j_r} = b_{ij}^r$ for $r \in \{1, \ldots, L\}$. Each arc of the form (i_r, i_{r+1}) in A_C has a cost of zero. Given the set of nodes S_r in N, define $S_r^r = \phi_r(S_r)$ to be the corresponding nodes in N_r. The node $1_1 \in S_1^1$ is declared the root. Fig. 2 shows the extended directed graph $ED(V, EA)$ corresponding to the undirected graph shown in Fig. 1. Given the solution in Fig. 1, we also show the corresponding solution (a Steiner arborescence) in Fig. 2 on the extended graph. Fig. 3a shows an example of a directed graph $D(N, A)$ with $|N| = 5$, $|A| = 8$ and $L = 4$

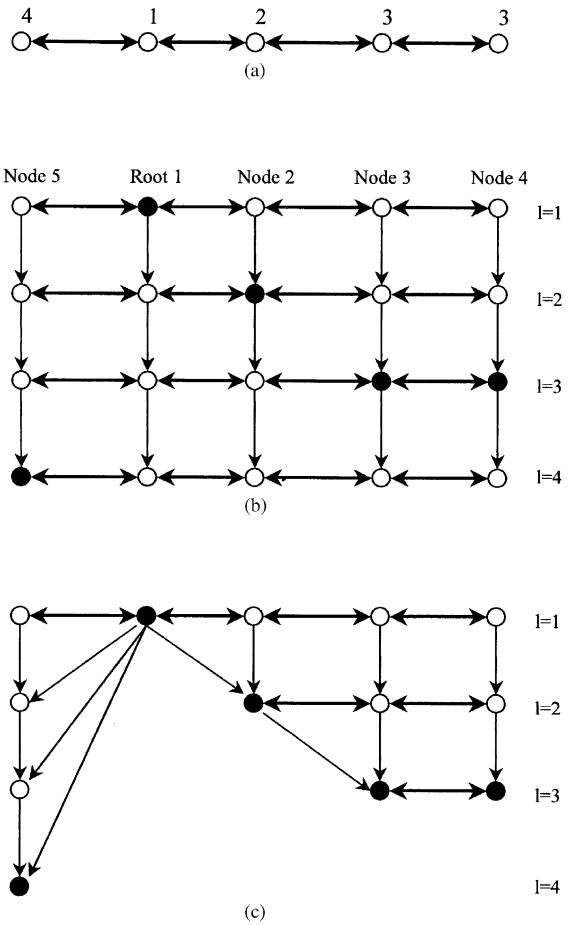

Fig. 3. (a) $D(N,A)$, $|N|=5$, $|A|=8$, $L=4$. (b) $ED(V,EA)$ constructed from Fig. 3a. (c) $ED(V,EA)$ in Fig. 3b after structural arc exclusions.

where the number located above a node i represents $\mathrm{lev}(i)$ and bidirectional arcs are represented by assigning arrows to both ends of a line segment. The graph in Fig. 3b corresponds to the graph $ED(V,EA)$ constructed from $D(N,A)$ with nodes in $\bigcup_{r=1}^{L} S_r^r$ (demand nodes in a Steiner tree problem) marked by filled circles. In this example, $|V|=20$, $|EA|=47$.

In the graph ED, we search for a minimum cost Steiner arborescence rooted at node 1_1 that spans all nodes in $\bigcup_{r=1}^{L} S_r^r$. Since all cross-level arcs have a cost of zero, once a node is reached at level i it can be reached at levels $i+1$ and beyond at zero cost. Thus no arc (i,j) will be used at two different levels in the optimal solution, i.e. if (i_r,j_r) appears in the optimal Steiner arborescence, then none of the arcs of the form (i_ℓ,j_ℓ) appear in the optimal arborescence for $\ell \neq r$. Further, in any Steiner arborescence, all

paths from the root are such that no arc from the set A_j can occur before A_i if $j > i$. Thus we restrict solutions considered to be of the form described in Proposition 2.

Given the directed graph ED, consider $X \subseteq V$, $1_1 \in X$, $|\{V \setminus X\} \cap \{\bigcup_{r=1}^{L} S_r^r\}| \geqslant 1$. Define the directed Steiner cut $\delta(X)$ where

$$\delta(X) = \{a = (p,q) \in EA \mid p \in X,\ q \in V \setminus X\}.$$

For each arc (p,q), define the variable z_{pq}, where

$$z_{pq} = \begin{cases} 1 & \text{if arc } (p,q) \text{ is in the Steiner arborescence,} \\ 0 & \text{otherwise.} \end{cases}$$

Given any node set X as defined above, each Steiner arborescence in ED satisfies the *directed Steiner cut inequality* (see the work of Chopra and Rao [10,11])

$$\sum_{a \in \delta(X)} z_a \geqslant 1. \tag{4}$$

Define the polytope

$$\mathrm{LP}_3 = \{1 \geqslant z \geqslant 0 \mid z \text{ satisfies all inequalities (4)}\}.$$

The problem MLND can thus also be formulated as

$$\text{Min} \sum_{a \in EA} c_a z_a \text{ s.t. } z \in \mathrm{LP}_3,\ z \text{ integer.}$$

This integer programming formulation is referred to as the *extended directed Steiner cut* (*EDSC*) *formulation* for MLND problem.

Given the optimal solution for the EDSC formulation, one can obtain the optimal solution to the DSC formulation as stated in the following proposition.

Proposition 3. *If $\hat{z}$ is the optimal solution to the EDSC formulation for MLND problem, then $\hat{u}$ is the optimal solution to the DSC formulation for MLND problem, where*

$$\hat{u}_{ij}^r = \hat{z}_{i_r, j_r}.$$

As in the above proposition, define the linear transformation ϕ from the z variables to the u variables where

$$u_{ij}^r = z_{i_r, j_r} \quad \text{for each arc } (i,j) \text{ and level } r.$$

It is easy to verify that for every vector $\bar{z} \in \mathrm{LP}_3$ there exists a corresponding vector $\bar{u} = \phi(\bar{z}) \in \mathrm{LP}_2$. However the converse is not true. There exist solutions $\hat{u} \in \mathrm{LP}_2$, where, on a path from the root 1 to a node k, the level of facilities assigned to the arcs decreases as we move from 1 to k, i.e. the grade of facility installed is higher as we move away from the root. However no such solutions are feasible in LP_3 since all cross-level arcs are directed from a level i to a level $i+1$, i.e. from a higher grade to a lower grade. Thus we obtain the following result.

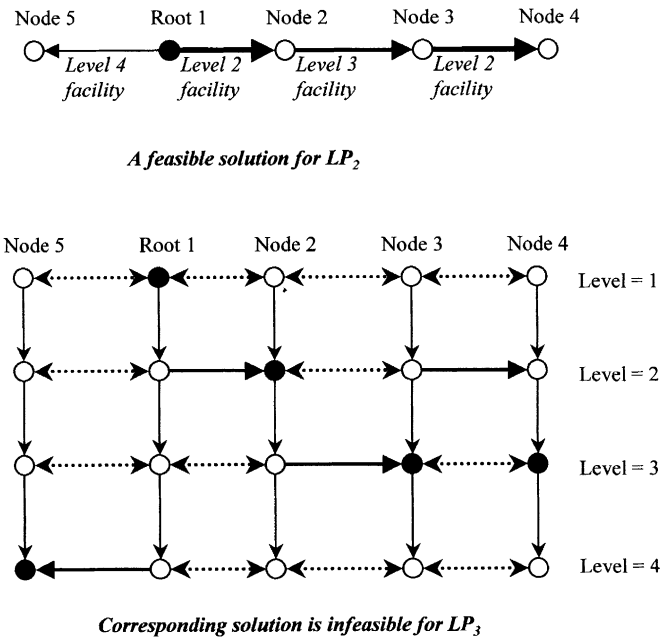

Fig. 4. A feasible solution to LP_2 but not LP_3.

