TECHNISCHE UNIVERSITÄT WIEN
Institut für Computergraphik und Algorithmen

The Generalized Minimum Edge Biconnected Network Problem: Efficient Neighborhood Structures for Variable Neighborhood Search

Bin Hu and Markus Leitner and Günther R. Raidl

Forschungsbericht / Technical Report

TR–186–1–07–02

17. November 2009

Favoritenstraße 9-11 / E186, A-1040 Wien, Austria
Tel. +43 (1) 58801-18601, Fax +43 (1) 58801-18699
www.cg.tuwien.ac.at

The Generalized Minimum Edge-Biconnected Network Problem: Efficient Neighborhood Structures for Variable Neighborhood Search

Bin Hu[1], Markus Leitner[2], Günther R. Raidl[1] *

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

[2]Department of Telematics and Network Engineering
Carinthia University of Applied Sciences, Austria
markus.leitner@fh-klagenfurt.at

Abstract

We consider the generalized minimum edge-biconnected network problem where the nodes of a graph are partitioned into clusters and exactly one node from each cluster is required to be connected in an edge-biconnected way. Instances of this problem appear, for example, in the design of survivable backbone networks. We present different variants of a variable neighborhood search approach that utilize different types of neighborhood structures, each of them addressing particular properties as spanned nodes and/or the edges between them. For the more complex neighborhood structures, we apply efficient techniques – such as a graph reduction – to essentially speed up the search process. For comparison purposes, we use a mixed integer linear programming formulation based on multi-commodity flows to solve smaller instances of this problem to proven optimality. Experiments on such instances indicate that the variable neighborhood search is also able to identify optimal solutions in the majority of test runs, but within substantially less time. Tests on larger Euclidean and random instances with up to 1280 nodes, which could not be solved to optimality by mixed integer programming, further document the efficiency of the variable neighborhood search. In particular, all proposed neighborhood structures are shown to contribute significantly to the search process.

Keywords: Network Design, Biconnectivity, Variable Neighborhood Search, Mixed Integer Programming

*This work is supported by the European Marie Curie RTN ADONET under grant 504438 and by the Austrian Research Promotion Agency (FFG) under grant 811378.

1 Introduction

The Generalized Minimum Edge-Biconnected Network Problem (GMEBCNP) is defined as follows. We consider a complete, undirected weighted graph $G = \langle V, E, c \rangle$ with node set V, edge set E, and edge cost function $c : E \to \mathbb{R}^+$. Node set V is partitioned into r pairwise disjoint clusters $V_1, V_2, \ldots, V_r,\ \bigcup_{i=1}^{r} V_i = V,\ V_i \cap V_j = \emptyset\ \forall i, j = 1, \ldots, r,\ i \neq j$.

A solution to the GMEBCNP defined on G is a subgraph $S = \langle P, T \rangle,\ P = \{p_1, \ldots, p_r\} \subseteq V$ connecting exactly one node from each cluster, i.e., $p_i \in V_i,\ \forall i = 1, \ldots, r$, and containing no bridges [4, 15, 17]; see Figure 1. A bridge is an edge which does not lie on any cycle and thus its removal would disconnect the graph. The cost of such an edge-biconnected network is its total edge cost, i.e., $c(T) = \sum_{(u,v)\in T} c(u, v)$, and the objective is to identify a feasible solution with minimum cost. This problem is $\mathcal{NP}$-hard since the task of finding a minimum cost biconnected network spanning all nodes of a given graph is already $\mathcal{NP}$-hard [4, 6], which is the special case with $|V_i| = 1,\ \forall i = 1, \ldots, r$.

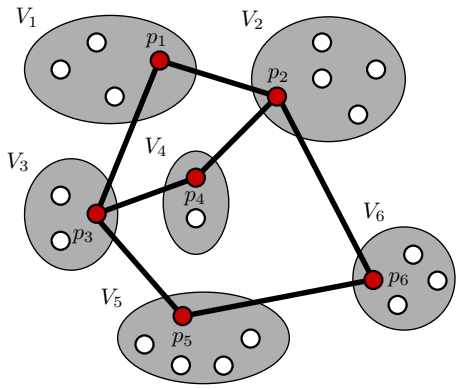

Figure 1: Example for a solution to the GMEBCNP.

The GMEBCNP was introduced by Huygens [15] and it arises in the design of backbones in large communication networks. For example, we can consider the possible access points of an existing local network as nodes of a cluster when designing a backbone network connecting multiple LANs. Survivability by means of a single link outage is covered via the considered edge redundancy [4].

In this paper, we propose variants of an improved *Variable Neighborhood Search* (VNS) approach for the GMEBCNP. VNS, combined with *Variable Neighborhood Descent* (VND) as local improvement procedure, is a metaheuristic which exploits systematically the idea of changing between different types of neighborhoods to head for superior local optima as well as a mechanism called shaking for reaching under-explored areas of the search space. For a more detailed description of VNS, see [10, 11]. We also propose a *Mixed Integer Programming* (MIP) formulation based on multi-commodity flows for solving smaller instances of this problem to provable optimality.

The remainder of this article is organized as follows. In Section 2, we give an overview of research done on the GMEBCNP and other related problems. Section 3 describes the components of our VNS approach in detail and Section 4 introduces the MIP formulation. Section 5 describes the instances we used for our computational experiments. Finally, we show the experimental results in Section 6 and conclude in Section 7.

2 Previous Work

Despite the importance of this problem in survivable network design, not much research has been done on this particular problem until now. Huygens [15] studied the GMEBCNP and provided integer programming formulations along with separation techniques, but no practical results on actual instances were published.

Though not identical, the GMEBCNP is related to the Generalized Minimum Spanning Tree Problem (GMSTP) [18]. As a matter of course, some concepts can be adopted from there. Hu et al. [13] approached the GMSTP from two different directions by utilizing dual representations and associated neighborhood structures within a VNS. By fixing the spanned nodes of each cluster, optimal edges can be efficiently computed via Kruskal's MST algorithm. On the other hand, by fixing the connections between clusters, an optimal choice of spanned nodes can be determined via dynamic programming in polynomial time. Though these concepts are not directly applicable for the GMEBCNP due to higher complexity, some neighborhood structures of the current work are also based on these ideas.

Another related problem is the Generalized Traveling Salesman Problem (GTSP) [12, 20, 21] that is also $\mathcal{NP}$-hard. Since every solution to the GTSP is obviously edge-biconnected and therefore also a solution to the GMEBCNP, its solution value can be regarded as an upper bound to the current problem. However, the GTSP is hard to handle as well, and especially on large graphs, these upper bounds become rather poor as the overall costs of solutions to the GMEBCNP can be substantially lower. Therefore, we will not consider the GTSP any further in this article.

The classical minimum edge-biconnected network problem has been shown to be $\mathcal{NP}$-hard by a reduction from the Hamiltonian cycle problem [6]. Khuller and Vishkin [16] proposed a factor two approximation algorithm and showed that approximating the optimal solution to within some additive constant is impossible in polynomial time unless P=NP. Czumaj and Lingas [2] presented more detailed results with respect to the approximability of the classical problem and gave a polynomial-time approximation scheme for the case of complete Euclidean graphs in $\mathbb{R}^d$.

3 Variable Neighborhood Search for the GMEBCNP

In this section, we will first describe the solution representation and the initialization procedure, then discuss our neighborhood structures along with techniques to optimize the search process. Finally we assemble our VND and VNS framework.

3.1 Terminology

For solution representation, graph reduction, and neighborhood structures, we use following terminology.

The global graph, denoted by $G^{\mathrm{g}} = \langle V^{\mathrm{g}}, E^{\mathrm{g}} \rangle$, consists of nodes corresponding to clusters in G, i.e., $V^{\mathrm{g}} = \{V_1, V_2, \ldots, V_r\}$, and the complete edge set $E^{\mathrm{g}} = \{(V_i, V_j) \mid V_i \in V^{\mathrm{g}} \wedge V_j \in V^{\mathrm{g}}\}$. Hereby, each *global connection* (V_i, V_j) represents all edges $\{(u, v) \in E \mid u \in V_i \wedge v \in V_j\}$ of graph G.

