Computing Generalized Minimum Spanning Trees with Variable Neighborhood Search

Bin Hu, Markus Leitner, Günther R. Raidl*

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{hu|raidl}@ads.tuwien.ac.at

Abstract

In the generalized version of the classical Minimum Spanning Tree problem, the nodes of a graph are partitioned into clusters and exactly one node from each cluster must be connected. This problem plays, for example, a role in the design of backbones in larger communication networks. We present a Variable Neighborhood Search (VNS) approach for this problem which is based on two different neighborhood types working in complementary ways to maximize the efficiency gained from the VNS concept. Both types of neighborhoods are large in the sense that they contain exponentially many candidate solutions, but efficient polynomial-time algorithms are used to identify best neighbors. Tests on Euclidean and random instances indicate in particular on instances with many nodes per cluster significant advantages of our VNS over previously published metaheuristic approaches.

Keywords: Generalized Minimum Spanning Tree, Variable Neighborhood Search, Dynamic Programming

1 Introduction

The Generalized Minimum Spanning Tree (GMST) problem is an extension of the classical Minimum Spanning Tree (MST) problem and is defined as follows. We consider a weighted complete

*This work is supported by the RTN ADONET under grant 504438 and the Austrian Science Fund (FWF) under grant P16263-N04.

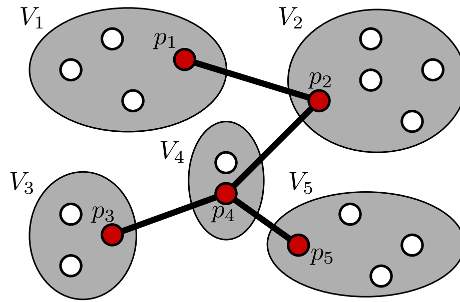

Figure 1: Example for a GMST solution

graph $G = \langle V, E, c\rangle$ with node set V, edge set E, and edge cost function $c : E \rightarrow \mathbb{R}^+$. The node set V is partitioned into r pairwise disjunct clusters $V_1, V_2, \ldots, V_r$ containing $d_1, d_2, \ldots, d_r$ nodes, respectively.

A spanning tree of a graph is a cycle-free subgraph connecting all nodes. A solution to the GMST problem defined on G is a graph $S = \langle P, T\rangle$ with $P = \{p_1, p_2, \ldots, p_r\} \subseteq V$ containing exactly one node from each cluster, i.e. $p_i \in V_i$ for all $1 \leq i \leq r$, and $T \subseteq P \times P \subseteq E$ being a spanning tree, see Figure 1. The costs of such a tree are its total edge costs, i.e. $C(T) = \sum_{(u,v)\in T} c(u, v)$, and the objective is to identify a solution with minimum costs.

In case each cluster contains only one node, i.e. $d_i = 1$ for all $1 \leq i \leq r$, the problem is reduced to the simple MST problem, which can be efficiently solved in polynomial time. In general, however, the GMST problem is NP-hard [8].

There are several real world applications of the GMST problem, e.g. in the design of backbones in large communication networks. Devices belonging to the same local area network can be seen as a cluster, and the global network connects one device per local network. For a more detailed overview on the GMST problem, see [1, 2, 8].

In this paper, we present a general Variable Neighborhood Search (VNS) approach for solving this problem. VNS is a metaheuristic which exploits the idea of local search in changing neighborhoods in order to head for a global optimum [6, 7]. As local improvement within VNS, we use Variable Neighborhood Descent (VND) utilizing two different types of exponentially large neighborhoods which can be seen as dual to each other.

The remainder of this article is organized as follows. In Section 2, we give an overview on research done for the GMST problem so far. In Section 3, we describe the neighborhoods used, as well as the

optimization techniques applied on them. After that, we present the details of our VNS algorithm in Section 4. We show experimental results including a comparison to previous approaches in Section 5 and finally conclude in Section 6.

2 Previous Work

The GMST problem was introduced by Myung et al. [8]. They proved that this problem is NP-hard and provided four different Integer Linear Programming (ILP) formulations. Feremans et al. [4] added another four formulations and did some in-depth investigation on all eight ILPs. Most of these approaches are based on subset analysis and therefore the number of constraints increases exponentially. Pop [9] introduced another ILP formulation, which proved to be more efficient than the others. Instances with up to 240 nodes divided into 30 clusters or 160 nodes divided into 40 clusters could be solved to optimality. Furthermore, Pop utilized the idea of his ILP formulation for a Simulated Annealing approach in order to heuristically solve larger instances. His work is essential for the design of the neighborhoods we present in this paper.

