# TECHNICAL RESEARCH REPORT

An Evolutionary Approach to the Multi-Level Capacitated Minimum Spanning Tree Problem

*by Ioannis Gamvros, S. Raghavan, Bruce Golden*

**CSHCN TR 2002-10**
**(ISR TR 2002-18)**


*The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.*

**Web site http://www.isr.umd.edu/CSHCN/**

# An Evolutionary Approach to the Multi-Level Capacitated Minimum Spanning Tree Problem

Ioannis Gamvros* S. Raghavan† Bruce Golden‡

March 2002

**Abstract**

Capacitated network design is a crucial problem to telecommunications network planners. In this paper we consider the Multi-Level Capacitated Minimum Spanning Tree Problem (MLCMST), a generalization of the well-known Capacitated Minimum Spanning Tree Problem. We present a genetic algorithm, based on the notion of grouping, that is quite effective in solving large-scale problems to within 10% of optimality.

**Keywords:** Network Design, Capacitated Minimum Spanning Tree, Multi-Level, Genetic Algorithms.

## 1 Introduction

One problem that arises often in practice in the design of local access communications networks is the Terminal Layout Problem. This problem is typically referred to in the Operations Research literature as the Capacitated Minimum Spanning Tree Problem (CMST) and has been extensively studied by many researchers over the years (see [9] for an extensive review). In the CMST problem, we are given a set of customer locations or terminals, each with its own traffic requirements that we wish to transport to a given *central* location (node). Furthermore, a single type of facility with a fixed capacity $K$ is available for installation, and we wish to design a feasible minimum cost tree network to carry the traffic. In practice, it seems unreasonable to assume that only a single type of facility is available to the network planner. Consequently, in this paper, we deal with a generalization of the CMST problem that we believe reflects the practical concerns that arise in the design of local access networks more closely. In our problem, we allow for

*The Robert H. Smith School of Business, University of Maryland, College Park, MD 20742

†The Robert H. Smith School of Business and Institute for Systems Research, University of Maryland, College Park, MD 20742

‡The Robert H. Smith School of Business, University of Maryland, College Park, MD 20742

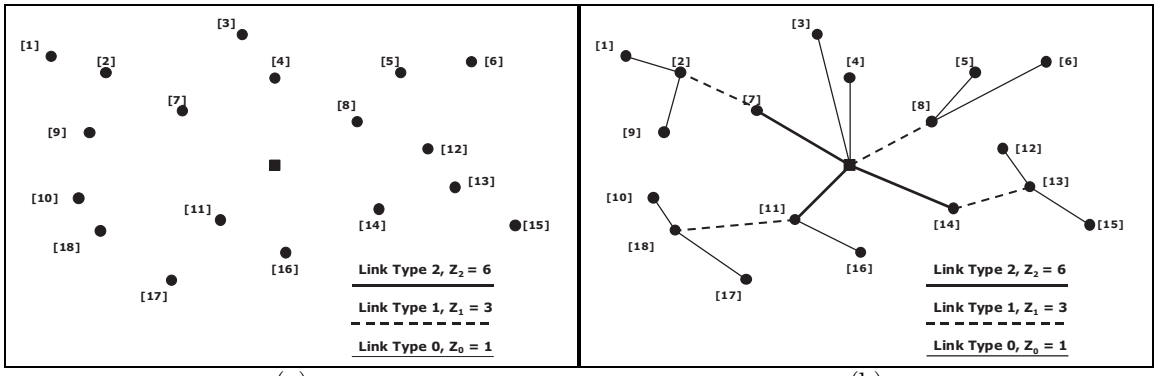



Figure 1: Multi-Level Capacitated Minimum Spanning Tree. (a) Points in the network. (b) Feasible Multi-Level Capacitated Spanning Tree.

the installation of multiple types of facilities with differing capacities. This problem we call the Multi-Level Capacitated Minimum Spanning Tree Problem (MLCMST).

Formally, the MLCMST is defined as follows: Given a graph $G = (N, E)$, with node set $N = \{0, 1, 2, \ldots, n\}$, where node 0 represents the central facility and the rest are customer locations, and edge set $E$, $W_i$ the traffic requirement (or weight) of node $i$ to be transported to the *central node* 0, facility types $0, 1, \ldots, L$ with capacities $Z_0 < Z_1 < \ldots < Z_L$ and cost function $C_{ij}^l$ denoting the cost of a facility of type $l$ installed between nodes $i$ and $j$; we wish to find a minimum cost tree network to carry the traffic.

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 3 types of links with capacities $Z_0 = 1, Z_1 = 3$, and $Z_2 = 6$, and the traffic generated from each node is 1 unit. Figure 1(b) shows a feasible multi-level capacitated spanning tree. Notice, the topology of the network is a tree, and traffic on any link is less than 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 customer nodes. Additionally, we restrict our attention to (realistic) cost functions that exhibit economies of scale that usually exist in communication networks. In other words, the cost function satisfies the relationship

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

for every edge $\{i, j\} \in E$. We also impose the restriction that only a single facility type is permitted to be installed on a link. This condition is actually not restrictive. If multiple facilities can be installed on a link, Salman et al. [18] point out, in the context of a related problem, that by applying a dynamic programming algorithm one can convert the problem 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 link types each representing one of the optimal combinations.

## 1.1 Related Literature

The MLCMST does not appear to have been given much attention by researchers previously. The most closely related problem is the so called Local Access Network Design (LAND) problem [3, 18] or the Telpak problem [17]. In the LAND problem, as in the MLCMST, we are given traffic demand from nodes in a network that need to be transported to a central node, varying 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. In a survey paper, Gavish [9] describes the Telpak problem, but restricts it to a tree. He presents a formulation for this problem, and also points to the lack of attention given to this important problem by researchers.

Berger et al. [3] propose a tabu search procedure for the LAND problem to obtain good heuristic solutions for problems with up to 200 nodes, and 9 cable 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. [18] study the version of the LAND problem that allows demand splitting, and propose a branch and bound procedure using a technique called search by objective relaxation. Using this technique, they solve to optimality problems with 10 nodes, and up to 9 cable types.

On the other hand, the CMST problem has attracted a lot of attention from the research community over the years. The most well-known heuristic for the problem is the Esau-Williams heuristic [5] that starts with a feasible star solution and uses a savings calculation to create and merge subtrees. This heuristic is still used as a benchmark against which other techniques are evaluated. Sharma [20] and McGregor at al. [15] propose algorithms based on a clustering (grouping) approach. They first assign the nodes into groups and then find a Minimum Spanning Tree (MST) for each group. Karnaugh [13], Kershenbaum et al. [14], and Gouveia and Lopes [12] describe so-called Second Order Greedy Algorithms (SOGA). These are local search heuristics that start with an initial solution generated typically by the Esau-Williams heuristic and try to improve it by either restricting the solution to include or exclude some links. Gavish and Altinkemer [10] describe a parallel savings heuristic that adds to the solution multiple links per iteration.

More recently, metaheuristic approaches such as simulated annealing and tabu search have been successfully used to obtain high-quality solutions for the CMST problem space. Sharaiha et al. [19] present a tabu search approach for the problem. Bourjolly et al. [4] use simulated annealing in order to search the solution

space. With the help of a modified version of Esau-Williams and a neighborhood description, they were able to generate effective solutions using the simulated annealing approach. They compared their results with Sharaiha et al.'s tabu search algorithm for 40, 50, 80, and 100 node size problems, both with unit demand and non-unit demand, and for the cases where the central node is at the center and at the edge of the graph. The simulated annealing algorithm outperformed the tabu search algorithm for smaller size non-unit demand problems, and on unit demand problems for which the central node was located at the center. Amberg et al. [2] compare a simulated annealing approach and three tabu search approaches with the Esau-Williams algorithm for different parameters for the meta-heuristics. They tested their algorithms for 40 and 80 node size unit-demand problems with the central node either at the center or at the edge and achieved 1% to 7% improvements over Esau-Williams when the parameters of the meta-heuristics were optimized. Recently, Ahuja et al. [1] present an improved neighborhood structure for the CMST problem, which they use in conjunction with a tabu search algorithm to generate solutions. They present improvements over the best known solutions for different sets of problems. For non-unit demand problems, their algorithm improved the best-known solution in terms of cost by 3% on average. Patterson and Pirkul [16] have embedded different heuristic processes into the topological design of a neural network and have used this network to generate solutions. They reported that the neural network that was based on the Esau-Williams algorithm was able to find solutions that were, on average, 0.5% away from the optimum with running times close to one hour for 150 node problems.