When given some feasible candidate solution $S = \langle P, T \rangle$ to the GMEBCNP, its corresponding *global structure* is defined as the induced global graph's subgraph $S^{\mathrm{g}} = \langle V^{\mathrm{g}}, T^{\mathrm{g}} \rangle$ with the global connections $T^{\mathrm{g}} = \{(V_i, V_j) \in E^{\mathrm{g}} \mid \exists (u, v) \in T \wedge u \in V_i \wedge v \in V_j\}$; see Figure 2.

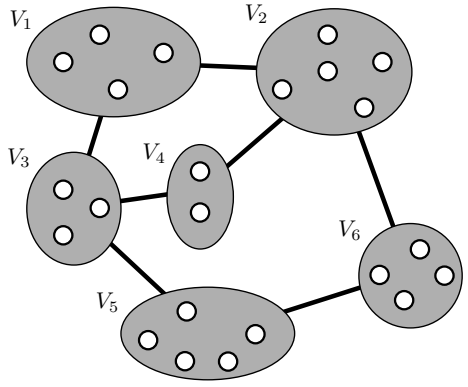

Figure 2: Example for the global structure of the solution in Figure 1.

Redundant edges of a candidate solution $S = \langle P, T \rangle$ are edges that can be removed without violating the edge-biconnectivity property.

3.2 Solution Representation

For each solution, we store the spanned nodes $P = \{p_1, \ldots, p_r\}$ and the global connections T^{g}. Spanned nodes $p_1, \ldots, p_r$ alone are insufficient to represent a solution, as finding the cheapest edges for them corresponds to the classical minimum edge-biconnected network problem which is $\mathcal{NP}$-hard [4, 6]. Similarly, a representation via global connections alone is also insufficient, since identifying a set of optimal nodes when restricted to a given global structure is also $\mathcal{NP}$-hard. Since the latter is not obvious, we prove it by a reduction from the graph coloring problem; for more details see also [17].

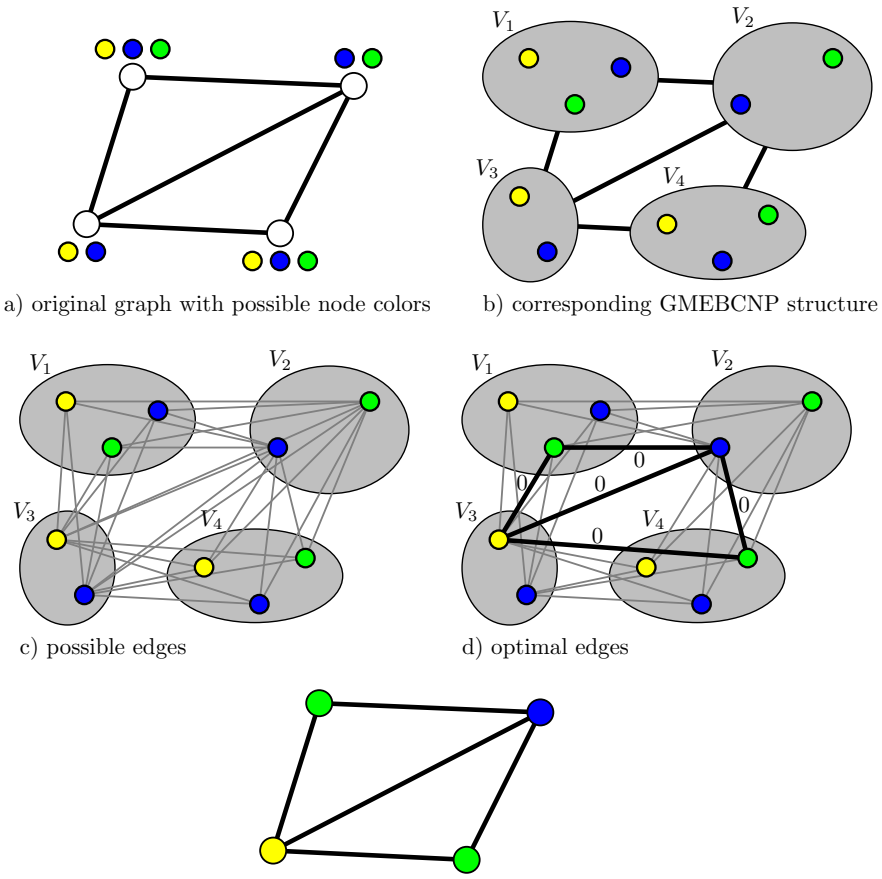

Figure 3: Transformation of the graph coloring problem into the problem of identifying an optimal node selection w.r.t. a given global structure.

Theorem 1 *Given an (edge-biconnected) global structure* $S^{\mathrm{g}} = \langle V^{\mathrm{g}}, T^{\mathrm{g}} \rangle$, $T^{\mathrm{g}} \subseteq E^{\mathrm{g}}$, *it is* $\mathcal{NP}$*-hard to identify an optimal selection of nodes* P *yielding a corresponding minimum cost GMEBCNP solution.*

Proof Consider the classical $\mathcal{NP}$-hard graph coloring problem [6] on an undirected graph $H = \langle U, F \rangle$ (Figure 3a): To each node, one color of a restricted set of colors needs to be assigned in such a way that any pair of adjacent nodes is differently colored. We consider the input graph H as the global structure S^{g}, and the clustered graph $G = \langle V, E \rangle$ is derived by the following procedure: Each node $i \in U$ becomes a cluster V_i and for each possible color c of i, we introduce a node v_i^c in cluster V_i (Figure 3b). For each edge $(i, j) \in F$, we create in the clustered graph edges (v_i^c, v_j^d) $\forall\ v_i^c \in V_i,\ \forall\ v_j^d \in V_j$ (Figure 3c). An edge's cost is 0 if $c \neq d$ and 1 otherwise.

If we are able to solve the problem of identifying the optimal nodes of the clusters in order to minimize the GMEBCNP's solution cost (Figure 3d), we also solve the original graph coloring problem on H: Suppose v_i^c is the selected node in cluster V_i, then c becomes the color of node $i \in U$ (Figure 3e). The validity of the

theorem thus follows from the $\mathcal{NP}$-hardness of the graph coloring problem. □

So far we considered arbitrary global structures that may contain redundant edges. As in our VND, we always only deal with edge-minimal global structures, i.e., global structures without redundant edges, we further show that the $\mathcal{NP}$-hardness even holds for this special case.

Theorem 2 *Given an edge-minimal edge-biconnected global structure $S^{\mathrm{g}} = \langle V^{\mathrm{g}}, T^{\mathrm{g}} \rangle$, $T^{\mathrm{g}} \subseteq E^{\mathrm{g}}$, it is $\mathcal{NP}$-hard to identify an optimal selection of nodes P yielding a corresponding minimum cost GMEBCNP solution.*

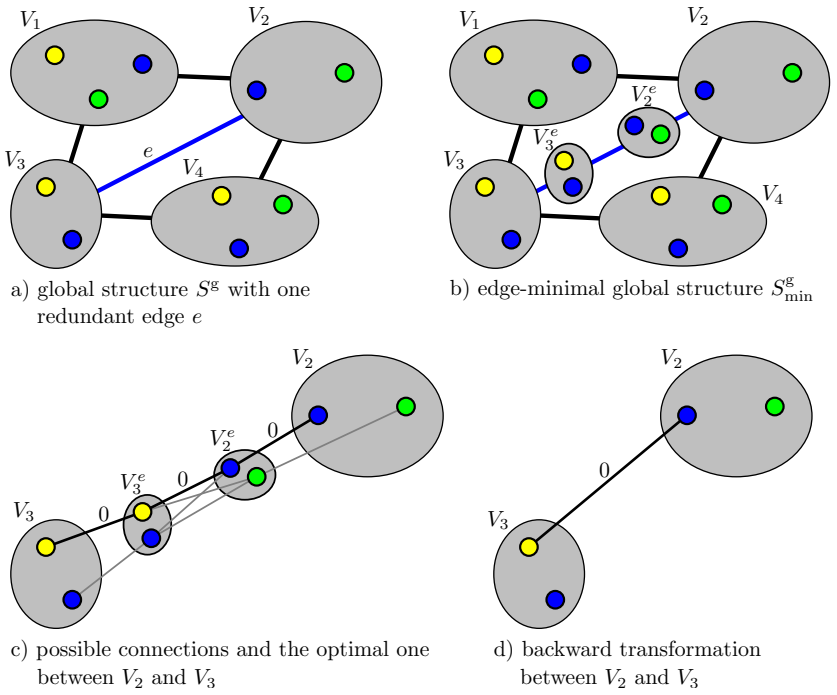

Figure 4: Transformation of the graph coloring problem: Extension towards an edge-minimal global structure.

