New Neighborhood Search Structures for the Capacitated Minimum Spanning Tree Problem

by
Ravindra K. Ahuja
James B. Orlin
Dushyant Sharma

WP #4040 November 1998

New Neighborhood Search Structures
for
the Capacitated Minimum Spanning Tree Problem

Ravindra K. Ahuja
Industrial & Systems Engineering
University of Florida
Gainesville, FL 32611
email: ahuja@ufl.edu

James B. Orlin
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
email: jorlin@mit.edu

Dushyant Sharma
Operations Research Center
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
email: dushyant@mit.edu

(September 1998)

New Neighborhood Search Structures for the Capacitated Minimum Spanning Tree Problem

Ravindra K. Ahuja[1], James B. Orlin[2], and Dushyant Sharma[3]

ABSTRACT

The capacitated minimum spanning tree problem is to find a minimum cost spanning tree with an additional cardinality constraint on the sizes of the subtrees incident to a given root node. The capacitated minimum spanning tree problem arises as a fundamental subproblem in many telecommunication network design problems. The capacitated minimum spanning tree problem is a hard combinatorial problem and existing exact algorithms can solve only small size problems. Therefore, there has been substantial interest in developing heuristic procedures for this problem. Currently, the best available heuristic procedures for the CMST problem are the tabu search algorithms due to Amberg et al. [1996] and Sharaiha et al. [1997]. These algorithms use neighborhood structures that are based on exchanging a single node or a set of nodes between two subtrees. In this paper, we generalize their neighborhood structures to allow exchanges of nodes among multiple subtrees simultaneously. Our first neighborhood structure allows exchanges of single nodes among several subtrees. Our second neighborhood structure allows exchanges that involve multiple subtrees. The size of each of these neighborhood structures grows exponentially with the problem size. To search the neighborhood efficiently we transform each feasible exchange into a "subset-disjoint" cycle in a graph, called the *improvement graph*. Our approach, which is based on the cyclic transfer neighborhood structure of Thompson and Orlin [1989], transforms a profitable exchange into a negative cost subset-disjoint cycle in the improvement graph. We heuristically identify these cycles by a modification of the shortest path label-correcting algorithm. Our computational results with local improvement and tabu search algorithms based on these neighborhood structures are very encouraging. For the unit demand case, our algorithms obtained the best known solutions for all benchmark instances, and improved some. For the heterogeneous demand case, our algorithms improved the best known solutions for most of the benchmark instances, with improvements by as much as 18%.

[1] Industrial & Systems Engineering, University of Florida, Gainesville, FL 32611.
[2] Sloan School of Management, MIT, Cambridge, MA 02139.
[3] Operations Research Center, MIT, Cambridge, MA 02139.

1. INTRODUCTION

The capacitated minimum spanning tree problem (CMST) is a fundamental problem in telecommunication network design. In this problem, we are given a central processor and a set of remote terminals with specified demands for traffic that must flow between the central processor and terminals. The objective is to build a minimum cost network to carry this demand. Between any pair of terminals or between the central processor and the terminals, there is a potential link that can be included for a given cost. The problem is to design a network connecting the terminal with the central processor so that the traffic on any arc of the network is at most K and the total design cost of the network is minimum. When the demands of all terminals are equal to one, then the problem reduces to finding a rooted spanning tree in which each of the subtrees incident to the root node contains at most K nodes. In the literature, this unit demand problem is referred to as *the capacitated minimum spanning tree problem*. More generally, we refer to the problem of non-unit demand case, that is, the heterogeneous demand case, as the capacitated minimum spanning tree problem. In this paper, we develop and analyse heuristic procedures for the capacitated minimum spanning tree problem.

Let G = (N, A) be a complete undirected graph. Let N = {0, 1, 2, ..., n} denote the node set, and let node 0 denote the *source node* or the central processor. Each node $i \in N\backslash\{0\}$ has an associated demand d_i that must be fulfilled by sending flow from the source node. Each arc (i, j) $\in$ A has an associated cost c_{ij}, and a capacity K that limits the maximum flow on the arc. The capacitated minimum spanning tree problem can be conceived of as a partitioning problem where the node set $N\backslash\{0\}$ is partitioned into L subsets R[1], R[2], ..., R[L] satisfying $\sum_{k\in R[p]} d_k \le K$ for each p, $1 \le p \le L$. Let $\mathbf{T}[p]$ denote a minimum (cost) spanning tree over the node subset $R[p]\cup\{0\}$ for each p, $1 \le p \le L$, and $\mathbf{T}^*$ denote the union of these trees. We define the cost of the tree $\mathbf{T}^*$ as $\sum_{\{(i,j)\in T^*\}} c_{ij}$. We illustrate these definitions using Figure 1, where we partition N into three subsets R[1] = {1, 4, 5, 10, 11, 15}, R[2] = {2, 6, 7, 12, 16}, and R[3] = {3, 8, 9, 13, 14}. We assume that each node has unit demand and K = 6. The minimum spanning trees over these node subsets are as shown. The cost of the tree $\mathbf{T}^*$ is 104. The capacitated minimum spanning tree problem is to identify the node partitions so that the resulting tree $\mathbf{T}^*$ has the minimum possible cost.

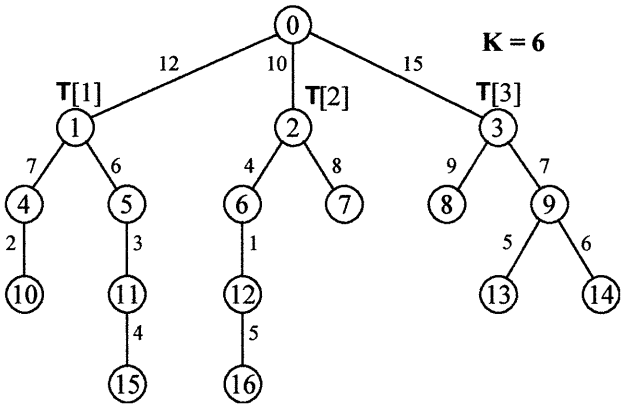


Figure 1. Example of a capacitated spanning tree where each node has unit demand.

There is a substantial research literature devoted to the capacitated minimum spanning tree problem. The survey paper by Gavish [1991] gives a detailed understanding of telecommunication design problems where the capacitated minimum spanning tree problem arises. The recent paper by Amberg et al. [1996] presents an excellent survey of exact and heuristic algorithms for the CMST problem. Typically, exact algorithms can solve problems of size less than 50 nodes. A cutting plane algorithm by Hall [1996], and Lagrangian relaxation based approaches by Gouveia [1995, 1996], and Gouveia and Martins [1996, 1998] obtain good solutions for the capacitated minimum spanning tree problem and gave linear programming based upper bounds on the percentage error. Amberg et al. [1996] and Sharaiha et al. [1997] have suggested tabu search algorithms for solving the capacitated minimum spanning tree problem. The website maintained by J.E. Beasley (http://www.ms.ic.ac.uk/info.html) provides benchmark instances of the capacitated minimum spanning tree problem. We obtained the best known solution values for these benchmark instances from Luis Gouveia.

In this paper, we focus on neighborhood search algorithms for the capacitated minimum spanning tree problem. The performance of a neighborhood search algorithm critically depends upon the neighborhood structure, the manner in which we define neighbors of a feasible solution. Currently, the best available neighborhood structures to our knowledge are those of Amberg et al. [1996] and Sharaiha et al. [1997]. Amberg et al.'s neighborhood structure is based on exchanging single nodes between two subtrees. The nodes can be anywhere in the subtrees and may not always be the leaf nodes of the subtrees. For example, in Figure 1, exchanging nodes 11 and 12 between **T**[1] and **T**[2] gives a neighbor of the solution shown. The neighborhood structure due to Sharaiha et al. moves a part of a subtree, from one subtree to another or to the root node. For example, in Figure 1, we can obtain a neighbor by deleting the node subset {5, 11, 15} from **T**[1] and attaching it to the root node. The number of neighbors in both of these neighborhood structures is no more that n^2. Both papers report computational results of tabu

search methods based on their neighborhood structures. Amberg et al. report results on the unit-demand instances and Sharaiha et al. report results on both unit-demand and heterogeneous demand instances. Amberg et al. obtained the best known solutions for all the benchmark instances for the unit demand case and Sharaiha et al. obtained the best known solutions for the heterogeneous demand case.

In this paper, we suggest two new neighborhood structures that can be regarded as generalizations of the above two neighborhood structures. Both the neighborhood structures are based on the cyclic exchange neighborhood structure developed by Thompson and Orlin [1989]. Glover [1996] refers to this type of exchange as *ejection chains*. Thompson and Psaraftis [1993] and Gendreau et al. [1998] have used this neighborhood structure to solve vehicle routing problems and obtained impressive results. We use cyclic exchange neighborhood structures to solve the capacitated minimum spanning tree problem.

Our first neighborhood structure allows exchanges of single nodes spanning several subtrees. For example, in Figure 1, one possible modification of **T** may be described as follows: (i) node 5 moves from **T**[1] to **T**[2], (ii) node 7 moves from **T**[2] to **T**[3], and (iii) node 9 moves from **T**[3] to **T**[1]. In general, a move can span some or all of the subtrees in **T**. Our second neighborhood structure allows moves which exchange subsets of nodes (a sub-subtree of a subtree) spanning several trees. For example, in Figure 1, one possible exchange consists of (i) node 5, 11, and 15 move from **T**[1] to **T**[2], (ii) nodes 12 and 16 move from **T**[2] to **T**[3], and (iii) nodes 9, 13, and 14 move from **T**[3] to **T**[1].

Our neighborhood structure allows exponentially large number of neighbors of a solution. Therefore, an explicit enumeration of the entire neighborhood will, in many cases, be prohibitively expensive. We develop a heuristic search technique based on the concept of *improvement graph* which converts each possible cyclic exchange into a "subset-disjoint" cycle in the improvement graph and converts a profitable cyclic exchange into a negative cost cycle. We heuristically identify negative cost subset-disjoint cycles using a modification of the shortest path label-correcting algorithm. To judge the efficacy of these neighborhood structures, we developed local improvement and tabu search algorithms and tested them on standard benchmark instances. For the unit demand case, our algorithms obtained the best known solutions for all benchmark instances and improved some. For the heterogeneous demand case, our algorithms improve the best known solutions for most of the benchmark instances.

2. NOTATION

In this section, we describe the notation used in the rest of the paper. We also use some graph notation in this paper, such as trees, subtrees, paths, and cycles. We omit the definitions of some standard terms here and refer the reader to the book of Ahuja, Magnanti and Orlin [1993].

Suppose that **T** is a spanning tree, where node 0 is a specially designated node, called the *source* node. We assume that arcs in the tree **T** denote the parent-child relationship, the node closer to the source node being the parent of the node farther from the source node. For each node i in **T**, we denote by T_i the subtree of **T** rooted at node i. We denote by S_i the set of descendants of node i, that is, children of node i, children of their children, and so on. For example, in Figure 1, T_5 denotes the subtree rooted at node 5 in **T**[1] and $S_5 = \{5, 11, 15\}$.

We refer to the children of the source nodes as *root* nodes and the trees rooted at the root node as *rooted subtrees*. We will often refer to the rooted subtrees simply as subtrees, when the rooted subtree will be obvious from the context. For any node i, we denote by **T**[i] as rooted subtree containing node i. We denote by **S**[i] the set of nodes contained in the subtree **T**[i]. For example, in Figure 1, for each node i = 4, 5, 10, 11, and 15, **T**[i] = **T**[1] and **S**[i] = {1, 4, 5, 10, 11, 15}.

For a subset S of nodes, we let $d(S) = \sum_{i \in S} d_i$. We say that the *subset* S is feasible if and only if $d(S) \leq K$. If S is feasible we denote by c(S) the cost of a minimum cost tree spanning the node set $S \cup \{0\}$. We say that the rooted subtree **T**[i] is *feasible* if $\sum_{j \in \mathbf{T}[i]} d_j \leq K$, and **T** is feasible if **T**[i] is feasible for each $i \in N \backslash \{0\}$.