Regarding approximation algorithms, Myung et al. [8] have shown the inapproximability of the GMST problem in the sense that no approximation algorithm with constant quality guarantee can exist unless P = NP. However, there are better results for some special cases of the problem. Pop et al. [10] described an approximation algorithm for the case when the cluster sizes are limited. If $|V_i| \leq d_{\max}$ for all $1 \leq i \leq r$, the total costs of the resulting solution are at most $2d_{\max}$ times the optimal solution value. Feremans et al. [3] provided a Polynomial Time Approximation Scheme (PTAS) for the GMST problem in case of grid clustering, in which all nodes are situated inside a planar integer grid.

As for metaheuristics, Ghosh [5] implemented and compared a Tabu Search (TS) with recency based memory, a TS with recency and frequency based memory, a Variable Neighborhood Descent Search, a Reduced VNS, a VNS with Steepest Descent and a Variable Neighborhood Decomposition Search. For all the VNS approaches, he used 1-swap and strict 2-swap neighborhoods, which exchange the used nodes within clusters. This type of neighborhoods will be utilized by us as well, see Section 3.1. Comparing these approaches on instances ranging from 100 nodes to 400 nodes, Ghosh concluded that TS with recency and frequency based memory performs best for small to medium sized graphs. For large instances, results are ambiguous.

3 Neighborhoods

In our VNS algorithm, we use two types of neighborhoods based on concepts from Ghosh [5] and Pop [9]. Pop approaches the GMST problem from the global view by first deciding which clusters are directly connected and then deriving the best suited nodes and edges for these "global" connections. On the other hand, Ghosh starts from the opposed direction by first fixing the nodes from each cluster and then connecting them in a best way.

3.1 Node Exchange Neighborhood (NEN)

In this neighborhood, which was originally proposed by Ghosh [5], a solution is represented by a vector $p = (p_1, \ldots, p_r)$ where p_i is the node to be connected from each cluster V_i, $i = 1, \ldots, r$. Knowing these nodes, there are still r^{r-2} possible spanning trees, but the best one can be efficiently derived by computing a classical MST on the subgraph of G induced by the chosen nodes.

The Node Exchange Neighborhood (NEN) of a solution p consists of all node vectors (and corresponding spanning trees) in which for precisely one cluster V_i the node p_i is replaced by a different node p'_i of the same cluster. This neighborhood therefore consists of $\sum_{i=1}^{r}(|V_i| - 1) = O(|V|)$ node vectors representing $O(|V| \cdot r^{r-2})$ trees. Since a single MST can be computed in $O(r^2)$ time, e.g. by Prim's algorithm, a straight-forward generation and evaluation of the whole neighborhood in order to find the best neighboring solution can be accomplished in $O(|V| \cdot r^2)$ time.

Using an incremental evaluation scheme, we can reduce the computational effort significantly. The goal is to derive a new minimum-cost tree S' when node p_i is replaced by node p'_i. Removing p_i and all its incident edges from the initial tree S results in a graph consisting of $k \geq 1$ connected components $T_1, \ldots, T_k$ where usually $k \ll r$.

The new minimum-cost tree S' will definitely not contain new edges within each component $T_1, \ldots, T_k$, because they are connected in the cheapest way as they where optimal in S. New edges are only necessary between nodes of different components and/or p'_i. Furthermore, only the shortest edges connecting any pair of components must be considered. So, the edges of S' must be a subset of

- edges of S after removing p_i and its incident edges,
- all edges $(p'_i, p_j) \mid j = 1, \ldots, r \wedge j \neq i$, and
- the shortest edges between any pair of the components $T_1, \ldots, T_k$.

To compute S', we therefore have to calculate the MST of a graph with $(r-k-1)+(r-1)+(k^2-k)/2 = O(r+k^2)$ edges only. Unfortunately, this does not change the worst case total computation time, because identifying the shortest edges between any pair of components may need $O(r^2)$ operations. However, in most cases it is faster to compute these shortest edges and to apply Kruskal's MST algorithm on the resulting thin graph. Especially when replacing a leaf node of the initial tree S, we only get a single component plus the new node and the incremental evaluation is much faster than the straight-forward approach.