Proof If the global structure S^{g}, after the previous transformation, is not edge-minimal, T^{g} contains at least one redundant connection (Figure 4a). For each such redundant connection $e = (V_i, V_j) \in T^{\mathrm{g}}$, we add new artificial clusters V_i^e and V_j^e, which are exact copies of V_i and V_j, respectively. The global connection (V_i, V_j) gets replaced by (V_i, V_i^e), (V_i^e, V_j^e), and (V_j^e, V_j) (Figure 4b). Let $S^{\mathrm{g}}_{\mathrm{min}} = \langle V^{\mathrm{g}}_{\mathrm{min}}, T^{\mathrm{g}}_{\mathrm{min}} \rangle$ denote the resulting structure, which obviously is edge-minimal.

When adding the clusters V_i^e and V_j^e, we have to modify the edges E in the clustered graph G as well. We replace each edge $(u, v) \in E$ with $u \in V_i \wedge v \in V_j$ by (u^e, v^e) with $u^e \in V_i^e \wedge v^e \in V_j^e$ where u^e and v^e are the new copies of u and v, respectively. Between V_i and V_i^e, we add edges (u, u^e) with costs 0 for all $u \in V_i$. The same procedure is applied for V_j and V_j^e (Figure 4c). Let $G' = \langle V', E', c \rangle$ denote the resulting modified graph. By determining an optimal selection of nodes in G' subject to the global structure $T^{\mathrm{g}}_{\mathrm{min}}$, we

get the optimal node set in G subject to the global structure T^{g} by simply ignoring the artificial clusters (Figure 4d). Thus, we also obtain the optimal solution to the original graph coloring problem on H by choosing the corresponding colors.

The backward transformation is valid because only one node (hence one color) is chosen per cluster as we solve the GMEBCNP containing exactly one node per cluster. Furthermore, the selected node of a cloned cluster is always the clone of the selected node in the original cluster due to the zero cost edges. □

3.3 Creating an Initial Solution

Our strategy for determining an initial solution for the GMEBCNP is inspired by the Christofides heuristic for the traveling salesman problem [1] and therefore is called the *Adapted Christofides Heuristic* (ACH). Its pseudo-code is listed in Algorithms 1 and 2. We start with a solution to the Generalized Minimum Spanning Tree Problem computed via the *Improved Kruskal Heuristic* (IKH) from Golden et al. [8]. This algorithm considers edges in increasing cost-order and adds an edge to the solution iff it does not introduce a cycle and does not connect a second node of any cluster. By fixing an initial node to be in the resulting generalized spanning tree, different solutions can be obtained. Therefore, this process is carried out $|V|$ times, once for each node to be initially fixed, and the overall cheapest spanning tree is adopted.

To augment this spanning tree to become a valid solution for the GMEBCNP, we then determine the set V_{o} of nodes with odd degree $\deg(v)$ and sort the edges induced by V_{o} and not contained in the spanning tree with respect to increasing edge costs. Next, we derive a matching T_M for the node set V_{o} by iterating through these edges and adopting any edge incident to two yet uncovered nodes until all nodes in V_{o} are covered. Note that this procedure, shown in Algorithm 2, does not necessarily generate a minimum cost matching.

Unfortunately, these steps still do not necessarily yield a solution completely satisfying the edge-biconnectivity property. More precisely, ACH will fail to find a perfect matching if the last two uncovered nodes u' and v' are adjacent in the spanning tree. However, it is easy to see that $S_0 = \langle P, T_0 \cup T_{\mathrm{M}} \rangle$ will consist of at most two edge-biconnected components even if no perfect matching is found, as both eventually existing components of $\langle P, T_0 \cup T_{\mathrm{M}} \setminus \{(u', v')\} \rangle$ are Eulerian in that case. Therefore Algorithm 1 adds the cheapest edge not yet part of the solution between the eventually remaining two edge-biconnected components.

At the end, we remove redundant edges which might occur due to the previous step with regard to decreasing edge costs. Ties that might appear due to edges having identical costs are broken at random.

The overall time complexity of ACH is $O(|E| \log |E| + r^3)$. The most expensive operations are generating a solution to the GMSTP by IKH with complexity $O(|E| \log |E|)$ [8], and finding and removing the redundant edges which can be done with complexity $O(r^3)$.

Algorithm 1: Adapted Christofides Heuristic

$S_0 = \langle P, T_0 \rangle$ = feasible GMST computed via *Improved Kruskal Heuristic*
T_{M} = compute matching (S_0) *// see Algorithm 2*
$S_0 = \langle P, T_0 \cup T_{\mathrm{M}} \rangle$
if S_0 has two edge-biconnected components **then**
  add cheapest edge $\in E \setminus T_0$ between the two edge-biconnected components
remove redundant edges

Algorithm 2: compute matching (GMST $S_0 = \langle P, T_0 \rangle$)

$T_{\mathrm{M}} = \emptyset$
$V_{\mathrm{o}} = \{v \in P \mid \deg(v) \text{ is odd}\}$
$E_{\mathrm{o}} = \{(u, v) \in E \mid u, v \in V_{\mathrm{o}} \wedge (u, v) \notin T_0\}$
sort E_{o} according to increasing costs, i.e., $c(e_1) \leq \cdots \leq c(e_{|E_{\mathrm{o}}|})$
$i = 1$
while $V_{\mathrm{o}} \neq \emptyset \wedge i < |E_{\mathrm{o}}|$ **do**
  // current edge $e_i = (u_i, v_i)$
  if $u_i \in V_{\mathrm{o}} \wedge v_i \in V_{\mathrm{o}}$ **then**
    $T_{\mathrm{M}} = T_{\mathrm{M}} \cup \{e_i\}$
    $V_{\mathrm{o}} = V_{\mathrm{o}} \setminus \{u_i, v_i\}$
  $i = i + 1$
return T_{M}

3.4 Neighborhood Structures

We propose four different types of neighborhood structures, each of them focusing on different aspects of solutions to the GMEBCNP. For two of them, there exist simple versions and advanced versions making use of the following graph reduction technique.

Graph Reduction: Though it is generally not possible to derive an optimal set of spanned nodes in polynomial time when a global structure S^{g} is given, this task becomes feasible once the spanned nodes in a few specific clusters are fixed. The underlying concept, called *graph reduction*, is based on the observation that good solutions to the GMEBCNP usually consist of only few clusters with spanned nodes of degree greater than two (*branching clusters*) and long paths of clusters with spanned nodes of degree two connecting them (*path clusters*). Once the spanned nodes within all branching clusters are fixed, it is possible to efficiently determine for each cluster path the optimal selection of remaining nodes by computing the shortest path between the two fixed branching cluster nodes in the subgraph of G represented by the cluster path.

Formally, for any global structure $S^{\mathrm{g}} = \langle V^{\mathrm{g}}, T^{\mathrm{g}} \rangle$, we can define a *reduced global structure* $S^{\mathrm{g}}_{\mathrm{red}} = \langle V^{\mathrm{g}}_{\mathrm{red}}, T^{\mathrm{g}}_{\mathrm{red}} \rangle$. V_{red} denotes the branching clusters, i.e., $V^{\mathrm{g}}_{\mathrm{red}} = \{V_i \in V^{\mathrm{g}} \mid \deg(V_i) \geq 3\}$ with $\deg(V_i)$ being the degree of cluster V_i in S^{g}. $T^{\mathrm{g}}_{\mathrm{red}}$ consists of edges which represent strings of path clusters connecting these branching clusters, i.e., $T^{\mathrm{g}}_{\mathrm{red}} = \{(V_a, V_b) \mid (V_a, V_{k_1}), (V_{k_1}, V_{k_2}), \ldots, (V_{k_{l-1}}, V_{k_l}), (V_{k_l}, V_b) \in T^{\mathrm{g}} \wedge V_a, V_b \in V^{\mathrm{g}}_{\mathrm{red}} \wedge V_{k_i} \notin V^{\mathrm{g}}_{\mathrm{red}},\ \forall i = 1, \ldots, l\}$.