3. EXISTING NEIGHBORHOOD STRUCTURES

A neighborhood search algorithm for the capacitated minimum spanning tree problem starts with a feasible tree **T**. Using a neighborhood structure, it defines neighbors of T; those solutions that can be obtained from **T** by performing an "exchange". It then identifies a suitable neighbor, replaces **T** by it, and repeats this process until a suitable termination criteria is reached. The performance of a neighborhood search algorithm crucially depends on the neighborhood structure. In this section, we review two existing neighborhood structures due to Amberg et al. [1996] and Sharaiha et al. [1997]. We assume in this section that each node $i \in N \backslash \{0\}$ has unit demand. We make this assumption for simplicity of notation. The non-unit demand case can be handled in a similar manner.

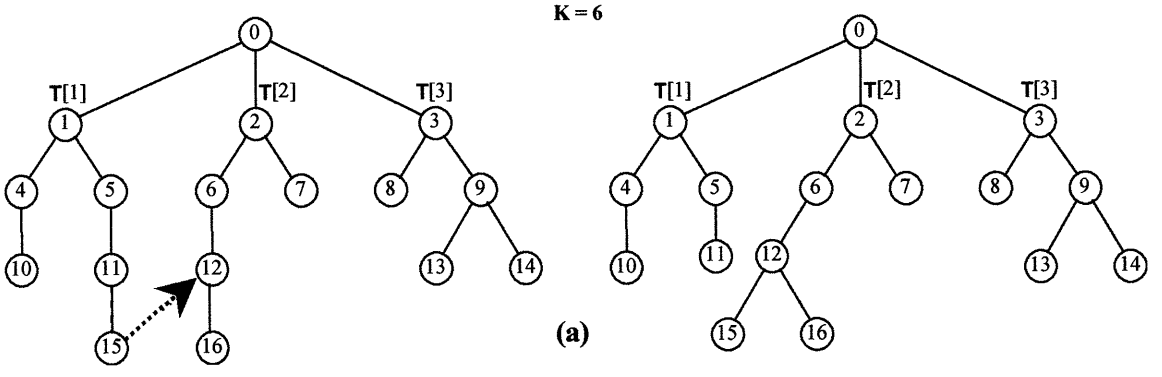
Neighborhood Structure due to Amberg et al.'s

The neighborhood search method due to Amberg et al. uses a node exchange procedure that transforms one feasible solution to a neighboring solution by changing the assignment of nodes in the subtrees; such a transformation is called a *move*. Their procedure considers two types of exchanges:

Node Shift: A node shift move chooses one node and moves it from its current subtree to another subtree.

Node Exchange: A node exchange chooses two nodes belonging to different subtrees and exchanges them.

The nodes selected for shift or exchange may not always be the leaf nodes of the subtrees they belong to. Further, after the shift or exchange move, the subtrees may readjust themselves so that each subtree is a minimum spanning tree over the node set it spans. The procedure allows only feasible exchanges, that is, moves which do not violate the capacity constraints. We show in Figure 2(a) a feasible node shift, where node 15 moves from the subtree **T**[1] to the subtree **T**[2]. We show in Figure 2(b) a feasible node exchange where node 11 in the subtree **T**[1] is exchanged with node 6 in the subtree **T**[2].



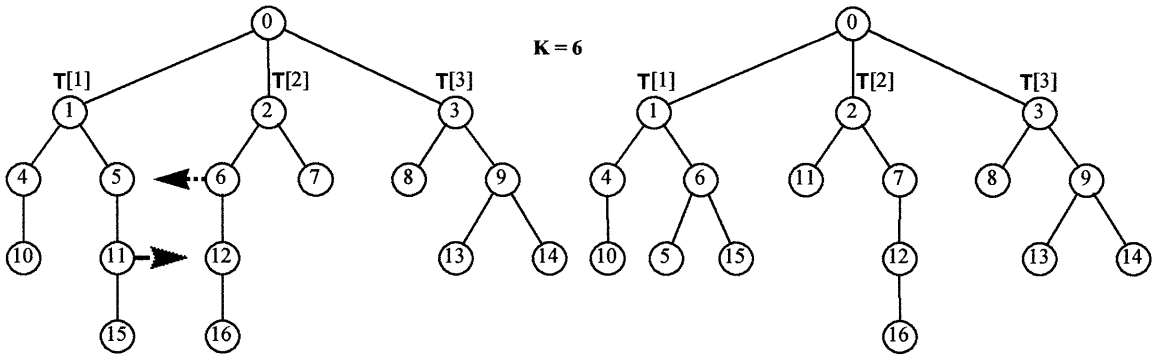


Figure 2. Illustrating neighborhood structure due to Amberg et al.

It is easy to see that there are $\Omega(nL)$ node shifts and $\Omega(n^2)$ node exchanges, where L is the total number of subtrees in the solution **T**. The cost of a node shift or a node exchange is calculated by determining the impact of the move on the objective function value. The cost of a node shift which shifts node i from the subtree **T**[i] to the subtree **T**[j] is $c(\mathbf{S}[i]\backslash\{i\}) - c(\mathbf{S}[i]) + c(\{i\}\cup\mathbf{S}[j]) - c(\mathbf{S}[j])$, and this can be determined by solving (or, reoptimizing) two minimum spanning tree problems. The cost of a node exchange which exchanges node i with node j is $c(\{j\}\cup\mathbf{S}[i]\backslash\{i\}) - c(\mathbf{S}[i]) + c(\{i\}\cup\mathbf{S}[j]\backslash\{j\}) - c(\mathbf{S}[j])$, and this too can be determined by solving two minimum spanning tree problems. It can be shown that each spanning tree problem can be solved in O(K) time for the unit demand case. Consequently, this method takes $O(n^2K)$ time to determine the most profitable node shift or a node exchange.

Neighborhood Structure due to Sharaiha et al.

The neighborhood search method due to Sharaiha's et al. performs cut and paste operations. A cut and paste operation consists of cutting either the whole subtree or a part of a subtree and then connecting (pasting) it either to the source node or to some other subtree. We illustrate this operation in Figure 3, where the part of the subtree **T**[1] rooted at node 5, the subsubtree T_5, is cut from **T**[1] and connected to the source node by using the minimum cost arc between T_5 and the source node. If the method deletes the arc (i, j) and adds in the arc (k, *l*), then the cost of the cut and paste operation is $c_{kl} - c_{ij}$. There are $\Omega(n^2)$ possibilities of cut and paste operations and it takes $O(K^2)$ time to identify the cost of each such operation. Consequently, the method takes $O(n^2K^2)$ time to determine the most profitable cut and paste operation.

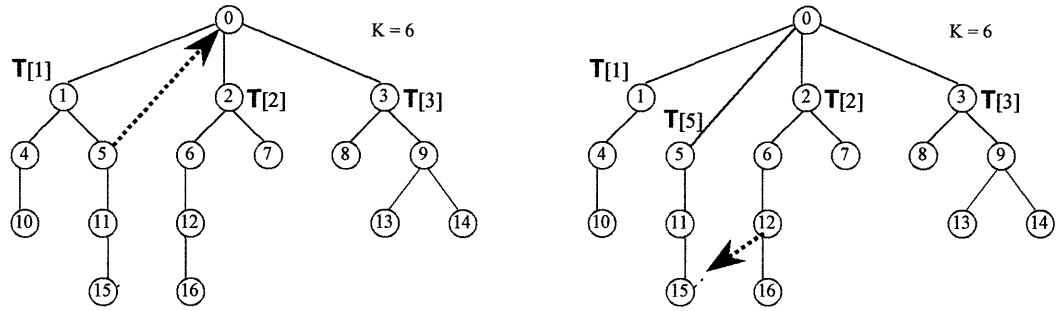


Figure 3. Illustrating the neighborhood structure due to Sharaiha et al.

4. NODE-BASED CYCLIC AND PATH EXCHANGE NEIGHBORHOOD STRUCTURES

Our first neighbourhood structure is a generalization of the neighbourhood structure of Amberg et al. [1996] and allows changes that involve more than two subtrees, we call such changes as multi-exchanges. We allow two types of multi-changes – *cyclic exchanges* and *path exchanges*.

Cyclic Exchanges

Let **T** be a tree satisfying the capacity constraints. We represent a *cyclic exchange* as i_1-i_2-i_3-…-i_r-i_1, where the nodes i_1, i_2, i_3, …, i_r belong to different rooted subtrees, that is, $\mathbf{T}[i_r] \neq \mathbf{T}[i_s]$ for $r \neq s$. The cyclic exchange i_1-i_2-i_3-…-i_r-i_1 represents the following changes: node i_1 moves from the subtree $\mathbf{T}[i_1]$ to the subtree $\mathbf{T}[i_2]$, node i_2 moves from the subtree $\mathbf{T}[i_2]$ to the subtree $\mathbf{T}[i_3]$, and so on, and finally node i_r moves from the subtree $\mathbf{T}[i_r]$ to the subtree $\mathbf{T}[i_1]$. We call the cyclic exchange i_1-i_2-i_3-…-i_r-i_1 *feasible* if the changed tree satisfies the capacity constraint. In other words, the cyclic exchange i_1-i_2-i_3-…-i_r-i_1 is a feasible move if only if the set of nodes $\{i_{p-1}\} \cup \mathbf{S}[i_p] \setminus \{i_p\}$ is feasible for every $p = 1, 2, \ldots, r$ where $i_0 = i_r$. Let $\mathbf{T}'$ denote the new feasible (capacitated) tree. We define the cost of this cyclic exchange as $c(\mathbf{T}') - c(\mathbf{T})$. Observe that

$$c(\mathbf{T}') - c(\mathbf{T}) = \sum_{p=1}^{r} \left(c\left(\{i_{p-1}\} \cup \mathbf{S}[i_p] \setminus \{i_p\}\right) - c\left(\mathbf{S}[i_p]\right) \right). \quad (1)$$

In other words, we solve minimum spanning trees over the new node subsets. The difference between the costs of new subtrees and the previous subtrees gives the cost of the

cyclic exchange. We call the cyclic exchange *profitable* if $c(\mathbf{T}') < c(\mathbf{T})$ and *non-profitable* otherwise. We illustrate the cyclic exchange using the numerical example shown in Figure 4.

A cyclic exchange may increase the number of subtrees of **T**. While computing $c(\{i_{p-1}\} \cup \mathbf{S}[i_p] \setminus \{i_p\})$ when it obtains a minimum spanning tree over the node subset $\{0, i_{p-1}\} \cup \mathbf{S}[i_p] \setminus \{i_p\}$, two or more arcs in it may be incident to the node 0; in this case a single subtree becomes two or more subtrees. A cyclic exchange cannot decrease the number of subtrees.

Path Exchanges

We now define a path exchange. We represent a path exchange as i_1-i_2- ...-i_r, where the nodes i_1, i_2, ... , i_r belong to different subtrees. The path exchange i_1-i_2- ...-i_r represents the following changes: node i_1 moves from the subtree $\mathbf{T}[i_1]$ to the subtree $\mathbf{T}[i_2]$, node i_2 moves from the subtree $\mathbf{T}[i_2]$ to the subtree $\mathbf{T}[i_3]$, and so on, and finally node i_{r-1} moves from the subtree $\mathbf{T}[i_{r-1}]$ to the subtree $\mathbf{T}[i_r]$. This exchange is similar to the cyclic exchange except that no node moves from the subtree $\mathbf{T}[i_r]$ to the subtree $\mathbf{T}[i_1]$. In this exchange, the subtree $\mathbf{T}[i_1]$ loses one node and the subtree $\mathbf{T}[i_r]$ gains one node. We call the path exchange i_1-i_2- ...-i_r *feasible* if the changed tree satisfies the capacity constraints. In other words, the path exchange i_1-i_2- ...-i_r is a feasible exchange if only if $\{i_{p-1}\} \cup \mathbf{S}[i_p] \setminus \{i_p\}$ is a feasible set of nodes for every p = 2, 3, ..., r. If $\mathbf{T}'$ denotes the changed tree after this path exchange has been performed, then the cost of this path exchange is

$$c(\mathbf{T}')-c(\mathbf{T}) = c(\mathbf{S}[i_1] \setminus \{i_1\}) + \sum_{p=2}^{r-1} c(\{i_{p-1}\} \cup \mathbf{S}[i_p] \setminus \{i_p\}) + c(\{i_{r-1}\} \cup \mathbf{S}[i_r]) - \sum_{p=1}^{r} c(\mathbf{S}[i_p]). \qquad (2)$$

This path exchange is *profitable* if $c(\mathbf{T}') \leq c(\mathbf{T})$ and *non-profitable* otherwise. We illustrate a path exchange using the numerical example shown in Figure 5. We point out that a path exchange can increase the number of subtrees in **T**, as in the case of cyclic exchange. But contrary to a cyclic exchange, a path exchange can also decrease the number of subtrees in **T**. We observe that in a path exchange i_1-i_2- ...-i_r, the subtree $\mathbf{T}[i_1]$ loses a node. If it consisted of a single node, then the number of subtrees will decrease.

