Solving Medium to Large Sized Euclidean Generalized Minimum Spanning Tree Problems

Diptesh Ghosh
P&QM Area, IIM Ahmedabad, Vastrapur, Ahmedabad 380015, India
diptesh@iimahd.ernet.in

WP. No. 2003-08-02, Working Paper Series, IIM Ahmedabad

Abstract

The generalized minimum spanning tree problem is a generalization of the minimum spanning tree problem. This network design problems finds several practical applications, especially when one considers the design of a large-capacity backbone network connecting several individual networks. In this paper we study the performance of six neighborhood search heuristics based on tabu search and variable neighborhood search on this problem domain. Our principal finding is that a tabu search heuristic almost always provides the best quality solution for small to medium sized instances within short execution times while variable neighborhood decomposition search provides the best quality solutions for most large instances.

Keywords: Generalized minimum spanning tree problem, neighborhood, tabu search, variable neighborhood search.

1 Introduction

In this paper we consider the following network design problem:

> Consider a weighted complete graph $G = (V, E, d)$, in which the vertex set V is partitioned into r subsets (called clusters), $V_1, V_2, \ldots, V_r$, containing $k_1, k_2, \ldots, k_r$ vertices respectively, and the cost function $d : E \times E \rightarrow \Re_+$. A solution to the Generalized Minimum Spanning Tree (GMST) Problem defined on G is a tree containing exactly one vertex from each of the subsets $V_1, V_2, \ldots, V_r$ (see Figure 1). The cost of a solution is the sum of the costs of the edges that make up the tree. The objective in the GMST problem is to obtain a minimum cost solution to an instance.

In this paper we study Euclidean GMST problems, which means that $d(i, j) + d(j, k) \geq d(i, k)$ for any i, j and k in V.

The GMST problem is one of several generalized network design problems mentioned in the literature (see Feremans, Labbé, and Laporte [3] for a review of generalized network design problems). It was first proposed in Myung, Lee, and Teha [12] and has been comprehensively reviewed in Feremans [5]. In the published literature, several alternate formulations for the GMST problem have been proposed in Faigle, Kern, Pop, and Still [2], Feremans [5], and Myung, Lee, and Teha [12]. NP-Hardness of this problem has been established in Myung Lee and Teha [12]. Exact solution algorithms based on branch and

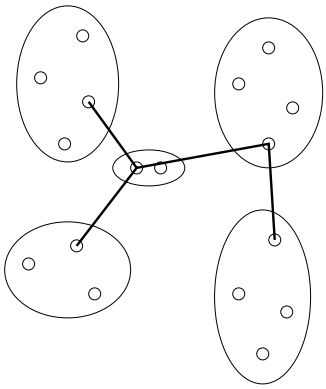

Figure 1: A solution to a GMST problem instance

cut has been proposed in Feremans [5], while approximation algorithms and heuristics have been proposed in Dror, Hauori, and Chaouachi [1], Myung, Lee, and Teha [12], and Pop, Kern, and Still [13]. To the best of our knowledge, sophisticated metaheuristic techniques have not been applied to this problem, although an elementary genetic algorithm was proposed in Dror, Hauori, and Chaouachi [1], and in Feremans [5] an elementary tabu search technique with short term memory is used to obtain upper bounds to the GMST problem. Problems closely related to the GMST problem have been dealt with in Dror, Hauori, and Chaouachi [1], Feremans, Labbé, and Laporte [4], and Ihler, Reich, and Widmayer [10].

In this paper, we study the performance of several tabu search and variable neighborhood search based metaheuristics on the GMST problem. We describe these metaheuristics in detail in the next section. Our computational experience with the heuristics described in Section 2 is presented in detail in Section 3. We conclude the paper with a brief summary in Section 4.

2 Description of metaheuristics considered

In this work, a solution to a GMST problem instance is represented as a r-vector, e.g. $S = \langle s_1, s_2, \ldots s_r \rangle$ where $s_i = j$ indicates that the jth vertex in the ith cluster is used in the solution. The solution itself is lowest cost tree connecting the vertices coded in the solution, and the cost of the solution is the cost of this tree. For instance, if we consider an instance of the GMST problem having four clusters containing 4, 5, 3, and 7 vertices respectively, then a solution $S_1 = \langle 3, 1, 2, 4 \rangle$ denotes the lowest cost tree connecting the third vertex in the first cluster, the first vertex in the second cluster, the second vertex in the third cluster, and the fourth vertex in the fourth cluster. It is important to note here that if there are more than one minimum cost trees connecting the vertices coded in the vector, the solution representation here does not specify which tree to choose. However, since all such trees will have the same cost, this ambiguity does not affect us in this work.

Tabu search (refer for example, to Gendreau [7]) and variable neighborhood search (refer for example, to Hensen and Mladenović [9]) are both improvement heuristics, which start out with a user-specified initial solution, and then tries to improve that solution. This improvement is carried out in steps, by moving from a solution to another solution in its *neighborhood*. The neighborhood that we consider is a 1-swap neighborhood common in

combinatorial optimization, in which two solutions are neighbors if and only if they differ in exactly one component. Therefore a solution $S_2 = \langle 3, 3, 2, 4\rangle$ is a neighbor of S_1, but $S_3 = \langle 4, 1, 3, 4\rangle$ is not (since it differs from S_1 in two components — the first and the fourth). An operation that transforms a solution to one of its neighbors is called a *move*.