Corresponding to the reduced global structure $S^{\mathrm{g}}_{\mathrm{red}} = \langle V^{\mathrm{g}}_{\mathrm{red}}, T^{\mathrm{g}}_{\mathrm{red}} \rangle$ we can define a *reduced graph* $G_{\mathrm{red}} = \langle V_{\mathrm{red}}, E_{\mathrm{red}} \rangle$ with all nodes of branching clusters $V_{\mathrm{red}} = \{v \in V_i \mid V_i \in V^{\mathrm{g}}_{\mathrm{red}}\}$ and edges between any pair of nodes whose clusters are adjacent in the reduced global structure, i.e., $(i,j) \in E_{\mathrm{red}} \Leftrightarrow (V_i, V_j) \in T^{\mathrm{g}}_{\mathrm{red}}, \forall i \in V_i, j \in V_j$. Each such edge (i,j) corresponds to the shortest path connecting i and j in the subgraph of G represented by the reduced structure's edge (V_i, V_j), and (i,j) therefore gets assigned this shortest path's cost; see Figure 5.

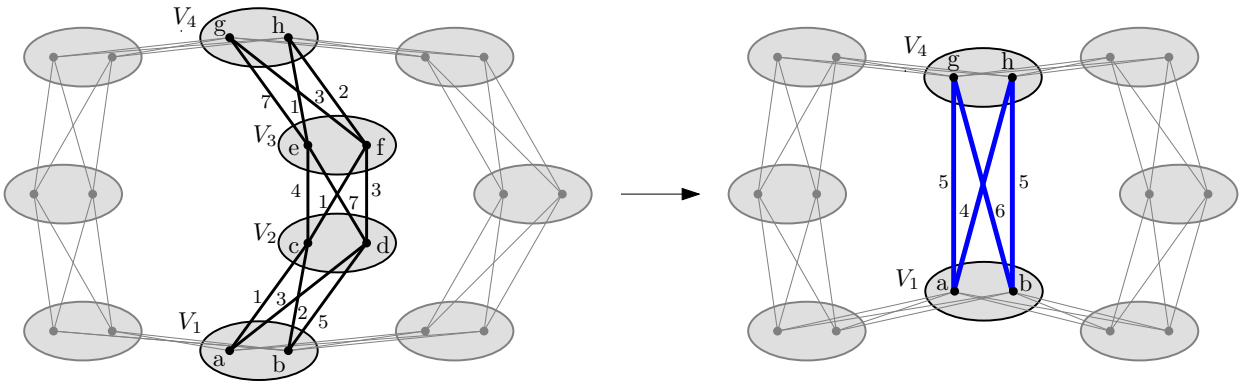

Figure 5: Computing the shortest paths between all node pairs of two branching clusters V_1 and V_4.

When fixing the spanned nodes in $V^{\mathrm{g}}_{\mathrm{red}}$ we can determine the cost of the corresponding solution S with optimally chosen nodes in path clusters efficiently by using the precomputed shortest path costs stored with the reduced graph's edges. Decoding the corresponding solution, i.e., making the optimal spanned nodes within path clusters explicit, is done by choosing all nodes on the shortest paths corresponding to used edges from E_{red}.

For details on how the graph reduction can be efficiently implemented, we refer to [17]. An edge-minimal solution to the GMEBCNP, as it is obtained from our initialization procedure, may consist of $O(r)$ edges only. When computing the corresponding reduced global structure and reduced graph, each solution edge is considered exactly once, and for each edge all combinations of nodes within three clusters need to be considered. The overall worst case time complexity is thus $O(r \cdot d^3_{\max})$, with $d_{\max}$ being the maximum number of nodes within a single cluster.

Figure 6 shows an example of reducing the number of clusters to be further considered from nine to two. Note that cyclic paths in T^{g}, i.e., $V_a = V_b$, will yield loops in $T^{\mathrm{g}}_{\mathrm{red}}$, as is the case with (V_6, V_6) in our example. Furthermore, multiple cluster paths may exist between two branching clusters, as for V_1 and V_6, and they lead to multi-edges in the reduced graph. We can get rid of such multi-edges by replacing them with corresponding simple edges and summing up the costs.

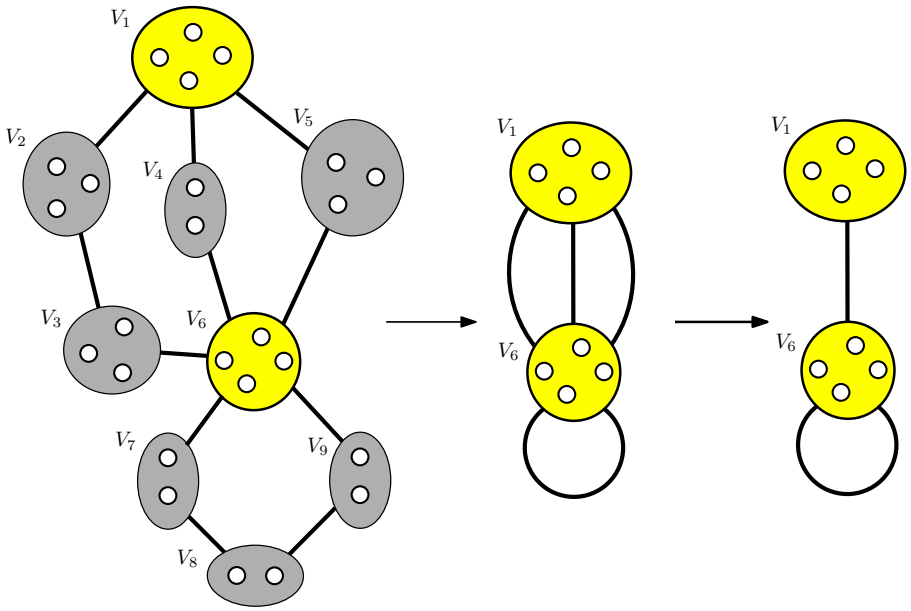

Figure 6: Example for graph reduction: V_1 and V_6 are branching clusters, while all others are path clusters.

3.4.1 Simple Node Optimization Neighborhood (SNON)

With this neighborhood structure we try to optimize a solution with respect to the spanned nodes within clusters while keeping the global structure. SNON consists of all solutions S' that differ from the current solution S by exactly one spanned node. A move within SNON (see Figure 7) is accomplished by changing $p_i \in V_i$ to $p'_i \in V_i, p_i \neq p'_i$, for $i \in \{1, \ldots, r\}$, removing all edges incident to p_i and adding edges from p'_i to all nodes that were incident to p_i in S; see Algorithm 3.

As the objective value can be updated in an incremental way, the time complexity of a complete search in SNON is $O(|V| \cdot d_{\max})$.

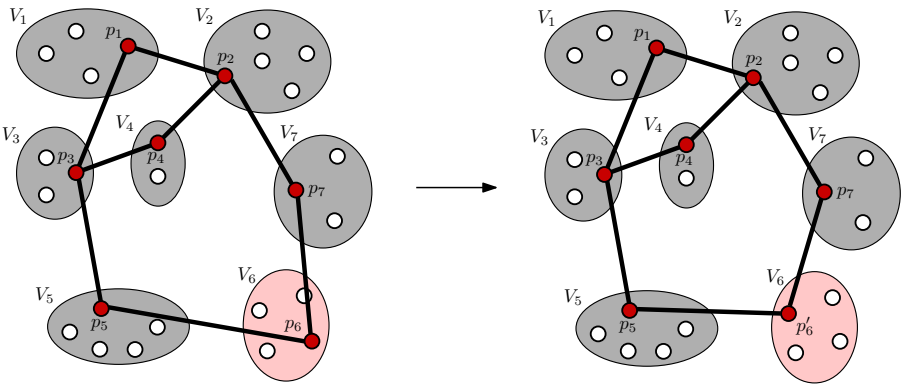

Figure 7: A SNON move, changing the spanned node of V_6 from p_6 to p'_6.

Algorithm 3: Simple Node Optimization (solution $S = \langle P, T \rangle$)

```
for i = 1, ..., r do
    forall v ∈ V_i \ p_i do
        change spanned node p_i of cluster V_i to v
        if current solution better than best then
            save current solution as best
        restore initial solution
restore and return best solution
```

3.4.2 Node Optimization Neighborhood (NON)

This neighborhood structure enhances SNON by utilizing the graph reduction technique. NON consists of all solutions S' that differ from S by at most two spanned nodes within branching clusters. Again, the global structure of the solution remains unchanged. By means of the graph reduction technique, spanned nodes of path clusters are selected in an optimal way once the best neighboring solution is identified on the reduced graph; see Algorithm 4.