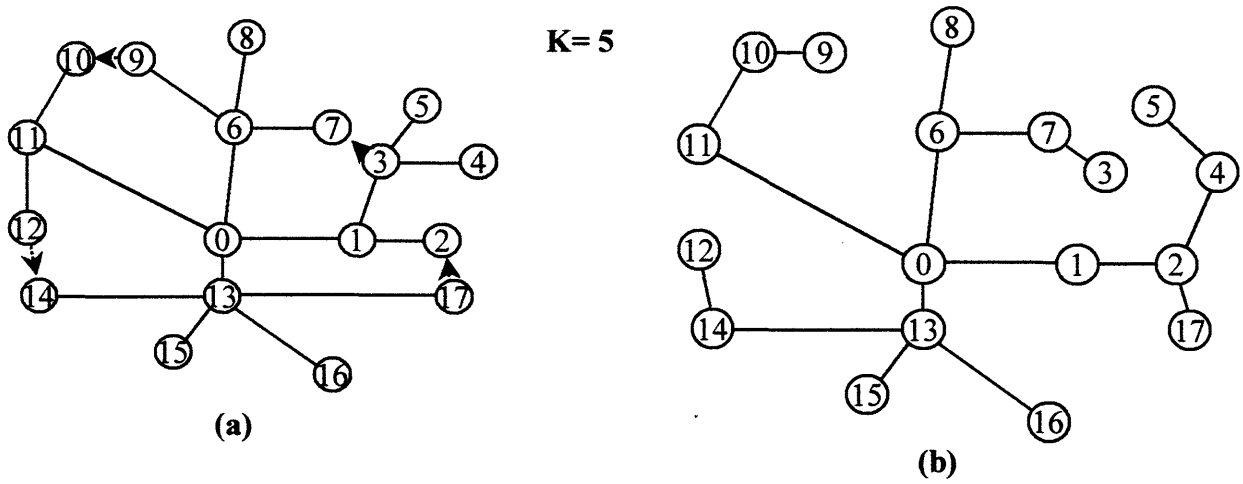


Figure 4. Illustrating the cycle move.
(a) A feasible tree before the cycle move.
(b) The tree after the cycle move 3-9-12-17-3.

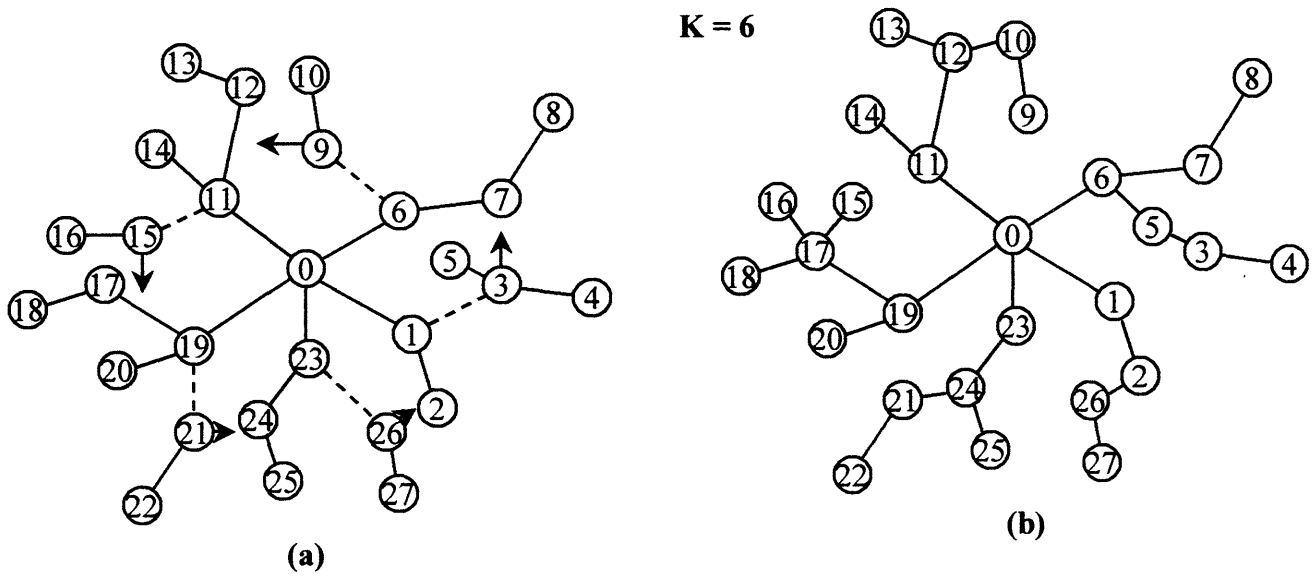


Figure 5. Illustrating the cycle move in tree-based neighborhood structure.
(a) The tree before the move.
(b) The tree after the move 3-9-15-21-26-3.

Neighborhood

For a given tree **T**, we define another tree **T′** as a *neighbor* of **T** if **T′** is a feasible capacitated tree and can be obtained from **T** by performing a cyclic or path exchange. We define the *neighborhood* of **T** as a collection of all trees which are neighbors of **T**. The neighborhood based on the cyclic and path exchanges is very large. Examining the entire neighborhood to identify a profitable exchange may be an extremely time-consuming task. For the unit-demand capacitated minimum spanning tree problem with n nodes and capacity K, the neighborhood of a tree may contain as many as $\Omega(K^{n/K}((n/K) - 1)!)$ solutions, which grows exponentially with the problem size n. We will suggest a heuristic that does not enumerate the entire neighborhood but is quite effective in practice. It identifies a profitable move in time comparable to those of other neighborhood structures due to Amberg et al. and Sharaiha et al. The basic idea underlying our scheme is the concept of the improvement graph, which we define next.

Improvement Graph

The concept of improvement graph is originally due to Thompson and Orlin [1989], and Thompson and Psaraftis [1993]. The improvement graph for the node-based neighborhood structure is defined with respect to a feasible tree and is represented by $\mathbf{G}^1(\mathbf{T})$. The graph $\mathbf{G}^1(\mathbf{T})$ is a directed graph with the same node set as the graph G. There is a one-to-one correspondence between nodes in G and those in $\mathbf{G}^1(\mathbf{T})$. The arc set in $\mathbf{G}^1(\mathbf{T})$ is defined in a manner so that each cyclic or path exchange with respect to **T** defines a directed cycle in $\mathbf{G}^1(\mathbf{T})$, and the cost of the cycle equals the cost of the corresponding exchange.

We now explain how to construct the improvement graph $\mathbf{G}^1(\mathbf{T})$ for a given tree **T**. The node set of $\mathbf{G}^1(\mathbf{T})$ is N, the same as that of the network G, but the arc set is in general different. A directed arc (i, j) in $\mathbf{G}^1(\mathbf{T})$ signifies that node i leaves the subtree **T**[i] and joins the subtree **T**[j] and simultaneously node j leaves the subtree **T**[j]. To construct the improvement graph, we consider every pair i and j of nodes in N, and add arc (i, j) if and only if (i) **T**[i] ≠ **T**[j], and (ii) $\{i\} \cup \mathbf{S}[j] \setminus \{j\}$ is a feasible subset of nodes. We define the cost α_{ij} of arc (i, j) as

$$\alpha_{ij} = c(\{i\} \cup \mathbf{S}[j] \setminus \{j\}) - c(S[j]). \tag{3}$$

We call a directed cycle i_1-i_2- …-i_r-i_1 in the improvement graph *subset-disjoint* if the subtrees $\mathbf{T}[i_1]$, $\mathbf{T}[i_2]$, … , $\mathbf{T}[i_p]$ are different rooted subtrees, that is, $\mathbf{T}[i_p] \neq \mathbf{T}[i_q]$ for $p \neq q$. Our algorithms use the following result.

Lemma 1. (Thompson and Orlin [1989]) *There is a one-to-one cost-preserving correspondence between cyclic exchanges with respect to the tree* $\mathbf{T}$ *and directed subset-disjoint cycles in* $\mathbf{G^1(T)}$.

Proof. Consider a cyclic exchange i_1-i_2- ...-i_r-i_1 with respect to the tree $\mathbf{T}$ and let $\mathbf{T}'$ denote the changed tree after the cyclic exchange. It follows from the definition of the cyclic exchange that $\mathbf{T}[i_p] \neq \mathbf{T}[i_q]$ for $p \neq q$, and that the set of nodes $\{i_{p-1}\} \cup \mathbf{S}[i_p] \setminus \{i_p\}$ is feasible for every p, $1 \leq p \leq r$. Consequently, i_1-i_2- ...-i_r-i_1 is a subset-disjoint directed cycle in $\mathbf{G^1(T)}$. The cost of the cycle $W = i_1$-i_2- ...-i_r-i_1 in $\mathbf{G^1(T)}$ is

$$c(W) = \sum_{p=1}^{r} \alpha_{i_{p-1},i_p} = \sum_{p=1}^{r} \left(c\left(\{i_{p-1}\} \cup \mathbf{S}[i_p] \setminus \{i_p\}\right) - c\left(\mathbf{S}[i_p]\right) \right) = c(\mathbf{T}') - c(\mathbf{T}), \tag{4}$$

where the first equality follows from (3) and the second equality follows from (1). We have thus shown that a cyclic exchange i_1-i_2- ...-i_r-i_1 with respect to $\mathbf{T}$ induces a subset-disjoint directed cycle i_1-i_2- ...-i_r-i_1 in the improvement graph $\mathbf{G^1(T)}$, and both have the same cost. The converse result, that a subset-disjoint directed cycle i_1-i_2- ...- i_r-i_1 in the improvement graph $\mathbf{G^1(T)}$ induces a cyclic exchange i_1-i_2- ...-i_r-i_1 of the same cost, can also be proved similarly. ♦

We next explain how to determine the costs of arcs in the improvement graph. We use the expression given in (3) to compute arc costs. The expression (3) contains two terms. We assume that the cost $c(\mathbf{S}[j])$ of each rooted subtree $\mathbf{T}[j]$ is already known. Hence to determine α_{ij}, we need to compute $c(\{i\} \cup \mathbf{S}[j] \setminus \{j\})$. This involves deleting a node j from $\mathbf{T}[j]$ and adding node i to it, and determining the cost of the minimum spanning tree. To solve a minimum spanning tree over the node set $\{i\} \cup \mathbf{S}[j] \setminus \{j\}$, we need to consider the arc set which is the union of the current spanning tree over $\mathbf{T}[j] \setminus \{j\}$ and the arcs (i, h) for every $h \in \mathbf{S}[j] \setminus \{j\}$. In the case of unit demands, the graph over which the minimum spanning tree problem is being solved contains O(K) nodes and O(K) arcs. This problem can be solved in O(K) time (Han et al. [1995]). We have thus shown that we can determine the cost of one arc in the improvement graph in O(K) time, and the costs of all the arcs in $O(n^2K)$ total time.