Exchanging More Than One Node

The above neighborhood can be easily generalized by simultaneously replacing $t \geq 2$ nodes. The computational complexity of a complete evaluation raises to $O(|V|^t \cdot r^2)$. While an incremental computation is still possible in a similar way as described above, the complete evaluation of the neighborhood becomes nevertheless impracticable for larger instances even when $t = 2$. We therefore use a Restricted Two Nodes Exchange Neighborhood (RNEN2) in which only pairs of clusters that are adjacent in the current solution S are simultaneously considered. In this way, the time complexity is reduced to $O(|V| \cdot r^2)$.

RNEN2 is still the most expensive neighborhood. Since its complete evaluation consumes too much time in case of large instances, we terminate its exploration after a certain time limit returning the so-far best neighbor instead of following a strict best-neighbor strategy.

3.2 Global Edge Exchange Neighborhood (GEEN)

Derived from Pop's local-global ILP and his Simulated Annealing approach [9], we use a second neighborhood type which is defined on a so-called "global graph". This graph $G^g = \langle V^g, E^g \rangle$ consists of nodes corresponding to the clusters in G, i.e. $V^g = \{V_1, V_2, \ldots, V_r\}$, and edge set $E^g = V^g \times V^g$.

We now consider a spanning tree $S^g = \langle V^g, T^g \rangle$ with $T^g \subseteq E^g$ on this global graph. This tree represents the set of all feasible generalized spanning trees on G which contain for each edge $(V_a, V_b) \in T^g$ a corresponding edge $(u, v) \in E \mid u \in V_a \wedge v \in V_b,\ a \neq b$. Such a set of trees on G that a particular global spanning tree represents is in general exponentially large with respect to the number of nodes. However, we can use dynamic programming to efficiently determine a minimum cost solution from this set. In this process, we root the global spanning tree at an arbitrary cluster $V_{root} \in V^g$ and

direct all edges towards the leafs. We traverse this tree in a recursive depth-first way calculating for each cluster $V_k \in V^g$ and each node $v \in V_k$ the minimum costs for the subtree rooted in V_k when v is the node to be connected from V_k. These minimum costs of a subtree are determined by the following recursion:

$$C(T^g, V_k, v) = \begin{cases} 0 \text{ if } V_k \text{ is a leaf of the global spanning tree} \\ \sum\limits_{V_l \in Succ(V_k)} \min\limits_{u \in V_l} \{c(v,u) + C(T^g, V_k, u)\} \text{ else,} \end{cases}$$

where $Succ(V_k)$ denotes the set of all successors of V_k in T^g. After having determined the minimum costs for the whole tree, the selected nodes can be easily derived in a top-down fashion by fixing for each cluster $V_k \in V^g$ the node $p_k \in V_k$ yielding the minimum costs. This dynamic programming algorithm requires $O(|V|^2)$ time.

As neighborhood for a given global tree T^g, we consider any other spanning tree differing from T^g by precisely one edge. If we determine the best neighbor by evaluating all possibilities of exchanging a global edge and naively perform the whole dynamic programming for each global candidate tree, the time complexity would be $O(|V|^2 \cdot r^2)$.

For a more efficient evaluation of the neighbors, we perform the whole dynamic programming only once, keep all costs $C(T^g, V_k, v)$, and only evaluate an update. According to the recursive definition of the dynamic programming approach, we only need to recalculate the values of a cluster V_i if it gets a new child, loses a child, or the costs of a successor change.

Computing a solution in this neighborhood means to exchange a single global connection (V_a, V_b) by a different connection (V_c, V_d) so that the resulting graph remains a valid tree. By removing (V_a, V_b), the subtree rooted at V_b is disconnected, hence V_a loses a child and V_a, as well as all its predecessors, must be updated. Before we add (V_c, V_d), we first need to consider the isolated subtree. If $V_d \neq V_b$, we have to re-root the subtree at cluster V_d. Thereby, the old root V_b loses a child. All other clusters which get new children or lose children are on the path from V_b up to V_d, and they must be reevaluated. Otherwise, if $V_d = V_b$, nothing changes within the subtree. When adding the connection (V_c, V_d), V_c gets a new successor and therefore must be updated together with all its predecessors on the path up to the root. In conclusion, whenever we replace a global connection (V_a, V_b) by (V_c, V_d), it is enough to update the costs of V_a, V_b, and all their predecessors on the way up to the root of the new global tree, see Figure 2.