Carrying out graph reduction in advance adds $O(r \cdot d_{\max}^3)$ to the time complexity. Since updating the objective value for a considered neighbor can be done in $O(d_{\max})$ time and $O(r^2)$ neighbors are to be considered, the overall time complexity of NON is $O(r^2 \cdot d_{\max}^2 + r \cdot d_{\max}^3)$.

Algorithm 4: Node Optimization (solution $S = \langle P, T \rangle$)

```
compute reduced structure S^g_red = ⟨V^g_red, T^g_red⟩
forall V_i, V_j ∈ V^g_red ∧ V_i ≠ V_j do
    forall u ∈ V_i ≠ p_i do
        change used node p_i of cluster V_i to u
        forall v ∈ V_j do
            change used node p_j of cluster V_j to v
            if current solution better than best then
                save current solution as best
            restore initial solution
restore best solution // fixes the spanned nodes in branching clusters
decode solution // by using precomputed shortest paths corresponding to used edges in E_red
return solution
```

3.4.3 Node Re-Arrangement Neighborhood (NRAN)

With this neighborhood structure we try to optimize a solution with respect to the arrangement of nodes. A neighbor solution S' in NRAN differs from S by exactly one swap move which exchanges for two nodes a and b their sets of adjacent nodes I_a and I_b as shown in Figure 8. Set I_a, with respect to solution $S = \langle P, T \rangle$, is defined as $I_a = \{w \in P \mid (a, w) \in T\}$. After this swap move, $S' = \langle P, T' \rangle$ consists of $T' = T \setminus I_a \setminus I_b \cup \{(a, v) \mid v \in I_b\} \cup \{(b, u) \mid u \in I_a\}$. The pseudocode for completely searching this neighborhood is given in Algorithm 5.

Updating the objective value for a single move means to subtract the costs of the original edges and to add the costs of the new ones. Therefore, a complete evaluation of NRAN, which consists of all solutions S' differing from S by exactly one swap move, can be done in time $O(r^2 \cdot d_{\max})$.

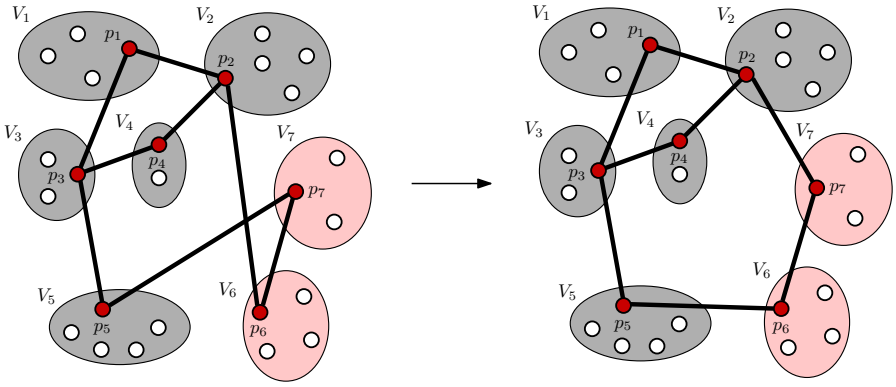

Figure 8: A NRAN move, swapping p_6 and p_7.

Algorithm 5: Node Re-Arrangement Optimization (solution $S = \langle P, T \rangle$)

```
for i = 1, ..., r − 1 do
    for j = i + 1, ..., r do
        swap adjacency lists of nodes p_i and p_j
        if current solution better than best then
            save current solution as best
        restore initial solution
restore and return best solution
```

3.4.4 Cluster Re-Arrangement Neighborhood (CRAN)

This neighborhood structure is an extension to NRAN which again makes use of the graph reduction technique. Moving from the current solution S to a neighbor solution S' in CRAN means swapping two nodes in an analogous way as for NRAN, then computing the reduced graph, and finally determining the best nodes in all path clusters. Since applying the whole graph reduction after each move is relatively time-expensive, only incremental updates of the reduced structure and associated information are carried out whenever two nodes of path clusters are swapped, which is in practice most of the time the case. Whenever two nodes a and b of degree two on the same reduced path are swapped, only this path has to be updated; if a and b belong to different paths, only the corresponding two paths must be recomputed. However, if at least one of these nodes belongs to a branching cluster, the graph reduction procedure must be completely re-applied as the structure of the whole solution graph may change. The pseudocode is given in Algorithm 6.

The worst case time complexity of completely examining CRAN is $O(r^3 \cdot d_{\max}^3)$ when graph reduction is applied after every move. Since the complete evaluation might require too much time on larger instances, we

abort the neighborhood exploration after a certain time limit is exceeded, returning the so-far best neighbor instead of following a strict best neighbor strategy.

Algorithm 6: Cluster Re-Arrangement Optimization (solution $S = \langle P, T \rangle$)

```
compute reduced structure S^g_red = ⟨V^g_red, T^g_red⟩
for i = 1, ..., r − 1 do
    for j = i + 1, ..., r do
        swap adjacency lists of nodes p_i and p_j
        if V_i or V_j is a branching cluster then
            recompute reduced solution S^g_red = ⟨V^g_red, T^g_red⟩
        else
            if V_i and V_j belong to the same reduced path P then
                update P in S^g_red
            else
                update the path containing V_i in S^g_red
                update the path containing V_j in S^g_red
        if current solution better than best then
            decode and save current solution as best
        restore initial solution and S^g_red
restore and return best solution
```

3.4.5 Edge Augmentation Neighborhood (EAN)

In this neighborhood structure, modifications on the edges are primarily considered. More precisely, EAN of a solution $S = \langle P, T \rangle$ consists of all solutions S' reachable from S by including a single additional edge $e \notin T$ and removing other, now redundant edges; see Figure 9 and Algorithm 7. Removing e itself is not allowed since this would obviously lead to the original solution S. We do not have to consider edges $e = (a, b)$ if $\deg(a) = \deg(b) = 2$ and a and b are part of the same reduced path. In these cases, adding e would lead to a graph where e is the only redundant edge. In practice, this restriction enables a large reduction of the search space since good solutions usually consist of only few branching clusters and thus only few, but long reduced paths.

Theoretically, EAN of a solution contains at most $O(r^2)$ possible moves and removing redundant edges has time complexity $O(r^3)$. Hence the overall time complexity for evaluating EAN is $O(r^5)$.

3.4.6 Node Exchange Neighborhood (NEN)

This neighborhood structure addresses both aspects, changing the spanned nodes as well as the edges connecting them. A neighbor solution in NEN differs from the original solution by exactly one spanned node and an arbitrary number of edges. A single move within NEN is accomplished by first changing $p_i \in V_i$ to $p'_i \in V_i, p_i \neq p'_i$, and removing all edges incident to p_i. This leads to a graph consisting of at least two and no

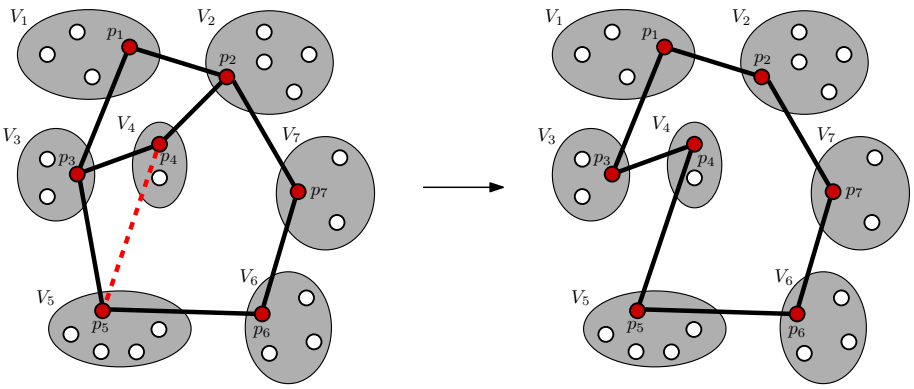

Figure 9: An EAN move, adding (p_4, p_5) and removing redundant edges (p_2, p_4) and (p_3, p_5).