In this work, a greedy heuristic is used to generate the starting solution for the heuristics. The vertices in each cluster are ordered according to the sum of the costs of the edges connecting them and vertices of other clusters, and the one with minimum sum of edge costs is chosen to be in the initial solution. Ties are broken arbitrarily.

We implemented and tested two variants of the tabu search heuristic in this work. The first variant, which we call TS1 incorporates recency based memory and aspiration rules. A description of this version follows.

First variant of tabu search (TS1)

Instance: A weighted graph $G = (V = \{V_1, \ldots, V_r\}, E, d)$, an initial solution S, a termination condition.

Output: A nearly optimal solution to the GMST problem instance.

Step 1 (Initialization): Define all $v \in V$ as non-tabu. Set $\mathtt{BestSolution} \leftarrow S$, and set $\mathtt{Iteration} \leftarrow 1$. Go to Step 1.

Step 2 (Termination): If a pre-defined termination condition is satisfied, output $\mathtt{BestSolution}$ and exit. Else go to Step 3.

Step 3 (Iteration): Consider all neighbors of S. All such neighbors for which a move to that neighbor requires the use of a vertex marked 'tabu' are tabu neighbors, while all others are 'non-tabu' neighbors. If the best tabu neighbor T of S has a cost lower than the cost of $\mathtt{BestSolution}$, go to Step 4, else replace S by the best non-tabu neighbor of S. Mark the vertices participating in this move (i.e. the vertex that has left the solution, and the vertex that has entered the solution to form the neighbor) as tabu for the next $\mathtt{TENURE}$ moves. If S is better than $\mathtt{BestSolution}$, then set $\mathtt{BestSolution} \leftarrow S$. Set $\mathtt{Iteration} \leftarrow \mathtt{Iteration} + 1$. Go to Step 2.

Step 4 (Aspiration): Set $\mathtt{BestSolution} \leftarrow T$, and $S \leftarrow T$. Remove the tabu status for all vertices. Set $\mathtt{Iteration} \leftarrow \mathtt{Iteration} + 1$. Go to Step 2.

The second version of tabu search that we implement here is called TS2. In addition to recency based memory and aspiration rules, this version incorporates frequency based memory. Frequency based memory uses an array (called $\mathtt{freq}$) to store the number of times that a certain move has been executed. It then discourages moves which have been executed a large number of times in the past. This allows the search to diversify and move into newer regions of the solution space. A description of this version is provided below.

Second variant of tabu search (TS2)

Instance: A weighted graph $G = (V = \{V_1, \ldots, V_r\}, E, d)$, an initial solution S, a termination condition.

Output: A nearly optimal solution to the GMST problem instance.

Step 1 (Initialization): Define all $v \in V$ as non-tabu. Set $\mathtt{freq}(i, j)$ for all moves that replace vertex i with vertex j to 0. Set $\mathtt{BestSolution} \leftarrow S$, and set $\mathtt{Iteration} \leftarrow 1$. Go to Step 1.

Step 2 (Termination): If a pre-defined termination condition is satisfied, output BestSolution and exit. Else go to Step 3.

Step 3 (Iteration): Consider all neighbors of S. All such neighbors for which a move to that neighbor requires the use of a vertex marked 'tabu' are tabu neighbors, while all others are 'non-tabu' neighbors. The costs of all the neighbors of S are adjusted depending on the freq values of the move from S to that neighbor. If the best tabu neighbor T of S has a cost lower than the cost of BestSolution, go to Step 4, else replace S by the best non-tabu neighbor of S. Mark the vertices participating in this move (i.e. the vertex that has left the solution, and the vertex that has entered the solution to form the neighbor) as tabu for the next TENURE moves. Increment the freq values of these vertices by 1. If S is better than BestSolution, then set $\texttt{BestSolution} \leftarrow S$. Set $\texttt{Iteration} \leftarrow \texttt{Iteration} + 1$. Go to Step 2.

Step 4 (Aspiration): Set $\texttt{BestSolution} \leftarrow T$, and $S \leftarrow T$. Remove the tabu status for all vertices. Set $\texttt{Iteration} \leftarrow \texttt{Iteration} + 1$. Go to Step 2.

For both the variants we set the tabu tenure i.e. the TENURE parameter, to 10. The adjustment of the costs of the neighbors in TS2 was achieved by multiplying their solution costs by a penalty factor. Our preliminary experiments showed that a good penalty factor is a gently increasing linear function of the iteration number and the number of times a move has been executed in the past. Setting the penalty factor to $1.0 + \texttt{Iteration} \times \texttt{freq}(i,j) \times 1 \times 10^{-5}$ resulted in the best solutions, and we use this penalty factor in our experiments.

We also implemented four variable neighborhood search based heuristics. All of these require more than one neighborhoods, and in our implementations we use two neighborhood structures. The first is the 1-swap neighborhood described earlier in this section. The other is a strict 2-swap neighborhood. In this neighborhood, a move changes vertices in exactly two of the clusters. Therefore the solution S_3 in our earlier example, which was not in the 1-swap neighborhood of S_1 is in its strict 2-swap neighborhood. Notice that the 1-swap and strict 2-swap neighborhoods of any solution are disjoint.

The first heuristic is the basic variable neighborhood descent heuristic. This is described below.

Variable neighborhood descent search (VND)

Instance: A weighted graph $G = (V = \{V_1, \ldots, V_r\}, E, d)$, an initial solution S.

Output: A nearly optimal solution to the GMST problem instance.

Step 1 (Initialization): Set $\texttt{BestSolution} \leftarrow S$, and $k \leftarrow 1$. Go to Step 2.