If the tree is not degenerated, we only need to update $O(\log r)$ clusters of G^g. Suppose each of

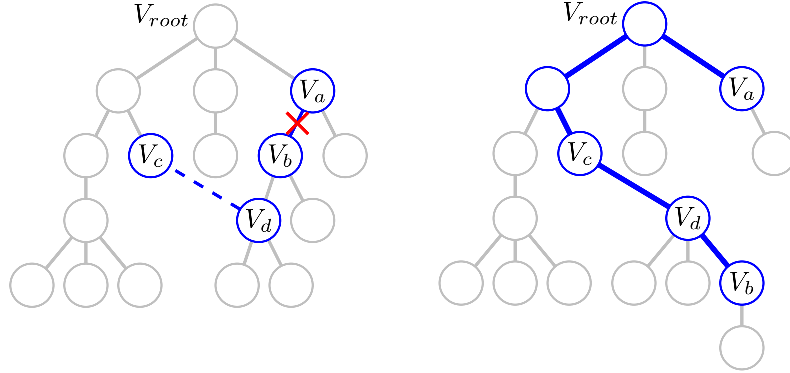

Figure 2: After removing (V_a, V_b) and inserting (V_c, V_d), all clusters on the paths from V_a and V_b up to V_{root} must be updated.

them contains no more than $d_{\max}$ nodes and has at most $s_{\max}$ successors. The time complexity of updating the costs of a single cluster V_i is $O(d_{\max}^2 \cdot s_{\max})$ and the whole process needs time that is bounded by $O(d_{\max}^2 \cdot s_{\max} \cdot \log r)$. In total we obtain an upper bound for the computation time which is, on not degenerated trees, much better than $O(|V|^2)$. An additional improvement is to further avoid unnecessary update calculations by checking if an update actually changes costs of a cluster. If this is not the case, we may omit the update of the cluster's predecessors as long as they are not affected in some other way.

Algorithm 1: global_edge_exchange(solution S)

```
forall global edges (V_i, V_j) ∈ E^g do
    remove (V_i, V_j)
    M_1 = preorder list of clusters V_k in component K_1^g containing V_i
    M_2 = preorder list of clusters V_l in component K_2^g containing V_j
    forall V_k ∈ M_1 do
        root K_1^g at V_k
        forall V_l ∈ M_2 do
            root K_2^g at V_l
            add (V_k, V_l)
            use dynamic programming to retrieve the objective value
            if current solution better than best then
                save current solution as best
            remove (V_k, V_l)
restore best solution
```

To examine the whole neighborhood of a current solution by using the improved method described above, it is a good idea to choose a processing order that supports incremental evaluation as well as possible. Algorithm 1 shows how this is done in detail. Removing an edge (V_i, V_j) splits our rooted tree into two components: K_1^g containing V_i and K_2^g containing V_j. The algorithm iterates through

all clusters $V_k \in K_1^g$ and makes them root. Each of these clusters is iteratively connected to every cluster of K_2^g in the inner loop. The advantage of this calculation order is that none of the clusters in K_1^g except its root V_k has to be updated more than once, because global edges are only added between the roots of K_1^g and K_2^g. Processing the clusters in preorder has another additional benefit: Typically most of the time very few clusters have to be updated when re-rooting either K_1^g or K_2^g.

4 VNS for the GMST Problem

We use the general VNS scheme with VND as local improvement for our approach. In VND, we alternate between NEN, GEEN, and RNEN2 in this order, see Algorithm 2. This sequence has been determined according to the computational complexity of searching the neighborhoods.

