INFORMS Journal on Computing
Vol. 18, No. 3, Summer 2006, pp. 348–365
ISSN 1091-9856 | EISSN 1526-5528 | 06 | 1803 | 0348

DOI 10.1287/ijoc.1040.0123

The Multilevel Capacitated Minimum Spanning Tree Problem

Ioannis Gamvros, Bruce Golden, S. Raghavan
The Robert H. Smith School of Business, University of Maryland, College Park, Maryland 20742-1815, USA
{igamvros@rhsmith.umd.edu, bgolden@rhsmith.umd.edu, raghavan@umd.edu}

In this paper, we consider the multilevel capacitated minimum spanning tree (MLCMST) problem, a generalization of the well-known capacitated minimum spanning tree (CMST) problem, that allows for multiple facility types in the design of the network. We develop two flow-based mixed integer programming formulations that can be used to find tight lower bounds for MLCMST problems with up to 150 nodes. We also develop several heuristic procedures for the MLCMST problem. First, we present a savings-based heuristic. Next, we develop local search algorithms that use exponential size, node-based, cyclic and path exchange neighborhoods. Finally, we develop a hybrid genetic algorithm for the MLCMST. Extensive computational results on a large set of test problems indicate that the genetic algorithm is robust and, among the heuristics, generates the best solutions. They are typically 6.09% from the lower bound and 0.25% from the optimal solution value.

Key words: networks; tree algorithms; heuristics; local search; multiexchange neighborhood; genetic algorithms
History: Accepted by Prakash Mirchandani, Area Editor for Telecommunications and E-Commerce; received June 2003; revised August 2004, September 2004, October 2004; accepted November 2004.

1. Introduction

The topological design of local access communication networks involves many analytical challenges (see Gavish 1991 for an overview). One of the problems that deals with the design of communication networks is the terminal layout problem, which is typically referred to as the capacitated minimum spanning tree (CMST) problem. In the CMST problem, we are given a set of terminal nodes, each with its own traffic requirements, that we wish to transport to a given location (i.e., the *central node*). Furthermore, a facility (communications link) with a fixed capacity is available for installation between any two nodes. We wish to design a feasible minimum-cost tree network to carry the traffic from the terminal nodes to the central node.

In practice, it seems unreasonable to assume that only a single facility type is available to the network planner. In this paper, we deal with a generalization of the CMST that addresses many of the practical concerns associated with local access network design. Specifically, we consider the multilevel capacitated minimum spanning tree (MLCMST) problem, in which there are multiple types of facilities (with different capacities and costs) that can be installed between two nodes in the network. This problem arises at telecommunications companies building fiber-optic based local access networks.

The MLCMST problem can be formally defined as follows. We are given a graph $G = (N, E)$, with node set N and edge set E. One of the nodes in N is the central node, which we will denote by c, and the rest are terminal nodes. W_i is the (integer) traffic requirement (or weight) of node i to be transported to the central node c. We are also given a set of facility types $\Lambda = \{0, 1, \ldots, L\}$ with capacities $Z_0 < Z_1 < \cdots < Z_L$ and cost functions C_{ij}^l denoting the cost of a facility of type l installed between nodes i and j. We wish to find a minimum-cost *tree network* on G to carry the traffic from the terminal nodes to the central node.

Figure 1 gives an example of the MLCMST problem. In Figure 1(a), the square node in the center is the central node to which traffic must be transported. There are three types of facilities with capacities $Z_0 = 1$, $Z_1 = 3$, and $Z_2 = 10$, and each node generates one unit of traffic. Figure 1(b) shows a feasible multilevel capacitated spanning tree. Notice that the topology of the network is a tree, and traffic on any link is less than or equal to the capacity of the facility installed on the link.

In general, the traffic requirements for each of the terminal nodes can be different. However, in this paper, we restrict our attention to unit-demand problems (i.e., problems with $W_i = 1$ for all terminal nodes). We discuss extensions of our work to the nonunit demand case in §7. Additionally, we restrict our attention to (realistic) cost functions that exhibit strict economies of scale that are typical in communication networks. In other words, the cost of each facility satisfies the relationship

$$C_{ij}^y < \frac{Z_y}{Z_x} C_{ij}^x$$

for every edge $\{i, j\} \in E$, and $x < y$.

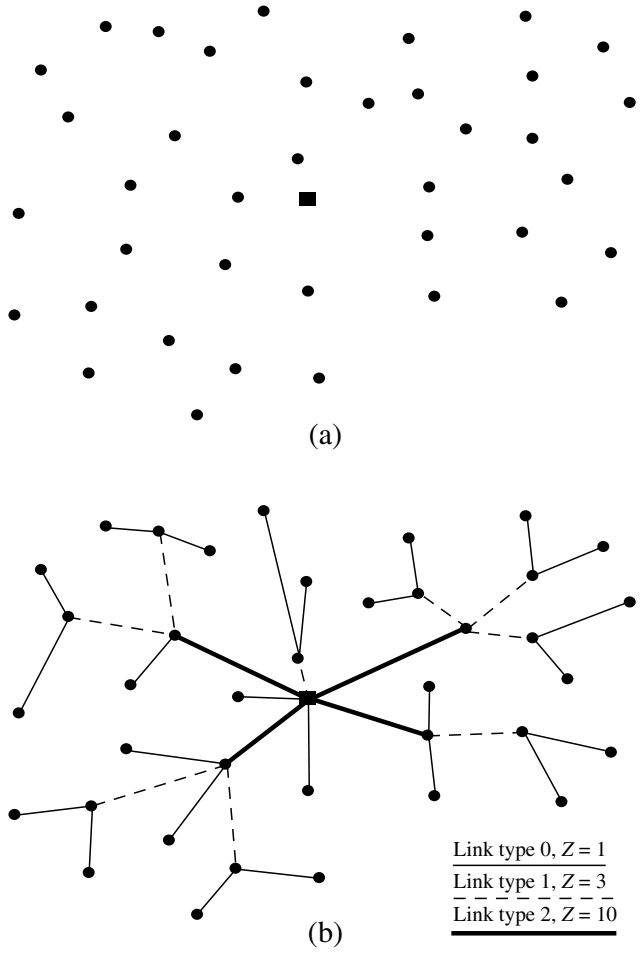


Figure 1 **Multilevel Capacitated Minimum Spanning Tree. (a) Nodes in the Network. (b) Feasible Multilevel Capacitated Spanning Tree**

We also impose the condition that only a single facility is installed on a link. This condition is, actually, not restrictive. If multiple facilities can be installed on a link, Salman et al. (2001) point out, in the context of a related problem, that by applying a dynamic programming algorithm, the problem can be converted to one where only a single facility type is installed on a link. This is done by determining the optimal combinations for all traffic levels, and creating new facility types, each representing one of the optimal combinations. For example, assume that three types of facilities are available for installation, with capacities $Z_0 = 1$, $Z_1 = 3$, and $Z_2 = 12$ and costs (per unit of traffic) of 1, 2, and 6, respectively. Then, by applying dynamic programming, we get the optimal facility combinations shown in Table 1. (Note that we calculate facility combinations in this manner up to the maximum traffic amount carried on any edge in the network.) Let f represent the traffic on a link. Then the table indicates that for $0 < f \leq 1$, it is optimal to install a facility of type 0, while for $6 < f \leq 7$, it is optimal to install a facility of type 0 and two facilities of type 1. By creating new facility types with costs and capacities indicated in columns three and four of the table, we obtain a problem where only a single facility is installed on a link.

Table 1 **Optimal Combinations of Facilities**

New facility type	Facility combination	Cost	Capacity
0	Z_0	1	1
1	Z_1	2	3
2	Z_0, Z_1	3	4
3	$2 \times Z_1$	4	6
4	$Z_0, 2 \times Z_1$	5	7
5	Z_2	6	12

Related Literature

The MLCMST does not appear to have been given much attention by researchers previously. A closely related problem that also deals with multiple facility types is the so-called local access network design (LAND) problem (Berger et al. 2000, Salman et al. 2001) or the so-called Telpak problem (Rothfarb and Goldstein 1971). In the LAND problem, as in the MLCMST, we are given traffic demand from nodes in a network that needs to be transported to a central node, several facility types with differing costs and capacities, and we wish to design a minimum-cost network to transport this traffic. However, the topology of the underlying network is not restricted to be a tree. Herein lies the distinction between the MLCMST problem and the LAND problem.

Berger et al. (2000) propose a tabu search procedure for the LAND problem to obtain good heuristic solutions for problems with up to 200 nodes and 9 facility types. In their problem, the demand from a node must travel to the central node along the same path (i.e., demand splitting is not allowed). Salman et al. (2001) study the version of the LAND problem in which demand splitting is allowed, and propose a branch-and-bound procedure using a technique called *search by objective relaxation*. Using this technique, they solve problems with ten nodes and up to nine facility types to optimality.

A generalization of the LAND problem that also involves discrete facility types to be installed on the edges of a graph is the network loading problem (NLP). In this problem, the traffic demands are distributed (i.e., demands exist between any two nodes in the network) as opposed to the centralized case where all demands are directed to a central node. Magnanti et al. (1993) consider the NLP with a single facility type and describe the convex hull of two core subproblems of the NLP. In a sequel paper Magnanti et al. (1995) provide two approaches—Lagrangian relaxation and a cutting plane approach—to solve

the NLP with two facility types. With this approach they are able to solve to integer optimality problems with up to 10 nodes and 23 edges. Bienstock et al. (1998) also consider the NLP with a single facility type. They describe two formulations, one based on metric inequalities and the other an aggregated multicommodity formulation. They develop cutting plane algorithms for both formulations, adding additional strong valid inequalities, and achieve comparable results with both formulations on two sets of real-world problems. Bienstock and Günlük (1996) study a generalization of the NLP called the capacity expansion problem (CEP). Here, an existing capacitated network is provided and the objective is to minimize the cost of additional facilities necessary to route the traffic. If there are no existing facilities this problem is identical to the NLP. They study the polyhedral structure of a mixed-integer programming formulation of this problem, with two facility types, and develop a cutting plane algorithm using facet defining inequalities. Dahl and Stoer (1998) consider the CEP with additional survivability requirements, and present a cutting plane algorithm that they test on a set of real-world problems.

In a survey paper, Gavish (1982) describes the Telpak problem and restricts it to a tree (thus considering the MLCMST problem). He presents a formulation for this problem and also points to the lack of research attention given to it. In fact, Gavish (1991) states: "The Telpak problem is a fundamental design problem in the local distribution of telephone systems. The problem is especially important in the design of fiber-based local access networks. It is surprising to find that in spite of its practical importance, the Telpak problem has received very little attention in the published literature. We are not aware of recent research on developing exact methods for solving the problem to optimality, or of the development of new types of heuristics for this class of problems." Our paper represents a step towards rectifying this situation for this important telecommunications problem.

The rest of this paper is organized as follows. In §2, we present two flow-based mixed-integer programming formulations that can provide exact solutions (for smaller problems) and lower bounds (for larger problems). In §3, we present a heuristic for the MLCMST. In §§4 and 5, we develop (two) local search approaches and a genetic algorithm, respectively. Finally, in §6, we present extensive computational experiments, and, in §7, we suggest directions for future work.

2. Mathematical-Programming Formulations

In this section, we describe some flow based mixed-integer programming formulations for the MLCMST problem. First, we describe a straightforward single commodity formulation (SCF), which is equivalent to the one presented by Gavish (1982) for the Telpak problem restricted to a tree. Next, we strengthen this formulation in a similar fashion to Gavish (1983) and Gouveia (1993) for the CMST problem, to obtain an enhanced single commodity formulation (ESCF). Finally, we develop a stronger multicommodity flow formulation (MCF) for the MLCMST problem.