Step 2 (Termination): If $k > 2$, output BestSolution and exit. Else go to Step 3.

Step 3 (Exploration): Find the best solution T in the k-swap neighborhood of S. If T is better than S, set $S \leftarrow T$, $\texttt{BestSolution} \leftarrow T$, and $k \leftarrow 1$. Else set $k \leftarrow k + 1$. Go to Step 2.

The second heuristic is a reduced variable neighborhood search heuristic. The main difference between this heuristic from VND is that the neighborhood of a solution is not searched exhaustively. Thus, for large problems, this heuristic can provide solutions much faster than VND. The heuristic is described below.

Reduced variable neighborhood search (RVNS)

Instance: A weighted graph $G = (V = \{V_1, \ldots, V_r\}, E, d)$, an initial solution S, a termination condition.

Output: A nearly optimal solution to the GMST problem instance.

Step 1 (Initialization): Set $k \leftarrow 1$ and $Iteration \leftarrow 1$. Go to Step 2.

Step 2 (Termination): If a termination condition is satisfied, output S and exit. Else go to Step 3.

Step 3 (Exploration): Generate a solution T at random in the k-swap neighborhood of S. If T is better than S, set $S \leftarrow T$, and $k \leftarrow 1$. Else if $k = 1$, set $k \leftarrow 2$, and if $k = 2$ set $k \leftarrow 1$. Set $Iteration \leftarrow Iteration + 1$. Go to Step 2.

The termination condition that we use here is a composite one. We stop if a pre-determined execution time is exceeded, or if more than 50 iterations have elapsed without improving the best solution at hand, S.

The third heuristic, the basic variable neighborhood search heuristic combines deterministic and stochastic changes of neighborhood. It is formally described below.

Variable neighborhood search (VNS)

Instance: A weighted graph $G = (V = \{V_1, \ldots, V_r\}, E, d)$, an initial solution S, a termination condition.

Output: A nearly optimal solution to the GMST problem instance.

Step 1 (Initialization): Set $k \leftarrow 1$ and $Iteration \leftarrow 1$. Go to Step 2.

Step 2 (Termination): If a termination condition is satisfied, output S and exit. Else go to Step 3.

Step 3 (Exploration): Generate a solution T at random in the k-swap neighborhood of S. Apply steepest descent local search starting with T and using a k-swap neighborhood to obtain a locally optimal solution T'. If T' is better than S, set $S \leftarrow T'$, and $k \leftarrow 1$. Else if $k = 1$, set $k \leftarrow 2$, and if $k = 2$ set $k \leftarrow 1$. Set $Iteration \leftarrow Iteration + 1$. Go to Step 2.

The last among the heuristics that we implement here is the variable neighborhood decomposition search. It differs from the previous three heuristics in the way it searches the neighborhood of a solution. It does not look at the whole neighborhood of the problem, but does an exhaustive search on a subset of clusters. We describe the heuristic below.

Variable neighborhood decomposition search (VNDS)

Instance: A weighted graph $G = (V = \{V_1, \ldots, V_r\}, E, d)$, an initial solution S, an integer $k_{max} < r$, and a termination condition.

Output: A nearly optimal solution to the GMST problem instance.

Step 1 (Initialization): Set $k \leftarrow 1$ and $Iteration \leftarrow 1$. Go to Step 2.

Step 2 (Termination): If a termination condition is satisfied, output S and exit. Else go to Step 3.

Step 3 (Exploration): Generate a solution T at random in the k-swap neighborhood of S. Let I be the set of clusters in which S and T use different vertices. Find the best possible solution T' obtainable from S by changing only vertices in I. If T' is better than S, set $S \leftarrow T'$, and $k \leftarrow 1$. Else if $k < K_{max}$, set $k \leftarrow k + 1$, and if $k = k_{max}$ set $k \leftarrow 1$. Set $Iteration \leftarrow Iteration + 1$. Go to Step 2.

We set k_{max} to 5 in our VNDS implementation. The termination condition in both VNS and VNDS is the same as the termination condition in RVNS.

3 Our computational experience

In this section we report the results of our computations with the heuristics described in Section 2 on Euclidean GMST problems. The heuristics were coded in C and compiled and executed on a Intel Pentium 4 personal computer with 1.7 GHz clock speed running a Linux operating system. Two types of GMST problem instances were used in our tests — randomly generated instances, and instances based on the TSPLIB (Reinelt [14]).

3.1 Randomly generated instances

For randomly generated GMST problem instances, we define a cluster as a pre-determined number p of points located in a square of side $SPAN$. The squares demarcating the clusters are themselves organized in the form of a rectangular grid, where the number of rows in the grid is denoted by m and the number of columns by n. Rows and columns are evenly spaced and the separation between two adjacent rows (or columns) is denoted by SEP. Figure 2 depicts the various parameters pictorially, where $m = 2$, $n = 3$ and $p = 3$, and the filled in circles correspond to the vertices.

In previous studies (see for example, Myung *et al.* [12] and Feremans [5]) a slightly different method is used to generate random instances. In these studies, all the vertices are generated as points in a square of side 100 units, and are randomly assigned to clusters, under the restriction that each cluster has the same number of vertices. By ensuring that $SEP < SPAN$ our generation method can generate similar instances. Refer, for example to Figure 3. This shows four clusters (depicted by boundaries with different line styles, and vertices that look different) in an instance with $\frac{SEP}{SPAN} = 0.75$.