Algorithm 2: Variable_Neighborhood_Descent(solution $S = \langle P, T \rangle$)

```
l = 1
repeat
    switch l do
        case 1 // NEN
            for i = 1, ..., r do
                forall v ∈ V_i \ p_i do
                    change used node p_i of cluster V_i to v
                    recalculate the MST T
                    if current solution better than best then
                        save current solution as best
            restore best solution
        case 2 // GEEN
            global_edge_exchange() //see Algorithm 1
        case 3 // RNEN2
            forall clusters V_i and V_j adjacent in the current solution do
                forall v ∈ V_i \ p_i and u ∈ V_j \ p_j do
                    change used node p_i of cluster V_i to v
                    change used node p_j of cluster V_j to u
                    recalculate the MST T
                    if current solution better than best then
                        save current solution as best
            restore best solution
    if solution improved then
        l = 1
    else
        l = l + 1
until l > 3
```

Shaking It turned out that using a shaking function which puts more emphasis on diversity yields good results for our approach, see Algorithm 3. This shaking process uses both, the NEN and the GEEN structures. For NEN, the number of random moves for shaking starts at three because we have a reduced 2-Opt NEN improvement; thus, shaking in NEN with smaller values would mostly lead to the same local optimum as reached before. Shaking in GEEN starts with two random moves for the same reason. The number k of random moves increases in steps of two up to $\lfloor \frac{r}{2} \rfloor$.

Algorithm 3: shake(solution $S = \langle P, T \rangle$, size k)

for $i = 1, \ldots, 2k$ **do**
  randomly change the used node p_i of a random cluster V_i
recalculate the MST T and derive T^g
for $i = 1, \ldots, 2k + 1$ **do**
  remove a randomly chosen global edge $e \in T^g$ yielding components K_1^g and K_2^g
  insert a randomly chosen global edge e' connecting K_1^g and K_2^g with $e' \neq e$
  recalculate the used nodes $p_1, \ldots, p_r$ by dynamic programming

Initialization We use Algorithm 4 to compute the initial solution. For each cluster, the node with the lowest sum of edge costs to all nodes in other clusters is used and a MST is determined.

Algorithm 4: initialize()

for $i = 1, \ldots, r$ **do**
  choose $p_i \in V_i$ with minimal $\sum_{v \in V \setminus V_i} c(p_i, v)$ as the used node
determine MST T on the used nodes $p_1, \ldots, p_r$
return solution $S = \langle P, T \rangle$

5 Computational results

We tested our algorithm on instances used by Ghosh [5] and some large TSPlib[1] instances which are geographically clustered as described in [2]. We extended Ghosh's benchmark by generating new instances with large number of nodes per cluster with the same algorithm as described in [5]. We compare our results with Ghosh's Tabu Search with recency and frequency based memory (TS2), his Variable Neighborhood Decomposition Search (VNDS), and Pop's Simulated Annealing (SA) approach [9]. All experiments were performed on a Pentium 4 2.8GHz PC with 2GB RAM. While TS2 is deterministic, we provide average results over 30 runs for VNDS and VNS, and over 10 runs for SA. For TS2, VNDS, and our VNS, runs were terminated when a certain CPU-time limit (as