In both the single commodity and the multicommodity formulations, following a common technique used to build strong formulations for tree problems, we develop our model on a directed graph. We create the directed graph by replacing each edge $\{i, j\} \in E$ in the original graph by two directed arcs(i, j) and (j, i), and denote the set of directed arcs by A. The cost of installing a facility of type l on an arc(i, j), C_{ij}^l, is identical to the cost of installing a facility of type l on edge $\{i, j\}$. Observe that any feasible multilevel capacitated spanning tree can be directed towards the central node to obtain a solution of identical cost on the directed graph. Consequently, on the directed graph, we wish to find a minimum-cost directed tree where every node has a directed path to the central node, and the traffic on every arc on the tree is less than or equal to the capacity of the facility installed on the arc. We note that, when a feasible multilevel capacitated spanning tree is directed towards the central node, none of its arcs are directed out of the central node. Consequently, we delete all arcs directed out of the central node from the arc set A. Also note, that when directed, (i) the outdegree of each node in the tree is one, and (ii) a feasible multilevel capacitated tree uses the single facility installed between two edges in one direction.

In the single commodity formulation, we create a single type of commodity and specify a supply of W_i units for each node $i \in N\backslash\{c\}$ and a demand of $W_c = \sum_{i \in N\backslash\{c\}} W_i$ units at the central node (i.e., node c). Note that in the unit-demand case, $W_i = 1$ for $i \in N\backslash\{c\}$ and $W_c = |N| - 1$. Let f_{ij} denote the flow on arc(i, j). Let y_{ij}^l be a binary decision variable representing whether or not a facility of type l is installed on arc(i, j). We can now state the single commodity formulation as follows:

Single Commodity Formulation (SCF)

$$\min \sum_{(i,j)\in A} \sum_{l=0}^{L} C_{ij}^l y_{ij}^l$$

$$\text{subject to} \sum_{j:(j,i)\in A} f_{ji} - \sum_{m:(i,m)\in A} f_{im} = \begin{cases} -W_i & \text{if } i \neq c, \\ W_c & \text{if } i = c, \end{cases} \quad \forall i \in N, \qquad (1)$$

$$f_{ij} \le \sum_{l=0}^{L} Z_l y_{ij}^l \quad \forall (i,j) \in A, \tag{2}$$

$$\sum_{j:(i,j)\in A} \sum_{l=0}^{L} y_{ij}^l = 1 \quad \forall i \in N \backslash \{c\}, \tag{3}$$

$$\sum_{l=0}^{L} (y_{ij}^l + y_{ji}^l) \le 1 \quad \forall \{i,j\} \in E,\ i,j \ne c, \tag{4}$$

$$y_{ij}^l \in \{0,1\} \quad \forall (i,j) \in A,\ \forall l \in \Lambda, \tag{5}$$

$$f_{ij} \ge 0 \quad \forall (i,j) \in A. \tag{6}$$

In the above formulation, constraint set (1) ensures that every node sends its demand to the central node. Constraint (2) ensures that the flow sent on an arc is less than the capacity of the facility installed on that arc. Constraint (3) ensures that there is exactly one arc, and one facility type, directed out of node i. Finally, constraint (4) guarantees that no more than one facility is installed between two nodes, and the facility is used in only one direction.

The linear programming relaxation of the SCF formulation is quite weak. We make the following observations that allow us to strengthen the single commodity flow formulation. For any arc(i,j), with $j \ne c$, that has a facility of type L installed on it, the flow on arc(i,j) is less than or equal to $Z_L - W_j$. Thus we may replace constraint (2) by

$$f_{ij} \le (Z_L - W_j) y_{ij}^L + \sum_{l=0}^{L-1} Z_l y_{ij}^l$$

when $j \ne c$. Additionally, if there is any facility on arc(i,j), there must be flow on arc(i,j). Further, if the facility installed on arc (i,j) is of type $l > 0$, then there must be at least $Z_{l-1} + 1$ units of flow on the arc (if there was less flow on the arc we could lower the cost of the solution by installing a lower capacity facility). Consequently, we can add the constraint

$$f_{ij} \ge y_{ij}^0 + \sum_{l=1}^{L} (Z_{l-1} + 1) y_{ij}^l \quad \forall (i,j) \in A \tag{7}$$

to the formulation. With these changes, the enhanced single commodity flow formulation may be stated as follows.

Enhanced Single Commodity Formulation (ESCF)

$$\min \sum_{(i,j)\in A} \sum_{l=0}^{L} C_{ij}^l y_{ij}^l$$

$$\text{subject to } (1),\ (3),\ (4),\ (5),\ (6),\ (7)$$

$$f_{ic} \le \sum_{l=0}^{L} Z_l y_{ic}^l \quad \forall (i,c) \in A, \tag{8}$$

$$f_{ij} \le (Z_L - W_j) y_{ij}^L + \sum_{l=0}^{L-1} Z_l y_{ij}^l \quad \forall (i,j) \in A,\ j \ne c. \tag{9}$$

The LP relaxation of ESCF produces much higher quality lower bounds than SCF. On a set of 200 test problems reported in §6, the gap (defined as 1 − the ratio of the optimal LP objective to the optimal IP objective) between the optimal LP objective and the optimal IP objective decreased from an average of 17.28% to 6.14% (see Table 3 for additional details). The increase in running time to solve the LP relaxation of ESCF compared to the LP relaxation of SCF is marginal (an average of 0.23 seconds vs. 0.2 seconds). We also note that we were able to use ESCF successfully to solve, to integer optimality, problems with 20 terminal nodes and problems with 30 terminal nodes with the central node in the center of the graph within four minutes of CPU time, on average. However, there was a wide variation in the CPU time (to solve ESCF to integer optimality) with the longest running time being just under 30 minutes of CPU time.

We now develop a multicommodity formulation whose linear programming relaxation provides tighter lower bounds than ESCF. In this formulation, we create a commodity for each terminal node, with a supply of one at each terminal node, and a demand of one at the central node. We denote the set of the different commodities by K. In our notation, the origin of commodity $k \in K$ is denoted by $O(k)$ and the destination of commodity k is the central node, c. We let f_{ij}^k denote the flow of commodity k on arc(i,j). It represents the fraction of the demand of terminal node $O(k)$ (i.e., $W_{O(k)}$) that is sent on a facility on arc (i,j). We also introduce two types of arc variables. x_{ij} is a binary variable denoting whether a facility (of any type) is installed on arc (i,j), and y_{ij}^l is a binary decision variable denoting whether or not a facility of type l is installed on arc(i,j). We now state the multicommodity formulation for the MLCMST.

Multicommodity Formulation (MCF)

$$\min \sum_{(i,j)\in A} \sum_{l=0}^{L} C_{ij}^l y_{ij}^l$$

$$\text{s.t.} \sum_{j:(j,i)\in A} f_{ji}^k - \sum_{m:(i,m)\in A} f_{im}^k = \begin{cases} -1 & \text{if } i = O(k), \\ 1 & \text{if } i = c, \\ 0 & \text{otherwise,} \end{cases} \quad \forall i \in N,\ \forall k \in K, \tag{10}$$

$$f_{ij}^k \le x_{ij} \quad \forall (i,j) \in A,\ \forall k \in K, \tag{11}$$

$$\sum_{k\in K} W_k f_{ic}^k \le \sum_{l=0}^{L} Z_l y_{ic}^l \quad \forall (i,c) \in A, \tag{12}$$

$$\sum_{k\in K} W_k f_{ij}^k \le (Z_L - W_j) y_{ij}^L + \sum_{l=0}^{L-1} Z_l y_{ij}^l \quad \forall (i,j) \in A,\ j \ne c, \quad (13)$$

$$\sum_{j\in N} x_{ij} = 1 \quad \forall i \in N \backslash \{c\}, \quad (14)$$

$$x_{ij} + x_{ji} \le 1 \quad \forall \{i,j\} \in E,\ i,j \ne c, \quad (15)$$

$$\sum_{l=0}^{L} y_{ij}^l = x_{ij} \quad \forall (i,j) \in A, \quad (16)$$

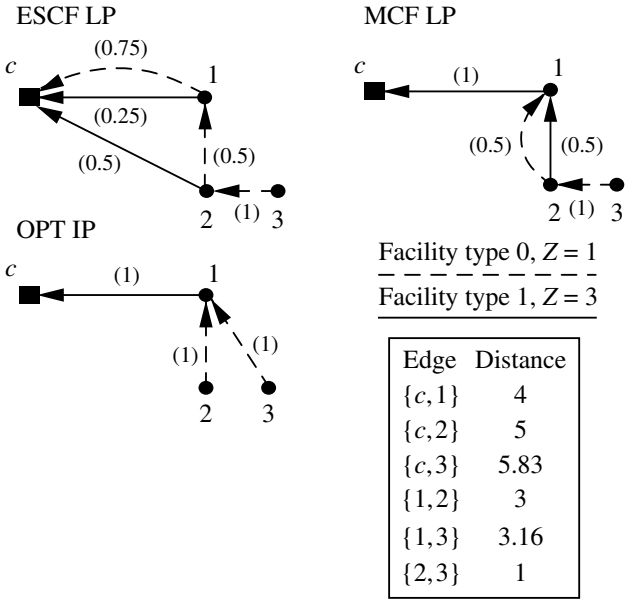
$$\sum_{k\in K} W_k f_{ij}^k \ge y_{ij}^0 + \sum_{l=1}^{L} (Z_{l-1} + 1) y_{ij}^l \quad \forall (i,j) \in A, \quad (17)$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A, \quad (18)$$

$$y_{ij}^l \in \{0,1\} \quad \forall (i,j) \in A,\ \forall l \in \Lambda, \quad (19)$$

$$f_{ij}^k \ge 0 \quad \forall (i,j) \in A,\ \forall k \in K. \quad (20)$$

Together, constraints (10), (11), (14), and (15) ensure that the network topology is a directed tree (directed into the central node). Constraints (12) and (13) guarantee that the total traffic on any arc is less than the capacity of the facility installed on that arc (with the strengthening, as in constraint (9)). Constraint (16) ensures that only one facility is installed on an arc, and only if the arc is part of the MLCMST tree (i.e., $x_{ij} = 1$). Finally, constraint (17) ensures that if a facility of type l is installed on arc(i,j), then the total traffic carried on that facility must be greater than or equal to $Z_{l-1} + 1$ units.



Edge	Distance
{c,1}	4
{c,2}	5
{c,3}	5.83
{1,2}	3
{1,3}	3.16
{2,3}	1

Figure 2 MCF Is Stronger than ESCF

We observe that any feasible solution $(\bar{\mathbf{f}}, \bar{\mathbf{y}}, \bar{\mathbf{x}})$ to the LP relaxation of MCF can be converted to a feasible solution $(\tilde{\mathbf{f}}, \tilde{\mathbf{y}})$ to the LP relaxation of ESCF, by setting $\tilde{y}_{ij}^l = \bar{y}_{ij}^l$, and $\tilde{f}_{ij} = \sum_{k\in K} W_k \bar{f}_{ij}^k$. On the other hand, as Figure 2 illustrates, there are solutions that are feasible to the LP relaxation of ESCF but infeasible to the LP relaxation of MCF, thus showing that MCF is a stronger formulation than ESCF. In Figure 2, node c is the central node and nodes 1, 2, and 3 have unit demand. There are two facilities, type 0 with a capacity of 1, and cost equal to the distance, and type 1 with a capacity of 3, and cost equal to twice the distance. Figure 2 shows the optimal solution to the LP relaxation of ESCF, the optimal solution to the LP relaxation of MCF, and the optimal integer solution to the problem. Observe that the optimal solution to the LP relaxation of ESCF has $y_{1c}^0 = 0.75$, $y_{1c}^1 = 0.25$, $y_{21}^0 = 0.5$, $y_{2c}^1 = 0.5$, $y_{32}^0 = 1$, $f_{1c} = 1.5$, $f_{21} = 0.5$, $f_{2c} = 1.5$, and $f_{32} = 1$. This solution has a cost of 12.5. The optimal solution to the LP relaxation of MCF has $x_{1c} = 1$, $x_{21} = 1$, $x_{32} = 1$, $y_{1c}^1 = 1$, $y_{21}^0 = 0.5$, $y_{21}^1 = 0.5$, $y_{32}^0 = 1$, $f_{1c}^1 = 1$, $f_{1c}^2 = 1$, $f_{1c}^3 = 1$, $f_{21}^2 = 1$, $f_{21}^3 = 1$, and $f_{32}^3 = 1$. This solution has a cost of 13.5 showing that the MCF formulation is stronger than ESCF. The optimal integer solution to the problem sets $y_{1c}^1 = 1$, $y_{21}^0 = 1$, and $y_{31}^0 = 1$ and has a cost of 14.16.