## 1.2 Overview of our Approach

In this paper we present a genetic algorithm for the MLCMST. Conceptually, our approach splits this problem into two separate sub-problems: a grouping problem and a network design problem. The grouping problem aims at finding the best assignment of the nodes into groups that correspond to subtrees of the central node. Consequently, it makes sure that the sum of the weights in a group does not exceed the capacity of the highest capacity link. In other words, for every group we require that $\sum W_i \leq Z_L$. The network design subproblem requires the construction of a minimum cost multi-level tree network that will connect all the nodes in each of the groups with the central node. Notice that this problem is identical to the MLCMST, but defined on each of the groupings. We apply our genetic algorithm (GA) to the grouping problem. The GA then calls upon the construction heuristic to construct subtrees on the groupings and provide the cost (fitness) of the solution to the problem.

Th notion of determining groupings (or nodes in a subtree) has been used previously by some researchers for the CMST problem [1, 2, 15, 19, 20]. Observe 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 it is identical to the original problem, albeit on a smaller graph.

### 1.3 Organization of this Paper

The rest of the paper is organized as follows. Section 2 describes our genetic algorithm procedure. Next, Section 3 describes the construction heuristic that the genetic algorithm uses to construct a multi-level tree with unit degree at the root. Then, Section 4 describes a mixed integer programming formulation for the problem. We used the LP-relaxation of this model to obtain lower bounds for our problem. We present our computational experiments in Section 5. Section 6 outlines some directions where, we believe, some improvements may be obtained for the solution procedure.

## 2 Genetic Algorithm

A genetic algorithm is a powerful evolutionary local search algorithm that typically consists of a few basic steps as shown in Figure 2. At first an initial population of feasible solutions (chromosomes) $P(t)$ has to be created. After the creation of the initial population, we evaluate the fitness of all the individuals in that population. This evaluation is required to check the termination condition and later on at the selection step. Usually, genetic algorithms terminate when there are no improvements in a number of successive populations (generations) or after a predefined number of generations. The selection step chooses different chromosomes from the old population $P(t-1)$ for reproduction. After the chromosomes are selected, they reproduce (crossover) to create children. The next population $P(t)$ is determined by taking some fraction of the best children, and some fraction of the best (elite) parents (this process is called elitism). At this stage, random mutations can arbitrarily change the genetic material found in the population. The purpose of mutations is to divert the search process to places that it wouldn't have examined otherwise. In the following sections, we elaborate on each of the steps of our genetic algorithm. Prior to that, we discuss the data structure or *representation* that we use to represent chromosomes in our GA.

We use a representation proposed by Falkenauer [6] in the context of bin packing. This representation, as shown in Figure 3, breaks up the chromosome representing the solution into 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) that the node belongs to. The group part contains a list of groups (subtrees) that make up the tree solution. The representation shown in Figure 3 indicates that nodes $1, 3, 4$, and 7 are in a group (group $\mathbf{A}$), nodes $2, 5$, and 6 are in a group (group $\mathbf{B}$), nodes $8, 9$, and 10 form a group (group $\mathbf{C}$), and node 11 forms a group (group $\mathbf{D}$). The group part lists the four groups $\mathbf{A}, \mathbf{B}, \mathbf{C}$,

```
Begin
    t ⟵ 0
    initialize P(t)
    evaluate P(t)
    while (not termination-condition) do
        t ⟵ t + 1
        select parents from P(t − 1)
        reproduce parents
        Determine P(t) from children and elite parents
        P(t): mutations
        evaluate P(t)
    end
end
```

Figure 2: Steps of a Genetic Algorithm

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 they be distinct from each other). Also note that since the number of subtrees of the root node will vary by solution, the length of the group part of the chromosome will vary.

The advantage of this representation is that it allows for a focus on the groups in a solution (via the group part), which is how we have approached the problem. Further, this representation satisfies the *Building Block Hypothesis*[1] which typically results in high quality genetic algorithms.

## 2.1 Initial Population

Since any genetic material - 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.

To create the initial population, we use the Esau-Williams heuristic to solve a CMST problem on our graph by setting the (subtree) capacity constraint to $Z_L$, the maximum capacity in our problem. Since the Esau-Williams heuristic gives a unique solution for a given graph, and we need our initial population to be diverse, we multiply the cost of each edge, $C_{ij}^0$ by a uniformly distributed random variable in the range $[(1-\epsilon),(1+\epsilon)]$. Varying the value of $\epsilon$ trades off between increased diversity of the initial population (for

[1]The Building Block Hypothesis [11] states that "A genetic algorithm seeks near optimal performance through the juxtaposition of short, low-order, high-performance schemata called the building blocks."

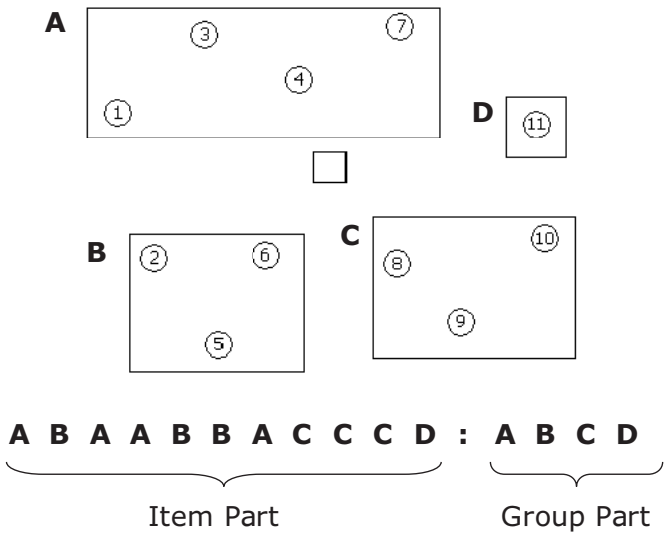



Figure 3: Representation of a solution in the Genetic Algorithm.

large $\epsilon$) and early termination to a solution further away from the global optimum (for small $\epsilon$).

To create increased variety in the initial population, we also experimented with changing the capacity constraint $K$ of the Esau-Williams algorithm. Specifically, we configured the capacity constraint as a uniformly distributed integer random variable in the range $[Z_L - \delta, Z_L]$. Each time we run the Esau-Williams heuristic to determine an initial solution, we also varied the capacity constraint in this manner.

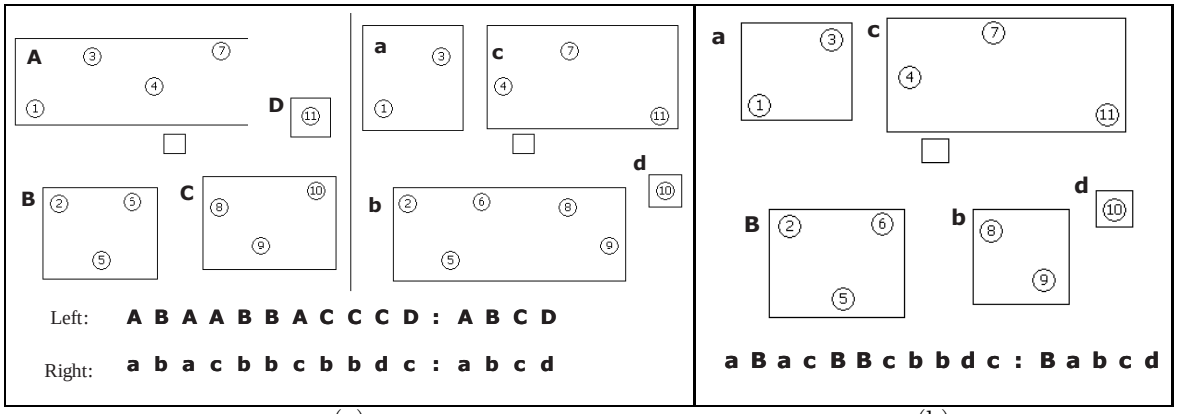
## 2.2 Crossover

The crossover operator is responsible for combining two chromosomes so that a new offspring chromosome can be generated. It is important to note here that the crossover operator should not only guarantee that the offspring chromosome will be a valid one (i.e., it will satisfy the constraints imposed by the problem) but it should also make sure that meaningful genetic material (i.e., building blocks) is passed on from the parents to the children.