[1]http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html

Grouped Euclidean Instances				TS2	VNDS		SA		VNS	
Name	$\|V\|$	r	$\|V\|/r$	$C(T)$	$\overline{C(T)}$	std dev	$\overline{C(T)}$	std dev	$\overline{C(T)}$	std dev
5_5_5_10_10 − 1	125	25	5	**141.1**	**141.1**	0.00	152.3	0.52	**141.1**	0.00
5_5_5_10_10 − 2	125	25	5	**133.8**	**133.8**	0.00	150.9	0.74	**133.8**	0.00
5_5_5_10_10 − 3	125	25	5	143.9	145.4	0.00	156.8	0.00	**141.4**	0.00
10_10_5_10_10 − 1	500	100	5	**566.7**	577.6	0.00	642.3	0.00	568.6	0.59
10_10_5_10_10 − 2	500	100	5	**578.7**	584.3	0.00	663.3	1.39	581.0	1.39
10_10_5_10_10 − 3	500	100	5	**581.6**	588.3	0.00	666.7	1.81	587.9	4.07
5_4_30_10_10 − 1	600	20	30	85.2	87.5	0.00	93.9	0.00	**84.8**	0.27
5_4_30_10_10 − 2	600	20	30	**87.9**	90.3	0.00	99.5	0.28	**87.9**	0.05
5_4_30_10_10 − 3	600	20	30	88.6	89.4	0.00	99.2	0.17	**88.5**	0.00
8_8_20_10_10 − 1	1280	64	20	327.2	329.2	0.00	365.1	0.46	**321.8**	2.41
8_8_20_10_10 − 2	1280	64	20	322.2	322.5	0.00	364.4	0.00	**316.3**	0.83
8_8_20_10_10 − 3	1280	64	20	**332.1**	335.5	0.00	372.0	0.00	334.3	2.13
General Euclidean Instances				TS2	VNDS		SA		VNS	
Name	$\|V\|$	r	$\|V\|/r$	$C(T)$	$\overline{C(T)}$	std dev	$\overline{C(T)}$	std dev	$\overline{C(T)}$	std dev
50_5_1000_1000 − 1	250	50	5	**2285.1**	2504.9	0.00	2584.3	23.82	2336.9	34.23
50_5_1000_1000 − 2	250	50	5	**2183.4**	2343.3	0.00	2486.7	0.00	2304.1	47.95
50_5_1000_1000 − 3	250	50	5	**2048.4**	2263.7	0.00	2305.0	16.64	2049.8	15.29
20_20_1000_1000 − 1	400	20	20	**557.4**	725.9	0.00	665.1	3.94	625.4	14.59
20_20_1000_1000 − 2	400	20	20	724.3	839.0	0.34	662.1	7.85	**595.3**	0.14
20_20_1000_1000 − 3	400	20	20	604.5	762.4	0.00	643.7	14.54	**588.8**	7.40
20_30_1000_1000 − 1	600	20	30	541.6	656.1	0.00	491.8	7.83	**443.5**	0.00
20_30_1000_1000 − 2	600	20	30	540.3	634.0	0.00	542.8	25.75	**535.2**	12.2
20_30_1000_1000 − 3	600	20	30	633.3	636.5	0.00	**469.5**	2.75	479.9	26.55
Non-Euclidean Instances				TS2	VNDS		SA		VNS	
Name	$\|V\|$	r	$\|V\|/r$	$C(T)$	$\overline{C(T)}$	std dev	$\overline{C(T)}$	std dev	$\overline{C(T)}$	std dev
20_10_1000 − 1	200	20	10	**71.6**	94.7	0.00	76.9	0.21	**71.6**	0.02
20_10_1000 − 2	200	20	10	**41.0**	76.6	0.00	41.1	0.02	**41.0**	0.00
20_10_1000 − 3	200	20	10	**52.8**	75.3	0.00	86.9	5.38	**52.8**	0.00
100_5_1000 − 1	500	100	5	**143.7**	203.2	0.00	200.3	4.44	173.4	8.40
100_5_1000 − 2	500	100	5	**132.7**	187.3	0.00	194.3	1.20	154.6	6.55
100_5_1000 − 3	500	100	5	**162.3**	197.4	0.00	205.6	0.00	180.1	3.67
20_30_1000 − 1	600	20	30	**14.5**	59.4	0.00	22.7	1.49	15.9	2.07
20_30_1000 − 2	600	20	30	17.7	23.7	0.00	22.0	0.82	**17.6**	1.75
20_30_1000 − 3	600	20	30	**15.1**	29.5	0.00	22.1	0.44	**15.1**	0.22

Table 1: Results on Ghosh [5] and newly created ($|V| = 600$) instances, 600s CPU-time (except SA).

indicated in the results tables) had been reached. In contrast, SA was run for a fixed number of iterations as specified in [9], which leaded to a much longer running time compared to the others.

In Table 1 and 2 we show instance names, numbers of nodes, numbers of clusters, average numbers of nodes per cluster, the (average) objective values and corresponding standard deviations of the final solutions of TS2, VNDS, SA, and VNS. All instances of Table 1 containing 600 nodes are new. The best values are printed in bold.