Algorithm 7: Edge Augmentation Optimization (solution $S = \langle P, T \rangle$)

```
for i = 1, ..., r − 1 do
    for j = i + 1, ..., r do
        if (i, j) ∉ T then
            if deg(i) ≠ 2 ∨ deg(j) ≠ 2 ∨ i and j are not part of the same reduced path then
                add (i, j)
                remove redundant edges
                if current solution better than best then
                    save current solution as best
                restore initial solution
restore and return best solution
```

more than $\deg(p_i) + 1$ components. We reconnect these parts by adding the cheapest edges between any pair of these components. Once this step is completed, edge-biconnectivity is restored using the advanced bridge covering strategy described below. Finally, redundant edges are removed; see Figure 10 and Algorithm 8.

The process of covering all bridges with additional edges can be expensive in practice. When disconnecting a node in a sparse graph, many bridges may arise. Therefore, we first determine all nodes with degree one and connect each of them with its cheapest partner. If only a single node with degree one exists, we connect it with the first reachable node of degree greater than two. This strategy helps to cover many bridges with only few edges. Remaining bridges are covered by simply adding the cheapest edges between pairs of edge-biconnected components. Even with this advanced bridge covering strategy, examining NEN still needs $O(|V| \cdot r^3)$ time. Therefore, analogous to CRAN, we stop the search of NEN after a time limit is exceeded and return the so-far best neighbor solution.

3.5 Variable Neighborhood Descent

We use the traditional general VNS scheme with VND as local improvement as described in [10, 11]. In order to be able to investigate in particular the efficiency of the more complicated neighborhoods based on

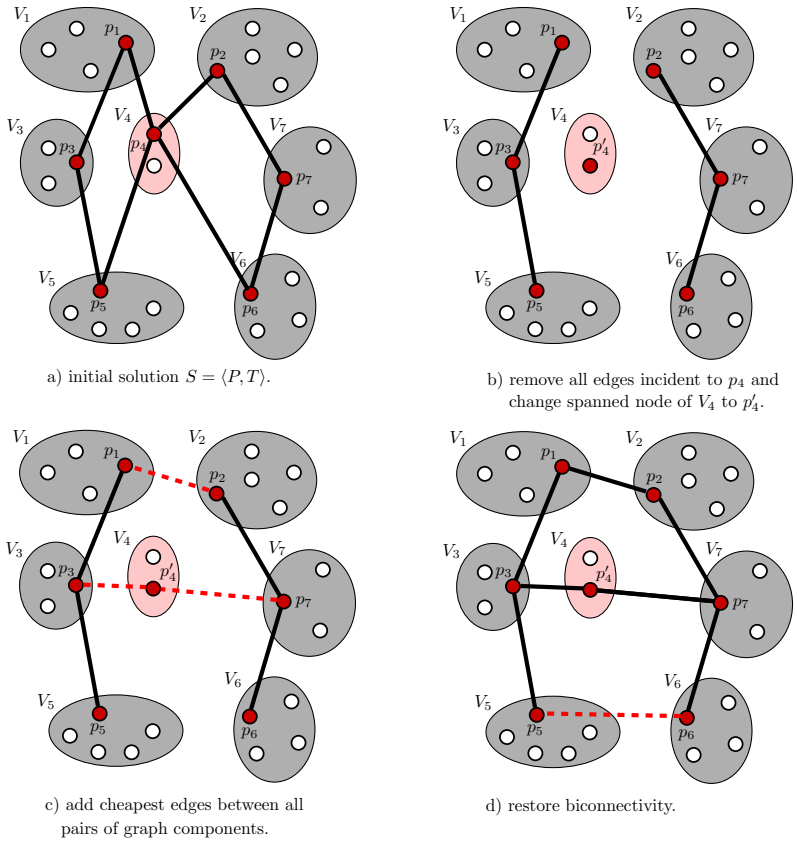

Figure 10: A NEN move, changing the spanned node of V_4, removing all adjacent edges, and re-augmenting the graph.

Algorithm 8: Node Exchange Optimization (solution $S = \langle P, T \rangle$)

```
for i = 1, ..., r do
    forall v ∈ V_i \ p_i do
        remove all edges incident to p_i
        change used node p_i of cluster V_i to v
        add cheapest edges between any two graph components
        restore biconnectivity
        remove redundant edges
        if current solution better than best then
            save current solution as best
        restore initial solution
restore and return best solution
```

graph reduction (NON, CRAN), two variants of VND differing in the used neighborhoods are considered. Furthermore, we examine the impact of using the more sophisticated *Self-Adaptive Variable Neighborhood Descent* (SAVND) with dynamic neighborhood-ordering as proposed in [14].

The first VND variant, VND1, is shown in Algorithm 9; it only applies the simpler neighborhood structures without graph reduction, i.e., SNON, NRAN, EAN, and NEN. This ordering has been determined taking

both the computational complexity as well as preliminary test results into account.

Algorithm 9: VND1 (solution $S = \langle P, T \rangle$)

```
l = 1
repeat
    switch l do
        case 1: // SNON
            S' = Simple Node Optimization (S)      // see Algorithm 3
        case 2: // NRAN
            S' = Node Re-Arrangement Optimization (S)      // see Algorithm 5
        case 3: // EAN
            S' = Edge Augmentation Optimization (S)      // see Algorithm 7
        case 4: // NEN
            S' = Node Exchange Optimization (S)      // see Algorithm 8
    if solution S' is better than S then
        S = S'
        l = 1
    else
        l = l + 1
until l > 4
return solution S
```

The second VND variant, VND2, is shown in Algorithm 10 and alternates between NON, NRAN, CRAN, EAN, and NEN. It therefore also uses the more sophisticated neighborhoods having higher computational complexity due to the applied graph reduction. SNON is not considered since it is fully contained in NON and preliminary experiments with both of them did not indicate advantages.

Algorithm 10: VND2 (solution $S = \langle P, T \rangle$)

```
l = 1
repeat
    switch l do
        case 1: // NON
            S' = Node Optimization (S)      // see Algorithm 4
        case 2: // NRAN
            S' = Node Re-Arrangement Optimization (S)      // see Algorithm 5
        case 3: // CRAN
            S' = Cluster Re-Arrangement Optimization (S)      // see Algorithm 6
        case 4: // EAN
            S' = Edge Augmentation Optimization (S)      // see Algorithm 7
        case 5: // NEN
            S' = Node Exchange Optimization (S)      // see Algorithm 8
    if solution S' is better than S then
        S = S'
        l = 1
    else
        l = l + 1
until l > 5
return solution S
```

Finally, the self-adaptive variable neighborhood descent (SAVND) uses the same neighborhood structures as VND2, but instead of a static order, the neighborhoods are rearranged automatically during the search process. Each neighborhood structure has associated a rating which is updated according to success probabilities and required times for evaluation. In this way, more effective neighborhood structures come to the fore and will be applied more frequently. For a more detailed description, see [14].

3.6 Variable Neighborhood Search Framework

The pseudocode for the general VNS scheme is given in Algorithm 11 and follows the traditional concept [10, 11].

Algorithm 11: VNS

```
create initial solution S
repeat
    k = 0
    repeat
        S' = S
        if k > 0 then
            // Shaking (S', k):
            add k randomly chosen edges from E \ T
            remove redundant edges
        VND1 (S')      // or VND2 (S') or SAVND (S')
        if solution S' is better than S then
            S = S'
            k = 1
        else
            k = k + 1
    until k == k_max
until a termination criterion is met
return solution S
```

Most of our neighborhood structures used in VND concentrate more on the optimization of the spanned nodes than on the global structure. In order to enhance diversity, our shaking procedure is therefore based on EAN. It augments a current solution by k randomly chosen new edges followed by a removal of other, now redundant edges. This process starts with $k = 1$ inserted edge, and as long as no improvement is achieved, k is incremented by one up to $k_{\max} = \lfloor \frac{r}{4} \rfloor$. In accordance to the used VND variant, we denote the three VNS variants VNS1, VNS2, and SAVNS, respectively.