We designed and conducted preliminary experiments using a branch and bound algorithm and the six heuristics on GMST problem instances with 12 clusters to detect any significant trends in execution times of exact algorithms and quality of solutions output by the heuristics when the value of p, or the $\frac{m}{n}$ ratio, or the $\frac{SEP}{SPAN}$ ratio was altered. We found that execution times of the exact algorithm increased when the value of p increased and in most cases, the exact algorithm took less execution time when the $\frac{m}{n}$ ratio decreased. However, the performance of the heuristics vis-à-vis the exact algorithm did not show any significant trends. Therefore, in our experiments with randomly generated instances, we generated thirty six problem sets containing five instances each, with two different number of clusters (100 and 400), two different values of p (4 and 5), three different $\frac{m}{n}$ ratios (1.0, 4.0, and 25.0), and three different $\frac{SEP}{SPAN}$ ratios (2.0, 1.0, and 0.5). In eighteen of these sets (Set 100-01 through 100-18), which consisted of relatively smaller instances, the six heuristics were allowed a maximum of 300 CPU seconds on each instance. In the others (Sets 400-01 through 400-18), which contained larger instances and the heuristics were allowed a

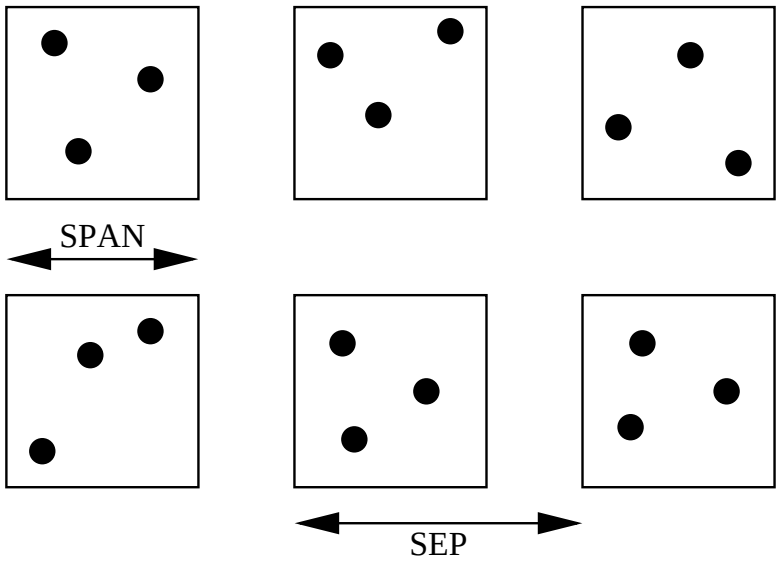

Figure 2: Layout of randomly generated instances

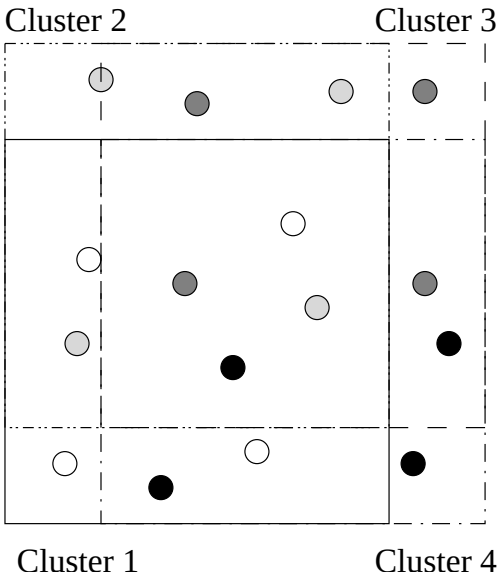

Figure 3: Overlapping clusters

maximum of 600 CPU seconds for each instance. Since all the instances were too large to solve optimally, we present a comparison of their *relative* performance, using the solution obtained by VNDS as the base. For each instance in each set, we compute the percentage by which the solution obtained by a particular heuristic improves over the solution obtained by the VNDS heuristic. A positive value indicates that the current heuristic output a better solution than VNDS, while a negative value shows that the current heuristic output a worse solution. The figure for a particular set is the average of the improvement percentages over all five instances in the set. Table 1 present our findings.