Our crossover operator is similar to that of Falkenauer [6], 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.




(a) (b)

Figure 4: Crossover Operator in Genetic Algorithm. (a) Parents, (b) Offspring

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. Finally, if a group from the second parent has lost items and it now has less than $k$ items, we reassign these items to other groups with probability $p_{\text{cr}}$. If any group is empty as a result, remove it from the group part.

We illustrate the procedure with the example shown in Figure 4. Consider the two chromosomes shown in Figure 4(a). 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.

$$\begin{array}{cccccccccccccccc} A & B & A & A & B & B & A & C & C & C & D & : & A & |B| & C & D \\ a & b & a & c & b & b & c & b & b & d & c & : & |a & b| & c & d \end{array}$$

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 that the node belongs to 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 4(b).

The final 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, for example,

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. Observe that the membership in the groups is either identical to the group in the parent, or is a subset of the group in the second parent. Our reassignment procedure focuses on such groups (i.e., where the group is inherited from the second parent but has fewer members than in the parent). If the number of items in such a group is less than a parameter $k$, then, with probability $p_{\text{cr}}$, each item is assigned to the group that the closest node not in its group belongs to.

For example, suppose $k = 3$, and $p_{\text{cr}} = 0.5$ and consider the result in Figure 4(b). 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 that node 10 (assuming 10 is closest to it and the group has sufficient capacity to accommodate node 9), and with probability 0.5 we reassign 8 to the group that node 6 is in (assuming that 6 is closest to it).

## 2.3 Mutation Operator

Our mutation operator aims at shuffling some of the items among groups. Specifically, our mutation operator is applied with probability $p_\mu$ to each node (item) of every chromosome and it searches for alternative groups that this node can be assigned to. The search is done by finding a set of nodes $M$ for which $C^0_{im} < C^0_{m0}$, where $m$ is the node that the operator acts on, $i$ are the nodes in $M$ and 0 is the central node. We then consider the nodes $i$, one by one, starting with the one with the smallest $C^0_{im}$ and check to see if the group that this node $i$ belongs to has available capacity. If there is available capacity and the group that $i$ belongs to is different from the current group that $m$ belongs to we reassign $m$ to the new group. Otherwise we move on to the next node in $M$. If no node in $M$ satisfies these constraints, node $m$ is not reassigned.

Consider the chromosome shown in Figure 4(b). Suppose we apply the mutation operator to node 4. For node 4, $M = \{3, 7\}$. Since 3 is closest node to 4 in $M$, it is considered first; and since it belongs to a different group, node 4 is assigned to that group creating the chromosome **aBaaBBcbbdc** : **Babcd**.

## 2.4 Selection

The selection procedure is an important step in the genetic algorithm. If the selective pressure is too strong (i.e., only the fittest individuals are selected for reproduction), then it is possible that some very fit chromosomes are going to dominate early in the search process and lead the algorithm to a local optimum,

thus terminating the search. On the other hand, if the selective pressure is too weak (i.e., even individuals with low fitness have a high probability of being selected for reproduction), then the algorithm is going to traverse the solution space aimlessly. We employ the classical roulette wheel mechanism to select the chromosomes that will participate in reproduction and create subsequent generations.

In the GA structure presented in Figure 2, we did not specify how many children to generate. Some options are to generate as many children as the population size (and have no elitism, i.e., carry no parents over to the next generation), or to generate more offspring than the population size and select only the best of those to make up the next generation.

Additionally, we employ elitism. We ensure that a percentage $E$ of each generation is made up of selected individuals (i.e., the alive chromosomes) from the previous generation. We select these *elite* individuals by evaluating and selecting the best parents.

# 3 Construction Heuristic

The GA uses the construction heuristic to construct subtrees on the groupings in order to evaluate them. Given that the GA will repeatedly call the construction heuristic, and use the solutions generated by it to evaluate the groupings, it is crucial that the construction heuristic is fast and provides near-optimal solutions.

The problem at hand is almost identical to the overall problem, except that it is defined on a smaller network (the subgraph defined by the grouping). However, for the construction heuristic, we impose the additional restriction that the degree of the central node is 1. The notion is that each grouping represents a subtree of the central node and, thus, when the subtree is viewed in isolation it has degree 1 at the central node. Notice that as a consequence, we ignore solutions that may be slightly better where the central node has a degree greater than 1. Figure 5 illustrates this point. Our rationale for keeping the degree of the central node 1 is that if a better solution exists where the degree of the central node is greater than 1, then that corresponds to new groupings which we leave for the GA to explore.

The first step of our heuristic consists of determining the node that is to be connected directly to the central node (i.e., the node in the subtree that will be directly connected to the root). For each node, we calculate the sum of distances to each node in the grouping including the central node, and select the node with the minimum value, i.e., the one closest to the *centroid*, to be in the set of candidate nodes $N_C$ that would be considered for connection to the central node. To this set, we add all nodes in the grouping that are closer to the central node than the "centroid node".

Initially, all nodes are connected to the node that is to be directly connected to the central node with the lowest capacity link (link type 0). Thus, if we ignore the connection to the central node, then this node takes

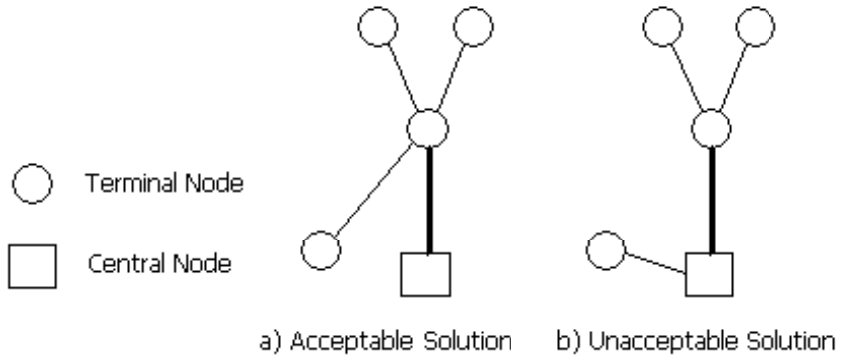



Figure 5: Construction heuristic must generate a solution where the central node has degree 1.

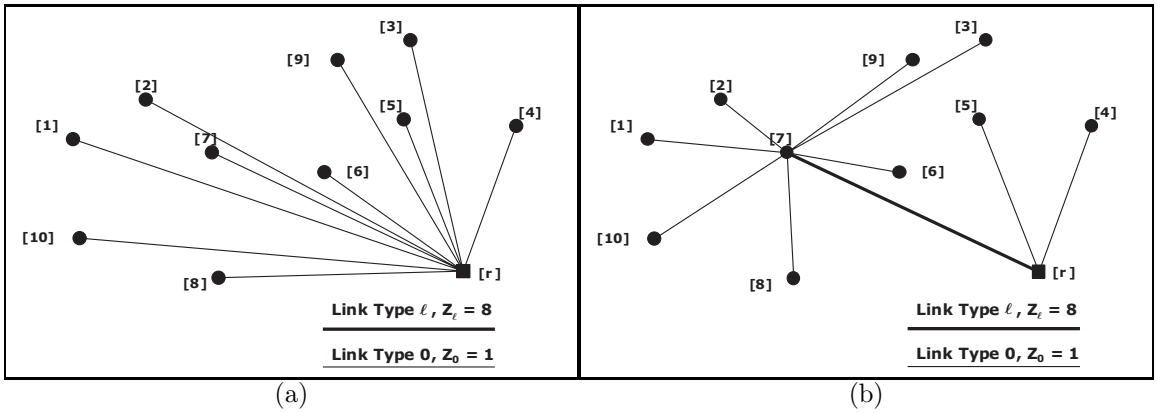



Figure 6: Core problem in construction heuristic for constructing tree on grouping. (a) All nodes are connected to a "root" node via links of capacity $Z_0$. (b) Determine savings by upgrading links connected to the "root" to capacity $Z_l$ and redirecting the connections to the "root" to the node whose connection to the root has been upgraded (i.e., in this example, node 7).

the role of a central node for the subtree, and all nodes are connected directly to it using the lowest capacity links in a star configuration. Our construction heuristic considers the savings obtained by upgrading the capacity of a link to the node that is directly connected to the central node, and connecting other nodes through this upgraded link. Specifically, our construction heuristic solves the following *core problem*.