Indentifying Path Exchanges

In the previous discussion, we explained how to convert a cyclic exchange with respect to a feasible tree $\mathbf{T}$ into a subset-disjoint cycle in the improvement graph $\mathbf{G^1(T)}$ with the same cost. We will now explain how to convert a path exchange with respect to $\mathbf{T}$ again into a subset-

disjoint cycle with the same cost. This transformation requires the augmentation of the improvement graph by adding some nodes and arcs. We create a *pseudonode* for each rooted subtree in **T**, and an extra node called the *origin node* v. For notational convenience, we will refer to each original node i in the improvement graph as a *regular node*. In the improvement graph, we connect the origin node v to each regular node i using the arc (v, i) and set its cost to c(**S**[i] \{i}) - c(**S**[i]) (see, for example, Figure 6). We also connect each pseudonode h to the origin node v using the arc (h, v) and set its cost to 0. We also connect each regular node i to each pseudonode h if node i does not belong to the rooted subtree represented by the pseudonode h, say **T**[h], and $\sum_{k \in T[h]} d_k + d_i \leq K$; this arc signifies that node i moves from the subtree **T**[i] to the subtree **T**[h], but no node moves out of **T**[h]. Consequently, we define the cost of the arc (i, h) as

$$\alpha_{ih} = c(\mathbf{S}[h] \cup \{i\}) - c(\mathbf{S}[h]). \tag{5}$$

It follows from (5) that α_{ih} denotes the cost of attaching node i to T_h. Using arguments similar to what we used for the cyclic exchange case, it can be shown that there is one-to-one cost-preserving correspondence between the path exchanges with respect to the tree **T** and subset-disjoint cycles in $\mathbf{G}^1(\mathbf{T})$ that pass through the origin node v. We will henceforth refer to the augmented improvement graph as the improvement graph $\mathbf{G}^1(\mathbf{T})$ since it is a data structure that helps us to identify both the profitable cyclic and path exchanges.

To summarise, using the concept of improvement graph, we have transformed the problem of finding profitable exchanges into identifying negative (cost) subset-disjoint cycles in $\mathbf{G}^1(\mathbf{T})$. We discuss next how to identify such cycles.

Identifying Negative Subset-Disjoint Cycles

A directed cycle i_1-i_2- …-i_r-i_1 in the improvement graph $\mathbf{G}^1(\mathbf{T})$ is a negative subset-disjoint cycle if each node in it belongs to a different subtree of **T** and its cost is negative. There exist several algorithms to identify negative (directed) cycles, but finding a negative subset-disjoint cycle is an NP-hard problem (Thompson and Orlin [1989]). Accordingly, we developed a heuristic algorithm to find negative cost subset-disjoint cycle. We now describe this heuristic algorithm.

Our heuristic algorithm is a simple modification of the well known label-correcting algorithm for the shortest path problem. We used the dequeue implementation of the label

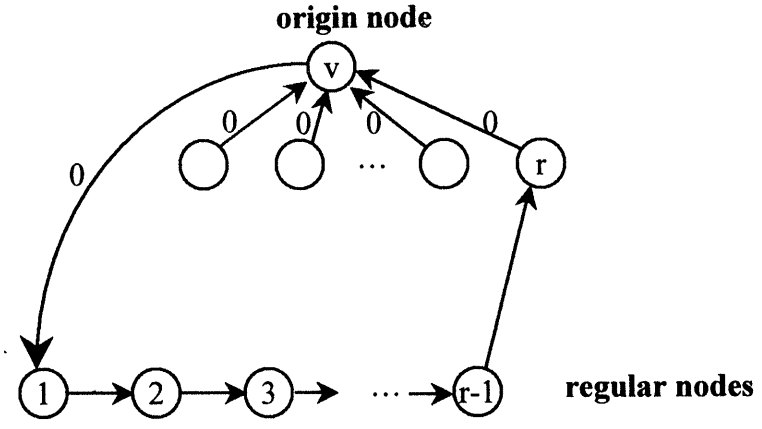


Figure 6. Illustrating the augmentation of the improvement graph to handle root-disjoint paths.

correcting algorithm (also known as Pape's [1980] algorithm) which is considered to be very efficient in practice. A label-correcting algorithm determines a shortest path from a specified node s to every other node in the network with arc costs given by α_{ij}'s. A label-correcting algorithm maintains two indices with each node j: d(j), the *distance label* of the node j, and pred(j), the *predecessor index* of node j. The distance label d(j) is either ∞, indicating that the algorithm has yet to discover a directed path from node s to node j, or it is the length of some directed path from the source to node j. The predecessor index, pred(j), records the node prior to node j in the current directed path of length d(j). The predecessor indices allow us to trace the current shortest path from the node j back to the node s. Let P[j] denote the current directed path from node s to node j. The optimality conditions for the shortest path problem require that $d(j) \leq d(i) + \alpha_{ij}$ for each arc (i, j) in the network. The basic step in a label-correcting algorithm is to identify an arc (i, j) violating its optimality condition, that is, $d(j) > d(i) + \alpha_{ij}$, and decrease the distance label d(j) to $d(i) + \alpha_{ij}$; this step is called the *distance update* step. The algorithm repeatedly performs distance update steps and terminates when all the arcs satisfy their optimality conditions. To identify an arc (i, j) violating its optimality condition efficiently, the algorithm maintains a list, LIST, of nodes with the property that if an arc (i, j) violates its optimality conditions then LIST must contain node i. At each iteration, the algorithm selects a node i from LIST, removes it from LIST, and examines it by performing a distance update step for all arcs emanating from node i.

To identify negative-cost subset-disjoint cycles in the improvement graph, we make just one modification in the label-correcting algorithm. We require that the current directed path P[j] from node s to each node j is a subset-disjoint path, that is, the nodes in the path P[j] belong to different rooted subtrees. Though this requirement is difficult to enforce for all nodes and at all steps of the algorithm, we try to enforce it as much as we can. While executing the label-correcting algorithm, whenever we remove a node i from LIST, we check whether its current path P[i] is a subset-disjoint path, that is, all nodes in it belong to different subtrees of **T**. If not, then we do not examine node i; otherwise we set d(i) equal to the length of the path P[i] and examine arcs emanating from it one by one. While examining the arc (i, j), we check whether $d(j) \leq d(i) + \alpha_{ij}$. If this condition is satisfied, we skip the arc (i, j) and examine another arc emanating from node i. If $d(j) > d(i) + \alpha_{ij}$, then there are three cases to consider:

Case 1. *Node j is a pseudonode.* If node j is a pseudonode, then arc (i, j) signifies that node i moves from the subtree represented by the pseudonode j. In this case, we check whether $d(i) + \alpha_{ij} < 0$, and if yes then we have discovered a profitable path exchange. To see this, recall from

our earlier discussion that each pseudonode j has a directed arc (j, v) of zero cost going into the origin node v and the origin node v has a zero cost arc (v, s) going into node s. Hence P[i]$\cup${(j, v)}$\cup${(v, s)} is a directed cycle. Further, since d(j) > d(i) + α_{ij}, the cost of this cycle is negative.

Case 2. *Node j is a regular node and P[i]$\cup${(i, j)} is a subset-disjoint path.* We update d(j) := d(i) + α_{ij} and set pred(j) = i. In this case, we simply extend the subset-disjoint path to node j by adding the arc (i, j) to the path to node i.

Case 3. *Node j is a regular node and P[i]$\cup${(i, j)} is not a subset-disjoint path.* We check whether node j $\in$ P[i]. This case will occur when node j belongs to a subset which has already been visited (that is, node j belongs to one of the subsets containing nodes in P[i]). It is also possible that we have visited node j itself earlier (that is, j $\in$ P[j]), in which case we have discovered a subset-disjoint cycle. Since d(j) > d(i) + α_{ij}, it is easy to see that this cycle has a negative cost.

We are now in a position to describe our method to identify negative subset-disjoint cycles. We apply the modified label-correcting algorithm, as described above, with some regular node as node s. The modified label-correcting algorithm during its execution can discover several profitable cyclic or path exchanges or none of them. We can quit the modified label-correcting algorithm when it finds the first profitable exchange, or let it run to completion and keep track of the most profitable exchange.

Our empirical investigations revealed that the success of the modified label-correcting algorithm in finding profitable exchanges depends upon which regular node is used as node s. If we apply it just once with some regular node as node s, we miss many profitable exchanges. We thus applied the modified label-correcting algorithm multiple times with different nodes as the starting nodes, once for each regular node. The running time of our method for finding profitable exchanges is n times the time taken by a single execution of the modified label-correcting algorithm. Our modifications to the label-correcting algorithm worsen its running time by at most a factor of L (the number of subtrees in **T**), but in practice, the increase in the running time is much less.

5. TREE-BASED NEIGHBORHOOD STRUCTURE

In this section, we describe another neighborhood structure for the capacitated minimum spanning tree problem. It is a variation of the node-based neighborhood structure described in the

previous section and can be thought of as a generalization of the neighborhood structure of Sharaiha et al., briefly described in Section 3. Whereas our node-based neighborhood structure uses cyclic or path exchange of single nodes, our tree-based neighborhood structure uses cyclic or path exchange of parts of the subtrees (often containing more than one node). The parts of subtrees that the tree-based neighborhood structure moves around are also trees; we call them *subsubtrees* to distinguish them from rooted subtrees.

As earlier, let **T** be a feasible tree with **T**[1], **T**[2], ..., **T**[L] as its subtrees. Recall that we denote by **T**[i] the subsubtree rooted at node i and by **S**[i] the set of nodes contained in **T**[i]. Similar to our first neighborhood structure, we allow two types of exchanges - cyclic exchanges and path exchanges. As earlier, we represent a cyclic exchange as i_1-i_2- ...-i_r-i_1, where the nodes i_1, i_2, ..., i_r belong to different subtrees. Then the cyclic exchange with respect to **T** in the tree-based neighborhood structure represents the following changes - nodes in the subsubtree T_{i_1} move from the subtree **T**[i_1] to the subtree **T**[i_2], nodes in the subsubtree T_{i_2} move from the subtree **T**[i_2] to the subtree **T**[i_3], and so on, and finally nodes in the subsubtree T_{i_r} move from the subtree **T**[i_r] to the subtree **T**[i_1]. We call such a cyclic exchange *feasible* if the changed tree satisfies the capacity constraints. In other words, the cyclic exchange i_1-i_2- ...-i_r-i_1 is a feasible exchange if and only if

$$\sum_{k \in \mathbf{T}[i_p]} d_k + \sum_{k \in T_{i_{p-1}}} d_k - \sum_{k \in T_{i_p}} d_k \leq K, \text{ for every } p = 1, 2, \ldots, r, \qquad (6)$$

where $T_{i_0} = T_{i_r}$. Let $\mathbf{T}'$ denote the new feasible capacitated tree obtained after this cyclic exchange. We define the cost of this cyclic exchange as the change in the cost of the tree, which is, $c(\mathbf{T}') - c(\mathbf{T})$. Observe that

$$c(\mathbf{T}') - c(\mathbf{T}) = \sum_{p=1}^{r} c\left(S_{i_{p-1}} \cup \mathbf{S}[p] \setminus S_{i_p}\right) - \sum_{p=1}^{r} c(\mathbf{S}[p]), \qquad (7)$$

where $S_{i_0} = S_{i_r}$. We call this cyclic exchange *profitable* if $c(\mathbf{T}') < c(\mathbf{T})$ and *non-profitable* otherwise. We can define a path exchange i_1-i_2- ...-i_r similar to the cyclic exchange, with the difference that no subsubtree moves out of the subtree T_{i_r}. We call a path exchange *feasible* if the modified tree after the path exchange is feasible, and call it *profitable* if the cost of the modified tree strictly decreases.

For a given tree **T**, we call another tree **T'** a *neighbor* to **T** if **T'** can be obtained from **T** by performing a cyclic or path exchange. We define the *neighborhood* of **T** as a collection of all neighbors of **T**. We point out that the neighborhood of **T** in the tree-based neighborhood structure is different from the neighborhood of **T** in the node-based neighborhood structure. The tree-based neighborhood structure cannot move just the non-leaf nodes to other subtrees and the node-based neighborhood structure cannot move more than one node.