Table 1: Relative Performance of heuristics on randomly generated instances

Problem					Improvement over VNDS				
Set	$m \times n$	p	SEP	SPAN	TS1	TS2	VND	RVNS	VNS
Set 100-01	10×10	4	10.0	5.0	0.30%	0.88%	0.61%	0.61%	0.61%
Set 100-02	10×10	5	10.0	5.0	0.45%	1.62%	0.80%	0.80%	0.80%
Set 100-03	20×5	4	10.0	5.0	0.13%	0.72%	0.45%	0.45%	0.45%
Set 100-04	20×5	5	10.0	5.0	0.11%	0.97%	0.38%	0.38%	0.38%
Set 100-05	50×2	4	10.0	5.0	0.06%	0.47%	0.34%	0.34%	0.34%
Set 100-06	50×2	5	10.0	5.0	0.11%	0.80%	0.32%	0.32%	0.32%
Set 100-07	10×10	4	10.0	10.0	0.48%	2.27%	1.66%	1.66%	1.66%
Set 100-08	10×10	5	10.0	10.0	0.96%	2.97%	1.77%	1.77%	1.77%
Set 100-09	20×5	4	10.0	10.0	0.94%	2.17%	1.48%	1.48%	1.48%
Set 100-10	20×5	5	10.0	10.0	0.29%	2.18%	0.96%	0.96%	0.96%
Set 100-11	50×2	4	10.0	10.0	0.09%	0.19%	0.15%	0.15%	0.15%
Set 100-12	50×2	5	10.0	10.0	0.01%	0.61%	0.62%	0.62%	0.62%
Set 100-13	10×10	4	5.0	10.0	1.55%	3.55%	2.48%	2.48%	2.48%
Set 100-14	10×10	5	5.0	10.0	0.84%	4.85%	1.86%	1.86%	1.86%
Set 100-15	20×5	4	5.0	10.0	0.25%	3.01%	2.03%	2.03%	2.03%
Set 100-16	20×5	5	5.0	10.0	0.72%	3.89%	2.45%	2.45%	2.45%
Set 100-17	50×2	4	5.0	10.0	0.54%	2.23%	1.90%	1.90%	1.90%
Set 100-18	50×2	5	5.0	10.0	0.34%	1.50%	1.02%	1.02%	1.02%
Average					0.45%	1.94%	1.18%	1.18%	1.18%
Set 400-01	20×20	4	10.0	5.0	-0.08%	-0.10%	-0.08%	-0.08%	-0.08%
Set 400-02	20×20	5	10.0	5.0	0.06%	0.09%	-0.24%	-0.24%	-0.24%
Set 400-03	40×10	4	10.0	5.0	0.35%	0.34%	-0.24%	-0.24%	-0.24%
Set 400-04	40×10	5	10.0	5.0	-1.34%	-1.37%	-0.13%	-0.13%	-0.13%
Set 400-05	100×4	4	10.0	5.0	0.32%	0.28%	0.02%	0.02%	0.02%
Set 400-06	100×4	5	10.0	5.0	-1.27%	-1.53%	-0.37%	-0.37%	-0.37%
Set 400-07	20×20	4	10.0	10.0	-3.34%	-2.45%	-0.16%	-0.16%	-0.16%
Set 400-08	20×20	5	10.0	10.0	-0.61%	-0.46%	-0.26%	-0.26%	-0.26%
Set 400-09	40×10	4	10.0	10.0	-0.23%	-0.25%	-0.20%	-0.20%	-0.20%
Set 400-10	40×10	5	10.0	10.0	-0.27%	-0.24%	-0.24%	-0.24%	-0.24%
Set 400-11	100×4	4	10.0	10.0	-0.15%	-0.23%	-0.17%	-0.17%	-0.17%
Set 400-12	100×4	5	10.0	10.0	0.00%	0.10%	-0.77%	-0.77%	-0.77%
Set 400-13	20×20	4	5.0	10.0	-4.99%	-3.95%	-0.70%	-0.70%	-0.70%
Set 400-14	20×20	5	5.0	10.0	0.26%	0.26%	-0.71%	-0.71%	-0.71%
Set 400-15	40×10	4	5.0	10.0	4.89%	4.79%	0.65%	0.65%	0.65%
Set 400-16	40×10	5	5.0	10.0	3.75%	3.65%	0.59%	0.59%	0.59%
Set 400-17	100×4	4	5.0	10.0	0.34%	0.27%	-1.30%	-1.30%	-1.30%
Set 400-18	100×4	5	5.0	10.0	-0.41%	-0.41%	-0.41%	-0.41%	-0.41%
Average					-0.15%	-0.07%	-0.26%	-0.26%	-0.26%

The results show that for instances with 100 clusters, TS2 generated the best solutions among the six heuristics. On an average, the solutions it generated are 1.94% better than those generated by VNDS. VND, RVNS, and VNS came next, and the solutions they output were 1.18% better than those output by VNDS. TS1 generated solutions that are only marginally better than VNDS. The performance of all the heuristics vis-à-vis VNDS improved when the $\frac{m}{n}$ or the $\frac{SEP}{SPAN}$ ratios decreased. However, when the value of p increased, the superiority of all the heuristics over VNDS decreased, with the exception of TS2.

For instances with 400 clusters, the results were very different. VNDS proved to be the one that output the best solutions for these problems. TS2 came second, and output solutions that were, on an average, 0.07% worse than those output by VNDS. TS1 generated solutions that were 0.15% worse, and VND, RVNS, and VNS generated solutions that were 0.26% worse than those output by VNDS. However, different trends were observed in their relative performances. At $\frac{m}{n} = 4.0$, all the five heuristics output solutions which were better than VNDS. However, when $\frac{m}{n} = 1.0$ or when $\frac{m}{n} = 25.0$ all of them output solutions worse than those output by VNDS. When $\frac{m}{n} = 1.0$, VND, RVNS, and VNS performed better than TS1 and TS2 on an average, but when $\frac{m}{n} = 25.0$, TS1 and TS2 output better solutions than the other three. At this $\frac{m}{n}$ ratio, TS1 was 0.2% worse than VNDS on an average, and TS2 was 0.25% worse. In this regard, recall that in our preliminary experiments with 12 clusters, we found that problems with high $\frac{m}{n}$ ratios require more execution times with branch and bound algorithms, and are prime candidates for the use of metaheuristic techniques. When the value of p increased, TS1 and TS2 output relatively better solutions. In fact when $p = 5$, the solutions they output were, on an average, better than those output by VNDS. The relative performance of VND, RVNS, and VNS deteriorated when p increased. When $\frac{SEP}{SPAN} = 0.5$ TS1 and TS2 output solutions that were better than VNDS on an average. VND, RVNS, and VNS output solutions which were worse than VNDS for these problems. However, when $\frac{SEP}{SPAN} = 2.0$ the solutions output by VND, RVNS, and VNS were better than those output by TS1 and TS2 on an average. All the five heuristics however output solutions that were worse than VNDS for these instances.