4 A Mixed Integer Programming Formulation for GMEBCNP

To obtain proven optimal solutions for small and medium sized GMEBCNP instances, we propose a multi-commodity flow MIP formulation based on the local-global approach which was originally suggested for the GMSTP [19]. We use following decision variables.

$$x_{u,v} = \begin{cases} 1 & \text{if the edge } (u,v) \text{ is included in the solution} \\ 0 & \text{otherwise} \end{cases} \qquad \forall (u,v) \in E$$

$$z_v = \begin{cases} 1 & \text{if the node } v \text{ is connected in the solution} \\ 0 & \text{otherwise} \end{cases} \qquad \forall v \in V$$

$$y_{i,j} = \begin{cases} 1 & \text{if cluster } V_i \text{ is connected to cluster } V_j \text{ in the global structure} \\ 0 & \text{otherwise} \end{cases} \qquad \forall (i,j) \in E^{\mathrm{g}}$$

$$f^k_{i,j} = \begin{cases} 1 & \text{if a flow } f \text{ of commodity } k \text{ exists from cluster } i \text{ to cluster } j \\ 0 & \text{otherwise} \end{cases} \qquad \begin{array}{l} \forall i,j = 1,\ldots,r \\ \forall k = 2,\ldots,r \end{array}$$

The MIP formulation consists of two parts: The multi-commodity flow part operates on the global structure and is based on sending from cluster V_1, which is defined to be the root, two units of flow f to every other cluster using edge-disjoint routes. Flows dedicated to different clusters are distinguished by their commodity k. The result is stored in the binary variables $y_{i,j}$ indicating the global connections. The local-global part, originally introduced by Pop [19] for the GMSTP, relates the local variables $x_{u,v}$ and z_v with the global connections.

(1) minimize $\displaystyle\sum_{(u,v)\in E} c_{u,v}\ x_{u,v}$

(2) subject to $\displaystyle\sum_{i=1}^{r} f^k_{i,j} - \sum_{l=1}^{r} f^k_{j,l} = \begin{cases} -2 & \text{if } j = 1 \\ 2 & \text{if } j = k \\ 0 & \text{else} \end{cases} \qquad \forall j = 1,\ldots,r,\ \forall k = 2,\ldots,r$

(3) $f^k_{i,j} + f^k_{j,i} \le 1 \qquad \forall i,j = 1,\ldots,r,\ \forall k = 2,\ldots,r$

(4) $\displaystyle\sum_{v\in V_k} z_v = 1 \qquad \forall k = 1,\ldots,r$

(5) $\displaystyle\sum_{u\in V_i, v\in V_j} x_{u,v} = y_{i,j} \qquad \forall (i,j) \in E^{\mathrm{g}}$

$$x_{u,v} \leq z_u \qquad \forall i = 1, \ldots, r, \ , \forall u \in V_i, \ \forall v \in V \setminus V_i \tag{6}$$

$$y_{i,j} \geq f^k_{i,j} \qquad \forall i, j = 1, \ldots, r, \ i \neq j, \ \forall k = 2, \ldots, r \tag{7}$$

$$f^k_{i,j} \geq 0 \qquad \forall i, j = 1, \ldots, r, \ \forall k = 2, \ldots, r \tag{8}$$

$$x_{u,v} \in \{0, 1\} \qquad \forall (u, v) \in E \tag{9}$$

$$y_{i,j} \in \{0, 1\} \qquad \forall (i, j) \in E^{\mathrm{g}} \tag{10}$$

$$z_v \in \{0, 1\} \qquad \forall v \in V \tag{11}$$

Constraints (2) ensure that two commodities k of flow f are produced in V_1, preserved by every cluster they are not dedicated for, and consumed by cluster V_k. To achieve edge-biconnectivity, inequalities (3) forbid the transportation of two commodities dedicated for the same cluster over the same connection. To obtain a valid global structure, inequalities (7) ensure global connections to be included in the solution if a flow variable is active on it. Constraints (4) guarantee that precisely one node is selected per cluster and equations (5) only allow edges between nodes of clusters which are connected in the global structure. Finally, inequalities (6) ensure that only edges incident to selected nodes are chosen.

5 Test Instances

We tested our algorithms on Euclidean TSPlib[1] instances with geographical center clustering according to [3, 5] and random instances originally introduced by Ghosh [7] for the GMSTP.

Geographical clustering on TSPlib instances is done as follows. First, r center nodes are chosen to be located as far as possible from each other. This is achieved by selecting the first center randomly, the second center as the farthest node from the first center, the third center as the farthest node from the set of the first two centers, and so on. Then, clustering is done by assigning each of the remaining nodes to its nearest center node. We consider the larger TSPlib instances with up to 442 nodes, 97461 edges, and 89 clusters; details are listed in Table 1. The values in the columns denote names of the instances, numbers of nodes, numbers of edges, numbers of clusters, and the average, minimal, and maximal numbers of nodes per cluster.

Ghosh [7] created so-called group Euclidean instances. For these instances, squares with side length *span* are associated with clusters and are regularly laid out on a grid of size $col \times row$ as shown in Figure 11. The nodes of each cluster are randomly distributed within the corresponding square. By changing the ratio between cluster separation *sep* and cluster span *span*, it is possible to generate instances with clusters that are overlapping or widely separated. The second type of benchmark instances is called random Euclidean; nodes of the same cluster are not necessarily close to each other. Such instances are created by simply scattering

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

Table 1: TSPlib instances with geographical clustering [3, 5]. Numbers of nodes vary for each cluster.

Instance name	$\|V\|$	$\|E\|$	r	$\|V\|/r$	$d_{\min}$	$d_{\max}$
gr137	137	9316	28	4.89	1	12
kroa150	150	11175	30	5.00	1	10
krob200	200	19900	40	5.00	1	8
ts225	225	25200	45	5.00	1	9
gil262	262	34191	53	4.94	1	13
pr264	264	34716	54	4.89	1	12
pr299	299	44551	60	4.98	1	11
lin318	318	50403	64	4.97	1	14
rd400	400	79800	80	5.00	1	11
fl417	417	86736	84	4.96	1	22
gr431	431	92665	87	4.95	1	62
pr439	439	96141	88	4.99	1	17
pcb442	442	97461	89	4.97	1	10

nodes randomly within a square of size 1000×1000 and making the cluster assignment independently at random. Finally, Ghosh also generated non-Euclidean random instances by choosing all edge costs randomly from the integer interval $[0, 1000]$. All graphs have a complete set of edges. The benchmark set contains instances with up to 1280 nodes, 818560 edges, and 64 clusters; details are listed in Table 2. For each type and size, we consider three different instances. The values in the columns denote names of the sets, numbers of nodes, numbers of edges, numbers of clusters, and numbers of nodes per cluster. In case of group Euclidean instances, numbers of columns and rows of the grid, as well as the cluster separation and cluster span values are additionally given.

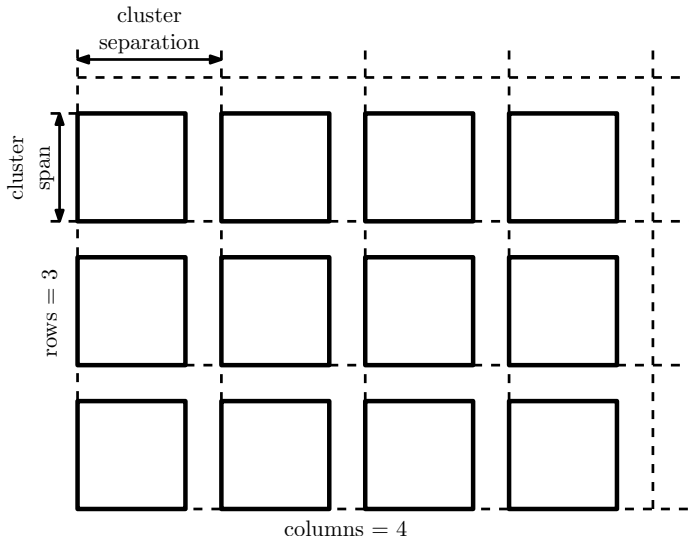

Figure 11: Structure of group Euclidean instances.

Table 2: Benchmark instance sets adopted from [7]. Each instance has constant number of nodes per cluster.