The strengthened formulation MCF obtained by disaggregating the flow variables comes at a cost in the size of the formulation. ESCF contains $|A|(|\Lambda|+1)$ variables, and $3|A| + 2|N| - 1 - |E|$ constraints (other than the nonnegativity restrictions). MCF contains $|A|(|\Lambda| + |N|)$ variables, and $|A||N| + 3|A| + |N|^2 - 1 - |E|$ constraints (other than the nonnegativity constraints). Thus, for a 21 node problem on a complete graph with 3 facility types, ESCF contains 1,600 variables (of which 1,200 are binary) and 1,031 constraints, while MCF contains 9,600 variables (of which 1,600 are binary) and 9,830 constraints.

In our computational tests comparing the LP relaxation of ESCF and MCF, we found that the gap between the LP relaxation of MCF and the optimal MIP solution was on average 0.6% smaller than the gap between the LP relaxation of ESCF and the optimal MIP solution (see Table 3 for additional details). On the other hand, the running time to solve the LP relaxation of MCF was two orders of magnitude greater than the running time to solve the LP relaxation of ESCF (see Table 4). As problem size gets larger, this difference is costly from a computational perspective. For example, for 50-node problems where the central node is in the center of the graph, we found that the LP relaxation of ESCF solves in 1.37 seconds on average while the LP relaxation of MCF solves in 536.96 seconds on average. Consequently, we decided to use ESCF to provide lower bounds for

the MLCMST problem (as some of our heuristic techniques worked quite successfully to provide solutions for problems with up to 150 nodes). We also note that, due to the size of MCF, we were unable to solve the MCF formulation to integer optimality for any of the 20-node instances in a reasonable amount of time.

As is evident from our discussion in this section, the MIP formulations were only able to consistently solve to optimality unit-demand problems with up to 20 and 30 nodes in a reasonable amount of computational time. Consequently, in the rest of this paper, we develop several heuristic solution procedures for the MLCMST problem.

3. Construction Heuristic

In this section, we describe a savings based construction heuristic for the MLCMST problem. First we present some notation. For a given graph $G = (N, E)$, let $\mathbf{T}$ denote a feasible tree for the MLCMST problem. For each node i in $\mathbf{T}$, we define a set P_i that contains all the nodes on the path from i to the central node (i.e., including node i, but excluding the central node). For example, in Figure 3, $P_2 = \{2, 4\}$ and $P_7 = \{7, 8\}$. We define the predecessor of node i as the first node on the path from i to the central node and denote that node by pred(i). We also use the notation $\mathbf{Z}(i)$, where $i \in N\backslash\{c\}$ to denote the capacity of the link that connects node i to its predecessor, and let $\lambda(i)$ denote its facility type. In our example, nodes $i = \{10, 11, 12, 13\}$ have pred(i) $= 16$, $\mathbf{Z}(i) = Z_0$, and $\lambda(i) = 0$. Additionally, we define the child-parent relationship between nodes where node i is the child of pred(i), or equivalently, node pred(i) is the parent of i. For each node i, we define the set J_i that consists of the subtree of i (i.e., node i, children of i, children of their children, and so on). Going back to our example, J_4 would be

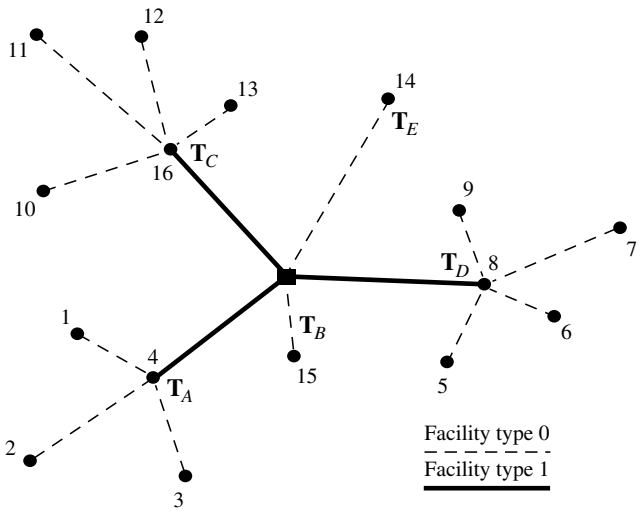


Figure 3 Example of a MLCMST Tree

$\{1, 2, 3, 4\}$. Finally, we refer to the subtrees that are rooted at the children of the central node as *rooted subtrees*.

Our construction heuristic starts with a star network in which all nodes are connected to the central node with the lowest capacity facilities possible (i.e., facility type 0 for the unit-demand case), as shown in Figure 4(b) for a three-facility example. To improve upon this solution, we would like to determine for which nodes it would be beneficial to upgrade their connections in the star network to facilities of type L. Obviously, upgrading the capacity of these connections will increase the cost. However, if we reconnect some of the other nodes that are directly connected to the central node to the nodes with upgraded connections, we might reduce the overall cost.

In order to calculate the savings generated by the upgrade and the reconnections, we first construct a savings value d_{ij} that captures the reconnection costs. We then use this value to evaluate the overall cost D_i^L (i.e., the cost of the upgrade to facility type L and the reconnections). The savings value $d_{ij} = C_{jc}^{\lambda(j)} - C_{ji}^{\lambda(j)}$, where c denotes the central node, represents the savings in removing the connection from node j to the central node and reconnecting node j to node i (in the unit-demand case $\lambda(j) = 0$). We then compute, for each node i, the overall savings D_i^L in upgrading link $\{i, c\}$ from a facility of the current type to a facility of type L. We compute

$$D_i^L = C_{ic}^{\lambda(i)} - C_{ic}^L + \max_{\substack{\{H: H\subset N,\, d_{ij}>0 \forall j\in H,\\ \sum_{t\in J_i^H} W_t \le Z_L\}}} \sum_{j\in H} d_{ij} \qquad (21)$$

for each node $i \in N\backslash\{c\}$. The first two terms in (21) represent the change in cost from upgrading link $\{i, c\}$ to a facility of type L. The third term represents the greatest savings obtained by removing the connections $\{j, c\}$ (since all nodes j were originally connected to the central node) and introducing the connections $\{j, i\}$, while ensuring that the capacity of facility type L is not violated. In other words, set H contains all the nodes that will send their traffic to i, which together with node i's traffic must be less than or equal to the capacity of facility type L. Here we have used J_i^H to denote the set J_i that would result *if* H was implemented. Once we have computed all D_i^L, we then select the largest and implement the upgrade and the reconnections corresponding to this choice. We repeatedly apply this procedure by recomputing the savings d_{ij} and D_i^L until there are no more positive savings (i.e., $D_i^L \le 0$). Figure 4(c) illustrates the network obtained after applying this procedure for a three-facility example.

At this point, we again seek to determine for which nodes it would be beneficial to upgrade their connections to type $L-1$. The only difference from the previous step is that, before, all connections considered for

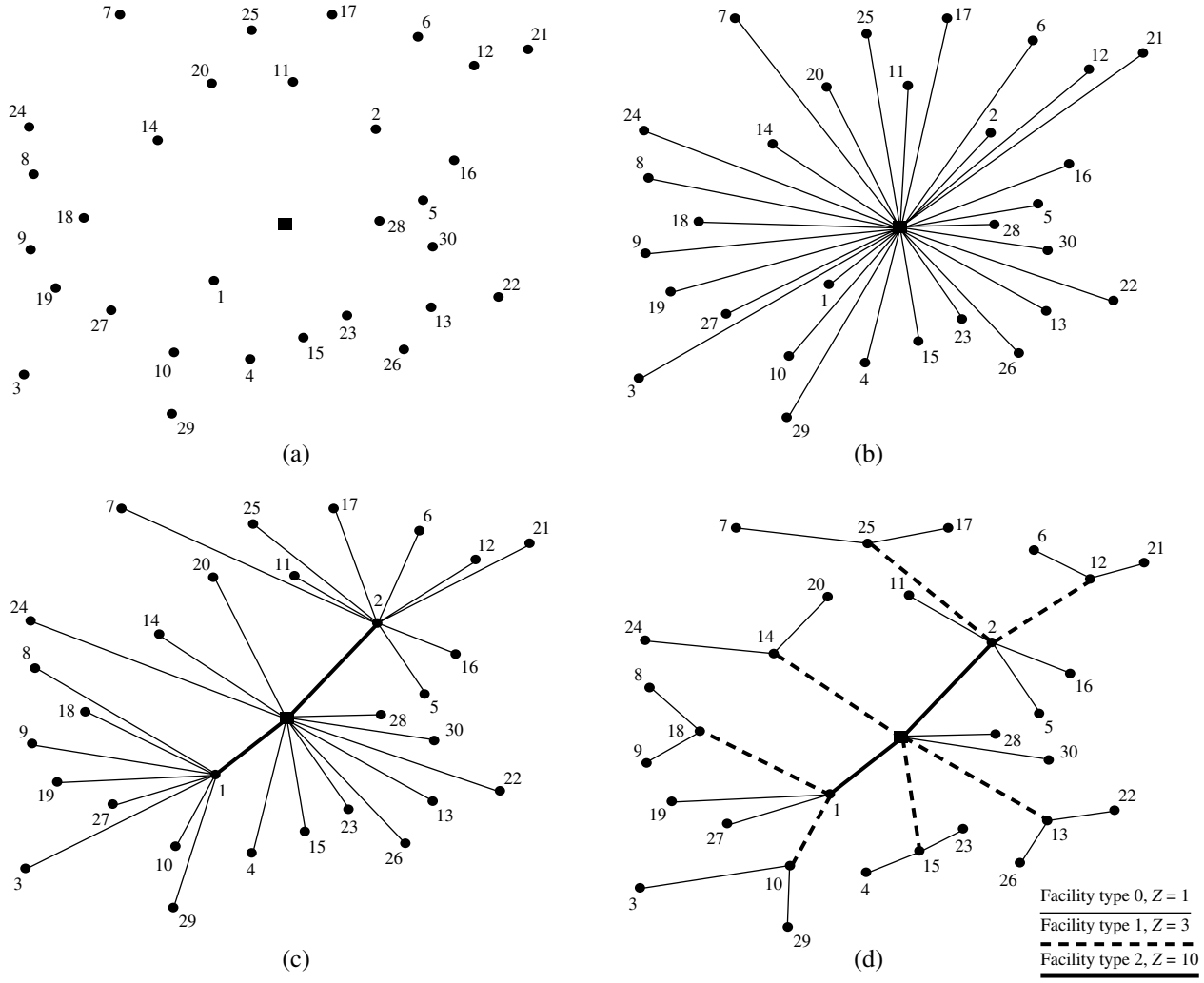


Figure 4 Construction Heuristic for a Three-Facility Unit-Demand Example

upgrading were directed to the central node but now there are some links directed to other nodes as well. We take this into account when calculating the savings value d_{ij}. In general $d_{ij} = C^{\lambda(j)}_{j\,\mathrm{pred}(j)} - C^{\lambda(j)}_{ji}$. We now calculate the overall savings D_i^{L-1} by upgrading the connection of node i to link type $L-1$ by computing

$$D_i^{L-1} = C^{\lambda(i)}_{i\,\mathrm{pred}(i)} - C^{L-1}_{i\,\mathrm{pred}(i)} + \max_{\substack{\{H:\,H\subset N,\,d_{ij}>0\ \forall\, j\in H,\\ \sum_{t\in J_i^H} W_t \le Z_{L-1},\\ \sum_{t\in J^H_{\mathrm{pred}(i)}} W_t \le \mathbf{Z}(\mathrm{pred}(i))\}}} \sum_{j\in H} d_{ij}. \tag{22}$$

Notice that, in (22), we are interested in upgrading the connection between node i and its predecessor. Also, we consider reconnecting some set of nodes H to node i only if the capacity of link $\{i, \mathrm{pred}(i)\}$ and the capacity of link $\{\mathrm{pred}(i), \mathrm{pred}(\mathrm{pred}(i))\}$ are not violated. These two conditions are expressed as $\sum_{t\in J_i^H} W_t \le Z_{L-1}$ and $\sum_{t\in J^H_{\mathrm{pred}(i)}} W_t \le \mathbf{Z}(\mathrm{pred}(i))$. We then select the largest D_i^{L-1} and implement the upgrade and the reconnections. We keep doing this until there are no more positive savings.