In summary, we would recommend TS2 for Euclidean GMST problems when dealing with medium sized instances, or large instances in which the clusters overlap. For other large instances, we recommend the VNDS heuristic.

3.2 TSBLIB based instances

TSPLIB (Reinelt [14]) is a library that provides a set of test instances for the symmetric traveling salesman problem. In this work we use an adaptation of a subset of these instances that form instances of the GMST problem. The instances that we use also form a subset of the instances used to test the performance of a branch and cut algorithm in Feremans [5].

In the literature, two types of clustering of the vertices in TSPLIB instances render them GMST problem instances. The first is called *geographical* clustering (simply called "clustering" in Fischetti *et al.* [6]). It fixes the number of clusters k to $\lceil \frac{|V|}{5} \rceil$. Then k centers are determined by considering k vertices as far away from each other as possible. Clusters are finally obtained by assigning each vertex to its nearest center. The second clustering approach is a *grid* clustering approach (see Feremans [5]) where the grid configuration is determined by a parameter μ corresponding to the average number of vertices in each cluster. The grid clustering approach works as follows. The vertices in a particular TSPLIB instance are plotted on a plane, and the smallest rectangle with sides parallel to the coordinate axes that encloses all vertices is obtained. This rectangle is divided into $NG \times NG$ equal sized

rectangles, and each rectangle containing at least one vertex corresponds to a cluster. The integer NG is determined as follows. Let cluster(H) be the number of non-empty clusters corresponding to a $H \times H$ grid. Then NG is the minimum integer such that cluster(NG) $\geq \frac{|V|}{\mu}$. In this work we use the cluster data used in Feremans [5].

In Feremans [5], TSPLIB-based instances are broadly divided into two categories, ones that the branch and cut algorithm developed therein could solve within 2 CPU hours on a Generic sun4u sparc SUNW, Ultra-5.10 workstation, and ones that could not be solved within that time. For our experiments, we choose those instances in the first category that used more than 600 CPU seconds to solve to optimality, and all instances in the second category.

While solving the instances of the first category, each of the six heuristics was alloted 90 CPU seconds of execution time. Table 2 presents the cost of the solutions output by each of the heuristics. TS2 was clearly the best among the six, obtaining optimal solutions in all but six of the thirty six instances solved and returning on an average, solutions that were 0.02% more expensive than the optimal solution. TS1 came second; although it could solve only seventeen instances to optimality, its average suboptimality is only 0.34%. VNDS returned the worst solutions among the six, and the solutions it output were 1.31% more costly than an optimal solution. The other three variants of variable neighborhood search heuristics output solutions with an average suboptimality of 0.51%.

Table 3 shows the time in seconds during the execution of each heuristic when it generated the solution which it finally output. Notice that apart from TS2, the other heuristics could not improve solutions that they generated very early in the search. Thus allowing these heuristics more execution time would most likely not appreciably improve the quality of solutions they output with an execution time of 90 CPU seconds. In a separate experiment, we allowed TS2 600 CPU seconds to solve each of the six kroA200 and kroB200 based instances that it could not solve to optimality within 90 CPU seconds, but it was not able to improve on any of the solutions within that time.

For the second category of TSPLIB-based instances, we allowed the six heuristics 600 CPU seconds of execution time, and compared the cost of the solutions output by each of the heuristics with the global upper bound presented in Feremans [5]. The results are shown in Table 4. Notice that apart from two of the instances (ts225 with geographical clustering, and gr202 with grid clustering and a μ value of 7), all the six heuristics improved the value of the global upper bound. Once again TS2 proved to be the best among the six, achieving an improvement in every single one of the instances and VNDS proved to be the worst. The average improvements ranged from 2.41% for TS2 to 1.11% for VNDS. A study of the execution time required by the heuristics to achieve the solution that they output showed that TS2 required the maximum time, 60.89 CPU seconds on an average, while RVNS and VNS required the minimum time, generating the final solution within the first CPU second of execution. VNDS required 11.95 CPU seconds on an average, VND required 5.32 CPU seconds, and TS1 required 2.11 CPU seconds.

4 A brief summary

In this paper, we study the performance of six heuristics based on tabu search and variable neighborhood search on the generalized network design problem. We define this problem in Section 1 and present a brief survey of published literature on this problem. In Section 2, we describe the six heuristics that we compare in this work. In addition to a formal description

Table 2: Costs of solutions output by the heuristics for the first category of TSPLIB-based instances