These results show that our VNS approach can compete well with Ghosh's TS2 and most of the

TSPlib Instances					TS2	VNDS		SA		VNS	
Name	$\|V\|$	r	$\|V\|/r$	time	$\overline{C(T)}$	$\overline{C(T)}$	std dev	$\overline{C(T)}$	std dev	$\overline{C(T)}$	std dev
gr137	137	28	5	150s	**329.0**	330.0	0.00	352.0	0.00	**329.0**	0.00
kroa150	150	30	5	150s	**9815.0**	**9815.0**	0.00	10885.6	25.63	**9815.0**	0.00
krob200	200	40	5	300s	11245.0	11353.0	0.00	12532.0	0.00	**11244.0**	0.00
ts225	225	45	5	300s	62366.0	63139.0	0.00	67195.1	34.49	**62280.5**	16.28
gil262	262	53	5	300s	**942.0**	979.0	0.00	1022.0	0.00	943.2	1.63
pr264	264	54	5	300s	**21886.0**	22115.0	0.00	23445.8	68.27	21890.8	5.92
pr299	299	60	5	450s	**20339.0**	20578.0	0.00	22989.4	11.58	20347.4	28.09
lin318	318	64	5	450s	18521.0	18533.0	0.00	20268.0	0.00	**18511.2**	9.70
rd400	400	80	5	600s	**5943.0**	6056.0	0.00	6440.8	3.40	5955.0	7.57
fl417	417	84	5	600s	7990.0	7984.0	0.00	8076.0	0.00	**7982.0**	0.00
gr431	431	87	5	600s	1034.0	1036.0	0.00	1080.5	0.51	**1033.0**	0.25
pr439	439	88	5	600s	51852.0	52104.0	0.00	55694.1	45.88	**51849.7**	39.30
pcb442	442	89	5	600s	**19621.0**	19961.0	0.00	21516.0	5.15	19729.3	50.90

Table 2: Results on TSPlib instances with geographical clustering, variable CPU-time.

time outperforms VNDS and Pop's SA. Compared to TS2, our algorithm provides significantly better results on random Euclidean instances in the case when clusters contain many nodes. On Ghosh-instances with five nodes per cluster, TS2 generally yields better results. In all other cases and on TSPlib instances, both algorithms provide comparable results.

6 Conclusion and Future Work

In this paper, we proposed a powerful VNS approach for solving the Generalized Minimum Spanning Tree problem by combining two complementary types of large neighborhoods. They can be seen as dual to each other. Results show that this concept outperforms previous metaheuristic approaches in particular on instances with a large number of nodes per cluster. This is due to the strength of the dynamic programming process of the Global Edge Exchange Neighborhood for computing the optimal node selection for a given global spanning tree.

In future work, we plan to consider further, more sophisticated neighborhoods for the VND. For example, they can be based on the existing ILP formulations and various ILP techniques can be used for finding the best neighbors.

References

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

[2] C. Feremans. *Generalized Spanning Trees and Extensions*. PhD thesis, Universite Libre de Bruxelles, 2001.

[3] C. Feremans and A. Grigoriev. An approximation scheme for the generalized geometric minimum spanning tree problem with grid clustering. Technical Report NEP-ALL-2004-09-30, Maastricht: METEOR, Maastricht Research School of Economics of Technology and Organization, 2004.

[4] C. Feremans, M. Labbe, and G. Laporte. A comparative analysis of several formulations for the generalized minimum spanning tree problem. *Networks*, 39(1):29–34, 2002.

[5] D. Ghosh. Solving medium to large sized Euclidean generalized minimum spanning tree problems. Technical Report NEP-CMP-2003-09-28, Indian Institute of Management, Research and Publication Department, Ahmedabad, India, 2003.

[6] P. Hansen and N. Mladenovic. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-heuristics, Advances and trends in local search paradigms for optimization*, pages 433–458. Kluwer Academic Publishers, 1999.

[7] P. Hansen and N. Mladenovic. A tutorial on variable neighborhood search. Technical Report G-2003-46, Les Cahiers du GERAD, HEC Montreal and GERAD, Canada, 2003.

[8] Y. S. Myung, C. H. Lee, and D. W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26:231–241, 1995.

[9] P. C. Pop. *The Generalized Minimum Spanning Tree Problem*. PhD thesis, University of Twente, The Netherlands, 2002.

[10] P. C. Pop, G. Still, and W. Kern. An approximation algorithm for the generalized minimum spanning tree problem with bounded cluster size. In H. Broersma, M. Johnson, and S. Szeider, editors, *Algorithms and Complexity in Durham 2005, Proceedings of the first ACiD Workshop*, volume 4 of *Texts in Algorithmics*, pages 115–121. King's College Publications, 2005.