In general, the steps of the construction procedure are shown in Figure 5. After the initial solution is generated, the algorithm calculates the savings generated when upgrading connections to facility type L. When no more savings can be achieved by these upgrades, we look at savings generated by upgrading connections to facility type $L-1$, then facility type $L-2$, and so on, until we consider savings with facility type 0 (if facility type 0 has capacity $Z_0 = 1$, facility type 0 need not be considered). The savings in all cases can be computed from

$$D_i^{l} = C^{\lambda(i)}_{i\,\mathrm{pred}(i)} - C^{l}_{i\,\mathrm{pred}(i)} + \max_{\substack{\{H:\,H\subset N,\,d_{ij}\ge 0\ \forall\, j\in H,\\ \sum_{t\in J_i^H} W_t \le Z_l,\\ \sum_{t\in J_m^H} W_t \le \mathbf{Z}(m),\,\forall\, m\in P_{\mathrm{pred}(i)}\}}} \sum_{j\in H} d_{ij}. \tag{23}$$

```
Begin
  Generate initial star solution
  l ← L
  while (l ≥ 0) do
    exit ← false;
    while (exit = false) do
      for every i not in the final solution do
        Compute D_i^l
      end for
      If (largest D_i^l > 0)
        Select largest D_i^l, implement upgrade and reconnections
        Add i in the final solution
      else (exit ← true)
    end while
    l ← l − 1
  end while
end
```

Figure 5 Steps of Construction Heuristic

The first two terms in (23) refer to the change in cost that comes by upgrading the link from i to its predecessor to a type l facility. The third term computes the best set of nodes H to connect to node i to maximize the savings achieved while ensuring that no facility capacity is exceeded along the path from node i to the central node (i.e., $\sum_{t\in J_i^H} W_t \le Z_l$, $\sum_{t\in J_m^H} W_t \le \mathbf{Z}(m)$, $\forall\, m \in P_{\text{pred}(i)}$). Figure 4(d) shows the final solution obtained by this procedure on a three-facility example.

We now discuss the worst-case running time of this heuristic for the unit-demand case. Observe that in (23), to find D_i^l we have to compute the set H that maximizes the savings in upgrading the connection from i to pred(i). We discuss this optimization in further detail. To facilitate the computation of the set H, we keep track of the following three variables at each node of the current tree: slack(i) $= \mathbf{Z}(i) - \sum_{t\in J_i} W_t$, which keeps track of the amount of unused capacity on the facility between i and pred(i), res(i) $= \min_{m\in P(i)}$ slack(m), the capacity of the path from i to the central node, and subnet(i), which is the last node on the path from i to the central node (this identifies all nodes in the same rooted subtree). We observe that for a given tree all three quantities can be computed on the tree in linear ($\mathcal{O}(|N|)$) time by making an upward pass from the leaf nodes to the central node (computing slack(i)) and a downward pass from the central node to the leaves (computing res(i) and subnet(i)).

To compute the set H that maximizes D_i^l, we first sort all $d_{ij} > 0$ in decreasing order. This takes $\mathcal{O}(|N|\log|N|)$ time. Next, we consider the $d_{ij} > 0$ in sorted order connecting j to i if it is feasible to do so (i.e., the path from i to the central node has sufficient capacity). Observe that if the connection from node i is upgraded, then initially res(i) $= \min(Z_l - 1,$ res(pred(i))). It may appear that we need only consider the first $\min(Z_l - 1,$ res(pred(i))) nodes on the sorted d_{ij} list and connect them to node i. However, this is not true. When subnet(j) $\neq$ subnet(i), connecting node j to node i reduces the available capacity on the path from node i to the central node by one. When subnet(j) = subnet(i) (i.e., nodes i and j are in the same rooted subtree of the central node) let t denote the node at which the path from node i to the central node and the path from node j to the central node intersect. If slack(k) > res(k) for all k on the path from node i to t, connecting node j to node i leaves res(i) unchanged (i.e., the available capacity on the path from node i to the central node remains unchanged). In other words, it may be possible to connect some nodes in i's rooted subtree (i.e., the rooted subtree to which i belongs) to node i to obtain savings, even if res(i) $= 0$. Consequently, to compute the maximizing set H we need to take into account whether or not the node j is in i's rooted subtree. When the node is from a different rooted subtree, we simply check that the residual capacity at node i is greater than 0. When the node is in the same rooted subtree we need to check res(k) and slack(k) for nodes k on the path $P(i)$, which takes $\mathcal{O}(\min(|\Lambda|, |N|))$ time. The maximum number of nodes in a subtree is Z_L, and thus the time to compute the maximizing set H is $\mathcal{O}(|N|\log|N| + |N| + Z_L \min(|\Lambda|, |N|))$ or $\mathcal{O}(|N|\log|N| + Z_L|N|)$.

With this discussion, we may now derive the worst-case running time as follows. Since we compute D_i^l in each iteration for all nodes that are not in the final solution and there are at most $|N| - 1$ such nodes, computing D_i^l takes at most $\mathcal{O}(|N|^2\log|N| + Z_L|N|^2)$ time. The number of iterations is bounded by $|N| + |\Lambda|$. Thus, the overall running time of the construction heuristic is $\mathcal{O}(|N|^2(\log|N| + Z_L)(|N| + |\Lambda|))$.

4. Local Search

We now describe two local search procedures for the MLCMST problem. Our local search procedures use a "node-based, multiexchange neighborhood structure" originally proposed by Ahuja et al. (2001) for the CMST problem. With this neighborhood structure, Ahuja et al. were able to achieve results that improved upon the previously best known solutions for CMST problems (specifically, improvements of 3.2% on average and 18% maximum were reported). As the MLCMST problem is a generalization of the CMST problem, it is likely that this neighborhood structure will prove effective in a local search technique for the MLCMST problem.

For the purposes of our presentation, we will use notation that is similar to that in Ahuja et al. (2001) except when it conflicts with our own. Recall from §3 that we denote the spanning tree that provides a solution to the MLCMST as $\mathbf{T}$. For each node i in $\mathbf{T}$, we denote by $\mathbf{T}[i]$ the rooted subtree of $\mathbf{T}$ that contains

node i. We also denote by $\mathbf{S}[i]$ the set of nodes that are contained in $\mathbf{T}[i]$. Note that we define $\mathbf{S}[i]$ to contain node i as well. For example, in Figure 3, for $\mathbf{T}[5] = T_D$, $\mathbf{S}[5] = \{5, 6, 7, 8, 9\}$ (in fact for all $j \in \mathbf{S}[5]$, $\mathbf{T}[j] = T_D$ and $\mathbf{S}[j] = \mathbf{S}[5]$). We say that $\mathbf{S}[i]$ is feasible if and only if $\sum_{i \in \mathbf{S}[i]} W_i \le Z_L$ where W_i is the traffic of node i and Z_L is the largest available capacity to us. If $\mathbf{S}[i]$ is feasible, we denote by $\mathscr{C}(\mathbf{S}[i])$ the cost of a minimum-cost multilevel capacitated tree spanning the node set $\mathbf{S}[i] \cup \{c\}$, where c is the central node. Also, a subtree $\mathbf{T}[i]$ is feasible only if the associated set $\mathbf{S}[i]$ is feasible and a tree $\mathbf{T}$ is feasible only if $\mathbf{T}[i]$ is feasible for every $i \in N \backslash \{c\}$.

4.1. Neighborhood Definition

We now present two types of node-based, multi-exchange neighborhood structures (described in Ahuja et al. 2001 for the CMST)—cyclic exchanges and path exchanges—as applied to the MLCMST problem.

Cyclic Exchanges. A cyclic exchange is denoted as $i_1 - i_2 - \cdots - i_r - i_1$ and represents the following changes: node i_1 moves from $\mathbf{T}[i_1]$ to $\mathbf{T}[i_2]$, node i_2 moves from $\mathbf{T}[i_2]$ to $\mathbf{T}[i_3]$, and so on, and finally node i_r moves from $\mathbf{T}[i_r]$ to $\mathbf{T}[i_1]$. A cyclic exchange is defined only if all nodes in the exchange belong originally to different rooted subtrees. Also a cyclic exchange is feasible only if the tree $\mathbf{T}'$ obtained after the exchange is feasible. We can compute the cost of a feasible exchange by calculating

$$\mathscr{C}(\mathbf{T}') - \mathscr{C}(\mathbf{T}) = \sum_{m=1}^{r} \big(\mathscr{C}(\{i_{m-1}\} \cup \mathbf{S}[i_m] \backslash \{i_m\}) - \mathscr{C}(\mathbf{S}[i_m])\big), \tag{24}$$

where i_0 is defined as i_r. The cyclic exchange is called *profitable* if $\mathscr{C}(\mathbf{T}') - \mathscr{C}(\mathbf{T}) < 0$ and *nonprofitable* otherwise. In Figure 6(a) we present a tree $\mathbf{T}$ and denote the cyclic exchange $10 - 14 - 6 - 3 - 10$ with arrows. Figure 6(b) shows the tree $\mathbf{T}'$ after the exchange.