Instance	Vertices	Clusters	Optimum	TS1	TS2	VND	RVNS	VNS	VNDS
Geographical Clustering									
gr202	202	34	135	135	135	136	136	136	140
pr124	124	25	30174	30416	30174	30174	30174	30174	30404
bier127	127	26	58150	58343	58150	58343	58343	58343	58355
gr137	137	28	329	330	329	330	330	330	330
pr144	144	29	40055	40085	40055	40085	40085	40085	40085
krob150	150	30	10048	10063	10048	10063	10063	10063	10084
pr152	152	31	39109	39109	39109	39116	39116	39116	39116
rat195	195	39	751	751	751	751	751	751	776
kroa200	200	40	11634	11698	11636	11698	11698	11698	11698
krob200	200	40	11244	11477	11245	11291	11291	11291	11477
Grid Clustering ($\mu = 3$)									
bier127	127	50	71221	71221	71221	71640	71640	71640	71696
gr137	137	49	391	393	391	393	393	393	393
pr144	144	48	43725	43725	43725	43725	43725	43725	43725
pr152	152	54	44253	44294	44253	44253	44253	44253	44294
u159	159	58	24214	24214	24214	24221	24221	24221	24221
rat195	195	81	1111	1116	1111	1116	1116	1116	1137
kroa200	200	72	14881	14966	14888	14885	14885	14885	14970
krob200	200	76	15320	15357	15328	15346	15346	15346	15449
Grid Clustering ($\mu = 5$)									
bier127	127	26	58989	58989	58989	58989	58989	58989	58989
gr137	137	32	338	339	338	339	339	339	339
pr144	144	30	36279	36279	36279	36279	36279	36279	36279
pr152	152	33	38143	38143	38143	38143	38143	38143	38143
rat195	195	49	796	796	796	806	806	806	830
kroa200	200	47	11628	11671	11628	11671	11671	11671	11671
krob200	200	48	11113	11115	11114	11115	11115	11115	11205
Grid Clustering ($\mu = 7$)									
bier127	127	19	52097	52097	52097	52242	52242	52242	52468
gr137	137	22	264	268	264	268	268	268	269
pr152	152	24	35429	35429	35429	35429	35429	35429	35429
rat195	195	36	639	639	639	641	641	641	667
kroa200	200	35	9640	9687	9686	9687	9687	9687	9688
krob200	200	36	9742	9780	9742	9818	9818	9818	9907
Grid Clustering ($\mu = 10$)									
pr152	152	16	33340	33340	33340	33340	33340	33340	33340
rat195	195	25	482	482	482	490	490	490	516
kroa200	200	25	6895	7102	6895	7102	7102	7102	7102
krob200	200	25	6922	6922	6922	7212	7212	7212	7318
pr226	226	27	43389	43389	43389	43389	43389	43389	43389
# suboptimal solutions (out of 36)				19	6	26	26	26	29

Table 3: Time of obtaining the final solution by the heuristics for the first category of TSPLIB-based instances

Instance	Vertices	Clusters	TS1	TS2	VND	RVNS	VNS	VNDS
Geographical Clustering								
gr202	202	34	2	2	15	0	0	0
pr124	124	25	0	4	0	0	0	0
bier127	127	26	0	3	1	0	0	0
gr137	137	28	0	59	0	0	0	0
pr144	144	29	0	23	0	0	0	0
krob150	150	30	0	5	1	0	0	0
pr152	152	31	1	0	0	0	0	0
rat195	195	39	4	4	5	0	0	0
kroa200	200	40	1	19	0	0	0	0
krob200	200	40	0	55	3	0	0	0
Grid Clustering ($\mu = 3$)								
bier127	127	50	1	1	0	0	0	0
gr137	137	49	0	12	0	0	0	0
pr144	144	48	0	0	0	0	0	0
pr152	152	54	0	10	1	0	0	0
u159	159	58	0	2	0	0	0	0
rat195	195	81	1	2	2	0	0	0
kroa200	200	72	1	26	1	0	0	0
krob200	200	76	1	22	2	0	0	0
Grid Clustering ($\mu = 5$)								
bier127	127	26	0	1	0	0	0	0
gr137	137	32	0	6	0	0	0	0
pr144	144	30	0	0	0	0	0	0
pr152	152	33	0	0	0	0	0	0
rat195	195	49	1	1	6	0	0	0
kroa200	200	47	1	4	1	0	0	0
krob200	200	48	1	89	2	0	0	0
Grid Clustering ($\mu = 7$)								
bier127	127	19	1	0	1	0	0	1
gr137	137	22	0	19	1	0	0	0
pr152	152	24	0	0	0	0	0	0
rat195	195	36	1	1	6	0	0	0
kroa200	200	35	1	0	1	0	0	0
krob200	200	36	1	3	2	0	0	0
Grid Clustering ($\mu = 10$)								
pr152	152	16	0	0	0	0	0	0
rat195	195	25	0	1	2	0	0	0
kroa200	200	25	0	34	0	0	0	0
krob200	200	25	0	1	2	0	0	0
pr226	226	27	0	0	0	0	0	0
Average			0.49	11.63	1.14	0	0	0.03

Table 4: Costs of solutions output by the heuristics for the second category of TSPLIB-based instances