Proposition 4. *Given the linear transformation* ϕ *as defined above, let* $\phi(\mathrm{LP}_3)$ *be the polytope obtained in the u space on transforming* LP_3. *Then*

$$\phi(\mathrm{LP}_3) \subseteq \mathrm{LP}_2, \qquad \phi(\mathrm{LP}_3) \neq \mathrm{LP}_2,$$

i.e. $\phi(\mathrm{LP}_3)$ *is a proper subset of* LP_2. *Thus* LP_3 *provides a stronger* LP*-relaxation to MLND problem than* LP_2.

As an example, consider the graph $D = (N, A)$ in Fig. 3a. Consider the solution $\hat{u}$ where $\hat{u}^4_{15} = \hat{u}^2_{12} = \hat{u}^3_{23} = \hat{u}^2_{34} = 1$, $\hat{u}^r_a = 0$ for all other r, a. It is easy to verify that $\hat{u} \in \mathrm{LP}_2$. Observe that this solution violates Proposition 2 since there exists a path $\{(1,2),(2,3),(3,4)\}$ from the root 1 to node 4 where arc $(2,3)$ comes before arc $(3,4)$, but arc $(2,3)$ has a level 3 facility while arc $(3,4)$ has a level 2 facility. The corresponding extended graph $ED = (V, EA)$ is shown in Fig. 4.

We now show that no corresponding solution $\hat{z}$ exists in LP_3. If such a solution did exist we would have $\hat{z}_a = 0$ for all arcs $a \in EA \backslash \{A_C \cup \{(1_2, 2_2), (2_3, 3_3), (3_2, 4_2), (1_4, 5_4)\}\}$. The arcs fixed at 0 are shown as dotted lines in the corresponding extended directed graph in Fig. 4. Observe that there does not exist a directed arborescence from the root spanning all demand nodes (shown as filled circles) that does not use at least one arc represented by a dotted line. Thus there does not exist a vector $\hat{z} \in \mathrm{LP}_3$ such that $\hat{u} = \phi(\hat{z})$. This example illustrates Proposition 4 and shows that LP_3 provides a tighter LP-relaxation than LP_2.

3. Branch-and-cut approach

The branch-and-cut approach is well understood (see [12,24]) and is not described here. In this section we discuss the design of the branch-and-cut approach for solving the EDSC formulation. The key tasks in the branch-and-cut approach include preprocessing, selection of the initial formulation, cut generation (to identify *directed Steiner cut* inequalities (4) violated by the current LP solution), and obtaining a good upper bound.

3.1. Preprocessing: graph reduction

The methods described in this section reduce problem size by identifying arcs that are always in an optimal Steiner arborescence and arcs that can never be in an optimal Steiner arborescence. Some of the reductions are done on the graph $G(N,E)$ and others are performed on the extended directed graph $ED(V,EA)$. Since all cross-level arcs $a \in A_C$ have cost zero, we can always obtain an optimal solution to EDSC where all cross-level arcs have value 1. This property is exploited in our solution approach. The preprocessing steps used by us are described below:

1. *Arc inclusions*: This is the primary arc inclusion used in Balakrishnan et al. [4] for TLND problems. The algorithm first finds a minimum spanning tree $T_1(N)$ on graph $G(N,E)$ using edge cost b_e^1 for $e \in E$. An edge $(i,j) \in T_1(N)$ is always in an optimal solution to an MLND problem if $i,j \in S_1$. Thus these edges can be contracted. At each level if parallel edges exist after contraction, only the one with the lowest edge cost is kept for constructing $ED(V,EA)$.
2. *Structural arc exclusions*: Consider any demand node j at level r, i.e. $j_r \in S_r^r$ for some level $r \in \{1,\ldots,L-1\}$. Every optimal solution to EDSC, contains a path from the root node to any demand node $j_r \in S_r^r$. By Proposition 2, no arcs below level r are included in this path, i.e. j_r must be reached using arcs in A_l, $l \leqslant r$. Because of the inclusion of all cross-level arcs in the optimal solution, no arcs of the form $(i_t,j_t) \in A_t$ can be in the Steiner arborescence for $t \in \{r+1,r+2,\ldots,L\}$. Thus, these arcs can be eliminated from ED. In addition, any node j_t, $t \in \{r+1,\ldots,L\}$ is connected to j_r through arcs $(j_r,j_{r+1}),(j_{r+1},j_{r+2}),\ldots,(j_{L-1},j_L)$ in A_C. Therefore we can delete these nodes and arcs and redirect arcs (j_t,p_t), $t \in \{r+1,r+2,\ldots,L\}$ to (j_r,p_t). Note that in this case $\mathrm{lev}(p) > \mathrm{lev}(j)$. Otherwise, node p_t should have been eliminated using the above approach. Fig. 3c shows the result of $ED(V,EA)$ in Fig. 3b after structural arc exclusions where $|V|$ and $|EA|$ are reduced from 20 and 47 to 13 and 27 respectively. This preprocessing step is very effective in reducing graph size.
3. *Heuristic arc exclusions*: We adopted the edge deletion heuristic used in Chopra et al. [9]. If $c_{i,j} > \max\{c_{i,k},c_{j,k}\}$ and $k \in \bigcup_{t=1}^{L} S_t^t$ then arcs (i,j) and (j,i) cannot be part of an optimal Steiner arborescence. Thus they can be deleted from the arc set EA. The arcs found by this heuristic are a subset of arcs that can be identified by the minimum spanning tree based *R-S*, *S-S* Deletion Algorithms in Balakrishnan

and Patel [6]. Since there are no reverse arcs for arcs in A_C, the algorithm can be implemented L times independently on $G(N,E)$ each time to a level i applying edge cost b_e^i.

The preprocessing steps used by us are quite effective in reducing the size of the graph. On test instances solved by us in our computational tests, preprocessing reduces the number of nodes by 0.67–43.6% and number of arcs by 34.5–81.7%.

3.2. Initial formulation

For a node $j_t \in S_t^t$, define $J(j_t) = \{j_1, j_2, \ldots, j_t\}$. The set $J(j_t)$ corresponds to all nodes in the graph $ED(V,EA)$ that are copies of j_t. Note that nodes $j_{t+1}, \ldots, j_L$ have been eliminated in the reduction stage using structural arc exclusions, for example $J(1_1) = \{1_1\}$. In our initial formulation, we include all EDSC cuts defined by the sets $J(j_t)$ for demand nodes $j_t \in \bigcup_{r=1}^{L} S_r^r$. Given a node set X, define $\gamma(X) = \{a = (p,q) \in A \mid p \in V \setminus X, q \in X\}$. The initial formulation can be written as

$$\text{Min.} \quad \sum_{a \in EA} c_a z_a$$

$$\text{s.t.} \quad \sum_{a \in \delta(1_1)} z_a \geqslant 1 \quad 1_1\text{: root node},$$

$$\sum_{a \in \gamma(J(j_t))} z_a \geqslant 1, \; j_t \in \bigcup_{r=1}^{L} S_r^r \setminus \{1_1\},$$

$$0 \leqslant z_a \leqslant 1, \quad a \in EA.$$

In the initial formulation, all variables corresponding to arcs remained after graph reduction are included except for those representing cross-level arcs and the number of constraints equals the number of demand nodes remained. During the solution process, variables may be added to or removed from the formulation depending on whether they appear in the EDSC cuts currently in the formulation or not.