Path Exchanges. A path exchange is denoted by $i_1 - i_2 - \cdots - i_r$ and represents the following changes: node i_1 moves from $\mathbf{T}[i_1]$ to $\mathbf{T}[i_2]$, node i_2 moves from $\mathbf{T}[i_2]$ to $\mathbf{T}[i_3]$, and so on, and finally node i_{r-1} moves from $\mathbf{T}[i_{r-1}]$ to $\mathbf{T}[i_r]$. The only difference between the cyclic exchange and the path exchange is that in the latter no node moves from subtree $\mathbf{T}[i_r]$ to subtree $\mathbf{T}[i_1]$. Similar to cyclic exchanges, path exchanges are defined only when all nodes in the exchange originally belong to different rooted subtrees. A path exchange is feasible only if the tree $\mathbf{T}'$ obtained after the exchange is feasible. The cost of a path exchange is given by

$$\mathscr{C}(\mathbf{T}') - \mathscr{C}(\mathbf{T}) = \mathscr{C}\big(\mathbf{S}[i_1] \backslash \{i_1\}\big) + \sum_{m=2}^{r-1} \mathscr{C}\big(\{i_{m-1}\} \cup \mathbf{S}[i_m] \backslash \{i_m\}\big) + \mathscr{C}\big(\{i_{r-1}\} \cup \mathbf{S}[i_r]\big) - \sum_{m=1}^{r} \mathscr{C}(\mathbf{S}[i_m]). \tag{25}$$

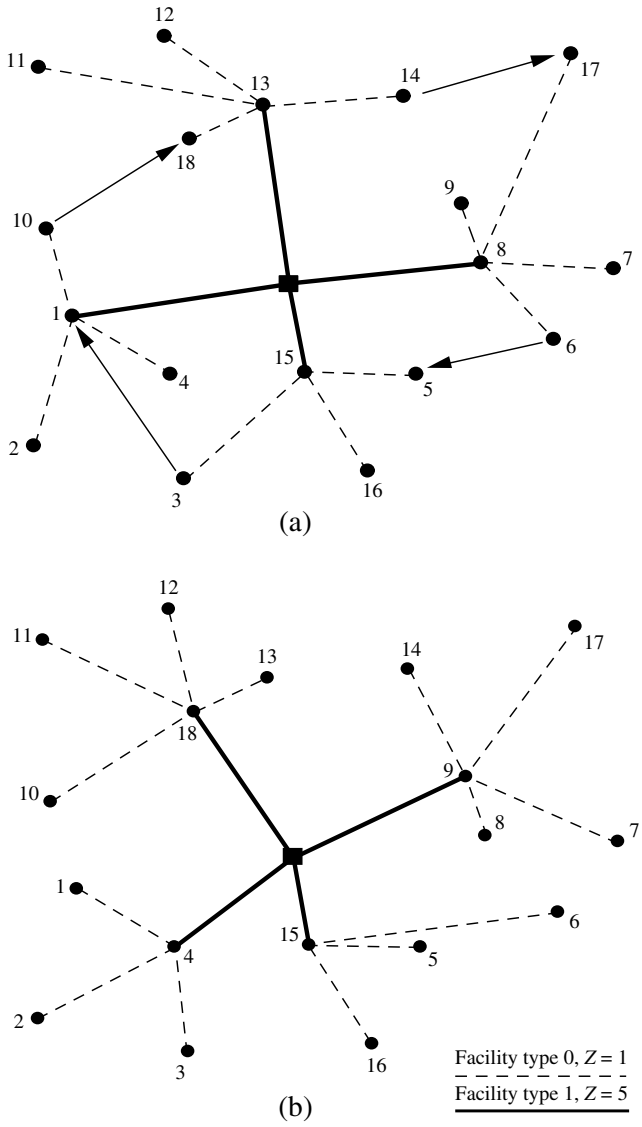


Figure 6 Cyclic-Exchange Example

The path exchange is called profitable if $\mathscr{C}(\mathbf{T}') - \mathscr{C}(\mathbf{T}) < 0$ and nonprofitable otherwise. In Figure 7(a) we present a tree $\mathbf{T}$ and denote the path exchange $10 - 14 - 6 - 3$ with arrows (note that the last node in the path exchange is not unique). Figure 7(b) shows the tree $\mathbf{T}'$ after the exchange. The neighbors of a tree $\mathbf{T}$ are defined as the set of feasible trees that can be obtained from $\mathbf{T}$ using a cyclic exchange or path exchange.

4.2. Neighborhood Construction and Exploration

The neighborhood defined above is of exponential size and is thus very large even for relatively small problems. An efficient exploration of this large neighborhood is achieved through the construction of an *improvement graph* (Thompson and Orlin 1989). We now briefly describe the construction of the improvement graph for cyclic and path exchanges as applied

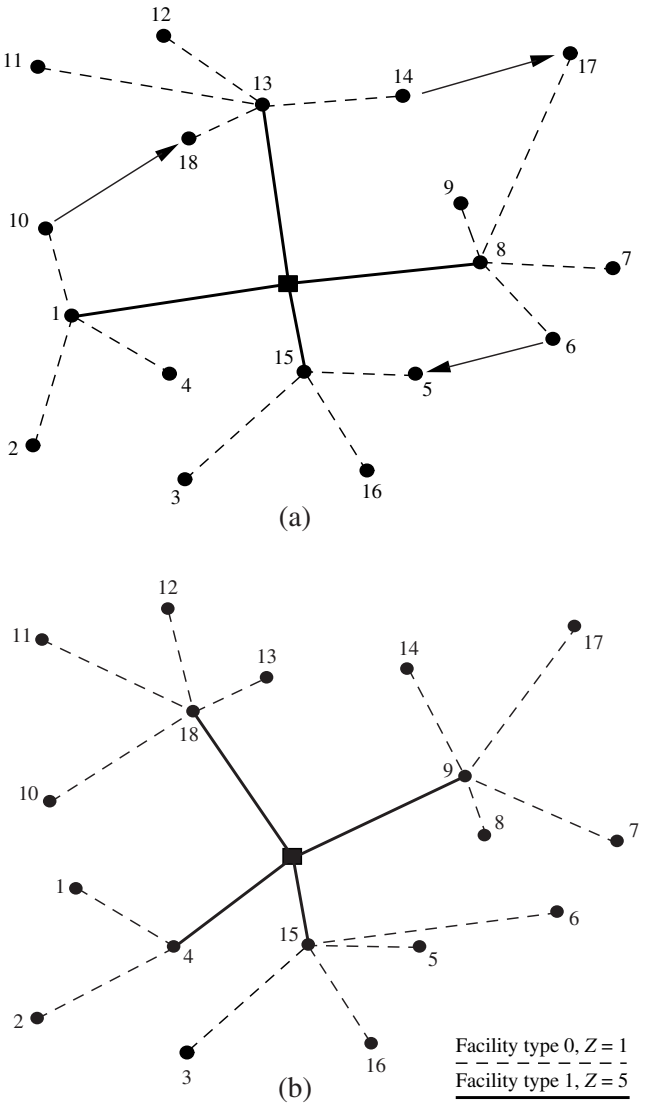


Figure 7 Path-Exchange Example

to the MLCMST problem, and explain how profitable cyclic and path exchanges can be identified.

Improvement Graph and Exchanges. An improvement graph for a specific feasible solution **T** to the MLCMST problem is denoted by **G**(**T**). The graph **G**(**T**) is directed and has the same node set as the original graph G of the MLCMST problem. Specifically, there is a one-to-one correspondence between a node i in the original graph and a node i in the improvement graph. However, the arc set of the **G**(**T**) graph is defined differently. For each pair of nodes $i, j \in N$, the arc(i, j) in **G**(**T**) is defined only if $\mathbf{T}[i] \neq \mathbf{T}[j]$ and $\{i\} \cup \mathbf{S}[j]\backslash\{j\}$ is a feasible subset of nodes (which is always true in the unit-demand case). Moreover, we denote the cost of arc(i, j) by α_{ij} and define it as

$$\alpha_{ij} = \mathcal{C}(\{i\} \cup \mathbf{S}[j]\backslash\{j\}) - \mathcal{C}(\mathbf{S}[j]).$$

Consider a directed cycle $i_1 - i_2 - \cdots - i_r - i_1$ where $\mathbf{T}[i_1] \neq \mathbf{T}[i_2] \neq \cdots \neq \mathbf{T}[i_r]$ (i.e., all nodes belong to different subtrees). Observe that the sum of the costs of the arcs on the cycle correspond exactly to the change in cost when the cyclic exchange $i_1 - i_2 - \cdots - i_r - i_1$ is implemented. A directed cycle $i_1 - i_2 - \cdots - i_r - i_1$ is called *subset-disjoint* if the subtrees $\mathbf{T}[i_1]$, $\mathbf{T}[i_2], \ldots, \mathbf{T}[i_r]$ are different rooted subtrees. Notice that every subset-disjoint cycle with negative cost on the graph **G**(**T**) represents a profitable node-based cyclic exchange for the tree **T** (Thompson and Orlin 1989).

To identify path exchanges, the improvement graph has to be modified slightly (see Ahuja et al. 2001 for complete details) by adding some nodes and arcs so that negative-cost subset-disjoint cycles represent both cyclic and path exchanges. We add an extra origin node v, and for each of the existing nodes j in the improvement graph create an arc(v, j) with cost $\mathcal{C}(\mathbf{S}[j]\backslash\{j\}) - \mathcal{C}(\mathbf{S}[j])$. Next, we create a pseudonode for each rooted subtree in **T**. For each pseudonode h, we create an arc(h, v) to the origin with cost 0, and an arc(i, h) from each of the nodes i in the original improvement graph (i.e., before we added extra nodes), if $\{i\} \cup \mathbf{S}[j]$ is a feasible subset of nodes, with cost $\mathcal{C}(\{i\} \cup \mathbf{S}[h]) - \mathcal{C}(\mathbf{S}[h])$.

Neighborhood Exploration. Once the improvement graph is constructed, we still have to identify subset-disjoint negative-cost cycles in order to identify profitable exchanges. Finding the most negative-cost subset-disjoint cycle is NP-hard. However, a subset-disjoint negative-cost cycle can be found using a modified shortest path algorithm as described in Ahuja et al. (2001). This label-correcting algorithm starts from a given node s and searches for the shortest path to all other nodes on the graph. The modifications made to the algorithm ensure that it finds only subset-disjoint cycles. The quality of the subset-disjoint cycles that are generated with this procedure depends heavily on the starting node s. Consequently, in our implementation, we select each node on the graph as a starting point, run the modified label-correcting algorithm, and then select the best overall cycle (i.e., the one with the smallest cost). Since we are interested in negative-cost subset-disjoint cycles if this procedure finds a subset-disjoint cycle with positive cost, we do not implement the exchange represented by the cycle.

4.3. MLCMST Implementation

Observe that the multiexchange neighborhood structure requires the calculation of a minimum-cost tree over the set of nodes $\mathbf{S}[i] \cup \{c\}$ that is necessary for the construction of the improvement graph. In the CMST problem, this tree is obtained by simply computing the minimum spanning tree (MST). However,

in the case of the MLCMST problem, finding this minimum-cost tree is also a MLCMST problem, but on a smaller graph. Specifically, for a MLCMST problem on a graph $G = (N, E)$ with $\Lambda = \{0, 1, \ldots, L\}$, the problem of finding a minimum-cost tree spanning the set $\mathbf{S}[i] \cup \{c\}$ will have, in the worst case (i.e., when all nodes in $\mathbf{S}[i]$ have unit demand), Z_L nodes. Since the cardinality of $\mathbf{S}[i]$ does not depend on the problem size $|N|$ but on Z_L, it is possible to have situations where, in order to evaluate $\mathscr{C}(\mathbf{S}[i])$, we need to solve a problem of size similar to the original. In practice, it is possible to compute $\mathscr{C}(\mathbf{S}[i])$ for smaller problems ($|N| \leq 20$) with the help of the ESCF model we presented in §2. However, this approach is computationally very expensive, as an MIP model has to be used to compute the cost of each arc in the improvement graph. Furthermore, as the structure of the current solution changes, the costs of the arcs in the improvement graph have to be recomputed, necessitating repeated use of an MIP model to compute the costs of the arcs.