Similar to the case of node-based neighborhood, we can define an improvement graph $\mathbf{G^2(T)}$ so that each cyclic or path exchange defines a subset-disjoint cycle in $\mathbf{G^2(T)}$ and the converse result is also true. The structure of the graph $\mathbf{G^2(T)}$ is similar to that of the graph $\mathbf{G^1(T)}$. An arc (i, j) in $\mathbf{G^2(T)}$ with $\mathbf{T}[i] \neq \mathbf{T}[j]$ signifies that the node subset S_i leaves the subtree $\mathbf{T}[i]$ and joins the subtree $\mathbf{T}[j]$ and simultaneously the node subset S_j leaves the subtree $\mathbf{T}[j]$. Hence we add the arc (i, j) to $\mathbf{G^2(T)}$ if and only if $\sum_{k \in \mathbf{S}[j]} d_k + \sum_{k \in S_i} d_k - \sum_{k \in S_j} d_k \leq K$. We define the cost of the arc (i, j) as

$$\beta_{ij} = c(S_i \cup \mathbf{S}[j] \setminus S_j) - c(\mathbf{S}[j]). \tag{8}$$

In other words, β_{ij} represents the cost of deleting the node set S_j from the subtree T_j and adding S_i to it. Using arguments similar to those we used in the node-based neighborhood structure, it can be shown that there is a one-to-one cost-preserving correspondence between cyclic exchanges with respect to **T** and subset-disjoint cycles in $\mathbf{G^2(T)}$.

We next consider the path exchanges with respect to the feasible tree **T**. Using the same method we used for the node-based neighborhood structure, we augment the improvement graph by adding a pseudonode for each subtree in **T** and an origin node v, and some arcs so that there is a one-to-one correspondence between path exchanges with respect to **T** and subset-disjoint cycles in $\mathbf{G^2(T)}$ passing through the origin node, and both have the same cost. The only difference is in the manner in which we define the cost of the arc (i, h) when h is a pseudonode, and the cost of the arc (v, i) where v is the origin node. In this case we define them as

$$\beta_{ih} = c(\mathbf{S}[h] \cup S_i) - c(\mathbf{S}[h]), \tag{9a}$$

$$\beta_{vi} = c(\mathbf{S}[i] \setminus S_i). \tag{9b}$$

where β_{ih} is the cost of adding S_i to $\mathbf{T}[h]$ and β_{vi} is the cost of removing the subsubtree T_i from $\mathbf{T}[i]$. These transformations convert a profitable cyclic or path exchange into a negative subset-

disjoint cycle in the improvement graph $\mathbf{G^2(T)}$. We can find negative coat subset-disjoint cycles in $\mathbf{G^2(T)}$ using exactly the same method we described in Section 4.

Determing β_{ij}'s

We next describe how to compute β_{ij}'s for all arcs in the improvement graph $\mathbf{G^2(T)}$. We use the expression (8) to compute β_{ij}'s. The second term in (8) is already known. We next consider the first term in (8). This term involves computing a minimum spanning tree over the node set $S_i \cup \mathbf{S}[j] \backslash S_j$. To do so, we delete the subsubtree T_j rooted at node j; this gives us a minimum spanning tree $\mathbf{T}[j] \backslash T_j$ over the node set $\mathbf{S}[j] \backslash S_j$. We next add to $\mathbf{T}[j] \backslash T_j$ the node set S_i and all arcs between S_i and $\mathbf{S}[j] \backslash S_j$ and solve a minimum spanning tree problem; this gives us a minimum spanning tree problem over the node set $S_i \cup \mathbf{S}[j] \backslash S_j$. In case all nodes have unit demands, we solve the minimum spanning tree problem over a subgraph with K nodes and K^2 arcs using Prim's [1957] algorithm and it takes $O(K^2)$ time. We thus take $O(K^2)$ time to compute a single β_{ij} and $O(n^2K^2)$ time to compute all β_{ij}'s.

6. NEIGHBORHOOD SEARCH ALGORITHMS

To determine the effectiveness of the two neighborhood structures developed in this paper, we used them in neighborhood search algorithms. We implemented local improvement algorithms and tabu search algorithms. We implemented efficient and straightforward versions of these algorithms and expect that there are opportunities for further enhancements of algorithms. All our neighborhood search algorithms start with a feasible solution $\mathbf{T}$ which is subsequently improved upon. Further, a neighborhood search algorithm is typically applied many times starting at different feasible solutions. We thus need an algorithm which can generate different starting solutions. In this section, we describe the details of the algorithm used to generate different feasible solutions, and also of our neighborhood search algorithms.

Determining Starting Solutions

A neighborhood search algorithm is often sensitive to its starting solution - better the starting solution, the faster is the convergence and, often, the quality of the local solution is better. In addition, a neighborhood search is typically applied many times starting with different feasible solutions. We thus need a mechanism to generate multiple good feasible solutions of the capacitated spanning tree problem. In our empirical investigations we have used a randomized

version of the Esau-William's algorithm (see, Esau and William [1966]), which is perhaps the most popular construction based heuristic algorithm for the capacitated minimum spanning tree problem. The Esau-William algorithm starts with each subtree containing a singleton node. In each iteration, the algorithm joins two subtrees into a single subtree so that the new subtree satisfies the capacity constraints and the savings achieved by the join operation are maximum. In our version of Esau-William algorithm, at each iteration we determine the p most profitable join operations for some small value p. We then generate an integer random number k uniformly distributed between 1 to p and perform the k^{th} most profitable join operation. This method in general provides a new feasible tree each time it is applied. Since at each step it performs one of the p most profitable join operations, the feasible tree obtained is generally a good tree. In our investigations, we used p = 3.

Local Improvement Algorithms

A local improvement algorithm is perhaps the simplest neighborhood search algorithm. Our local improvement algorithm for the capacitated minimum spanning tree problem starts with a feasible solution **T** obtained through the randomised Esau-William algorithm. Each subtree in the starting solution **T** is converted into a minimum spanning tree over the subset of nodes it contains. The local improvement algorithm checks if there exists a profitable exchange with respect to **T**. If yes, it performs the exchange, obtains a new solution, and replaces **T** by the new improved solution. This process is repeated until there is no profitable exchange with respect to **T**. At this point, the solution **T** is a locally optimal solution. This completes one run of the local improvement algorithm. In the next run, we start with another feasible solution of the capacitated spanning tree problem obtained again through the randomised Esau-William algorithm and convert it into a locally optimal solution by performing a sequence of profitable exchanges. We continue in this manner until either we have performed enough runs or the total time taken by the algorithm has reached a specified upper bound.

We implemented local improvement algorithm using both the node-based and tree-based neighborhood structures. We refer to the local improvement algorithm with the node-based neighborhood structure as the algorithm "IMP1" and the local improvement algorithm with the tree-based neighborhood structure as the algorithm "IMP2". In both algorithms, we identify profitable exchanges by identifying negative subset-disjoint cycles in the improvement graph. We had the option to improve the current solution using the first profitable exchange found or using the most profitable exchange in the neighborhood. In both local search algorithms, we improved the current solution using the first profitable exchange identified. We also specified an

upper bound on the total time and allowed the algorithms to perform as many runs as possible within the specified time.

Tabu Search Algorithms

Tabu search is a more sophisticated neighborhood search algorithm and has been very successful in solving difficult combinatorial optimization problems. Amberg et al. [1996] and Sharaiha et al. [1997] developed tabu search algorithms, which are currently the two best available neighborhood search algorithms to solve the capacitated minimum spanning tree problem. We refer the reader to the survey papers of Glover [1989, 1990] and the book by Glover and Laguna [1997] for additional material on tabu search algorithms.

Our tabu search algorithm for the capacitated minimum spanning tree problem uses only the "short term memory". It starts with a feasible solution **T** obtained through randomized Esau-William algorithm. It checks if there is a profitable exchange with respect to **T** that honors the tabu rules. If yes, it performs it; otherwise, it selects the least costly interchange that it has found using the heuristic described in Section 5 and which is not currently on the tabu list. It performs this interchange and the reversal of the applied exchange is added to the tabu list unless this exchange improved the best solution over all previous iterations. Our tabu criteria is the following: if the exchange involves the nodes $i_1, i_2, \ldots, i_r$ then we add each one of them to a tabu list for some time; that is, we do not allow the nodes $i_1, i_2, \ldots,$ or i_r to be part of any exchange for a certain number of iterations, called *tabu time*. We used a tabu time of 10 for problems less than 100 nodes. For larger size problems, we set the tabu time to be 10% of the number of nodes. For our tabu search algorithms, we set an upper limit of 50 or 100 on the number of such iterations depending upon the problem size. When the tabu search algorithm terminates in this manner it completes one run of the algorithm. In the next run, we start with another feasible solution of the capacitated spanning tree problem obtained again through the randomized Esau-William algorithm and apply the tabu search algorithm again. We continue in this manner until either we have performed enough runs or the total time taken by the algorithm has reached a specified upper bound. We implemented the tabu search algorithm using both the node-based and tree-based neighborhood structures. We refer to the tabu search algorithm using the node-based neighborhood structure as the algorithm "TABU1" and the tabu search algorithm using the tree-based neighborhood structure as the algorithm "TABU2". In both algorithms, we specified an upper bound on the total time and allowed the algorithms to perform as many runs as possible within the specified time limit.

7. COMPUTATIONAL RESULTS

In this section, we present preliminary computational results of our neighborhood search algorithms using the two neighborhood structures suggested in this paper. We wrote computer programs for our algorithms in C programming language, and tested them on the DEC alpha computer at the University of Massachusetts, Amherst. We tested our algorithms on four classes of benchmark problems given below, of which the first three problem classes are available at the website with URL: http://www.ms.ic.ac.uk/info.html. We developed the fourth problem class ourselves. Some details of these problem classes are given below:

tc problem class: Randomly generated problems; nodes in a square grid; all pairs of nodes are connected; arc costs between different nodes are linearly related to the Euclidean distance between the nodes; two problem sizes: 40 and 80 nodes; K = 5, 10, and 20; and the root node is in the center of the grid. There are 35 problems in this class.

te problem class: Same as the tc problem class except that the root node is in the corner of the grid instead of the center. This problem class is considered to be somewhat harder than the tc problem class. There are 25 problems in this class.

cm problem class: Randomly generated problems; nodes have heterogeneous demands which varies from 0 to 100; K = 200, 400, and 800; problem sizes: 50, 100 and 200 nodes. There are 45 problems in this class.

aos problem class: Randomly generated problems; problem sizes 100, 200, 300, 400, and 500; half the problems have unit demands and the other half have heterogeneous demands; half the problems have K = 200 and the other half have K = 400. There are 40 problems in this class.

We applied all the four algorithms, IMP1, IMP2, TABU1, and TABU2, to all the above benchmark problems. We applied our algorithms for 100 seconds for problems with fewer than 50 nodes, for 200 seconds for problems with at least 50 nodes and at most 100 nodes, for 500 seconds for problems with at least 100 nodes. We compared the best objective function value obtained by these algorithms with the best previously available objective function values; we obtained these values from Luis Gouveia. For the aos problem class, no best known solutions were available; for these problems we noted the improvements over the solutions obtained by the Esau-William's method. We give in Tables 1, 2, and 3, the percent deviations from of the best known solutions for all of our four algorithms for the three problem sets, tc class, te class, and cm class, respectively. For all problem classes, tabu search algorithms obtains better results than

local improvement algorithms. We also plot the percent deviations of the tabu search algorithms for the tc, te, and cm classes in Figures 7, 8, and 9. Our principle findings are the following:

1. For the tc and te problem classes, TABU1, the tabu search algorithm using the node-based neighborhood structure, obtains the best available objective function value for all problems, and improves the solution values of three instances.

2. For the cm problem class, TABU2, the tabu search algorithm using the tree-based neighborhood search algorithm, improves almost all the previous best known solutions. The average improvement is around 3% and the maximum improvement is 18%.

3. We find that the two neighborhood structures have different strengths. The node-based neighborhood structure is quite effective for solving unit demand problems, and the tree-based neighborhood structure is very effective in solving heterogeneous demand problems.

We have the following observations that may partially explain to justify the relative performance of the two tabu search algorithms. For the unit-demand case, exchanges of two nodes i and j belonging to two rooted subtrees are always possible but exchanges of two subsubtrees T_i and T_j may not be possible because the subsubtrees may contain a different number of nodes. Consequently, the neighborhood for TABU1 is generally a larger neighborhood than that of TABU2. Perhaps this explains in part why TABU1 performs better than TABU2. For the heterogeneous demand case, exchanges of two nodes are not always possible because nodes different demands and subtrees may not have enough spare capacity. In order to add one node to a subtree to T_i (say, of large demand), we may have to remove several nodes of smaller capacity from T_i. The neighborhood of TABU2 admits these possibilities. Perhaps this explains in part why TABU2 performs better than TABU1 for the heterogeneous case.