We are given a complete graph with node set $N^*$, a root node $r$, two link types, 0 and $l$, with capacities $Z_0 < Z_l$, costs $C_{ij}^0$ and $C_{ij}^l$ respectively, and an initial tree network that is constructed as a star with $r$ at the center using link type 0. We would like to determine where to upgrade the connections in the star network to facilities of type $l$, and reconnect some of the links with capacity $Z_0$ to this connection instead of their direct connection to the root node. Obviously, upgrading the capacity of the connection will increase the cost, but if the cost of the lower capacity connections is reduced, that might reduce the overall cost. Figure 6 illustrates the core problem.

Our solution approach to the core problem consists of the following steps. We start with the star network

and construct a savings value $d_{ij} = C^0_{jr} - C^0_{ij}$, where $r$ denotes the root node for the problem. This represents the savings in connecting node $j$ to node $i$. We then compute for each node $D_i$ the overall savings in upgrading link $\{i, r\}$ to a facility of type $l$. We compute

$$D_i = C^0_{ir} - C^l_{ir} + \max_{\{J:|J|<Z_l\}} \sum_{\{j:d_{ij}>0, j\in J\}} d_{ij}$$

for each node $i$ directly connected to the root $r$ by a facility of type 0. The first two terms represent the change in cost by upgrading link $\{i, r\}$ from a facility of type 0 to a facility of type $l$. The third term represents the greatest savings obtained by changing the connections $\{j, r\}$ to the connections $\{j, i\}$ ensuring that the capacity of facility $l$ is not violated. We then select the largest $D_i$. This represents the largest savings obtained by upgrading link $\{i, r\}$ and implementing it. We repeatedly apply this procedure, recomputing the savings $d_{ij}$ and $D_i$ to the core problem until there are no more savings (i.e., $D_i <= 0$). Note that once we construct a subtree of the root by this savings procedure, we will not consider the nodes it contains in computing further savings.

Our overall procedure starts with the highest capacity level that is greater than or equal to the size of the group. For example if the group size is 4, and the smallest capacity that is greater than or equal to 4 is one with a capacity of 5, the procedure starts with the facility of type, say $k$, with the capacity of 5. We identify the set of candidate nodes $N_C$ that will connect the group to the central node. We select a node in $N_C$ and create a star network connecting the nodes in the grouping to it with facilities of type 0. We then consider the core problem with facilities of type $l = k - 1$ (one lower than what we started with). We apply the savings procedure until there are no more savings. We then consider the network defined by the lowest capacity facilities in the current solution. For each of the stars in this network, we consider the core problem with facilities of type $l = k - 2$, and apply the savings heuristic. We repeat this procedure, constructing the network defined by the lowest capacity facilities in the current solution, and solving the core problem, lowering the value of $l$ by 1 each time, until we are left with $l = 0$. The final level (i.e., when $l = 0$) deals with the lowest capacity levels. If the lowest level facility has capacity 1, we stop. Otherwise, our last step is to perform the familiar Esau-Williams heuristic on each of the star networks defined by the lowest capacity links. We perform these steps for each of the nodes in $N_C$, and so have $|N_C|$ different solutions. Consequently, we select as the solution of the construction heuristic the one that is of lowest cost among these $|N_C|$ solutions.

Figure 7 gives an example of our construction heuristic procedure. Figure 7(a) shows the set of nodes and the set $N_C$ consisting of nodes $6, 7, 13$, and 18. As shown in Figure 7(b), we select node 7, as it is in $N_C$, and create a star connecting all nodes to 7 (except the central node) with link type 0. We solve the core

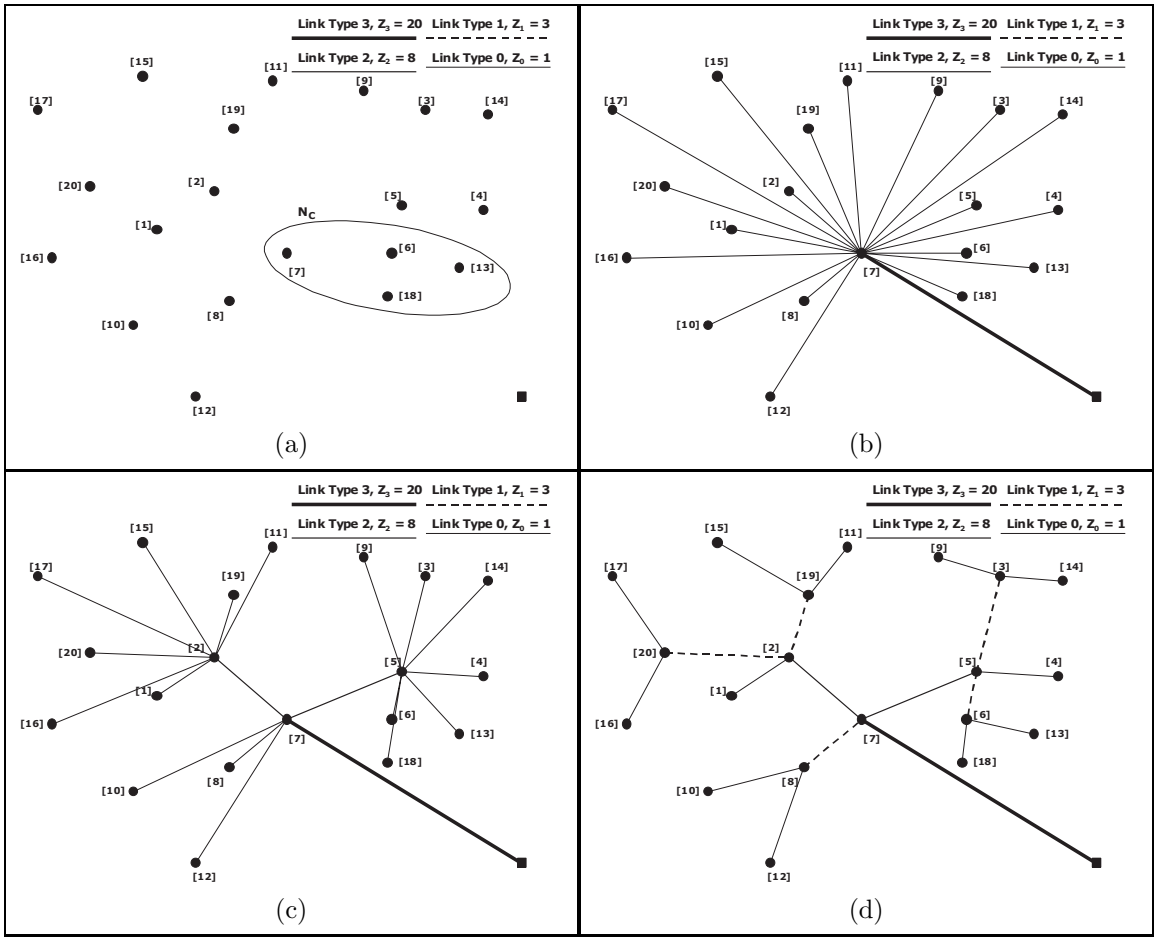



Figure 7: Construction heuristic example.

problem on this star network with $l = 2$, to obtain the network shown in Figure 7(c). Observe link $\{2, 7\}$ has been upgraded to a facility of type 2, and nodes $1, 11, 15, 16, 17, 19$, and 20, are directly connected to node 2. Also, link $\{5, 7\}$ has been upgraded, and nodes $3, 4, 6, 9, 13, 14$, and 18, are directly connected to node 5. We now lower the value of $l$ to 1, and consider the star networks defined by the lowest capacity facilities. There are three star networks: one at node 2 with direct links to nodes $1, 11, 15, 16, 17, 19$, and 20; one at node 7 with direct links to nodes $8, 10$, and 12; and one at node 5 with direct links to nodes $3, 4, 6, 9, 13, 14$, and 18. We solve the core problem on these three star networks to obtain the solution shown in Figure 7(d). We lower $l$ to 0. Since $l = 0$ and $Z_0 = 1$ we stop.