Instead, we propose the use of the construction heuristic we developed in §3 to compute an estimate $\mathscr{C}'(\mathbf{S}[i])$ of the cost of an arc in the improvement graph. Observe that when we use the construction heuristic to compute the costs of arcs in the improvement graph, the cost of a cyclic or path exchange corresponds to the difference in cost between the tree $\mathbf{T}$ prior to the exchange (given by $\mathscr{C}'(\mathbf{T})$) and the tree $\mathbf{T}'$ after the exchange (given by $\mathscr{C}'(\mathbf{T}')$) when the rooted subtrees in the trees $\mathbf{T}$ and $\mathbf{T}'$ are obtained by the construction heuristic. Thus, a negative-cost subset-disjoint cycle in the improvement graph represents a cyclic or path exchange that yields an improved solution. Clearly, using the construction heuristic to compute costs entails some risks (in terms of the quality of the improved solutions found). On the other hand, it is very advantageous to use the heuristic due to running-time considerations. Our computational results indicate that, despite using the construction heuristic to compute costs, the performance of the local search procedures we propose is quite reasonable.

4.4. Local Search Algorithms for the MLCMST

We now present two local search procedures that use the two node-based, multiexchange neighborhood structures discussed earlier.

The first procedure starts from a feasible solution that is generated with the MLCMST construction heuristic. It then builds the improvement graph for this solution and looks for negative-cost subset-disjoint cycles starting from all nodes in the graph. The costs of the arcs in the improvement graph are computed using the MLCMST construction heuristic. We then select, among the subset-disjoint cycles found, the one with the lowest cost and we implement the exchange (i.e., either cyclic or path) represented by this cycle. Once the new solution is obtained, we build the improvement graph corresponding to this new solution and look for cycles once again. The procedure repeats these steps as long as we can identify negative-cost subset-disjoint cycles. When no improving exchanges can be found, the algorithm stops.

The second procedure, which we call randomized start local search (RSLS), considers many different random starting solutions. The random starting solutions are obtained by running the MLCMST construction heuristic on a perturbed version of the original graph. The perturbation is achieved by multiplying all the costs C_{ij}^l by a random variable ε uniformly distributed in the range $[0.7, 1.3]$. Note that, in order to keep the original cost structure consistent, we have to make sure that for each pair of nodes $i, j \in N$, the costs C_{ij}^l, $\forall l \in \Lambda$ are perturbed using the same value of ε. We generate ten different starting solutions and we perform the local search procedure presented earlier on each one individually. The final solution of the RSLS approach is the best solution obtained from the different runs.

5. Genetic Algorithm

Genetic algorithms (GA) are powerful procedures motivated by ideas from the theory of evolution and have been successfully used for a variety of problems (see Michalewicz 1996). Our GA consists of the steps shown in Figure 8. First, an initial population of individual, feasible solutions (or chromosomes) $P(t)$ is created. Next, the fitness of all the chromosomes in that population is evaluated. The fitness of an individual summarizes in a single scalar the quality of the solution represented by the individual. The evaluation of the fitness of the population is required to check the termination condition of the algorithm and during the selection of chromosomes. At first, we select r chromosomes (parents) from the old population $P(t-1)$ for reproduction (crossover). The crossover operator is essentially a set of rules that defines the way in which characteristics (or genes) from two solutions (parents) are combined to create new individuals (children). After the crossover step, the r new chromosomes (children) are assigned to the new population, $P(t)$. Next, $pop\_size - r - m$ individuals are selected from $P(t-1)$ and are copied to $P(t)$. Additionally, the best m individuals are chosen from $P(t-1)$ to be mutated. The mutated individuals then become part of $P(t)$. Finally, the new population is evaluated and the procedure starts over.

Conceptually, our genetic-algorithm approach splits the MLCMST problem into two parts: a grouping problem and a network design problem. The grouping problem aims at finding the best assignment of

```
Begin
  t ← 0
  initialize P(t)
  evaluate P(t)
  while (not termination-condition) do
    t ← t + 1
    select r parents from P(t − 1)
    let the r parents reproduce (crossover) and generate r
      offspring
    insert the r offspring to P(t)
    select (pop_size − r − m) individuals from P(t − 1) and
      copy them to P(t)
    take the best m individuals of P(t − 1)
      and mutate them
    insert the mutated individuals to P(t)
    evaluate P(t)
  end while
end
```

Figure 8 Steps of the Genetic Algorithm

nodes into groups that correspond to subtrees of the central node. In other words, we try to find the nodes i that belong in $\mathbf{S}[i]$ and form $\mathbf{T}[i]$. Recall that the set $\mathbf{S}[i]$ is feasible only if the sum of the weights in the set does not exceed the capacity of the highest capacity link. In other words, for every group, we require that $\sum_{i \in \mathbf{S}[i]} W_i \leq Z_L$. The cost of an assignment made during the grouping part of the problem is determined within our genetic algorithm by solving the network design subproblem on the set $\mathbf{S}[i] \cup \{c\}$ (where c is the central node) and determining $\mathbf{T}[i]$ for each node $i \in N$. This network design problem requires the construction of a minimum-cost multilevel tree network that will connect all the nodes in each of the groups with the central node, and the genetic algorithm uses the heuristic presented in §3 to construct trees on the groupings. The sum of the costs of the different subtrees constructed by the heuristic corresponds to the quality (fitness) of the grouping assignment by the GA.

The notion of determining groupings (or nodes in a subtree) has been used previously by some researchers for the CMST problem (Ahuja et al. 2001, Amberg et al. 1996, McGregor and Shen 1977, Sharaiha et al. 1997, Sharma 1983). Recall from our discussion in §4 that, in the case of the CMST problem, interconnecting nodes that belong to the same subtree is quite simple as it is the minimum spanning tree problem, while in the case of the MLCMST, this interconnection of nodes is identical to the original problem, albeit on a smaller graph. We now elaborate on the details of our genetic algorithm.

5.1. Representation

Our genetic algorithm uses a representation proposed by Falkenauer (1996) in the context of bin packing. This representation, as shown in Figure 9, consists of two parts, an item part and a group part. In the item part, the nodes are assumed to be ordered in increasing order, and the characters represent the group (subtree of the central node) to which the node belongs. The group part contains a list of groups (subtrees) that make up the tree solution. The representation shown in Figure 9 indicates that nodes 1, 3, 4, and 7 are in a group (group **A**), nodes 2, 5, and 6 are in a group (group **B**), nodes 8, 9, and 10 form a group (group **C**), and node 11 forms a group (group **D**). The group part lists the four groups **A**, **B**, **C**, and **D**. Observe that the order of the groups in the group part does not alter the grouping represented by the chromosome. Further, the mnemonic used to identify groupings is largely irrelevant (other than that they be distinct from each other). Also note that since the number of subtrees of the central node will vary by solution, the length of the group part of the chromosome will also vary.

5.2. Initial Population

Since any genetic material (solution characteristics)—apart from mutations—found in the final solution will come from the individuals in this population, the choice of the initial population is a very important aspect of the whole search procedure. If it is too specific, then the search will be limited to a small region of the solution space, leading to a local optimum. On the other hand, if the initial population is very diverse, then the algorithm will spend valuable computational resources exploring a variety of promising areas of the search space.

We create the initial population, by using the Esau-Williams heuristic (Esau and Williams 1966) for the CMST problem for half of the initial population and our construction heuristic from §3 for the other half of the initial population. (The Esau-Williams heuristic is the most popular heuristic for the CMST problem. It starts with a star solution, and in each iteration of the

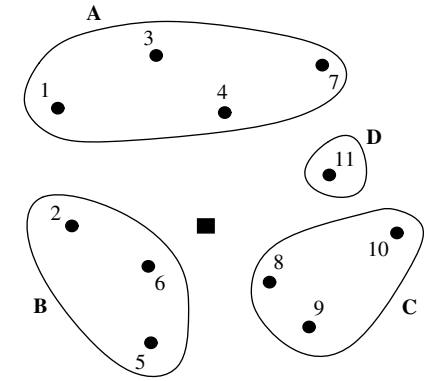


Figure 9 Example of a Group Assignment and the Respective Representation

algorithm merges two subtrees into a single subtree so that the new subtree satisfies the capacity constraints, and the savings achieved by the merge operation are maximum.) Since both of these heuristics give a unique solution for a given problem instance, and we need our initial population to be diverse, we generate multiple instances by multiplying the cost C_{ij}^l of each edge by a uniformly distributed random variable in the range $[1-\epsilon, 1+\epsilon]$. Varying the value of ϵ trades off between increased diversity of the initial population (for large ϵ) and early termination to a solution further away from the global optimum (for small ϵ). We also make sure that each individual in the initial population is unique, in order to ensure diversity, regardless of the value of ϵ. In the Esau-Williams heuristic we use the capacity and costs of facility type L, the maximum facility capacity in our problem.

We convert the solutions generated by both heuristics to the format of the GA representation by placing nodes that belong to the same subtree of the central node to the same group.

5.3. Selection Methods

We use the cost $f(i)$, obtained with the MLCMST heuristic, of the trees represented by the individuals in $P(t)$ as the fitness value. We then select the chromosomes that will participate in reproduction and create subsequent generations employing the classical roulette-wheel mechanism (Michalewicz 1996). This mechanism selects an individual i from the population $P(t)$ with probability

$$\frac{f_{\max} - f(i)}{\sum_{j \in P(t)} (f_{\max} - f(j))},$$

where

$$f_{\max} = \max_{j \in P(t)} f(j) + 1.$$

Notice the probabilities in the roulette-wheel mechanism are well defined regardless of the sign of the fitness value (i.e., even for negative fitness values).

For a given population, we discovered that the fitness values of different individuals vary only slightly. This is especially true for large problems (i.e., 100 nodes) where moving a node from one group to a neighboring group has only a small effect on the cost of the tree and the fitness value of the individual. In order to focus on the differences in the fitness of the individuals rather than the absolute cost values, we use a fitness function called *sigma truncation* (Michalewicz 1996). This modifies the original fitness value $f(i)$ to a new fitness value $f'(i)$ by computing

$$f'(i) = f(i) + \bar{f} - \gamma\sigma,$$

where $\bar{f}$ is the average of the old fitness values over the entire population, σ is the standard deviation, and γ is a constant, usually in the range $[1, 5]$. Larger values of γ correspond to higher selective pressure (i.e., fitter individuals have a larger probability of being selected). These new fitness values $f'(i)$ are then used with the roulette-wheel mechanism.

5.4. Crossover Operator

Our crossover operator is nearly identical to that of Falkenauer (1996), with a small change specific to the MLCMST problem. It is applied to the group part of the chromosome structure, and is able to work with chromosomes of varying length. It consists of the following steps:

1. Select at random two crossing sites, which define the crossing section, on the group part of the two parent chromosomes.
2. Inject the contents between the two crossing sites of the first parent just before the first crossing site of the second parent.
3. Update the membership of the items as follows. All items will belong to the group specified in the second parent, unless the group to which the item belongs in the first parent is injected into the second parent. In that case, the item would have a new membership specified by the group of the first parent. If any group is empty as a result, remove it from the group part.
4. If a group from the second parent has lost items and it now has fewer than k items, we reassign these items to other groups with probability p_{cr}. If any group is empty as a result, remove it from the group part. (This is the step that differs from Falkenauer 1996.)
5. Repeat Steps(1)–(4) reversing the role (i.e., order) of the parents.