3.3. Cut generation

Given a solution to the current LP-relaxation, our goal in this stage is to identify Directed Steiner Cut inequalities (4) that are violated by it. We use two basic algorithms for this purpose, namely Breadth First Search (BFS) and Minimum Cut.

These algorithms are run on the graph $\widehat{ED}(V,\widehat{EA})$ induced by a solution $\hat{z}$ where the capacity of arc a is determined by the value of $\hat{z}_a$ in the solution. $\widehat{EA}$ only includes arcs that have $\hat{z}_a > 0$. Recall that the *directed steiner cut* inequalities (4) require that all cuts in $\widehat{ED}$ separating the root node and any other terminal (demand) node must have capacity at least 1. As a result the following violated cuts may be generated.

1. BFS cuts: This separation performs a BFS on $\widehat{ED}$ to find a BFS tree originating from the root node. A cut is generated if any of the demand nodes $\bigcup_{r=1}^{L} S_r^r \setminus \{1_1\}$

is not in the tree. This identifies a violated EDSC inequality (4) where we have a cut of capacity 0 in $\widehat{ED}$.

2. Mincuts: We use the root node 1_1 as the source to run maxflow-mincut algorithm each time with a randomly selected sink node from the set $\bigcup_{r=1}^{L} S_r^r \setminus \{1_1\}$. If the capacity of the mincut is less than one, a violated EDSC inequality (4) is found and is referred to as a mincut inequality.

Observe that both BFS cut and mincut will result in the minimum capacity cut closest to the root node being generated. As observed by Chopra et al. [9], it is very likely that some of the mincuts generated by the regular mincut algorithm for different sinks could be identical. Attempting to identify more distinct mincuts, we try to generate minimum capacity cuts closest to the sink nodes. For this we construct a reverse graph $\overline{ED}(V,\overline{EA})$ from $\widehat{ED}(V,\widehat{EA})$ where $\overline{EA} = \{(i,j) \mid (j,i) \in \widehat{EA}\}$ and $\bar{z}_{ij} = \hat{z}_{ji}$. Observe that the minimum capacity cuts in $\overline{ED}$ and $\widehat{ED}$ have the same value. On the graph $\overline{ED}$ we run the following variants of the previous two separation algorithms. Since the cuts generated are likely to be closer to the sinks, the chance of generating distinct cuts is higher, especially when the induced graph is not completely connected.

1. Reverse BFS cuts: We choose a demand node at random from $\bigcup_{r=1}^{L} S_r^r \setminus \{1_1\}$ as the root node. If in the BFS tree node 1_1 is not reachable from the root node chosen, a violated EDSC inequality (4) is found and is referred to as a reverse BFS cut. Unlike the regular BFS cut, this algorithm could generate more than one violated reverse BFS cuts on an induced graph $\overline{ED}(V,\overline{EA})$.
2. Reverse Mincuts: Here we randomly select a node from $\bigcup_{r=1}^{L} S_r^r \setminus \{1_1\}$ as the source and 1_1 as the sink. If the capacity on the mincut is less than one, we have a violated EDSC inequality (4) that is referred to as a reverse mincut. Our experiment shows that the reverse mincut algorithm usually generates more distinct violated EDSC inequalities than its regular counterpart.

As the complexity of a maxflow-mincut algorithm ($\mathcal{O}(|V||EA|\log|V|)$ [27]) is higher than that of a *breadth first search* ($\mathcal{O}(|EA|)$), the design of the algorithm tries to utilize BFS more frequently whenever possible. As long as the graph induced by the current LP solution is not connected, we state that the algorithm is in stage 1 and use BFS and reverse BFS to generate violated inequalities (4). If the graph induced by the current LP solution is connected, the algorithm is in stage 2 and reverse mincut is used to generate violated inequalities. In each iteration the number of cuts generated is up to a threshold. In our experiments, the threshold is set at perc $\times |N|$ where perc is a parameter between 0 and 1. In stage 1 the order in which we try and generate violated inequalities is BFS, reverse BFS and then reverse mincut. Thus we only resort to reverse mincut if the total number of violated inequalities generated by BFS and reverse BFS is less than the threshold. In this stage, BFS is run in every iteration and reverse BFS is attempted in every iteration except that in every freq iterations, reverse mincut is run in place of reverse BFS, where freq is a parameter. In stage 2, reverse mincut is run in every iteration except that in every freq iterations, reverse BFS is

run in place of reverse mincut. If some violated reverse BFS cuts are found, we have disconnected components again on the graph and the process switches back to stage one. Regular mincut is used only for optimality test in stage one when no BFS cut is found before constructing $\overline{ED}(V,\overline{EA})$.

Because the root/source/sink nodes in the separation algorithms are randomly selected, each demand node has an equal opportunity of being examined for violated cuts based on it. From our computational test, randomizing is found to be more effective than fixing an order on the nodes and searching for violated inequalities in that sequence.

If no violated cuts are found, the current solution solves the LP-relaxation LP_3. If the solution is integral, we have an optimal solution. Else we proceed to the branching phase. In the branching phase, we limit the number of iterations in which new cuts are generated to maxiter iterations at each branch before creating a new one. The values of perc, freq, and maxiter are chosen from calibration tests in our computational study.

3.4. Heuristic upper bound

We develop a heuristic algorithm to obtain an upper bound for the optimal solution based on the fractional solution to the LP-relaxation. The approach is to sequentially fix arcs to be in the solution, using the current LP solution as a guide when fixing variables.

Given a fractional solution, each variable with value larger than 0.7 is fixed to 1 with probability equal to its fractional value. We have found this randomized fixing to be more effective than fixing all variables with value larger than 0.7 to 1. Given the specific structure of the extended graph and the fact that we are seeking a Steiner arborescence, a fixing of one variable allows us to fix several other variables as well. However, we need to ensure that between the arcs fixed to 1 and the arcs not yet fixed, there exists a feasible solution to the LP-relaxation. For this reason we also unfix variables as needed.

If a variable $z_{i_pj_q}$ is fixed to 1, the variable $z_{j_qi_p}$ is unfixed if it was previously fixed. This is because the underlying problem is on an undirected graph and the unfixing allows us the flexibility of reversing the direction of the arc (i_p,j_q) in a solution to get a new solution of the same value.

Recall that all cross-level arcs have a cost of 0. Thus if the variable $z_{i_pj_q}$ is fixed to 1, we can reach any of the nodes j_s for $s \geqslant q$ at zero cost using the cross-level arcs. Thus we can fix all arcs of the form $(u_r,j_s) \neq (i_p,j_q)$ to have $z_{u_rj_s}=0$ for $s \geqslant q$. This includes any such arcs that may previously have been fixed to 1. In case we reach a situation where no feasible solution exists among the variables not fixed to 0, we run a *breadth first search* using arcs not fixed to zero. If a demand node cannot be reached from the root, we obtain a cut which contains arcs fixed to zero. We then unfix all arcs in the cut and resolve the resulting linear program.

As stated earlier, we fix only a subset of the fractional variables at each stage. Ideally, we would like to solve the new LP-relaxation to optimality before fixing new

variables since this would likely give better information on the value of the variable at optimality. However, to speed up the process, we run only maxiter (the same parameter used in the branching phase) iterations of cut generation between successive stages of variable fixing.