# 4 Mixed-Integer Programming Model

We now describe a strong Mixed-Integer Programming Model for the MLCMST. We obtained lower bounds to evaluate our Genetic Algorithm by solving the Linear Programming relaxation of the model. A related

paper [8] provides additional details on the strength of the formulation.

We model the problem on a directed graph. To do this we replace each edge $\{i, j\}$ in the graph by two directed arcs $(i, j)$ and $(j, i)$. 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 $\{i, j\}$. We will introduce two types of arc variables in our model. Let $x_{ij}$ denote whether a facility is installed on arc $(i, j)$, being 1 if a facility is installed on the arc, and 0 otherwise. Let $y_{ij}^k$ be 1 if a facility of type $k$ is installed on arc $(i, j)$, and 0 otherwise. Thus, it identifies the type of facility installed on arc $(i, j)$.

We create a commodity for each terminal node, with a supply of 1 at the terminal node, and a demand of 1 at the central node. In our notation, the origin of commodity $k$ is node $k$ and the destination of commodity $k$ is node 0. We let $f_{ij}^k$ denote the flow of commodity $k$ on arc $(i, j)$. Our multicommodity directed flow formulation for the problem is:

$$\text{Minimize} \quad \sum_{(i,j)\in A} \sum_{l=0}^{L} C_{ij}^l y_{ij}^l \tag{1}$$

$$\text{subject to} \quad \sum_{j\in N} f_{ji}^k - \sum_{l\in N} f_{il}^k = \begin{cases} -1 & \text{if } i = k; \\ 1 & \text{if } i = 0; \quad \text{for all } i \in N \text{ and } k \in K \\ 0 & \text{otherwise;} \end{cases} \tag{2}$$

$$\sum_{k\in K} f_{ij}^k \le \sum_{l=0}^{L} Z^l y_{ij}^l \quad \forall (i, j) \in A, \tag{3}$$

$$\sum_{l=0}^{L} y_{ij}^l = x_{ij} \quad \forall (i, j) \in A, \tag{4}$$

$$f_{ij}^l \le x_{ij} \quad \forall (i, j) \in A, l = 0, 1, \ldots, L, \tag{5}$$

$$\sum_{j\in N} x_{0j} = 0 \tag{6}$$

$$\sum_{j\in N} x_{ij} = 1 \quad \forall i \in N, i \neq 0, \tag{7}$$

$$x_{ij} + x_{ji} \le 1 \quad \forall (i, j) \in A, \tag{8}$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A, \tag{9}$$

$$y_{ij}^l \in \{0, 1\} \quad \forall (i, j) \in A, l = 0, 1, \ldots, L, \tag{10}$$

$$f_{ij}^k \ge 0 \quad \forall (i, j) \in A, k \in K. \tag{11}$$

Constraints 2, 5, 6, 7, and 8 ensure that the topology of the underlying network is a tree that is directed towards the central node. Constraint 4 ensures that only one type of facility is installed on an arc, and only if the arc is selected to be in the multi-level capacitated tree (i.e., only if $x_{ij} = 1$). Constraint 3 ensures that

**GA Parameters**

| Category | Description | Name | Value |
|---|---|---|---|
| ***General*** | | | |
| | *Population Size* | - | 100 |
| | *Number of Offspring* | - | 300 |
| ***Selection*** | | | |
| | *Selection of Parents* | - | Roulette |
| | *Elitism* | **E** | 50% |
| ***Mutation*** | | | |
| | *Classic Mutation Rate* | $\mathbf{P}_{\mu}$ | 1% |
| ***Initial Population*** | | | |
| | *Cost Matrix Perturbation* | $\varepsilon$ | 0.1 |
| | *Group Size Range* | $\delta$ | 0 |
| ***Stopping Criteria*** | | | |
| | *Maximum Number of Generations* | - | 50 |
| | *Number of Generations w/out improvement* | - | 10 |
| ***Crossover*** | | | |
| | *Probability of Reassignment* | $\mathbf{p}_{\alpha}$ | 1 |
| | *Number of Nodes for Reassignment* | **k** | <6 |

Table 1: Genetic Algorithm Parameters.

the flow sent on a link is less than the capacity installed on the link.

# 5 Computational Experiments

We now report on some computational experiments with the GA for the MLCMST problem. We coded our GA and construction heuristic in Visual C++. We conducted all runs on a Dual Processor Pentium III PC running Windows 2000, 1GHz clock speed, and with 512MB RAM.

We generated three sets of 50 terminal node problems—one with the root node in the center, one with the root node in the corner, and one where the root node is located randomly—each containing 50 problem instances (together with the root node there are a total of 51 nodes in these problem instances). The points in these problems are generated randomly on a $20 \times 20$ grid. We also generated one 100 terminal node problem set—containing 50 problem instances—with the root node in the center. For all test problems, we used three link types with capacities 1, 3, and 10 respectively. The cost of link type 0, $C_{ij}^0$, is equal to the Euclidean distance between the two end points of the link. The cost of link type 1 is equal to twice the cost of link type 0, and the cost of link type 2 is equal to six times the cost of link type 0.

To compare the results of the GA to a lower bound, we solved the LP relaxation of the mixed integer programming formulation presented in Section 4. We coded this formulation in the ILOG OPL Studio programming language and ran it on the same computer as the GA.

After significant computational testing on the different GA parameters [7] we selected the values shown in Table 1.

Tables 2, 3, and 4 show the results of our computations for 50 node problems. The gap shown in the Table represents the difference between the GA solution and the lower bound divided by the lower bound, expressed in percentage terms. This gap provides an upper bound, or a guarantee, on how far the GA solution is from optimality. When the central node is in the center, the GA provides solutions on average in 34 seconds. The average gap is about 9.95%. When the central node is in the corner, the running time of the GA goes up to an average of 79 seconds, but the quality of the solutions improves, with an average gap of 5.90%. When the central node is selected randomly, the results are in between these two cases.

Table 5 shows the results for a set of 100 node problems with the central node in the center. The average running time of the GA has doubled to about 67 seconds. Surprisingly, as the problem size has increased, the quality of the solutions has improved with an average gap of 7.68%. It is not clear why this is the case, but it appears that as the problem size increases the GA does well in finding good groupings that are close to the maximum link capacity.

# 6 Conclusions

In this paper, we presented a genetic algorithm for the MLCMST problem. The genetic algorithms generates solutions very rapidly, and we have been able to solve up to 100 node problems to within 10% of optimality.[2] As noted earlier the actual performance of the GA is significantly better, as these lower bounds are obtained by solving the LP relaxation of a MIP formulation for the problem.

There are several directions in which we are pursuing our ongoing research on the MLCMST. Although the GA has performed reasonably well, we believe there may be opportunity for improvements. In computational testing we have found that the solutions generated by the construction heuristic are generally within 0.5% of optimality, on average, when the degree of the central node is restricted to be 1. On the other hand, when the degree of the central node need not be 1, the construction heuristic solution is within 2.5% of optimality, on average. Thus one direction of our research is to try and improve the construction heuristic, allowing for the degree of the central node to be greater than 1. This would result in improvements in the overall GA solution. Another direction is the development of alternate heuristic solution procedures to the genetic algorithm approach for the MLCMST problem. A third direction is the improvement of the genetic algorithm operators and parameters.