We illustrate the procedure with the example shown in Figure 10. Consider the two chromosomes and their respective representations shown in Figures 10(a) and 10(b). To perform a crossover, we generate at random two crossing sites for each chromosome. They are just prior to and after group **B** for the first parent, and just prior to group **a** and after group **b** for the second parent, as shown in Figures 10(a) and 10(b). Injecting the contents of the crossing section of the first parent (|**B**|) to the second parent at the first crossing site of the second parent, we obtain **Babcd** as the new grouping for the child. The group membership of the nodes in the child follow from the second parent, unless the group to which the node belongs in the first parent has been injected into the group part. Since **B** is the only group injected from the first parent, the item part of the child is **aBacBBcbbdc**. Consequently, we obtain the chromosome **aBacBBcbbdc** : **Babcd** from the crossover (as shown in Figure 10(c)). Observe that each group of the offspring is either identical to a group in the first

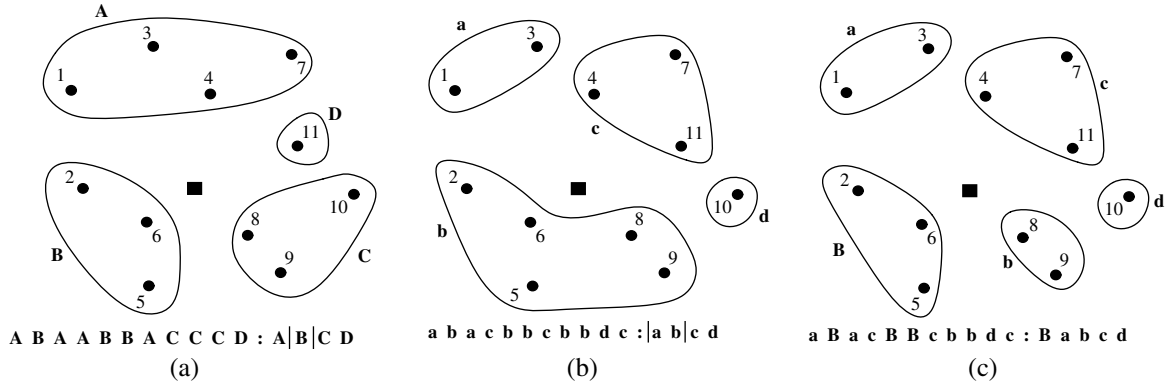


Figure 10 Crossover Operator in Genetic Algorithm

Note. Parents are represented in (a) and (b); offspring is represented in (c).

parent, or is a subset of a group in the second parent. Thus, if the parents were feasible to begin with (i.e., the total weight of the items in a grouping is less than or equal to Z_L), then the offspring will also be feasible.

The fourth step in the crossover procedure is specifically designed for our problem and it aims at improving the result of the crossover operator. In other grouping problems, like the bin-packing problem, it is desired that all the bins are close to or at their maximum capacity. Thus, usually a heuristic like first fit decreasing is applied to reassign nodes in groups with fewer items. In our case, however, it is not wise to reassign nodes to any group just because it has available capacity. Actually, it is not certain that there will be any gains by reassigning these nodes at all. Consequently, we design our reassignment procedure as follows. Our reassignment procedure focuses on groups that are (strict) subsets of a group in the second parent. If the number of items in such a group is less than a parameter k, then, with probability p_{cr}, each item is assigned to the group to which the closest (using facility type 0 costs) node not in its group belongs.

For example, suppose $k = 3$ and $p_{cr} = 0.5$, and consider the result in Figure 10(c). The only group that qualifies and must be considered for reassignments is group b. Then, with probability 0.5, we reassign node 9 to the group to which node 10 (assuming 10 is closest to it and the group has sufficient capacity to accommodate node 9) belongs, and with probability 0.5 we reassign 8 to the group to which node 6 belongs (assuming that 6 is closest to it and its group has enough capacity).

5.5. Mutation

Our mutation operator is a local search operator and is based on the cyclic and path exchange neighborhood structure presented in §4. Every time the mutation operator is to be applied to a given chromosome, we first construct an improvement graph based on the solution represented by that chromosome. We then find negative-cost subset-disjoint cycles on that graph by starting from different nodes in the graph, and we select the cycle with the lowest cost. The reassignments corresponding to the negative cycle are then implemented on the original chromosome. We apply the mutation operator on the m best individuals of each population and copy the solutions to the new population.

This approach is different from traditional mutation operators that are usually applied with low probability on any chromosome in the population (i.e., not necessarily the best individuals) and do not guarantee an improvement. This is usually done because mutation operators can significantly alter the genetic material of a chromosome. However, in our case (i) the operator will always improve a chromosome, and (ii) the genetic material of the chromosome that is being mutated will not be lost since these chromosomes (i.e., the best in the population) have a high probability of participating in crossovers or being copied directly to the next generation.

6. Computational Experiments

We now report on several computational experiments with our heuristics for the MLCMST problem. We coded our construction heuristic, local search procedures, and genetic algorithm in Visual C++. We conducted all runs on a dual-processor Pentium III PC running Windows 2000, 1 GHz clock speed, with 512 MB RAM. After computational testing on the different parameters for the GA, we selected the values shown in Table 2 (which are robust across our test set).

Our computational experiments fall into two categories. Experiments on small problems with 20 and 30 terminal nodes, and experiments on larger problems with 50, 100, and 150 terminal nodes (not counting the central node). We ran the construction heuristic, local search procedures, and the genetic algorithm on all the test problems with 100 or fewer terminal

Table 2 Genetic-Algorithm Parameters

Category	Description	Name	Value
General	Population size	—	100
	Number of parents	—	70
Selection	Selection method	—	Roulette
	Sigma truncation constant	γ	3
Mutation	Number of mutation	—	10
Initial population	Cost matrix perturbation	ϵ	0.5
Stopping criteria	Maximum number of generations	—	10
	Generations without improvement	—	5
Crossover	Probability of reassignment	p_{cr}	1
	Number of nodes for reassignment	k	<6

nodes. For the 150-node problems, we present results only with the construction heuristic and the first local search procedure since the running times of the RSLS procedure and the genetic algorithm were excessive. Further, we computed lower bounds using the LP relaxation of ESCF on all of the test problems. Additionally, for the small test problems, we also computed the optimal integer solution using the ESCF formulation.

For each problem size (i.e., 20, 30, 50, etc.) we generated three problem types—one with the central node in the center, one with the central node at the edge, and one where the central node is located randomly—each with 50 instances. We denote each problem set by specifying the number of terminal nodes in the problems followed by a lowercase letter c, e, or r, indicating whether the central node is at the center, edge, or randomly selected. In all these test problems, the nodes are generated randomly in a 20×20 square grid. The problem allows the use of three different facility types with capacities 1, 3, and 10 units of traffic, and the cost of facility type 0, C^0_{ij}, was set equal to the Euclidean distance between nodes i and j. The cost of facility type 1 was set to $2C^0_{ij}$ and the cost of facility type 2 was set to $6C^0_{ij}$. Recall that the traffic requirement for each terminal node is one unit of traffic.

6.1. Comparing SCF with MCF

Table 3 shows our computational analysis on the mathematical-programming formulations we presented in §2. We present the percentage gap of the LP relaxation for SCF, ESCF, and MCF to the optimal solutions for all 20 terminal node problems and for the 30 terminal node problems for which the central node is positioned in the center. The percentage gaps show that the average gap over all 200 problem instances for SCF is 17.28%, while the same gap is only 6.14% for ESCF and slightly better, at 5.54%, for MCF.

Table 3 Percentage Gaps of LP Relaxations of SCF, ESCF, and MCF to Optimal Integer Solution

Problem set	LP SCF (%)		LP ESCF (%)		LP MCF (%)	
	Average	Range	Average	Range	Average	Range
20c	19.78	17.65–22.56	6.11	3.08–10.06	5.62	2.86–9.61
20e	13.62	10.96–15.52	5.47	3.88–7.38	4.71	2.98–6.57
20r	17.47	13.28–22.54	6.61	2.96–9.17	5.97	1.82–8.45
30c	18.24	16.27–19.90	6.38	4.48–9.22	5.85	3.97–8.50
Aggregate	17.27	10.96–22.56	6.14	2.96–10.06	5.54	1.82–9.61

Table 4 shows the running times (in seconds) for all three LP relaxations and the IP solutions. Notice that the enhancements to the SCF result in only minor additional computational effort. The average over all 200 instances for the LP relaxation of SCF is 0.2 seconds, while the same average for the LP relaxation of ESCF is 0.23 seconds. However, the increase in computational time is significant for the LP relaxation of MCF. The average over all 200 instances for LP MCF is 18 seconds, almost 100 times greater than for the single commodity formulations.

We now remark on an edge-reduction procedure that we applied quite effectively to reduce the size of the mathematical-programming formulations. We replaced the arc set A used by the formulations with a new set, $A' \subseteq A$, by using the following rule, originally mentioned in Rothfarb and Goldstein (1971): every arc$(i, j) \in A$ is also in A' only if it satisfies $C^0_{ij} \leq C^0_{ic}$, where c is the central node. We eliminate these arcs from consideration since they cannot be part of an optimal solution. This relationship simply states that we cannot have flow directed out of node i and into node j if it is cheaper to send it directly to the central node c instead. We point out that this rule can be applied only if the costs of all the facilities are a function of the distance between i and j. This reduc-

Table 4 Running Times (in Seconds) for the Different Formulations

Problem set	LP SCF		LP ESCF		LP MCF		IP ESCF	
	Average	Range	Average	Range	Average	Range	Average	Range
20c	0.11	0.09–0.14	0.13	0.11–0.16	2.07	1.08–4.69	3.82	0.39–38.66
20e	0.24	0.22–0.27	0.29	0.25–0.33	29.18	13.86–44.00	192.53	17.30–1,751.38
20r	0.17	0.11–0.23	0.20	0.13–0.30	10.71	1.77–34.30	73.26	1.88–1,415.55
30c	0.28	0.27–0.33	0.33	0.30–0.36	30.05	10.33–74.92	244.77	9.19–1,724.45
Aggregate	0.20	0.09–0.33	0.23	0.11–0.36	18.00	1.08–74.92	128.59	0.39–1,751.38

Table 5 Percentage Gap of Heuristic Solutions to Optimal Solution

Problem set	HEU			LS			RSLS			GA		
	Average (%)	Range (%)	No. of opt.	Average (%)	Range (%)	No. of opt.	Average (%)	Range (%)	No. of opt.	Average (%)	Range (%)	No. of opt.
20c	3.47	0.00–8.97	2	0.76	0.00–3.31	21	0.27	0.00–2.46	33	0.17	0.00–1.54	37
20e	5.45	0.61–11.92	0	3.66	0.00–9.04	1	0.86	0.00–3.55	15	0.46	0.00–2.12	21
20r	3.25	0.00–8.18	2	1.05	0.00–4.54	11	0.47	0.00–3.25	22	0.12	0.00–1.66	36
30c	5.00	1.81–10.64	0	1.81	0.00–5.23	3	0.85	0.00–3.59	6	0.27	0.00–1.56	22
Aggregate	4.29	0.00–11.92	4	1.82	0.00–9.04	36	0.61	0.00–3.59	76	0.25	0.00–2.12	116

tion procedure is particularly effective as the problem size increases. For the three 50-node problem sets, the average reduction in the number of arcs by applying this procedure over all 150 instances is 46.51%, and the average reduction in running time due to the reduction in problem size is also 46.51%.

6.2. Heuristic Procedures

Table 5 compares the heuristic solutions with the optimal solutions (computed with the ESCF model). It reports on the percentage gap between the heuristic solution and the optimal solution, and also reports the number of optimal solutions that each heuristic was able to find. "HEU" stands for the construction heuristic, "LS" and "RSLS" stand for the two local search procedures presented in §4, and "GA" stands for the genetic algorithm of §5. Our results indicate that the construction heuristic finds the optimal solution for four out of the 200 problems and is, on average, within 4.29% of the optimal solution value. LS is able to find the optimal solution for 36 out of the 200 problems and is, on average, within 1.82% of the optimal solution value. RSLS is able to achieve even better results and finds the optimal solution for 76 problems and is, on average, within 0.61% of optimality. Finally, GA finds the optimal solution in 116 instances and is, on average, within 0.25% of optimality.