4. Computational results

To test the effectiveness of our branch-and-cut algorithm, we generate 420 instances of various sizes and structures. Forty of those generated instances are used for calibrating some key parameters used in our branch-and-cut solver. Those tests allow us to tune the solver for the best performance. Once the best parameters are found, the remaining 380 instances were solved to test the effectiveness of the EDSC formulation and the solver. For each instance, we allow 20 min of CPU time. If branching is required, the heuristic is solved to obtain an initial upper bound. If an optimal solution is not found after 75% of the allowable time and the process has not proceeded to the branching stage, we also solve the heuristic to find an upper bound. The choice of 75% is to reserve 5 min for the heuristic so that an upper bound is available in case an instance is not solved to optimality within the allowable time. Tailing-off criterion adopted by the solver is similar to that in [24]. All instances are solved on a Dell Pentium III 500 MHz personal computer with 128 MB of RAM running *windows operating system*. The LP solver is ILOG CPLEX 6.5 [21] by ILOG, Inc. To compare the performance of our branch-and-cut algorithm with the dual ascent method [4,22], we implement the dual ascent approach described in [22] as a benchmark method.

4.1. Problem generator

Test instances are generated the same way as those in Mirchandani [22]. Instances are characterized by four features: size, cost function, cost ratio and the distribution of demand nodes. Problem size is determined by the numbers of nodes, edges, and levels. Nodes are generated uniformly distributed over a 1000 by 1000 grid as in [22]. Cost function is specified by the four different ways with which edge costs are calculated, namely, Euclidean distance, Manhattan distance, infinity norm and randomly generated cost. Cost ratio is either fixed or general. It determines the cost ratio between two successive levels. A fixed ratio problem has a constant ratio of $b_{ij}^{r+1}/b_{ij}^{r}=\rho^{r}$ for all edges $(i,j)\in E$ where $0<\rho^{r}<1$ and $r\in\{1,\ldots,L-1\}$ whereas a general (variable) ratio problem could have various cost ratios for different edges, i.e. $b_{ij}^{r+1}/b_{ij}^{r}=\rho_{ij}^{r}$, $0<\rho_{ij}^{r}<1$. The values of ρ^{r} and ρ_{ij}^{r} are generated uniformly from ranges given by Mirchandani [22]. The combination of cost functions and cost ratios results in eight different cost structures. The distribution of demand nodes dictates the number of demand nodes assigned to each level. Since $|N|=\sum_{r=1}^{L}|S_r^r|$, given a fixed $|N|$ different instances can be generated by changing the assignment of $|S_r^r|$, $r\in\{1,\ldots,L\}$. In measuring the performance of the algorithm on each set of instances, we take the average solution

time from solving five randomly generated instances with the same characteristics. We also report the standard deviation of the solution times, average optimality gap, and number of instances solved to optimality among the five.

4.2. Selection of parameters

In practice, the performance of the branch-and-cut algorithm depends on the values of some crucial parameters mentioned in Sections 3.3 and 3.4. The test to find the best parameters for the branch-and-cut approach is conducted on 40 problem instances with eight different configurations whose characteristics are listed below, where $|N| \times |E|$ denotes the number of nodes and edges in the original graph $G(N,E)$; L represents the number of levels and the distribution of demand nodes among levels are written as $|S_1^1|/|S_2^2|/\cdots/|S_L^L|$. Five test instances are generated from each of the configurations:

I. 400×1400, $L = 4$, (50/80/120/150) Euclidean, fixed ratio,
II. 400×1400, $L = 4$, (150/120/80/50) Euclidean, fixed ratio,
III. 600×2100, $L = 4$, (150/150/150/150) Random, general ratio,
IV. 600×2100, $L = 4$, (150/150/150/150) Euclidean, general ratio,
V. 800×2800, $L = 4$, (200/200/200/200) Random, fixed ratio,
VI. 800×2800, $L = 4$, (200/200/200/200) Random, general ratio,
VII. 1000×3500, $L = 4$, (250/250/250/250) Manhattan, general ratio,
VIII. 1000×3500, $L = 4$, (250/250/250/250), Infinity, fixed ratio,

In each test, we vary the value of a single parameter while keeping others fixed at the best parameter values found from other tests. For each parameter value (p) and test set combination ($i \in \{\mathrm{I}, \mathrm{II}, \ldots, \mathrm{VIII}\}$), we identify an *effectiveness index* $\mathrm{EI}(p,i)$, where

$$EI(p,i) = (\text{Average time to solve test set } i \text{ with parameter value } p)/ (\text{Average time to solve test set } i \text{ using best parameter value}).$$

Thus the best parameter value has an *effectiveness index* of 1. In general, we seek a parameter value that has an effectiveness index close to 1 over all eight test sets.

Parameter perc: In any branch-and-cut algorithm, most of the time is spent either generating violated inequalities or solving the resulting LP. We use $\text{perc} \times |N|$ to define the maximum number of violated inequalities generated before a new LP is solved where perc is set to a value between 0 and 1. Increasing perc results in more and (possibly) stronger inequalities leading to a greater improvement in the objective function. However, it takes longer time in generating violated inequalities of which some become inactive later. It also increases the computer memory requirement due to the larger size of the resulting LP. Thus perc is an important parameter to be calibrated. We consider six choices of the threshold $0.5\% \times |N|$, $1.0\% \times |N|$, $5.0\% \times |N|$, $10.0\% \times |N|$, $20.0\% \times |N|$, and $50.0\% \times |N|$. The results are reported in the six perc columns in Table 1. Observe that the effectiveness index varies significantly (the range is from 1.00 to 12.44) as we change the value of perc. Our conclusion from this calibration is

Table 1
Calibration of key computational options

Problem set	perc (effectiveness index)						gencut t(seq)/t(rnd)
	0.005	0.01	0.05	0.1	0.2	0.5	
I	1.91	2.02	1.14	**1.00**	1.28	2.58	0.52
II	1.92	1.91	1.12	**1.00**	1.70	4.32	1.25
III	1.25	1.01	1.01	**1.00**	1.70	3.39	1.74
IV	3.28	2.16	1.00	1.18	1.73	7.16	1.38
V	2.24	**1.00**	1.81	2.21	3.73	9.56	1.29
VI	2.07	**1.00**	1.88	3.03	5.39	12.44	1.22
VII	1.37	**1.00**	1.30	1.47	1.82	3.31	1.03
VIII	1.43	**1.00**	1.70	1.88	2.41	3.59	1.66

to set perc $= 10\%$ for graphs with $|N| \leqslant 600$ and perc $= 1\%$ otherwise. This approach gives an overall *effectiveness index* very close to 1 across all test sets. This clearly demonstrates that one should not generate too many cuts in each iteration before re-solving the LP and as the problem size grows the value of perc should be reduced. To keep the size of the LP manageable, we also purge inactive constraints every five iterations.

Parameters freq, maxiter: These parameters do not exhibit significant differences among different parameter values. Hence, we only report the best choices without providing numerical details from these tests. Parameter freq controls the switch between stage one and stage two, allowing the process to focus more on connecting the graph in stage one using BFS and reverse BFS, and once the graph is connected devoting more time in reverse mincut to remove fractional values. However, in stage two we still run reverse BFS once in a while so that when the graph is disconnected again, the process returns to stage one, while in stage one reverse mincut is executed once in a few iterations to get better improvement on the objective function. The best choice of freq is five chosen among every 1, 5, 10, 15, 20, and 30 iterations. When a problem instance takes too much time to solve, our goal is to find a good feasible solution in a given amount of computational time instead of finding an optimal solution. Limiting the number of iterations on each branch of a branching tree and in the heuristic before fixing more variables might reach a feasible solution sooner. However, it can have a negative impact on the optimality gap of the upper bound thus obtained. We tested 15, 30, and 45 iterations on a few instances requiring branching and found no significant impact on both solution times and optimality gaps. maxiter $= 30$ provides the best balance between computational times and the quality of upper bounds.