We also believe we should perform our computational tests on a wider variety of problems. These problems should not only vary in size (i.e., number of nodes) and number of link types but should also cover a wide variety of cost function combinations. One additional challenge concerning the construction heuristic

[2]The GA can solve larger problems quite easily. The bottleneck is solving the LP relaxation of the formulation that provides us with lower bounds.

| Problem # | GA solutions | GA Running Time (sec) | Lower Bounds | Gap % |
|---|---|---|---|---|
| 50_node_rc 1 | 586.442 | 39.0 | 534.097 | 9.80% |
| 50_node_rc 2 | 558.807 | 24.8 | 504.369 | 10.79% |
| 50_node_rc 3 | 582.100 | 28.4 | 526.688 | 10.52% |
| 50_node_rc 4 | 573.491 | 25.6 | 528.805 | 8.45% |
| 50_node_rc 5 | 555.518 | 24.4 | 511.617 | 8.58% |
| 50_node_rc 6 | 625.846 | 34.2 | 570.824 | 9.64% |
| 50_node_rc 7 | 590.903 | 31.8 | 527.432 | 12.03% |
| 50_node_rc 8 | 596.850 | 29.8 | 553.266 | 7.88% |
| 50_node_rc 9 | 649.982 | 32.0 | 590.250 | 10.12% |
| 50_node_rc 10 | 656.978 | 30.4 | 601.044 | 9.31% |
| 50_node_rc 11 | 571.287 | 31.6 | 529.999 | 7.79% |
| 50_node_rc 12 | 619.719 | 27.2 | 565.144 | 9.66% |
| 50_node_rc 13 | 650.407 | 28.8 | 593.949 | 9.51% |
| 50_node_rc 14 | 564.759 | 19.0 | 503.929 | 12.07% |
| 50_node_rc 15 | 588.561 | 19.2 | 536.241 | 9.76% |
| 50_node_rc 16 | 581.574 | 38.0 | 533.807 | 8.95% |
| 50_node_rc 17 | 607.259 | 28.8 | 543.791 | 11.67% |
| 50_node_rc 18 | 598.116 | 38.2 | 539.310 | 10.90% |
| 50_node_rc 19 | 630.942 | 36.0 | 568.868 | 10.91% |
| 50_node_rc 20 | 617.805 | 26.2 | 573.500 | 7.73% |
| 50_node_rc 21 | 575.973 | 56.4 | 516.353 | 11.55% |
| 50_node_rc 22 | 614.343 | 48.6 | 557.419 | 10.21% |
| 50_node_rc 23 | 601.310 | 33.0 | 535.659 | 12.26% |
| 50_node_rc 24 | 596.128 | 32.0 | 529.082 | 12.67% |
| 50_node_rc 25 | 657.205 | 30.6 | 586.309 | 12.09% |
| 50_node_rc 26 | 578.097 | 33.0 | 541.006 | 6.86% |
| 50_node_rc 27 | 585.631 | 28.4 | 534.086 | 9.65% |
| 50_node_rc 28 | 576.380 | 48.6 | 532.920 | 8.16% |
| 50_node_rc 29 | 591.938 | 27.8 | 551.045 | 7.42% |
| 50_node_rc 30 | 595.380 | 45.0 | 534.230 | 11.45% |
| 50_node_rc 31 | 626.098 | 22.4 | 572.630 | 9.34% |
| 50_node_rc 32 | 606.540 | 33.8 | 546.559 | 10.97% |
| 50_node_rc 33 | 602.449 | 38.2 | 562.516 | 7.10% |
| 50_node_rc 34 | 591.008 | 44.4 | 532.474 | 10.99% |
| 50_node_rc 35 | 616.839 | 40.0 | 554.201 | 11.30% |
| 50_node_rc 36 | 588.037 | 39.2 | 533.374 | 10.25% |
| 50_node_rc 37 | 598.908 | 53.4 | 537.203 | 11.49% |
| 50_node_rc 38 | 543.987 | 25.4 | 499.420 | 8.92% |
| 50_node_rc 39 | 616.809 | 26.0 | 558.822 | 10.38% |
| 50_node_rc 40 | 570.685 | 46.8 | 509.843 | 11.93% |
| 50_node_rc 41 | 569.135 | 34.2 | 522.888 | 8.84% |
| 50_node_rc 42 | 597.173 | 27.2 | 548.139 | 8.95% |
| 50_node_rc 43 | 601.475 | 33.4 | 543.439 | 10.68% |
| 50_node_rc 44 | 615.054 | 26.4 | 552.021 | 11.42% |
| 50_node_rc 45 | 605.699 | 30.0 | 560.601 | 8.04% |
| 50_node_rc 46 | 596.436 | 37.4 | 545.328 | 9.37% |
| 50_node_rc 47 | 592.667 | 33.8 | 547.014 | 8.35% |
| 50_node_rc 48 | 640.039 | 27.4 | 573.516 | 11.60% |
| 50_node_rc 49 | 592.014 | 35.8 | 544.614 | 8.70% |
| 50_node_rc 50 | 592.128 | 33.4 | 534.740 | 10.73% |
| Average: | 598.858 | 33.3 | 544.688 | 9.95% |

Table 2: Results for 50 node problems, central node in center.

| Problem # | GA solutions | GA Running Time (sec) | Lower Bounds | Gap % |
|---|---|---|---|---|
| 50_node_re 1 | 1119.562 | 72.2 | 1065.368 | 5.09% |
| 50_node_re 2 | 1173.596 | 89.8 | 1101.844 | 6.51% |
| 50_node_re 3 | 1034.424 | 87.4 | 964.639 | 7.23% |
| 50_node_re 4 | 1115.090 | 76.6 | 1039.421 | 7.28% |
| 50_node_re 5 | 1134.852 | 77.2 | 1075.757 | 5.49% |
| 50_node_re 6 | 1115.464 | 87.2 | 1046.318 | 6.61% |
| 50_node_re 7 | 1012.236 | 66.4 | 956.250 | 5.85% |
| 50_node_re 8 | 1050.942 | 85.4 | 993.783 | 5.75% |
| 50_node_re 9 | 1087.234 | 70.0 | 1033.087 | 5.24% |
| 50_node_re 10 | 1132.760 | 68.6 | 1061.181 | 6.75% |
| 50_node_re 11 | 1130.490 | 75.0 | 1060.322 | 6.62% |
| 50_node_re 12 | 1096.984 | 73.6 | 1036.208 | 5.87% |
| 50_node_re 13 | 1024.120 | 57.0 | 962.587 | 6.39% |
| 50_node_re 14 | 1164.298 | 90.8 | 1117.859 | 4.15% |
| 50_node_re 15 | 1112.578 | 78.0 | 1041.785 | 6.80% |
| 50_node_re 16 | 1093.076 | 77.6 | 1037.546 | 5.35% |
| 50_node_re 17 | 1057.454 | 71.0 | 1002.112 | 5.52% |
| 50_node_re 18 | 1070.682 | 78.6 | 1012.761 | 5.72% |
| 50_node_re 19 | 1075.930 | 70.0 | 1017.786 | 5.71% |
| 50_node_re 20 | 1056.270 | 80.2 | 994.261 | 6.24% |
| 50_node_re 21 | 1052.194 | 86.4 | 1004.719 | 4.73% |
| 50_node_re 22 | 1143.660 | 84.2 | 1086.435 | 5.27% |
| 50_node_re 23 | 1123.628 | 99.0 | 1064.507 | 5.55% |
| 50_node_re 24 | 1029.266 | 93.0 | 971.645 | 5.93% |
| 50_node_re 25 | 1087.582 | 74.0 | 1022.928 | 6.32% |
| 50_node_re 26 | 1043.210 | 66.0 | 980.506 | 6.40% |
| 50_node_re 27 | 1127.960 | 85.8 | 1080.119 | 4.43% |
| 50_node_re 28 | 955.497 | 69.2 | 897.726 | 6.44% |
| 50_node_re 29 | 1066.754 | 85.2 | 1012.379 | 5.37% |
| 50_node_re 30 | 1143.726 | 69.4 | 1080.133 | 5.89% |
| 50_node_re 31 | 1052.052 | 82.0 | 1002.793 | 4.91% |
| 50_node_re 32 | 1115.622 | 79.4 | 1067.499 | 4.51% |
| 50_node_re 33 | 1141.652 | 78.2 | 1088.609 | 4.87% |
| 50_node_re 34 | 1086.802 | 86.8 | 1016.569 | 6.91% |
| 50_node_re 35 | 1039.868 | 74.2 | 979.604 | 6.15% |
| 50_node_re 36 | 1058.304 | 84.8 | 996.094 | 6.25% |
| 50_node_re 37 | 1073.226 | 75.0 | 1008.998 | 6.37% |
| 50_node_re 38 | 1009.634 | 75.4 | 953.870 | 5.85% |
| 50_node_re 39 | 1091.734 | 79.8 | 1022.297 | 6.79% |
| 50_node_re 40 | 1046.746 | 81.8 | 991.004 | 5.62% |
| 50_node_re 41 | 1052.942 | 88.4 | 996.890 | 5.62% |
| 50_node_re 42 | 1058.820 | 63.6 | 995.357 | 6.38% |
| 50_node_re 43 | 1065.930 | 67.0 | 1003.914 | 6.18% |
| 50_node_re 44 | 1060.270 | 84.6 | 1001.630 | 5.85% |
| 50_node_re 45 | 1026.000 | 89.0 | 971.827 | 5.57% |
| 50_node_re 46 | 1077.796 | 77.0 | 1012.852 | 6.41% |
| 50_node_re 47 | 1120.072 | 75.0 | 1053.246 | 6.34% |
| 50_node_re 48 | 1123.666 | 82.6 | 1063.170 | 5.69% |
| 50_node_re 49 | 1036.996 | 87.2 | 984.062 | 5.38% |
| 50_node_re 50 | 1118.700 | 81.2 | 1046.620 | 6.89% |
| Average: | 1081.767 | 78.8 | 1021.578 | 5.90% |