Tables 6 and 7 present the percentage gap between the heuristic solutions and the lower bounds that were computed by the LP relaxation of ESCF. Table 6 presents the results for "HEU" and "LS" for all problem sets while Table 7 shows the performance of "RSLS" and "GA" for problem sets with at most 100 terminal nodes. Looking at the average gap from the lower bound across all problem sets, we notice that the gaps decrease slightly as problem size increases. We note the average gaps from the lower bounds over 600 instances (not counting the 150 terminal node problems) are 9.91%, 6.99%, 6.30%, and 6.09% for HEU, LS, RSLS, and GA respectively. For the 150 terminal node problems, we obtained heuristic solutions only with the construction heuristic and LS procedure. The average gap for these, over all problem instances, was 9.29% and 6.47%, respectively.

Tables 8 and 9 present the running times of the heuristics for all problem sets. The construction heuristic, as expected, is the fastest of the four procedures. In fact, even for 150-node problems, it

Table 6 Percentage Gap of HEU and LS from Lower Bound

Problem set	HEU (%)		LS (%)	
	Average	Range	Average	Range
20c	10.24	4.54–18.81	7.35	3.17–11.58
20e	11.56	7.45–17.17	9.67	6.00–15.20
20r	10.58	4.58–19.00	8.22	3.05–11.88
30c	12.17	7.55–20.00	8.77	5.47–13.70
30e	9.98	5.01–15.00	6.57	4.20–9.68
30r	11.02	6.48–16.00	7.61	4.63–10.89
50c	11.15	7.16–15.63	8.15	5.92–11.21
50e	8.14	5.51–11.23	5.15	4.13–6.77
50r	9.53	6.12–13.50	6.47	4.63–9.96
100c	9.93	7.73–12.03	6.64	5.28–7.69
100e	6.16	4.93–7.82	3.85	3.21–4.56
100r	8.52	6.47–11.52	5.47	4.19–7.21
Aggregate(100)	9.91	4.54–20.00	6.99	3.05–15.20
150c	8.52	7.21–10.16	5.51	4.30–6.60
150e	5.05	4.07–5.99	3.24	2.71–3.84
150r	6.75	4.93–8.81	4.37	3.22–5.47
Aggregate	9.29	4.07–20.00	6.47	2.71–15.20

Note. The row Aggregate(100) presents aggregate statistics for problems with up to 100 nodes.

Table 7 Percentage Gap of RSLS and GA from Lower Bound

Problem set	RSLS (%)		GA (%)	
	Average	Range	Average	Range
20c	6.83	3.17–11.58	6.73	3.17–11.18
20e	6.70	4.07–9.90	6.29	4.07–9.30
20r	7.60	3.53–11.25	7.23	3.05–10.15
30c	7.75	4.90–12.61	7.12	4.79–10.65
30e	5.57	4.20–8.20	5.69	4.04–7.61
30r	6.88	4.17–9.91	6.64	3.72–9.49
50c	7.56	3.86–10.27	7.30	3.97–9.60
50e	4.77	3.43–5.66	4.69	3.58–5.36
50r	6.01	4.49–8.23	5.78	4.24–7.48
100c	6.45	5.27–7.39	6.38	5.08–7.40
100e	3.96	3.28–4.62	3.80	3.13–4.40
100r	5.55	4.49–6.75	5.39	4.19–6.64
Aggregate	6.30	3.17–12.61	6.09	3.05–11.18

Table 8 Running Times (in Seconds) for HEU and LS

Problem set	HEU Average	LS Average	LS Range
20c	<0.01	0.45	0.23–0.88
20e	<0.01	0.94	0.55–2.83
20r	<0.01	0.74	0.33–2.27
30c	<0.01	2.38	1.02–6.06
30e	<0.01	6.14	2.03–14.11
30r	<0.01	3.96	1.48–11.56
50c	<0.01	17.31	7.11–40.50
50e	<0.01	29.86	9.39–63.84
50r	<0.01	19.82	6.14–34.38
100c	0.04	194.16	68.38–323.33
100e	0.04	284.91	181.23–497.59
100r	0.03	237.84	109.98–412.66
150c	0.09	807.94	434.99–1,280.34
150e	0.10	1,015.32	633.38–1,629.39
150r	0.09	833.67	471.67–1,222.36

runs in under a tenth of a second! The local search procedure LS is fairly rapid, and can take about 1,000 seconds (15 minutes), on average, for 150-node problems. What is interesting to note here is that RSLS becomes computationally more expensive than the GA, as problem size increases. Consequently, we can say that RSLS is dominated by the GA, since it not only gives poorer quality solutions but it also requires more time.

In summary, the construction heuristic achieves an acceptable level of performance for a greedy heuristic and the local search procedure is able to provide higher quality solutions in a relatively short amount of time. We found that LS improves the construction heuristic solutions by 2.56%, on average, and the maximum improvement over the construction heuristic solution was 7.8%. The RSLS procedure improves upon the solutions of the local search algorithm slightly, but these improvements come at a high computational cost. Finally, the best solutions are obtained for problems with up to 100 nodes using the GA.

Table 9 Running Times (in Seconds) for RSLS and GA

Problem set	RSLS Average	RSLS Range	GA Average	GA Range
20c	5.34	2.97–7.94	11.43	8.31–29.03
20e	12.24	7.73–18.63	25.37	16.70–33.83
20r	9.52	3.59–16.33	15.52	8.34–33.48
30c	27.18	16.50–43.22	26.49	19.19–43.19
30e	91.89	64.13–119.83	76.81	57.75–96.31
30r	56.19	29.08–111.44	51.67	27.30–92.19
50c	256.61	183.83–378.17	170.57	117.95–214.95
50e	668.14	548.22–818.75	369.08	272.50–567.00
50r	384.40	246.63–606.45	225.58	135.02–353.82
100c	3,557.17	2,924.41–4,048.28	2,503.97	1,620.69–3,756.26
100e	6,982.16	5,934.83–8,570.88	2,974.92	1,302.28–5,258.36
100r	4,748.49	3,132.99–7,447.30	2,155.69	935.11–3,247.06

7. Conclusions and Future Work

In this paper, we described the multilevel capacitated minimum spanning tree problem, a generalization of the well-known CMST problem. The MLCMST encompasses practical concerns of local access network design but has not attracted much attention in the past. Like most capacitated network design problems, the MLCMST is an extremely challenging problem to solve. In this paper, we developed both heuristic techniques and lower bounding mechanisms for the MLCMST. To our knowledge, this work represents the first comprehensive study of the problem.

To begin, we presented two mixed-integer programming formulations, ESCF and MCF, that provide strong lower bounds. The LP relaxation of ESCF, a single commodity model, solves rapidly and is most useful for bounding purposes. The second model MCF, a multicommodity model, is stronger and can provide tighter bounds, but even to solve its LP relaxation for large problems (100 nodes and greater) is computationally challenging. The improved bounds provided by MCF are only marginally better, and, thus, on balance, we recommend the use of the ESCF model to calculate lower bounds.

We then developed a savings-based heuristic for the MLCMST problem that finds solutions that are, on average, 9.29% away from the lower bound. The heuristic is extremely fast and runs in under a second, even for 150 node problems. Next, we adapted a neighborhood search technique, originally proposed for the CMST problem, and developed two local search procedures (LS and RSLS). The LS procedure achieves results that are, on average, within 6.47% of the lower bound, and the RSLS algorithm is able to improve marginally on the results of LS. Novel in our development of the very large scale neighborhood search procedures for the MLCMST is the use of a heuristic to compute the costs of arcs on the improvement graph (instead of using exact costs). The local search procedure LS improves the construction heuristic solutions by 2.56%, on average, and has successfully been used for problems with up to 150 nodes. Finally, we presented a hybrid genetic algorithm for the MLCMST problem. The GA, compared to our other heuristics, obtains the best solutions. For smaller problems, the GA is able to provide solutions whose objective is within 0.25% of optimality and, for larger problems, the GA solutions is about 4% to 7% away from the lower bound, depending on the location of the central node. The GA can be used to provide solutions for problems with up to 100 nodes.

We now make a few observations regarding future work. First of all, we note that the construction heuris-

tic is critical to all our approaches. By improving the quality of the solutions found by the heuristic, we could possibly improve the performance of all our algorithms.

Another direction for future work is the nonunit demand case. Observe that our MIP formulations (ESCF and MCF) directly apply to the nonunit demand case. In the nonunit demand case, our construction heuristic can be used to construct a feasible multilevel capacitated spanning tree based on the same savings calculated by equation (23). However, unlike the unit-demand case, finding the maximizing set H corresponds to a knapsack problem. Thus, it may be appropriate to use a heuristic (as a surrogate) to find the maximizing set H in (23). Of course, this approach and other possible alternatives should be explored through extensive computational testing so that their performance can be evaluated.

Observe that our metaheuristic procedures can be applied to the nonunit demand case without significant modifications, *assuming* that a construction heuristic is available for the nonunit demand case. Specifically, the neighborhood structure used by our local search procedure directly applies to the nonunit demand case. Additionally, the genetic algorithm representation and crossover operator also directly apply to the nonunit demand case. Thus, no significant modifications are required.

Finally, one additional challenge is to develop better mathematical-programming formulations that can find the optimal solutions for large problems and generate tighter bounds. Although our heuristics can run on problems with up to 150 nodes, we are unable to find exact solutions for problems with more than 30 nodes. Stronger formulations will allow for a better understanding of the quality of the heuristic solutions on larger problems. In connection with this, we are making available to other researchers the 750 problems considered in our computational work in this paper.

References

Ahuja, R. K., J. B. Orlin, D. Sharma. 2001. Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Math. Programming* **91** 71–97.

Amberg, A., W. Domschke, S. Voß. 1996. Capacitated minimum spanning trees: Algorithms using intelligent search. *Combin. Optim. Theory Practice* **1** 9–40.

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

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

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

Dahl, G., M. Stoer. 1998. A cutting plane algorithm for multicommodity survivable network design problems. *INFORMS J. Comput.* **10** 1–11.

Esau, L. R., K. C. Williams. 1966. On teleprocessing system design. *IBM System J.* **5** 142–147.

Falkenauer, E. 1996. A hybrid grouping genetic algorithm for bin packing. *J. Heuristics* **2** 5–30.

Gavish, B. 1982. Topological design of centralized computer networks—Formulations and algorithms. *Networks* **12** 355–377.

Gavish, B. 1983. Formulations and algorithms for the capacitated minimal directed tree problem. *J. ACM* **30** 118–132.

Gavish, B. 1991. Topological design of telecommunications networks—Local access design methods. *Ann. Oper. Res.* **33** 17–71.

Gouveia, L. 1993. A comparison of directed formulations for the capacitated minimum spanning tree problem. *Telecomm. Systems* **1** 51–76.

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

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

McGregor, P. M., D. Shen. 1977. Network design: An algorithm for access facility location problems. *IEEE Trans. Comm.* **25** 61–73.

Michalewicz, Z. 1996. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York.

Rothfarb, B., M. C. Goldstein. 1971. The one-terminal Telpak problem. *Oper. Res.* **19** 156–169.

Salman, F., S. R. Ravi, J. Hooker. 2001. Solving the local access network design problem. Working paper, Krannert Graduate School of Management, Purdue University, West Lafayette, IN.

Sharaiha, Y. M., M. Gendreau, G. Laporte, I. H. Osman. 1997. A tabu search algorithm for the capacitated minimum spanning tree problem. *Networks* **29** 161–171.

Sharma, R. L. 1983. Design of an economical multidrop network topology with capacity constraints. *IEEE Trans. Comm.* **31** 590–591.

Thompson, P., J. B. Orlin. 1989. Theory of cyclic transfers. Working paper, Operations Research Center, MIT, Cambridge, MA.