In Figure 10, we report our results on the aos problems with unit demands. We plot the percent improvements obtained by our two tabu search algorithms over the solutions obtained by the Esau-William's algorithm. In the figure, DEC denotes problems where arc costs are proportional to the Euclidean distance between the two endpoints of the arc, and SDEC denotes problems where arc costs are proportional to the square of the Euclidean distances. We find a marked difference in the improvements for the Euclidean and the squared Euclidean cases. It appears that the Euclidean case is relatively easy to solve and these exist a large number of solutions that are not too far from the optimal solution and it is easy to find such a solution. In the squared Euclidean case, it appears much more difficult to get close to the optimal solution.

Table 1. Table of results of algorithms on unit-demand problems with central depot in the center.

Problem-ID	Best Known	percentage deviation from best known solutions					New Best Known
		EW	IMP1	TABU1	IMP2	TABU2	
n=40K=10tc-1	498	1.606	0.000	0.000	0.000	0.000	498
n=40K=10tc-2	490	2.857	0.000	0.000	0.000	0.000	490
n=40K=10tc-3	500	4.800	1.600	0.000	0.000	0.000	500
n=40K=10tc-4	512	3.516	0.000	0.000	0.000	0.000	512
n=40K=10tc-5	504	0.000	0.000	0.000	0.000	0.000	504
n=40K=10tc-6	498	0.803	0.000	0.000	0.000	0.000	498
n=40K=10tc-7	508	0.787	0.000	0.000	0.000	0.000	508
n=40K=10tc-8	485	0.000	0.000	0.000	0.000	0.000	485
n=40K=10tc-9	516	0.000	0.000	0.000	0.000	0.000	516
n=40K=10tc-10	517	1.934	0.000	0.000	0.000	0.000	517
n=40K=5tc-1	586	1.877	0.000	0.000	0.000	0.000	586
n=40K=5tc-2	578	1.730	0.000	0.000	0.000	0.000	578
n=40K=5tc-3	577	5.373	0.000	0.000	0.000	0.000	577
n=40K=5tc-4	617	3.566	0.000	0.000	0.000	0.000	617
n=40K=5tc-5	600	2.333	0.000	0.000	0.500	0.833	600
n=40K=5tc-6	590	1.525	0.000	0.000	0.000	0.000	590
n=40K=5tc-7	609	1.970	0.000	0.000	0.000	0.000	609
n=40K=5tc-8	553	2.170	0.000	0.000	0.181	0.000	553
n=40K=5tc-9	599	8.681	0.000	0.000	2.671	0.000	599
n=40K=5tc-10	600	5.500	0.000	0.000	0.000	0.000	600
n=80K=10tc-1	888	7.095	0.000	0.000	0.676	0.450	888
n=80K=10tc-2	877	5.245	0.000	0.000	1.368	0.000	877
n=80K=10tc-3	880	3.864	0.227	0.000	0.682	0.000	880
n=80K=10tc-4	868	6.106	0.000	0.000	0.922	0.461	868
n=80K=10tc-5	1002	2.695	0.000	0.000	0.000	0.000	1002
n=80K=20tc-1	834	0.480	0.000	0.000	0.000	0.000	834
n=80K=20tc-2	820	3.659	0.000	0.000	0.000	0.000	820
n=80K=20tc-3	828	1.449	0.000	0.000	0.000	0.000	828
n=80K=20tc-4	820	1.220	0.000	0.000	0.000	0.000	820
n=80K=20tc-5	916	3.493	1.856	0.000	0.000	0.000	916
n=80K=5tc-1	1099	6.187	0.091	0.000	3.549	2.639	1099
n=80K=5tc-2	1100	6.364	0.182	0.000	3.091	2.545	1100
n=80K=5tc-3	1073	6.151	0.000	0.000	2.050	1.771	1073
n=80K=5tc-4	1080	6.852	0.000	0.000	2.963	1.944	1080
n=80K=5tc-5	1287	5.905	0.078	0.000	1.943	1.632	1287
Average deviation		3.366	0.115	0.000	0.588	0.351	

Table 2. Table of results of algorithms for unit-demand problems with central depot in the corner

Problem-ID	Best Known	percentage deviation from best known solutions					New Best Known
		EW	IMP1	TABU1	IMP2	TABU2	
n=40c=10te-1	596	7.215	0	0	0.671	0	596
n=40c=10te-2	573	8.551	0	0	0	0	573
n=40c=10te-3	568	7.746	0	0	0.176	0	568
n=40c=10te-4	596	0.671	0	0	0	0	596
n=40c=10te-5	572	3.671	0.35	0	0.35	0.35	572
n=40c=5te-1	830	3.976	0	0	1.807	0.964	830
n=40c=5te-2	792	5.177	0	0	1.01	0.379	792
n=40c=5te-3	797	4.642	0	0	1.255	1.757	797
n=40c=5te-4	814	6.265	0	0	0.246	1.106	814
n=40c=5te-5	784	4.464	0	0	0.383	0.383	784
n=80c=10te-1	1651	3.937	2.023	0	2.698	3.25	1651
n=80c=10te-2	1643	5.904	0.061	**-0.243**	0.365	1.096	**1639**
n=80c=10te-3	1688	5.391	0.237	**-0.059**	1.066	0.77	**1687**
n=80c=10te-4	1629	8.840	0.368	0	2.149	1.842	1629
n=80c=10te-5	1603	6.550	0.561	0	1.123	1.185	1603
n=80c=20te-1	1275	2.588	0	0	0.235	0.392	1275
n=80c=20te-2	1225	6.122	**-0.082**	0	0.245	0.408	**1224**
n=80c=20te-3	1267	5.919	1.342	0	0	0	1267
n=80c=20te-4	1265	8.379	0	0	0.711	0	1265
n=80c=20te-5	1240	4.032	0	0	0.081	0.081	1240
n=80c=5te-1	2544	3.223	0	0	0.354	0.432	2544
n=80c=5te-2	2551	1.960	0.196	0	0.706	0.862	2551
n=80c=5te-3	2612	2.489	0	0	1.417	1.531	2612
n=80c=5te-4	2558	2.893	0.156	0	1.212	1.407	2558
n=80c=5te-5	2469	1.620	0	0	0.324	0.486	2469
Average deviation		4.889	0.208	-0.012	0.743	0.747	

Table 3. Table of results of algorithms on the heterogeneous demand problems.

Problem-ID	Best Known	percentage deviation from best known solutions					New Best Known
		EW	IMP1	TABU1	IMP2	TABU2	
CM50R1K200	1135	0.617	**-3.26**	**-3.26**	-2.996	-2.996	**1098**
CM50R2K200	1023	-2.151	**-4.79**	**-4.79**	-3.812	-4.203	**974**
CM50R3K200	1229	1.627	-2.929	**-3.499**	-1.709	-1.465	**1186**
CM50R4K200	811	2.219	**-1.356**	**-1.356**	-1.356	-1.233	**800**
CM50R5K200	970	-1.753	-4.021	**-4.33**	-4.124	-4.33	**928**
CM50R1K400	726	-1.102	**-6.198**	**-6.198**	-4.821	-5.096	**681**
CM50R2K400	642	2.492	**-1.713**	-1.09	-1.558	-0.467	**631**
CM50R3K400	741	-0.270	**-0.81**	**-0.81**	**-0.81**	**-0.81**	**735**
CM50R4K400	583	-2.058	-2.573	**-2.744**	**-2.744**	-2.573	**567**
CM50R5K400	628	1.752	**-2.707**	**-2.548**	-2.548	-2.548	**612**
CM50R1K800	544	-4.779	-8.64	**-9.007**	**-9.007**	**-9.007**	**495**
CM50R2K800	531	2.825	-2.825	**-3.013**	**-3.013**	**-3.013**	**515**
CM50R3K800	554	1.083	-2.527	**-3.971**	-3.249	-3.43	**532**
CM50R4K800	472	4.025	0.636	0.636	0.636	**-0.212**	**471**
CM50R5K800	501	2.395	-1.397	-1.397	**-1.796**	-1.397	**492**
CM100R1K200	551	21.960	-2.904	-4.174	-4.537	**-5.626**	**520**
CM100R2K200	616	28.409	-1.461	-2.11	-1.786	**-2.273**	**602**
CM100R3K200	608	20.888	-5.428	-6.579	-9.211	**-9.704**	**549**
CM100R4K200	445	21.798	3.371	3.146	**-0.225**	**-0.225**	**444**
CM100R5K200	442	33.258	2.941	1.131	-0.226	**-3.394**	**427**
CM100R1K400	259	17.761	4.633	3.861	-1.544	**-2.317**	**253**
CM100R2K400	278	24.460	2.878	3.597	0	**0**	278
CM100R3K400	238	21.429	4.622	2.101	**-0.84**	-0.42	**236**
CM100R4K400	223	20.628	4.036	4.933	**-1.794**	**-1.794**	**219**
CM100R5K400	227	22.907	4.405	2.643	**-1.322**	-0.881	**224**
CM100R1K800	182	18.132	2.198	4.396	**0**	**0**	182
CM100R2K800	179	15.084	1.117	2.235	**0**	**0**	179
CM100R3K800	175	14.286	3.429	4	**0**	**0**	175
CM100R4K800	183	8.197	2.732	3.825	**0**	**0**	183
CM100R5K800	187	9.091	2.674	2.139	**-0.535**	**-0.535**	**186**
CM200R1K200	1147	16.739	-8.195	-8.457	-8.893	**-9.59**	**1037**
CM200R2K200	1505	9.834	-15.748	-15.814	**-18.272**	**-18.272**	**1230**
CM200R3K200	1464	20.287	-3.62	-2.527	-5.943	**-6.626**	**1367**
CM200R4K200	1017	23.206	-2.95	-2.852	-5.9	**-7.375**	**942**
CM200R5K200	1145	18.777	-13.188	-10.218	-13.712	**-14.323**	**981**
CM200R1K400	421	16.627	3.325	1.663	-4.276	**-5.226**	**399**
CM200R2K400	498	31.526	4.217	8.835	-1.807	**-2.41**	**486**
CM200R3K400	587	22.828	1.874	2.044	-3.237	**-3.578**	**566**
CM200R4K400	404	23.020	5.198	4.95	**-1.733**	-1.485	**397**
CM200R5K400	442	16.290	1.81	0.452	-3.394	**-3.846**	**425**
CM200R1K800	256	25.781	4.688	6.25	0.391	**0**	256
CM200R2K800	296	11.149	4.73	3.716	0.338	**-0.676**	**294**
CM200R3K800	362	8.564	3.315	3.315	0	**0**	362
CM200R4K800	276	15.217	5.797	7.971	0	**0**	276
CM200R5K800	295	10.508	4.746	6.102	-0.339	**-0.678**	**293**
Average deviation		12.790	-0.442	-0.373	-2.927	-3.201	