Table 3: Results for 50 node problems, central node in corner.

| Problem # | GA solutions | GA Running Time (sec) | Lower Bounds | Gap % |
|---|---|---|---|---|
| 50_node_rr 1 | 610.255 | 61.0 | 558.770 | 9.21% |
| 50_node_rr 2 | 754.853 | 63.0 | 695.080 | 8.60% |
| 50_node_rr 3 | 719.658 | 56.6 | 669.121 | 7.55% |
| 50_node_rr 4 | 701.551 | 63.0 | 646.854 | 8.46% |
| 50_node_rr 5 | 882.590 | 51.8 | 819.670 | 7.68% |
| 50_node_rr 6 | 969.545 | 65.0 | 905.207 | 7.11% |
| 50_node_rr 7 | 782.362 | 51.6 | 730.728 | 7.07% |
| 50_node_rr 8 | 759.105 | 64.0 | 700.610 | 8.35% |
| 50_node_rr 9 | 703.088 | 54.4 | 650.251 | 8.13% |
| 50_node_rr 10 | 849.200 | 66.6 | 797.342 | 6.50% |
| 50_node_rr 11 | 881.175 | 67.0 | 812.673 | 8.43% |
| 50_node_rr 12 | 685.858 | 69.8 | 637.340 | 7.61% |
| 50_node_rr 13 | 738.496 | 64.8 | 673.989 | 9.57% |
| 50_node_rr 14 | 803.492 | 69.8 | 751.839 | 6.87% |
| 50_node_rr 15 | 815.114 | 61.0 | 744.074 | 9.55% |
| 50_node_rr 16 | 885.607 | 79.4 | 823.878 | 7.49% |
| 50_node_rr 17 | 732.542 | 74.4 | 664.230 | 10.28% |
| 50_node_rr 18 | 608.798 | 72.2 | 553.443 | 10.00% |
| 50_node_rr 19 | 708.509 | 67.0 | 661.677 | 7.08% |
| 50_node_rr 20 | 832.586 | 65.4 | 767.024 | 8.55% |
| 50_node_rr 21 | 884.901 | 67.4 | 839.776 | 5.37% |
| 50_node_rr 22 | 830.286 | 69.6 | 764.893 | 8.55% |
| 50_node_rr 23 | 701.086 | 70.2 | 656.505 | 6.79% |
| 50_node_rr 24 | 744.620 | 74.6 | 685.745 | 8.59% |
| 50_node_rr 25 | 827.542 | 49.4 | 769.567 | 7.53% |
| 50_node_rr 26 | 642.110 | 48.2 | 590.763 | 8.69% |
| 50_node_rr 27 | 866.022 | 73.0 | 816.258 | 6.10% |
| 50_node_rr 28 | 723.650 | 51.6 | 669.898 | 8.02% |
| 50_node_rr 29 | 793.500 | 59.2 | 732.451 | 8.33% |
| 50_node_rr 30 | 1081.390 | 70.8 | 1019.981 | 6.02% |
| 50_node_rr 31 | 731.157 | 59.2 | 678.932 | 7.69% |
| 50_node_rr 32 | 746.974 | 78.6 | 689.648 | 8.31% |
| 50_node_rr 33 | 985.220 | 82.8 | 928.788 | 6.08% |
| 50_node_rr 34 | 994.286 | 67.0 | 930.782 | 6.82% |
| 50_node_rr 35 | 822.314 | 53.6 | 753.169 | 9.18% |
| 50_node_rr 36 | 561.240 | 55.2 | 524.467 | 7.01% |
| 50_node_rr 37 | 723.518 | 88.8 | 664.338 | 8.91% |
| 50_node_rr 38 | 782.324 | 81.2 | 729.349 | 7.26% |
| 50_node_rr 39 | 671.847 | 57.0 | 622.688 | 7.89% |
| 50_node_rr 40 | 704.057 | 74.0 | 649.060 | 8.47% |
| 50_node_rr 41 | 676.663 | 84.2 | 613.323 | 10.33% |
| 50_node_rr 42 | 811.927 | 48.4 | 746.731 | 8.73% |
| 50_node_rr 43 | 551.855 | 61.0 | 507.171 | 8.81% |
| 50_node_rr 44 | 728.951 | 43.0 | 689.220 | 5.76% |
| 50_node_rr 45 | 660.621 | 69.0 | 591.593 | 11.67% |
| 50_node_rr 46 | 820.265 | 58.8 | 759.493 | 8.00% |
| 50_node_rr 47 | 564.290 | 70.8 | 515.980 | 9.36% |
| 50_node_rr 48 | 775.379 | 68.2 | 730.892 | 6.09% |
| 50_node_rr 49 | 727.045 | 85.2 | 647.559 | 12.27% |
| 50_node_rr 50 | 643.565 | 64.0 | 599.998 | 7.26% |
| Average: | 764.060 | 65.4 | 707.656 | 8.08% |

Table 4: Results for 50 node problems, central node selected randomly.