Instance	Vertices	Clusters	global	Percentage improvement over UB					
			UB*	TS1	TS2	VND	RVNS	VNS	VNDS
Geographical Clustering									
d198	198	40	7232	2.24%	2.34%	1.12%	1.12%	1.12%	0.65%
gr202	202	41	250	3.20%	3.20%	0.00%	0.00%	0.00%	0.00%
ts225	225	45	62506	-1.50%	0.22%	-0.24%	-0.24%	-0.24%	-1.50%
pr226	226	46	55971	0.81%	0.81%	0.81%	0.81%	0.81%	0.60%
Grid Clustering ($\mu = 3$)									
d198	198	67	8599	2.91%	3.67%	2.85%	2.85%	2.85%	2.34%
gr202	202	68	302	2.32%	2.98%	1.66%	1.66%	1.66%	0.66%
ts225	225	75	81962	3.42%	3.59%	3.42%	3.42%	3.42%	3.29%
pr226	226	84	63148	0.92%	0.98%	0.94%	0.94%	0.94%	0.92%
Grid Clustering ($\mu = 5$)									
d198	198	40	7291	1.19%	2.51%	2.65%	2.65%	2.65%	1.21%
gr202	202	41	243	4.53%	4.53%	0.82%	0.82%	0.82%	0.41%
ts225	225	45	62242	1.14%	2.38%	2.01%	2.01%	2.01%	1.34%
pr226	226	50	56822	0.13%	0.18%	0.18%	0.18%	0.18%	0.13%
Grid Clustering ($\mu = 7$)									
d198	198	32	6759	3.77%	3.82%	3.82%	3.82%	3.82%	3.77%
gr202	202	31	207	1.93%	1.93%	-2.90%	-2.90%	-2.90%	-2.90%
ts225	225	35	53661	5.21%	5.26%	5.21%	5.21%	5.21%	5.21%
pr226	226	33	48254	0.01%	0.01%	0.01%	0.01%	0.01%	0.01%
Grid Clustering ($\mu = 10$)									
d198	198	25	6351	2.61%	2.61%	2.60%	2.60%	2.60%	2.60%
gr202	202	21	185	2.16%	2.70%	2.16%	2.16%	2.16%	1.08%
ts225	225	25	41162	1.44%	2.00%	1.66%	1.66%	1.66%	1.33%
Average Improvement over UB				2.02%	2.41%	1.51%	1.51%	1.51%	1.11%

$\star$: UB quoted in Feremans [5]

of the heuristics, the section also contains details about our implementations. Section 3 describes the results of our computational experiments. We experimented with two types of instances, some of which were randomly generated, and some others were modified from symmetric traveling salesman problem instances from the TSPLIB (Reinelt [14]). In case of randomly generated instances, we describe an alternative scheme of generating random problems, which is more elaborate than the one commonly used in the literature (e.g. in Myung *et al.* [12] and Feremans [5]). Our experiments in this section demonstrate that this scheme allows us a richer analysis of the performance of the heuristics for these problems. For the second type of instances, we used a subset of the instances used in Feremans [5].

Our results show that for small and medium sized instances, which in this study include all the TSPLIB based instances and randomly generated instances with 100 clusters, TS2, a tabu search heuristic incorporating both recency based memory and frequency based memory was the heuristic of choice. For large instances, i.e. the ones with 400 clusters, the recommendation is divided. If the clusters overlap, then TS2 is still the best option available. Otherwise, we recommend variable neighborhood decomposition search.

The basic neighborhood structure that we use for tabu search heuristics is a 1-swap neighborhood, in which we allow a change in the choice of a vertex in exactly one cluster. One could use a 2-swap neighborhood structure, in which one allows a change in the the choice of vertices in a maximum of two clusters. Such a neighborhood is not necessary for small and medium sized problems, in which tabu search almost always generates an optimal solution using a 1-swap neighborhood. However in larger problems the 2-swap neighborhood may generate solutions better than those generated with a 1-swap neighborhood structure, although the time required to search the neighborhood would be longer. Finally, hybrids of tabu search and variable neighborhood decomposition search could yield better composite heuristics for the generalized minimum spanning tree problem.

Acknowledgement:

The author thanks Corrine Feremans for providing us a copy of her dissertation and data on the TSPLIB based instances, and Saral Mukherjee for many useful suggestions. Most of the computational experiments were carried out in the Manufacturing and Supply Chain Laboratory at IIM Ahmedabad.

References

[1] Dror, M., Haouari, M., and Chaouachi, J. (2000), Generalized spanning trees, *European Journal of Operational Research* 120, 583–592.

[2] Faigle, U., Kern, W., Pop, P.C. and Still, G. (2000), The generalized minimum spanning tree problem, *Working Paper, Department of Operations Research and Mathematical Programming, University of Twente, The Netherlands.*

[3] Feremans, C., Labbé, M. and Laporte, G. (2003), Generalized network design problems, *European Journal of Operational Research* 148, 1–13.

[4] Feremans, C., Labbé, M. and Laporte, G. (2001), On generalized minimum spanning trees, *European Journal of Operational Research* 134, 457–458.

[5] Feremans, C. (2001), Generalized Spanning Trees and Extensions, *Ph.D. Dissertation, Université Libre de Bruxelles.*

[6] Fischetti, M., Salazar, J.J., and Toth, P. (1995), The symmetric traveling salesman polytope, *Networks* 26, 113–123.

[7] Gendreau, M. (2003), An introduction to tabu search, Chapter 2 in [8], 37–54.

[8] Glover, F., and Kochenberger, G.A. (2003), *Handbook of Metaheuristics*, Kluwer Academic Publishers, Boston.

[9] Hansen, P., and Mladenović, N. (2003), Variable neighborhood search, Chapter 6 in [8], 145–184.

[10] Ihler, E., Reich, G. and Widmayer, P. (1999), Class Steiner trees and VLSI design, *Discrete Applied Mathematics* 90, 173–194.

[11] McKay, M.D., Conover, W.J. and Beckman, R.J. (1979), A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, *Technometrics* 21, 239–245.

[12] Myung, Y.S., Lee, C.H. and Teha, D.W. (1995), On the generalized minimum spanning tree problem, *Networks* 26, 231–241.

[13] Pop, P.C., Kern, W. and Still, G.J. (2001), An approximation algorithm for the generalized minimum spanning tree problem with bounded cluster size, *Working Paper, Department of Operations Research and Mathematical Programming, University of Twente, The Netherlands.*

[14] Reinelt, G. (1995), TSPLIB-95, `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`