Sequence of examining potential violated cuts: When running reverse BFS or reverse mincut in each iteration, we select a demand node $i \neq 1_1$ as the root or source node. The selection of the root (source) node can be done following either a fixed order for all iterations or a process randomly choosing a node from the candidate nodes not selected yet. Since our algorithm stops searching for more violated cuts and solve the resulting LP once perc $\times |N|$ violated inequalities are found, the choice between the

two approaches may affect the solution time. The fixed order search tends to grow the tree to connect the demand nodes to node 1_1 following the fixed sequence. The random order process, on the other hand, grows the tree to connect the demand nodes to 1_1 more evenly among the demand nodes. The last column of Table 1 shows the ratio of the solution time between the sequential and random approaches. It indicates that except for the first test set, the random process has an edge over the sequential approach. Thus, it is adopted in our design of the solver.

4.3. Analysis of computational times

In our computational experiments, there are three questions we seek to answer:

1. How effective is the EDSC formulation for multi-level network design problem?
2. How effective is the branch-and-cut approach (applied to the EDSC formulation) for multi-level network design problem?
3. What problem characteristics have a significant impact on solution time and quality?

We answer those questions by observing the behavior of the branch-and-cut algorithm in solving 380 randomly generated instances with graph sizes varying from 400 to 1200 nodes, from 1200 to 4200 edges, and from two to five levels. All test instances are solved by both the proposed approach and the benchmark approach. Tables 2–6 summarize the result. Each row in the table represents the average result from solving five problem instances of the same configuration. Rows are then grouped into sets by the number of nodes, edges, levels and demand node distributions. These problem characteristics are the most influential ones in solution times. Rows in the same set, following a title row showing those problem characteristics, differ only in their cost functions (Random, Euclidean, Manhattan, and Infinity) and cost ratios (fixed, general). Cost factors appear to be less conclusive in their influence on solution times. Thus, comparisons are made mostly from the 'Average' row, which contains the overall average from the rows in the same set.

Effectiveness of EDSC formulation: Our results indicate that EDSC is a very effective formulation for multi-level network design problem since 356 (94%) of the 380 instances attempted are solved to optimality without branching compared to 112 (29%) instances solved to optimality by the dual ascent method. An instance is solved to optimality by an approach if the approach shows zero optimality gap without comparing its solution to that obtained from the other approach. In each of those 356 instances, the LP-relaxation given by the EDSC formulation provides the integer optimum. For the remaining 24 test instances, the optimality gap is closed to within 2.5% for two and within 2% for three and within 1% for the remaining 19 instances. We consider that the extended directed Steiner cuts generated from the branch-and-cut approach provide a very good description of the facial structure of the integer polytope near the optimal extreme point. Note that in the dual ascent method, we allow three iterations of the Add-Drop process as in [22]. The more iterations allowed, the longer solution time it takes but with a potentially smaller optimality gap. Our implementation of the dual

Table 2
Computations to study impact of changing cost structure[a]

Approach	Branch-and-cut on EDSC				Dual ascent			
Problem type	Average time (s)	SD of time	Average gap (%)	# solved to optimal	Average time (s)	SD of time	Average gap (%)	# solved to optimal
$\|N\| = 400$, $\|E\| = 1400$, 4 levels, demand node distribution: 100/100/100/100								
Random (fixed)	206	378	0	5	179	50	0.10	1
Random (general)	22	11	0	5	118	99	0.08	1
Euclidean (fixed)	24	8	0	5	213	23	0.24	0
Euclidean (general)	18	6	0	5	144	27	0.27	0
Manhattan (fixed)	23	10	0	5	205	34	0.20	0
Manhattan (general)	17	4	0	5	135	27	0.25	0
Infinity (fixed)	29	5	0	5	225	22	0.24	0
Infinity (general)	25	9	0	5	122	10	0.28	0
Average	46	54	0	5.0	168	36	0.21	0.3
$\|N\| = 600$, $\|E\| = 2100$, 4 levels, demand node distribution: 150/150/150/150								
Random (fixed)	281	403	0	5	409	66	0.08	0
Random (general)	135	156	0	**5**	297	65	0.24	0
Euclidean (fixed)	158	143	0	5	332	163	0.17	0
Euclidean (general)	93	63	0	5	269	75	0.29	0
Manhattan (fixed)	92	53	0	5	384	128	0.14	0
Manhattan (general)	56	32	0	5	235	37	0.29	0
Infinity (fixed)	109	74	0	5	395	99	0.10	0
Infinity (general)	82	71	0	5	256	63	0.18	0
Average	125	124	0	5.0	322	87	0.19	0.0
$\|N\| = 800$, $\|E\| = 2800$, 4 levels, demand node distribution: 200/200/200/200								
Random (fixed)	55	57	0	**5**	75	115	0.00	3
Random (general)	34	16	0	5	26	7	0.03	0
Euclidean (fixed)	120	204	0	5	400	306	0.01	4
Euclidean (general)	123	203	0	**5**	199	235	0.05	0
Manhattan (fixed)	136	250	0	5	193	149	0.01	3
Manhattan (general)	187	366	0	**5**	209	219	0.03	1
Infinity (fixed)	98	164	0	5	277	273	0.02	3
Infinity (general)	95	160	0	**5**	117	205	0.04	0
Average	106	178	0	5.0	187	189	0.02	1.8

[a]Bold figures: one of the instances solved to optimality in branching phase without branching any nodes.

ascent algorithm takes longer average solution times to solve problem instances of the same sizes compared to the result in [22] in exchange of a much smaller optimality gap. This may be caused by differences in some implementational details that are not fully disclosed in [22].

Effectiveness of the branch-and-cut approach: We can also claim that the branch-and-cut approach is an effective solution methodology since 356 of the 380 test instances are solved to optimality within 20 min. In terms of average solution times, the branch-and-cut approach consistently outperforms the dual ascent method except for the smallest set of test instances (1000 nodes, 3500 edges, 2 levels). In terms of the quality of solutions, it also has a smaller average optimality gap for most of the test instances

Table 3
Computations to study impact of changing demand node distributions among levels[a]