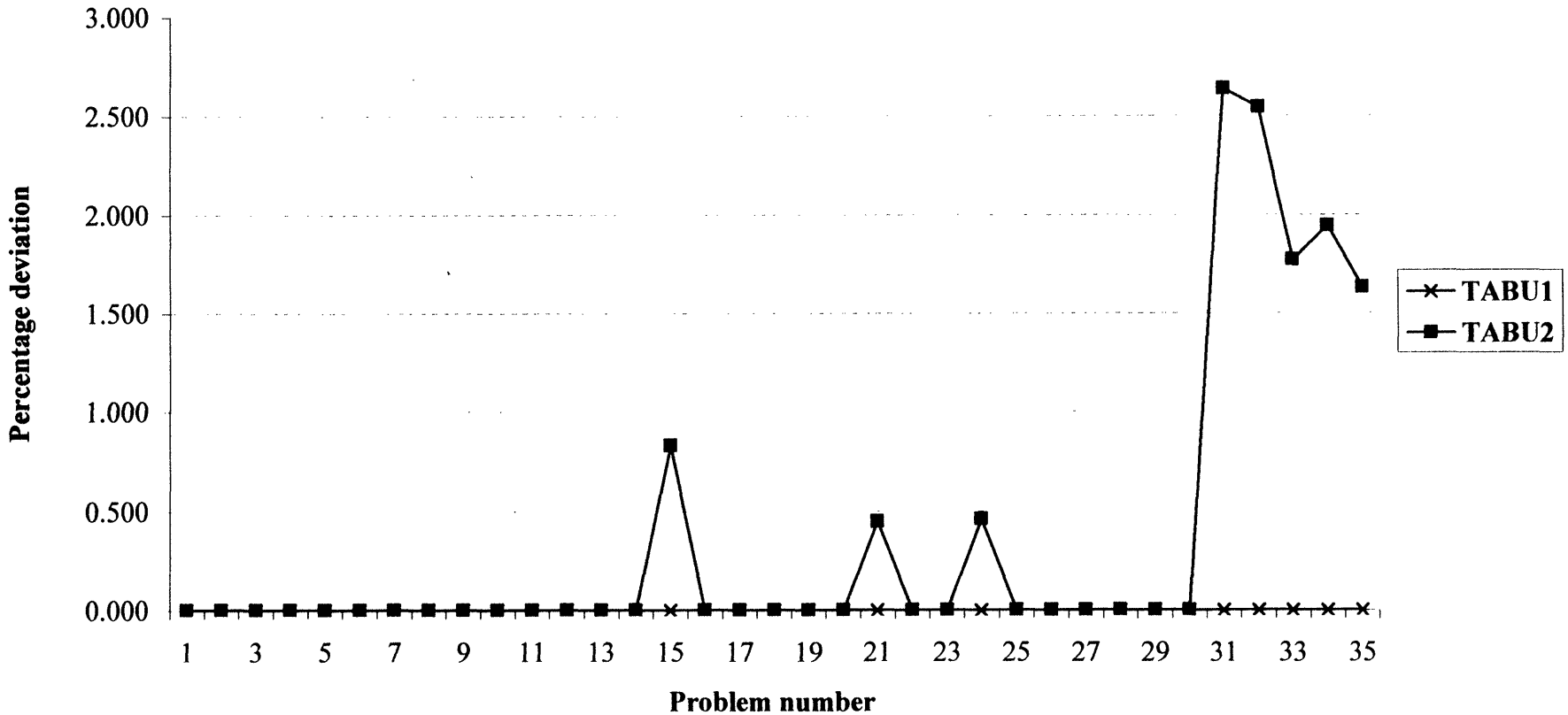


Figure 7. Comparison of algorithms for unit-demand problems with central depot in the center.

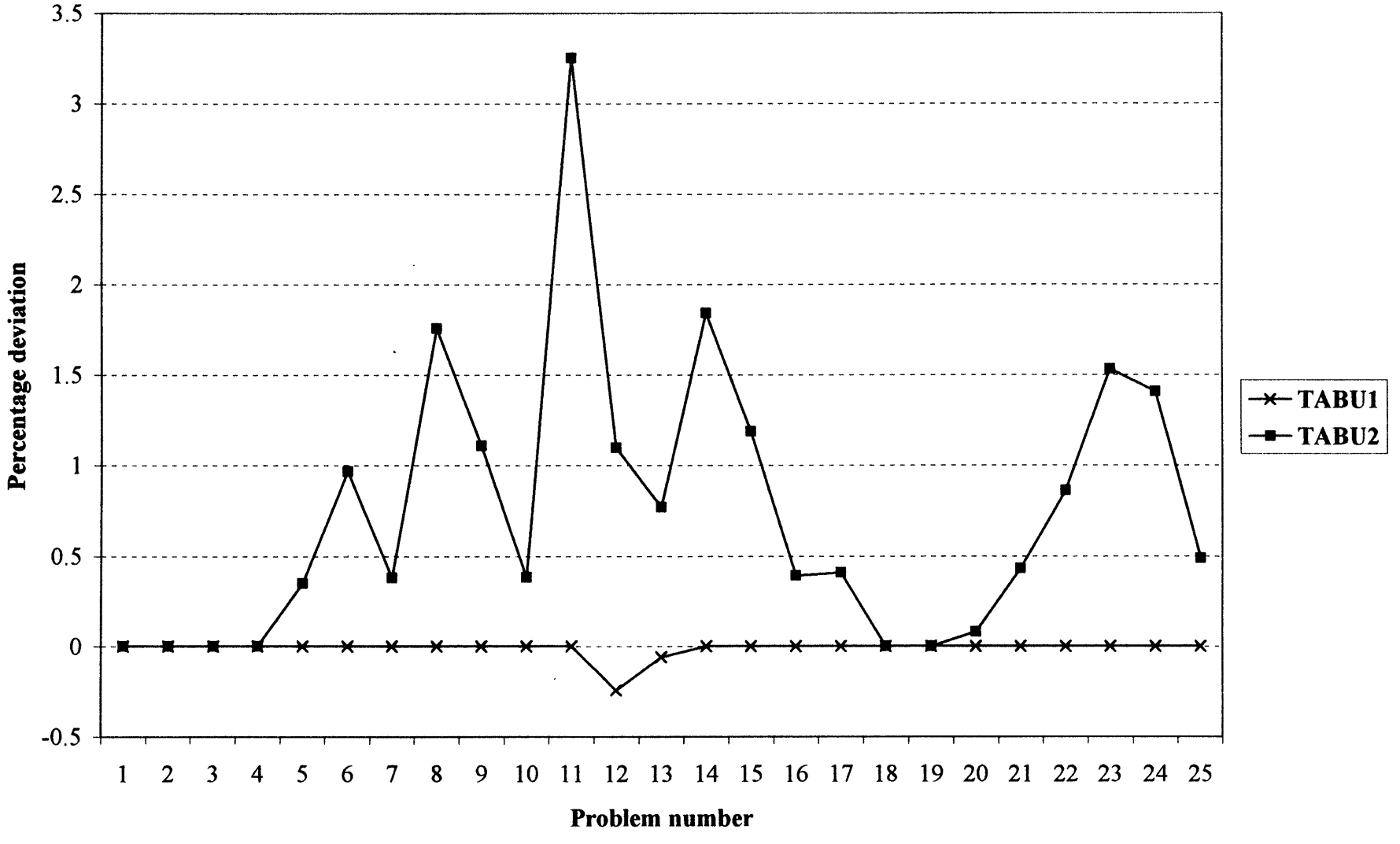


Figure 8. Comparison of algorithms on problems with central depot in the corner.

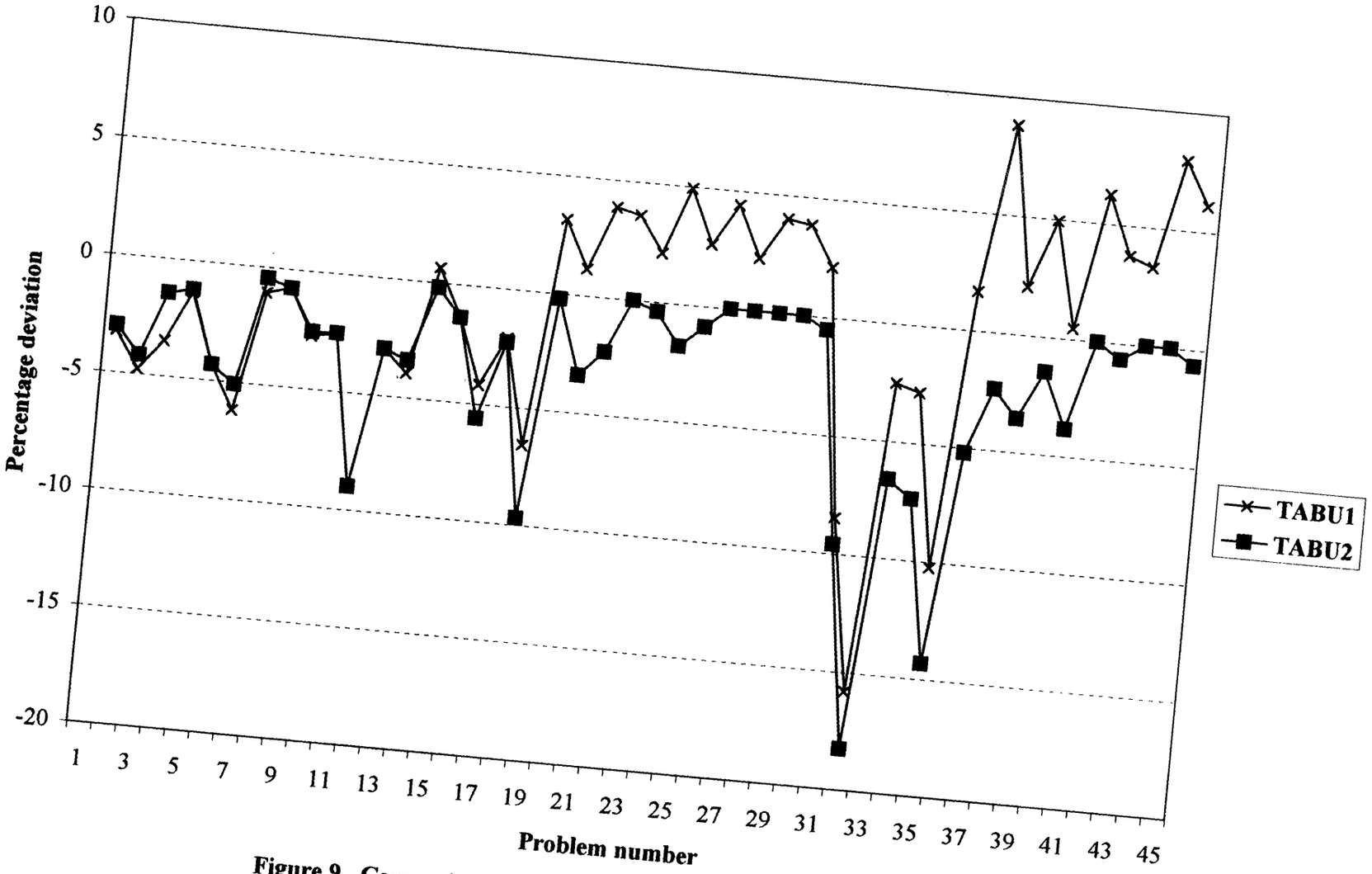


Figure 9. Comparison of algorithms on hetergeneous demand problems.

Figure 10. Comparison of tabu search procedures on node and tree based neighborhood structures for unit-demand problems, K=10