| Problem # | GA solutions | GA Running Time (sec) | Lower Bounds | Gap % |
|---|---|---|---|---|
| 100_node_rc 1 | 1100.844 | 56.4 | 1021.960 | 7.72% |
| 100_node_rc 2 | 1130.898 | 61.0 | 1046.689 | 8.05% |
| 100_node_rc 3 | 1131.024 | 71.2 | 1050.233 | 7.69% |
| 100_node_rc 4 | 1120.026 | 67.2 | 1046.991 | 6.98% |
| 100_node_rc 5 | 1092.292 | 72.2 | 1018.994 | 7.19% |
| 100_node_rc 6 | 1138.986 | 62.8 | 1055.471 | 7.91% |
| 100_node_rc 7 | 1083.662 | 74.4 | 985.551 | 9.95% |
| 100_node_rc 8 | 1168.724 | 92.6 | 1078.915 | 8.32% |
| 100_node_rc 9 | 1127.646 | 61.8 | 1052.296 | 7.16% |
| 100_node_rc 10 | 1170.880 | 60.2 | 1075.294 | 8.89% |
| 100_node_rc 11 | 1062.028 | 74.0 | 989.938 | 7.28% |
| 100_node_rc 12 | 1071.886 | 57.6 | 991.443 | 8.11% |
| 100_node_rc 13 | 1114.166 | 73.2 | 1041.033 | 7.03% |
| 100_node_rc 14 | 1094.690 | 57.4 | 1015.852 | 7.76% |
| 100_node_rc 15 | 1071.142 | 76.8 | 997.692 | 7.36% |
| 100_node_rc 16 | 1124.400 | 48.8 | 1036.613 | 8.47% |
| 100_node_rc 17 | 1101.556 | 61.2 | 1033.530 | 6.58% |
| 100_node_rc 18 | 1219.632 | 64.8 | 1135.519 | 7.41% |
| 100_node_rc 19 | 1154.948 | 67.8 | 1057.025 | 9.26% |
| 100_node_rc 20 | 1088.270 | 56.0 | 1010.279 | 7.72% |
| 100_node_rc 21 | 1125.670 | 65.4 | 1053.878 | 6.81% |
| 100_node_rc 22 | 1106.576 | 65.2 | 1029.692 | 7.47% |
| 100_node_rc 23 | 1056.130 | 78.0 | 974.723 | 8.35% |
| 100_node_rc 24 | 1116.250 | 60.4 | 1032.265 | 8.14% |
| 100_node_rc 25 | 1138.074 | 68.4 | 1037.066 | 9.74% |
| 100_node_rc 26 | 1106.626 | 55.4 | 1030.699 | 7.37% |
| 100_node_rc 27 | 1070.384 | 59.0 | 1001.511 | 6.88% |
| 100_node_rc 28 | 1081.568 | 77.6 | 1009.329 | 7.16% |
| 100_node_rc 29 | 1088.274 | 73.0 | 1011.166 | 7.63% |
| 100_node_rc 30 | 1155.576 | 60.2 | 1074.143 | 7.58% |
| 100_node_rc 31 | 1063.984 | 73.0 | 987.769 | 7.72% |
| 100_node_rc 32 | 1079.964 | 74.0 | 1011.384 | 6.78% |
| 100_node_rc 33 | 1082.864 | 76.2 | 999.343 | 8.36% |
| 100_node_rc 34 | 1123.736 | 73.8 | 1051.224 | 6.90% |
| 100_node_rc 35 | 1090.768 | 63.2 | 1020.212 | 6.92% |
| 100_node_rc 36 | 1155.266 | 89.8 | 1069.918 | 7.98% |
| 100_node_rc 37 | 1140.170 | 66.0 | 1065.392 | 7.02% |
| 100_node_rc 38 | 1143.992 | 64.8 | 1057.484 | 8.18% |
| 100_node_rc 39 | 1155.468 | 69.2 | 1071.248 | 7.86% |
| 100_node_rc 40 | 1118.640 | 52.2 | 1041.582 | 7.40% |
| 100_node_rc 41 | 1106.170 | 59.0 | 1028.042 | 7.60% |
| 100_node_rc 42 | 1084.732 | 50.0 | 1015.577 | 6.81% |
| 100_node_rc 43 | 1014.684 | 58.4 | 953.365 | 6.43% |
| 100_node_rc 44 | 1082.410 | 74.8 | 1005.343 | 7.67% |
| 100_node_rc 45 | 1131.210 | 66.4 | 1045.948 | 8.15% |
| 100_node_rc 46 | 1168.932 | 77.8 | 1073.781 | 8.86% |
| 100_node_rc 47 | 1135.820 | 64.6 | 1061.260 | 7.03% |
| 100_node_rc 48 | 1154.732 | 65.6 | 1076.947 | 7.22% |
| 100_node_rc 49 | 1086.654 | 61.4 | 1015.103 | 7.05% |
| 100_node_rc 50 | 1055.772 | 68.6 | 975.614 | 8.22% |
| Average: | 1111.776 | 66.6 | 1032.446 | 7.68% |

Table 5: Results for 100 node problems, central node in center.

would be to make it capable of dealing with non-unit demands. This capability is in a way already "built-in" in the case of the genetic algorithm since the representation we used is, as we mentioned earlier, popular with grouping problems that attempt to partition items with non-unit weights. However, this might not be as easy for the construction heuristic and it will definitely add more complexity to the procedure.

# References


[1] Ahuja, R. K., J. B. Orlin, D. Sharma,"Multi-Exchange Neighborhood Structures for the Capacitated Minimum Spanning Tree Problem," *Mathematical Programming*, **91**, pp. 71–97, 2001.

[2] Amberg, A., W. Domschke, S. V. Darmstadt, "Capacitated Minimum Spanning Trees: Algorithms Using Intelligent Search," *Combinatorial Optimization: Theory and Practice*, **1**, pp. 9–40. 1996.

[3] Berger, D., B. Gendron, J. Y. Potvin, S. Raghavan, P. Soriano, "Tabu Search for a Network Loading Problem with Multiple Facilities," *Journal of Heuristics*, **6**(2), pp 253–267, 2000.

[4] Bourjolly, J., D. Tomiuk, G. H. M. Kapantow, "Using Simulated Annealing to Minimize the Cost of Centralized Telecommunications Networks," *INFOR* **37**(3), 1999.

[5] Esau, L. R., K. C. Williams, "On Teleprocessing System Design: Part II," *IBM System Journal*, **5**(3), pp. 142–147, 1966.

[6] Falkenauer, E., "A Hybrid Grouping Genetic Algorithm for Bin Packing", *Journal of Heuristics*, pp. 5–30, 1996.

[7] Gamvros, I., "An Evolutionary Approach for the Multi-Level Capacitated Minimum Spanning Tree," M.S. in Telecommunications' Project, University of Maryland, 2001.

[8] Gamvros, I., B. L. Golden, S. Raghavan, "The Multi-Level Capacitated Spanning Tree Problem," *In preparation*, 2002.

[9] Gavish, B. "Topological Design of Telecommunications Networks - Local Access Design Methods," *Annals of Operations Research*, **33**, pp 17–71, 1991.

[10] Gavish, B., K. Altinkemer, "A Parallel Savings Heuristic for the Topological Design of Local Access Tree Networks," *Proc. IEEE-INFOCOM*, pp. 130–139. 1986

[11] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[12] Gouveia, L., M. J. Lopes, "Using Generalized Capacitated Trees for Designing the Topology of Local Access Networks," *Telecommunication Systems*, **7**, pp. 315–337, 1997.

[13] Karnaugh, M., "A New Class of Algorithms for Multipoint Network Optimization," *IEEE Transactions on Communications*, **24**(5), pp. 500–505, 1976.

[14] Kershenbaum, A., R. Boorstyn, R. Oppenheim, "Second-Order Greedy Algorithms for Centralized Network Design," *IEEE Transactions on Communications*, **22**(11), pp. 1835–1838, 1980.

[15] McGregor, P. M., D. Shen, "Network Design: An Algorithm for Access Facility Location Problems," *IEEE Trans. Comm.*, **25**, pp. 61–73, 1977.

[16] Patterson, R., H. Pirkul, "Heuristic Procedure Neural Networks for the CMST Problem", *Computers and Operations Research*, **27**, pp. 1171–1200, 2000.

[17] Rothfarb, B., M. C. Goldstein, "The One-Terminal Telepak Problem," *Operations Research*, **19**, pp. 156–169, 1971.

[18] Salman F. S., R. Ravi, J. Hooker, "Solving the Local Access Network Design Problem,", Working Paper, Krannert Graduate School of Management, Purdue University, 2001.

[19] Sharaiha Y. M., M. Gendreau, G. Laporte, I. H. Osman, "A Tabu Search Algorithm for the Capacitated Minimum Spanning Tree Problem," *Networks*, **29**, pp. 161–171, 1997.

[20] Sharma R. L., "Design of an Economical Mutlidrop Network Topology with Capacity Constraints," *IEEE Trans. Comm. COM*, **31**, pp. 590-591, 1983.