Approach / Problem type	Branch-and-cut on EDSC				Dual ascent			
	Average time (s)	SD of time	Average gap (%)	# solved to optimal	Average time (s)	SD of time	Average (%) gap	# solved to optimal
$\|N\| = 800$, $\|E\| = 2800$, 4 levels, demand node distribution: 80/160/240/320								
Random (fixed)	62	66	0.000	5	120	209	0.002	4
Random (general)	105	111	0.000	**5**	35	13	0.037	0
Euclidean (fixed)	484	612	0.466	4	351	285	0.002	4
Euclidean (general)	405	525	0.480	4	597	514	0.042	0
Average	264	328	0.237	4.5	276	255	0.021	2.0
$\|N\| = 800$, $\|E\| = 2800$, 4 levels, demand node distribution: 100/170/230/300								
Random (fixed)	68	81	0.000	**5**	197	233	0.000	4
Random (general)	51	43	0.000	**5**	187	347	0.030	1
Euclidean (fixed)	425	639	0.226	4	327	277	0.006	2
Euclidean (general)	325	501	0.076	**4**	544	477	0.022	1
Average	217	316	0.076	4.5	314	333	0.015	2.0
$\|N\| = 800$, $\|E\| = 2800$, 4 levels, demand node distribution: 200/200/200/200								
Random (fixed)	55	57	0	**5**	75	115	0.002	4
Random (general)	34	16	0	5	26	7	0.032	0
Euclidean (fixed)	120	204	0	5	400	306	0.006	4
Euclidean (general)	123	203	0	**5**	199	235	0.046	0
Average	83	120	0	5.0	175	166	0.022	2.0
$\|N\| = 800$, $\|E\| = 2800$, 4 levels, demand node distribution: 300/230/170/100								
Random (fixed)	19	8	0	5	14	5	0.000	5
Random (general)	20	8	0	5	17	7	0.022	3
Euclidean (fixed)	35	39	0	5	84	62	0.000	5
Euclidean (general)	36	37	0	5	28	21	0.043	0
Average	27	23	0	5.0	36	24	0.016	3.3
$\|N\| = 800$, $\|E\| = 2800$, 4 levels, demand node distribution: 320/240/160/80								
Random (fixed)	15	5	0	5	12	4	0.000	5
Random (general)	17	5	0	5	15	5	0.010	3
Euclidean (fixed)	33	40	0	5	71	56	0.000	5
Euclidean (general)	32	39	0	5	23	21	0.046	1
Average	24	22	0	5.0	30	22	0.014	3.5

[a]Bold figures: one of the instances solved to optimality in branching phase without branching any nodes.

compared to the benchmark method. In many cases, the higher average solution times and gaps of the branch-and-cut approach are caused by few instances which take much longer time to solve than other instances of the same configuration. The following observations are made from studying the behavior of the solver on some selected instances.

1. 14 problems are solved to optimality in branching phase without branching any nodes due to the cuts generated from the heuristic. Recall that we solve the heuristic as the first step in the branching phase and the resulting LP with the new cuts

Table 4
Computations to study impact of changing number of levels[a]

Approach / Problem type	Branch-and-cut on EDSC				Dual ascent			
	Average time (s)	SD of time	Average gap (%)	# solved to optimal	Average time (s)	SD of time	Average gap (%)	# solved to optimal
$\|N\| = 1000$, $\|E\| = 3500$, 2 levels, demand node distribution: 500/500								
Random (fixed)	21	13	0.000	**5**	15	5	0.254	4
Random (general)	18	3	0.000	5	110	206	0.004	4
Euclidean (fixed)	262	525	0.042	4	46	37	0.000	5
Euclidean (general)	34	26	0.000	5	91	159	0.004	4
Average	84	142	0.011	4.8	66	102	0.066	4.3
$\|N\| = 1000$, $\|E\| = 3500$, 3 levels, demand node distribution: 333/333/334								
Random (fixed)	22	4	0	5	27	10	0.000	5
Random (general)	26	4	0	5	29	9	0.010	1
Euclidean (fixed)	65	64	0	5	216	158	0.002	3
Euclidean (general)	68	60	0	5	273	281	0.027	0
Average	45	33	0	5.0	136	114	0.010	2.3
$\|N\| = 1000$, $\|E\| = 3500$, 4 levels, demand node distribution: 250/250/250/250								
Random (fixed)	34	5	0.000	5	889	1237	0.000	5
Random (general)	35	5	0.000	5	230	424	0.012	1
Euclidean (fixed)	306	503	0.014	4	821	797	0.004	3
Euclidean (general)	304	507	0.014	4	285	459	0.058	0
Average	170	255	0.007	4.5	556	729	0.019	2.3
$\|N\| = 1000$, $\|E\| = 3500$, 5 levels, demand node distribution: 200/200/200/200/200								
Random (fixed)	48	15	0.000	5	752	647	0.002	4
Random (general)	50	17	0.000	5	360	697	0.040	0
Euclidean (fixed)	358	478	0.174	4	1012	1493	0.002	4
Euclidean (general)	397	472	0.174	4	408	626	0.032	1
Average	213	245	0.087	4.5	633	866	0.019	2.3

[a]Bold figures: one of the instances solved to optimality in branching phase without branching any nodes.

added but without the variable fixing constraints is solved again before branching any nodes. If the resulting LP solution is optimal, no branching nodes are created. Those 14 instances do not take significantly more time than those not requiring branching. For these instances, the process reaches the branching phase because no violated inequalities are found rather than tailing-off.

2. In order to study the behavior of the solver on challenging instances, we conduct further investigation on the 9 instances not solved to optimality from the last set of Table 5. This is the set of 20 instances in which we solve the least number of them to optimality. No branching nodes are created for those nine instances due to the time limit. All nine instances have optimality gaps within 1.09% from the heuristic.
3. In the next experiment, we quadruple the maximum allowable time from 20 to 80 min. Five out of the nine instances are solved to optimality in 24 min without branching. On average, there are 21 612 cuts generated for these five instances. Because violated cuts are added and inactive constraints are purged during the solution process, we measure the maximum number of constraints reached in all iterations

Table 5
Computations to study the impact of increasing number of edges[a]

Approach / Problem type	Branch-and-cut on EDSC				Dual ascent			
	Average time (s)	SD of time	Average gap (%)	# solved to optimal	Average time (s)	SD of time	Average gap (%)	# solved to optimal
$\|N\| = 600$, $\|E\| = 1200$, 5 levels, demand node distribution: 80/100/120/140/160								
Random (fixed)	96	36	0	5	379	174	0.200	0
Random (general)	108	113	0	5	225	18	0.388	0
Euclidean (fixed)	150	157	0	**5**	442	211	0.408	0
Euclidean (general)	130	160	0	5	371	224	0.486	0
Average	121	116	0	5.0	354	157	0.371	0.0
$\|N\| = 600$, $\|E\| = 1600$, 5 levels, demand node distribution: 80/100/120/140/160								
Random (fixed)	94	39	0	5	685	464	0.208	0
Random (general)	60	18	0	5	271	13	0.448	0
Euclidean (fixed)	161	163	0	**5**	622	199	0.348	0
Euclidean (general)	90	69	0	5	283	35	0.564	0
Average	101	72	0	5.0	465	178	0.392	0.0
$\|N\| = 600$, $\|E\| = 2000$, 5 levels, demand node distribution: 80/100/120/140/160								
Random (fixed)	460	457	0.034	4	789	274	0.440	0
Random (general)	612	546	0.000	5	535	359	0.450	0
Euclidean (fixed)	647	654	0.655	3	905	405	0.478	0
Euclidean (general)	538	524	0.044	4	470	248	0.706	0
Average	564	545	0.183	4.0	675	321	0.519	0.0
$\|N\| = 600$, $\|E\| = 2400$, 5 levels, demand node distribution: 80/100/120/140/160								
Random (fixed)	811	586	0.373	2	880	392	0.276	0
Random (general)	745	612	0.052	3	811	680	0.450	0
Euclidean (fixed)	612	653	0.090	3	876	293	0.388	0
Euclidean (general)	587	601	0.282	3	424	51	0.942	0
Average	689	613	0.199	2.8	748	354	0.514	0.0

[a]Bold figures: one of the instances solved to optimality in branching phase without branching any nodes.

as an indicator of the degree of difficulty for each instance. The average maximum number of constraints for the five instances is 1967. For the remaining four instances, two instances did not reach branching stage and the other two have one and two branching nodes created respectively in 80 min. Their average integrality gap is improved from 0.45% to 0.26%. The average number of cuts generated and average maximum constraints reached are 44 455 and 3808 respectively, which are twice as many as those from the five solved instances.