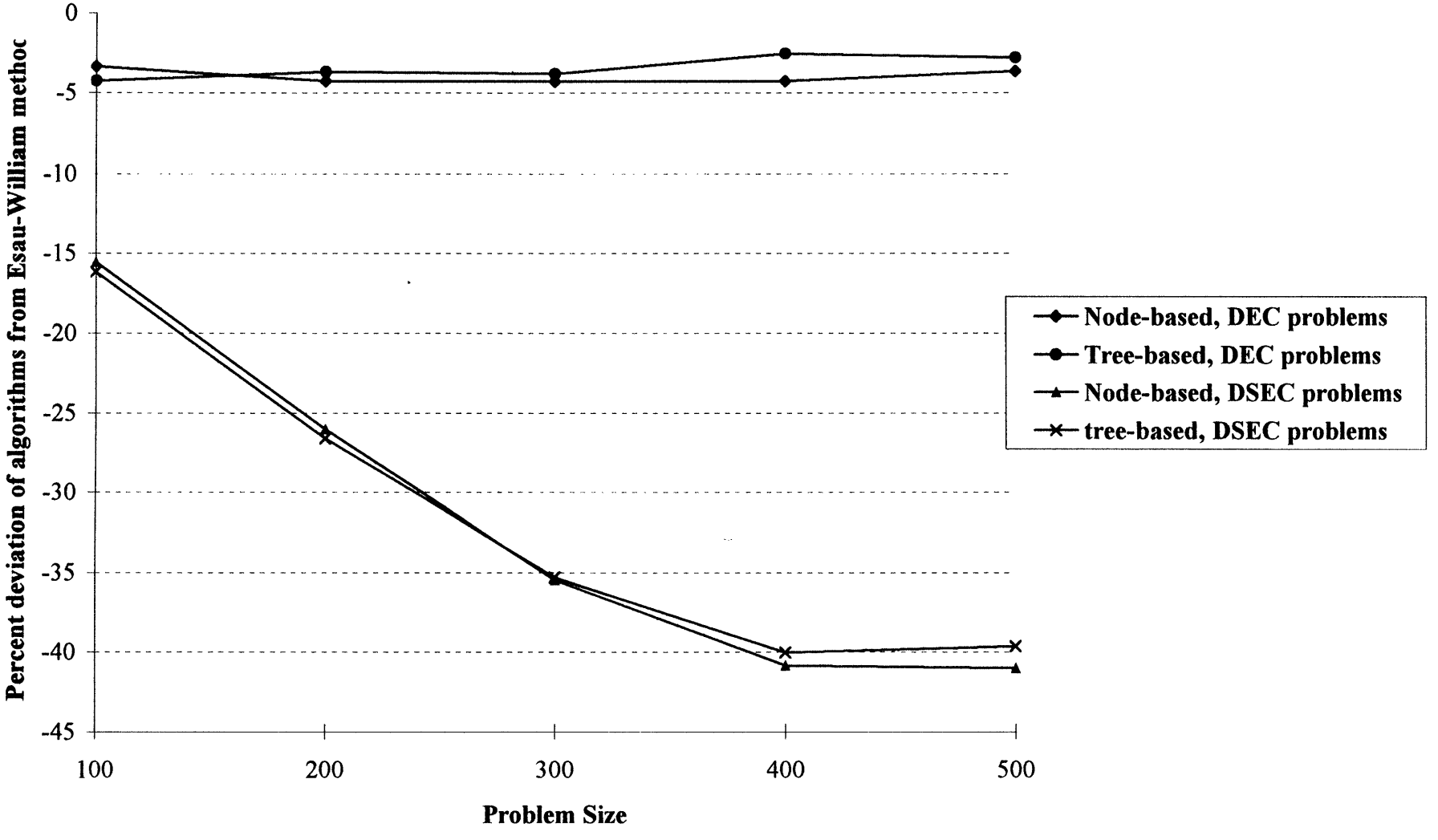
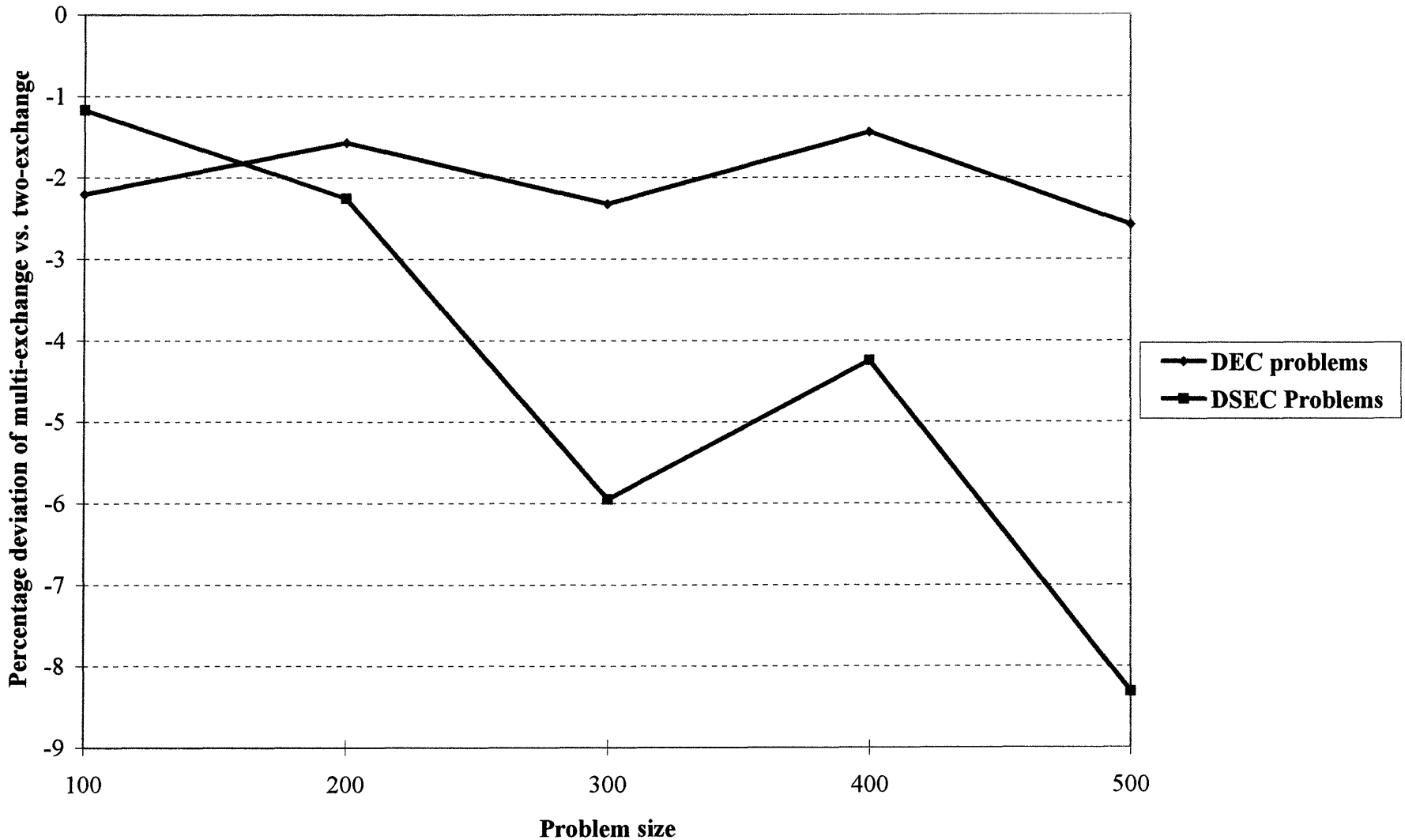


Figure 11. Comparison of multi-exchange neighborhood vs. two-exchange neighborhood.



Accordingly, our tabu search algorithms are able to obtain solutions with percent improvement as much as 40%.

We performed additional tests to determine the effectiveness of the multi-exchange neighborhood structure over the two-exchange neighborhood structure. We considered variations of our tabu search algorithms where we performed only two-exchanges and do not allow changes involving more than two subtrees. We allowed both algorithms to run for the same time. We plot in Figure 11, the percentage deviation of the tabu search algorithm with the multi-exchange neighborhood vs. the tabu search algorithm with two-exchange neighborhood on aos problems with unit demand and squared Euclidean costs. We observe that the algorithm with the multi-exchange neighborhood exhibits the superior performance and its performance improves with the increase in problem size.

We wanted to determine the bottleneck operation in the algorithm. The algorithm repeatedly performs two major operations: (i) construction of the improvement graph, and (ii) determining negative cost subset-disjoint directed cycles in the improvement graph. Using profiling we determined the time spent by the computer problem in these two operations. We found that the algorithm spends about 2/3 of its time in constructing improvement graphs, and about 1/3 of its time in identifying negative cost subset-disjoint cycles. The reader may recall that the construction of an improvement graph requires solving spanning tree problems which is a time-consuming step. Faster spanning tree algorithms may reduce the time taken by this operation.

8. SUMMARY AND CONCLUSIONS

In this paper, we suggested two new node-based and tree-based neighborhood structures for the capacitated minimum spanning tree problem by generalizing two existing neighborhood structures. Whereas the existing neighborhood structures consider changes involving only two subtrees, our neighborhood structures consider changes involving several subtrees, which could be as many as the total number of subtrees in the current solution. The number of such exchanges grow exponentially with the problem size, but we suggested a shortest path based technique that heuristically identifies profitable changes in a low order polynomial time. Our preliminary computational results are very encouraging and both the neighborhood structures have their own relative strengths. The node-based neighborhood structure is more effective for solving capacitated minimum spanning tree problems where all nodes have unit (or, homogenous) demand, and the tree-based neighborhood structure is more effective for problems with heterogeneous demands. A tabu search algorithm using the node-based neighborhood

structure obtained the best known solutions for all the 60 benchmark instances with unit demands and improved three of these solutions. Another tabu search algorithm using the tree-based neighborhood structure when applied to 35 benchmark instances improved most of the best known solutions; with the average improvement around 3.2% and the maximum improvement as much as 18%.

In our empirical investigations, we have implemented fairly straightforward tabu search algorithms and made no specific efforts to fine tune them. We believe that more sophisticated tabu search algorithms will perform even better; we leave this issue as an issue for future research. We can also develop simulated annealing algorithms using the suggested neighborhood structures; we again leave this issue for future research. The basic ideas underlying our neighborhood structures are also applicable to those combinatorial optimization problems where the problem can be conceived of as a set partitioning problem and each part can be optimized separately. Some problems where this structure arises naturally are the vehicle routing problem, the multi-traveling salesman problem, parallel machine scheduling problems, facility location problems, exam scheduling problems, clustering and graph partitioning, and graph coloring problems. We intend to apply the ideas contained in this paper for such problems.

ACKNOWLEDGEMENTS

We thank Luis Gouviea for providing us with valuable information related to the available research on the capacitated minimum spanning tree problem. We also thank Leslie Hall for providing timely help at various stages of this research. The research contained in this paper is supported in part by the National Science Foundation SGER Grant # DMI-9810359.

REFERENCES

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows : Theory, Algorithms and Applications*. Prentice Hall, New Jersey.

Amberg A., W. Domschke, and Stefan Voβ. 1996. Capacitated minimum spanning trees: Algorithms using intelligent search. *Combinatorial Optimization: Theory and Practice* **1**, 9-39.

Esau, L. R., and K.C. Williams. 1966. On teleprocessing system design. Part II - A method for approximating the optimal network. *IBM Systems Journal* **5,** 142-147.

Gavish, B. 1991. Topological design telecommunications networks - Local access design methods. *Annals of Operations Research* **33**, 17-71.

Gendreau, M., F. Guertin, J-Y Potvin, and R. Seguin. 1998. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. CRT Research Report No. CRT-98-10, Centre for Research on Transportation, University of Montreal, Montreal, CANADA.

Glover, F. 1989. Tabu search - Part I. *ORSA Journal on Computing* **1**, 190-206.

Glover, F. 1989. Tabu search - Part II. *ORSA Journal on Computing* **2**, 4-32.

Glover, F. 1996. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics* **65**, 223-253.

Glover, F., and M. Laguna. 1997. Tabu Search. Kluwer Academic Publishers, Norwell, MA.

Gouveia, L. 1995. A 2n-constraint formulation for the capacitated minimal spanning tree problem. *Operations research* **43,**130-141.

Gouveia, L. 1996. Using variable redefinition for computing minimum spanning and steiner trees with hop constraints. Working Paper No. 2/96, Centro de Investigacao Operacional, University of Lisbon, Lisbon, Portugal.

Gouveia, L., and P. Martins. 1996. The capacitated minimal spanning tree problem: An experiment with a hop-indexed model. Working Paper No. 1/96, Centro de Investigacao Operacional, University of Lisbon, Lisbon, Portugal.

Gouveia, L., and P. Martins. 1998. A hierarchy of hop-indexed models for the capacitated minimum spanning tree problem. Working Paper No. 2/98, Centro de Investigacao Operacional, University of Lisbon, Lisbon, Portugal.

Hall L. A. 1996. Experience with a cutting plane approach for the capacitated spanning tree problem. *ORSA Journal on Computing* **8**, 219-234.

Han, X., Kelsen, P., Ramachandran, V., Tarjan, R. 1995. Computing minimal spanning subgraphs in linear time. *SIAM Journal on Computing* **24**, 1332-1358.

Kruskal, J.B. 1956. On the shortest spanning tree of graph and the traveling salesman problem. *Proceedings of the American Mathematical Society* **7**, 48-50.

Pape, U. 1980. Algorithm 562: Shortest path lengths. *ACM Transactions on Mathematical Software* **6**, 450-455.

Prim, R.C. 1957. Shortest connection networks and some generalizations. *Bell System Technical Journal* 36, 1389-1401.

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

Thompson, P. M., and J. B. Orlin. 1989. The theory of cyclic transfers. Working Paper OR200-89, Operations Research Center, MIT, Cambridge, MA.

Thompson, P. M., and H. N. Psaraftis. 1993. Cyclic transfer algorithms for multi-vehicle routing and scheduling problems. *Operations Research* **41,** 935-946.