4. To further test the effect of branching on the four difficult instances, we increase the threshold of tailing-off, forcing the process to get into branching phase earlier. Thus, more time is spent in the branching phase. However, since the maximum number of constraints goes as high as 8000, with more than 55 000 cuts generated, it reaches the limit of computer memory and dramatically slows down the solver due to swapping between hard drive and main memory. As a result, only 1 to 2 more branching nodes were created in the 80 min time period.

Table 6
Computations to study impact of increasing number of nodes and edges[a]

Approach / Problem type	Branch-and-cut on EDSC				Dual ascent			
	Average time (s)	SD of time	Average gap (%)	# solved to optimal	Average time (s)	SD of time	Average gap (%)	# solved to optimal
$\|N\| = 400$, $\|E\| = 1400$, 4 levels, demand node distribution: 100/100/100/100								
Random (fixed)	206	378	0	5	179	50	0.100	1
Random (general)	22	11	0	5	118	99	0.079	1
Euclidean (fixed)	24	8	0	5	213	23	0.242	0
Euclidean (general)	18	6	0	5	144	27	0.270	0
Average	68	101	0	5.0	163	50	0.173	0.5
$\|N\| = 600$, $\|E\| = 2100$, 4 levels, demand node distribution: 150/150/150/150								
Random (fixed)	281	403	0	5	409	66	0.084	0
Random (general)	133	154	0	5	297	65	0.236	0
Euclidean (fixed)	157	142	0	5	332	163	0.170	0
Euclidean (general)	91	61	0	5	269	75	0.286	0
Average	166	190	0	5.0	327	92	0.194	0.0
$\|N\| = 800$, $\|E\| = 2800$, 4 levels, demand node distribution: 200/200/200/200								
Random (fixed)	55	57	0	**5**	75	115	0.005	3
Random (general)	34	16	0	5	26	7	0.032	0
Euclidean (fixed)	120	204	0	5	400	306	0.006	4
Euclidean (general)	123	203	0	**5**	199	235	0.046	0
Average	83	120	0	5.0	175	166	0.022	1.8
$\|N\| = 1000$, $\|E\| = 3500$, 4 levels, demand node distribution: 250/250/250/250								
Random (fixed)	34	5	0.000	5	889	1237	0.000	5
Random (general)	35	5	0.000	5	230	424	0.012	1
Euclidean (fixed)	305	502	0.014	4	821	797	0.004	3
Euclidean (general)	303	503	0.014	4	285	459	0.058	0
Average	169	254	0.007	4.5	556	729	0.019	2.3
$\|N\| = 1200$, $\|E\| = 4200$, 4 levels, demand node distribution: 300/300/300/300								
Random (fixed)	71	21	0.000	5	280	481	0.000	5
Random (general)	88	28	0.000	**5**	74	24	0.254	0
Euclidean (fixed)	305	501	0.232	4	856	543	0.006	3
Euclidean (general)	344	485	0.170	**4**	1259	862	0.021	1
Average	202	259	0.101	4.5	617	478	0.070	2.3

[a]Bold figures: one of the instances solved to optimality in branching phase without branching any nodes.

From the analysis, we observe that instances requiring branching because of no violated cuts found are not particularly difficult as seen from the 14 instances solved without creating branching nodes. Problem instances that are hard because of their size can be solved to optimality given more time such as the 5 instances solved to optimality in 24 min. Problem instances that are hard because their polyhedral structures cause tailing off, can not be solved efficiently to optimality by the branch-and-cut approach. Therefore, the best solution strategy for the solver is to avoid branching even if more computationally time is available. For large size but less computationally challenging instances, this allows us to solve more instances to optimality. For difficult instances

involving tailing-off, this strategy allows more time spent in improving lower bound from which a better heuristic upper bound can be expected.

Effect of problem characteristics: To identify problem characteristics that may have a significant impact on solution times or quality, we consider the following three major attributes.

1. *Cost structures*: In Table 2, we compare the average solution times obtained from solving 80 instances of different cost structures (Random, Euclidean, Manhattan, Infinity cost functions for both fixed and general cost ratios) on graphs with four levels and sizes varying from 400 nodes, 1400 edges to 800 nodes and 2800 edges. In each case, the demand nodes are equally distributed across the four levels. The results indicate that random cost function instances with fixed cost ratios take longer to solve for 400-node and 600-node instances. For larger problem sizes (800 nodes) Euclidean, Manhattan, and Infinity cost functions tend to be harder. In general, there is no consistent pattern in solution times among different cost functions especially given the large standard deviations. Thus we use Euclidean cost function to represent all three geographic functions for the rest of our computational experiment. The two remaining cost functions, Euclidean and Random, are the most commonly studied cost functions in literature (for example [2,8,9,16,25]). The dual ascent approach takes much longer average time to solve each set of instances and does not show any pattern in favor of any particular cost functions either. From Tables 2–6, there is no concrete evidence showing the impact of cost functions and cost ratios on solution times.
2. *Distributions of demand nodes*: The impact of changing the distribution of demand nodes (across different levels) is significant from the result detailed in Table 3. In Table 3, we consider 800-node, 2800-edge, and 4-level instances. The five distributions of demand nodes considered are 80/160/240/320, 100/170/230/300, 200/200/200/200, 300/230/170/100, 320/240/160/80 from level 1 to level 4. Results in Table 3 clearly indicate that as the proportion of nodes requiring higher grade facilities increases, the problem becomes easier to solve for both approaches. The decrease in solution times is significant except for one instance set in the dual ascent approach. For the branch-and-cut approach, this phenomenon can be explained by the smaller graph size due to effective preprocessing in such instances as well as the fact that separations tend to be cheaper since more demand nodes are closer to the root in the extended graph. For example, the 20 instances tested with demand node distribution 80/160/240/320 have average node reduction of 6% and arc reduction of 57% compared to 31% and 80% respectively for the 20 instances with demand node distribution 320/240/160/80. Again, the branch-and-cut approach takes less solution times and solves more instances to optimality compared to the dual ascent approach in all five test sets. It also has smaller average optimality gaps in three sets.
3. *Problem sizes*: The impact of problem size on solution times is significant from three experiments whose results are found in Tables 4–6. In Table 4, we consider

graphs with 1000 nodes, 3500 edges, and number of levels from 2 to 5. When converted to the corresponding extended graph ED, the largest instances solved in this test have 5000 nodes and 39 000 arcs. In each case the number of demand nodes in each level is kept the same at $|N|/L$. We are able to handle instances up to five levels for this problem size within 20 min, solving 75 of the 80 instances attempted to optimality. It is clear that the longer average solution times and higher standard deviations in five rows of Table 4 are caused by the one instance not solved to optimality in that row. The table also shows that increasing the number of levels increases the time taken to solve the instance with an exception in the two-level set due to an outlier. The 20 five-level instances take on average 272 iterations before branching with a range between 95 and 501 and less than 200 iterations in the branching stage for the 2 instances requiring branching. There are 2833 cuts generated on average, among which, 315, 2018, 0, and 500 come from BFS, reverse BFS, mincut, and reverse mincut, respectively. This shows that we are able to count more on the less time consuming reverse BFS than on the reverse mincut due to the design of the switch between stage one and stage two. Note that the regular mincut serves only as an optimality check under certain circumstances in stage one. The dual ascent approach also exhibits longer solution times when the number of levels increases. However, the branch-and-cut approach has smaller optimality gaps and much shorter average solution times except for the two-level set.

In Table 5, we consider the impact of changing the number of edges in the graph. The number of nodes is fixed at 600 and the number of levels is fixed at 5. As a result, the largest problems include 3000 nodes and 26 400 arcs in the extended graph. Except for one set in the branch-and-cut approach, it is clear that increasing the edge density of the graph makes the instance harder to solve. Since these instances have more demand nodes at lower level, it is expected that they are harder instances for the branch-and-cut approach from previous discussions. Indeed, in the 2400-edge set, 9 out of 20 instances are not solved to optimality. Those 20 instances have 1.7% of nodes and 36% of arcs removed by the graph reduction algorithm. As a result, in the initial formulation, there are on average 590 (roughly $600 \times (1 - 0.017)$) constraints and 15 445 variables (approximately $2400 \times 2 \times 5 \times (1 - 0.36)$). During the solution process, the maximum number of constraints in the formulation reaches 2206 on average, about 3.7 times of its initial size while there are, on average, 12 484 constraints generated. It shows that the selection of parameter perc and purging inactive constraints keep the LP under manageable size for the available computing resources. For easier instances, the maximum number of constraints in the LP is less than 1.5 times of those in the initial formulation. Comparing to the benchmark approach, our approach has shorter average solution times and smaller average optimality gaps for all four test sets. In addition, the dual ascent method solves none of the 80 instances to optimality while the branch-and-cut approach solved 67 instances to optimality.

In Table 6, we increase the size of the graph from 400 to 1200 nodes keeping the number of levels fixed at 4 and the edge density fixed at $|E| = 3.5|V|$. It is clear

that the problems become more difficult to solve as size increases with an exception from the 600-node set. However, we are able to solve 96 out of 100 instances to optimality and the average times for all five sets are less than 4 min. In the 1000-node and 1200-node sets, each has two instances not solved to optimality. The optimality gaps left by those from the larger size (1200 nodes) instances are bigger than those left by the smaller size (1000 nodes) instances. Average solution times from the branch-and-cut approach are much lower than those from the dual ascent method. A closer look at the twenty 400-node instances shows that, on average, 3.4% of the computational time is spent in preprocessing; 31% in solving LP by CPLEX; 29% in generating reverse BFSs; 15% in generating reverse mincuts; 11% in purging rows; 3% and 4.5% in generating regular BFSs and mincuts, respectively. This again shows that the design of the solver allows us to depend more on the less time consuming reverse BFS cut than the reverse mincut and to balance between the time spent in solving LPs and the time taken by separation algorithms. From Tables 4–6, we demonstrate that the size of an instance has a significant impact on solution times.

In conclusion, our study shows that MLND problems can be solved efficiently by a branch-and-cut approach with limited amount of computer resources on a personal computer. The conversion of MLND problem to directed Steiner tree problem not only allows us to treat the problem as a standard Steiner tree problem but also provides a tighter LP relaxation to the problem supported by the fact that 356 of the 380 instances are solved to optimality without branching. Finally, the EDSC formulation can be modified to include problems with switching equipment required at the node connecting one grade of facility to another by associating the equipment costs with the cross-level arcs. However, the problem is expected to be harder as graph reduction methods and heuristics relying on the property of free ride on the cross-level arcs are no longer valid.

References

[1] K. Altinkemer, Z. Yu, Two matching based algorithm for tree network design, J. Inform. Optim. Sci. 15 (2) (1994) 213–226.

[2] Y.P. Aneja, An integer linear programming approach to the Steiner tree problem in graphs, Networks 10 (2) (1980) 167–178.

[3] C. Arnst, This is the official day the telecom wars begin, BusinessWeek (October 13, 1997), pp. 32–33.

[4] A. Balakrishnan, T.L. Magnanti, P. Mirchandani, A dual-based algorithm for multi-level network design, Management Sci. 40 (5) (1994) 567–581.

[5] A. Balakrishnan, T.L. Magnanti, P. Mirchandani, Modeling and Heuristic worst-case performance analysis of the two-level network design problem, Management Sci. 40 (7) (1994) 846–867.

[6] A. Balakrishnan, N.R. Patel, Problem reduction methods and a tree generation algorithm for the Steiner network problem, Networks 17 (1987) 65–85.

[7] T. Barnea, D. Benanav, K. Dutta, I. Eisenberg, J. Euchner, E. Gilbert, A. Goodarzi, E. Lee, Y.L. Lin, J. Martin, J. Peterson, R. Pope, R. Salgame, S. Sardana, G. Sevitsky, Arachne: planning the interoffice facilities network at NYNEX, Interfaces 26 (1) (1996) 85–101.

[8] J.E. Beasley, An SST-based algorithm for the Steiner tree problems in graphs, Networks 19 (2) (1989) 1–16.
[9] S. Chopra, E. Gorres, M.R. Rao, Solving the Steiner tree problem on a graph using branch and cut, ORSA J. Comput. 4 (3) (1992) 320–335.
[10] S. Chopra, M.R. Rao, The Steiner tree problem I: formulations, compositions and extension of facets, Math. Programming 64 (2) (1994) 209–230.
[11] S. Chopra, M.R. Rao, The Steiner tree problem II: formulations, compositions and extension of facets, Math. Programming 64 (2) (1994) 231–248.
[12] H. Crowder, E. Johnson, M. Padberg, Solving large scale zero-one linear programming problems, Oper. Res. 31 (1983) 803–834.
[13] J. Current, H. Pirkul, The hierarchical network design problem with transshipment facilities, European J. Oper. Res. 52 (1991) 338–347.
[14] J. Current, C. ReVelle, J. Cohon, The hierarchical network design problem, European J. Oper. Res. 27 (1986) 57–66.
[15] C. Duin, A. Volgenant, Reducing the hierarchical network design problem, European J. Oper. Res. 39 (6) (1989) 332–344.
[16] C. Duin, A. Volgenant, Heuristics for the hierarchical network design problem, in: J. Lenstra, H. Tijms, A. Volgenant (Eds.), Twenty-five Years of Operations Research in the Netherlands, CWI Tract 70, Centre for Mathematics and Computer Science, Amsterdam, 1990, pp. 23–24.
[17] C. Duin, A. Volgenant, The multi-weighted Steiner tree problem, Ann. Oper. Res. 33 (1991) 451–469.
[18] B. Gavish, Topological design of centralized computer networks — formulations and algorithms, Networks 12 (1982) 355–377.
[19] B. Gavish, K. Altinkemer, Backbone network design tools with economic tradeoffs, ORSA J. Comput. 2 (3) (1990) 236–252.
[20] B. Gavish, H. Pirkul, Computer and database location in distributed computer systems, IEEE Trans. Comput. C-35 (7) (1986) 583–590.
[21] ILOG Inc., ILOG CPLEX Reference Manual, ILOG, Inc., Mountain View, CA, 1999.
[22] P. Mirchandani, The multi-tier tree problem, INFORMS J. Comput. 8 (3) (1996) 202–218.
[23] I. Neuman, Class dependent routing in backbone computer networks, INFOR 28 (3) (1990) 247–265.
[24] M. Padberg, G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Rev. 33 (1) (1991) 60–128.
[25] H. Pirkul, J. Current, S. Nagarajan, The hierarchical network design problem: a new formulation and solution procedures, Transportation Sci. 25 (3) (1991) 175–182.
[26] H. Pirkul, S. Narasimhan, Primary and secondary route selection in backbone computer networks, ORSA J. Comput. 6 (1) (1994) 50–60.
[27] R.E. Tarjan, Data Structures and Network Algorithm, The Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1983, p. 108.