Instance set	$\|V\|$	$\|E\|$	r	$\|V\|/r$	col	row	sep	$span$
Group Eucl 125	125	7750	25	5	5	5	10	10
Group Eucl 500	500	124750	100	5	10	10	10	10
Group Eucl 600	600	179700	20	30	5	4	10	10
Group Eucl 1280	1280	818560	64	20	8	8	10	10
Random Eucl 250	250	31125	50	5	-	-	-	-
Random Eucl 400	400	79800	20	20	-	-	-	-
Random Eucl 600	600	179700	20	30	-	-	-	-
Non-Eucl 200	200	19900	20	10	-	-	-	-
Non-Eucl 500	500	124750	100	5	-	-	-	-
Non-Eucl 600	600	179700	20	30	-	-	-	-

The above test instances are too large to be practically solved to optimality by the proposed MIP approach. For comparison purposes, we therefore derived additional smaller instances by reducing some original benchmark instances. Their properties are listed in Table 3.

Table 3: Small instances for comparison with the MIP approach.

Instance	$\|V\|$	$\|E\|$	r	$\|V\|/r$	col	row	sep	$span$
Group Eucl 40	40	780	8	5	4	2	10	10
Group Eucl 50	50	1225	10	5	5	2	10	10
Group Eucl 60	60	1770	12	5	6	2	10	10
Random Eucl 40	40	780	8	5	-	-	-	-
Random Eucl 50	50	1225	10	5	-	-	-	-
Random Eucl 60	60	1770	12	5	-	-	-	-
Non-Eucl 40	40	780	8	5	-	-	-	-
Non-Eucl 50	50	1225	10	5	-	-	-	-
Non-Eucl 60	60	1770	12	5	-	-	-	-

6 Computational Results

All experiments have been performed on a Pentium 4, 2.6 GHz PC with 1GB RAM. To test the performance of the MIP formulation, we used the general purpose MIP solver CPLEX version 10.0.1. In order to compute average values and standard deviations for the VNS, we performed for each algorithm variant and each instance 30 independent runs.

For the two complex neighborhood structures CRAN and NEN, we set the time limit for each evaluation to 5s; i.e., after 5s, if not all neighbors of the current solution could be evaluated, VND continues with the so far-best neighbor.

6.1 Results on Small Instances

We first consider the small instances derived in particular for testing the MIP approach. Table 4 shows the corresponding results. For the MIP approach, the obtained optimal solution values $C(T^*)$ and the required CPU times for identifying and proving them are listed. For the three VNS variants VNS1 (including only simpler neighborhoods), VNS2 (including neighborhoods based on graph reduction), and SAVNS (VNS2 with self-adaptive ordering of neighborhoods), the CPU time was limited to one second per run. Obtained average solution values $\overline{C(T)}$, corresponding standard deviations, and success rates of how often the optimal solution was found are listed.

Table 4: Results on small instances. Time limit for all VNS approaches is 1s per run.

	MIP		VNS1			VNS2			SAVNS		
Instance, $\|V\|$	$C(T^*)$	time	$\overline{C(T)}$	std dev	Opt.	$\overline{C(T)}$	std dev	Opt.	$\overline{C(T)}$	std dev	Opt.
Group Eucl, 40	79.4	4.6s	79.4	0.00	30/30	79.4	0.00	30/30	79.4	0.00	30/30
Group Eucl, 50	82.1	31.2s	82.2	0.25	25/30	82.1	0.00	30/30	82.1	0.00	30/30
Group Eucl, 60	91.8	539.8s	91.9	0.40	27/30	91.8	0.00	30/30	91.8	0.00	30/30
Random Eucl, 40	989.0	6.3s	989.0	0.00	30/30	989.0	0.00	30/30	989.0	0.00	30/30
Random Eucl, 50	1310.2	762.0s	1311.1	4.75	29/30	1310.2	0.00	30/30	1310.2	0.00	30/30
Random Eucl, 60	1318.0	3370.6s	1374.5	121.21	17/30	1349.8	87.99	26/30	1318.6	3.36	29/30
Non-Eucl, 40	152.8	3.8s	176.0	34.96	17/30	164.4	21.50	20/30	160.6	9.77	18/30
Non-Eucl, 50	216.2	16.1s	269.1	59.28	13/30	249.7	47.16	18/30	242.8	41.84	20/30
Non-Eucl, 60	165.5	276.9s	221.7	52.16	9/30	201.6	46.08	15/30	195.5	36.56	16/30

We observe that the MIP approach was able so solve all instances with up to 60 nodes to proven optimality. The required CPU times are, however, relatively large and substantially increase with the number of nodes, especially for random Euclidean instances. For the three VNS variants, most of these small instances turned out to be no real challenge: On group Euclidean and random Euclidean instances, optimal solutions could often be found in fractions of a second. Non Euclidean instances proved to be significantly more difficult in our experiments. VNS1 performed worse than VNS2 and SAVNS due to the absence of the more complex neighborhood structures.

6.2 Results on Larger Instances

We now turn to the larger TSPlib instances and random instances from Ghosh. First of all, Table 5 shows the results of the construction heuristic ACH. Though the solution quality is only moderate compared to the final solutions obtained by the VNS variants, ACH only requires fractions of a second for small and medium instances and never more than a few seconds for the largest instances.

Table 5: Results of construction heuristic ACH

Random instance sets				ACH	
Instance	$\|V\|$	r	$\|V\|/r$	$\overline{C(T)}$	time
gr137	137	28	4.89	562.0	0.01s
kroa150	150	30	5.00	17234.0	0.02s
krob200	200	40	5.00	17779.0	0.03s
ts225	225	45	5.00	83729.0	0.03s
gil262	262	53	4.94	1434.0	0.05s
pr264	264	54	4.89	39860.0	0.21s
pr299	299	60	4.98	28684.0	0.08s
lin318	318	64	4.97	28039.0	0.08s
rd400	400	80	5.00	9605.0	0.16s
fl417	417	84	4.96	12177.0	0.32s
gr431	431	87	4.95	1681.0	1.41s
pr439	439	88	4.99	86968.0	0.52s
pcb442	442	89	4.97	29573.0	0.17s
Random instance sets				ACH	
Instance	$\|V\|$	r	$\|V\|/r$	$\overline{C(T)}$	time
Group Eucl 125	125	25	5	227.1	0.01s
	125	25	5	209.5	0.01s
	125	25	5	230.9	0.01s
Group Eucl 500	500	100	5	939.6	0.23s
	500	100	5	993.6	0.24s
	500	100	5	943.7	0.24s
Group Eucl 600	600	20	30	172.6	1.41s
	600	20	30	151.0	1.28s
	600	20	30	179.0	1.01s
Group Eucl 1280	1280	64	8	590.2	6.03s
	1280	64	8	585.4	4.22s
	1280	64	8	562.5	5.18s
Random Eucl 250	250	50	5	4398.9	0.08s
	250	50	5	5110.0	0.12s
	250	50	5	4975.1	0.12s
Random Eucl 400	400	20	20	3237.8	0.49s
	400	20	20	2582.8	0.40s
	400	20	20	2308.6	0.71s
Random Eucl 600	600	20	30	2984.3	2.56s
	600	20	30	2964.1	1.87s
	600	20	30	2550.8	1.62s
Non-Eucl 200	200	20	10	1569.7	0.02s
	200	20	10	1223.9	0.02s
	200	20	10	1465.6	0.02s
Non-Eucl 500	500	100	5	2045.9	0.13s
	500	100	5	2073.6	0.11s
	500	100	5	1565.0	0.11s
Non-Eucl 600	600	20	30	1469.6	0.41s
	600	20	30	1754.6	0.41s
	600	20	30	414.3	0.50s

Long runs: Tables 6 and 7 show the results of the three VNS variants when using a fixed time limit as the termination criterion. For TSPlib instances, the allowed CPU time roughly depends on the instance size as indicated in Table 6 (between 150s and 600s), while for the random instance sets, each run is terminated after 600s. These limits include the time required for finding the initial solutions. Depending on the instance size and other properties such as type and/or number of clusters, a shorter time limit would be enough to let VNS fully converge. As a result, the final best solutions might be found rather early for small instances whereas they are obtained at the end of a run for large instances. The search process takes up all the time nevertheless. By keeping track of when they are found during the runs, we are able to observe the convergence behavior of the different VNS variants on the different types and sizes of the instances. This is the reason why we decided to use the same time limit for all instances of the random sets.

We list the objective values of the best solutions found among 30 runs $C(T_{\text{best}})$, the average values $\overline{C(T)}$, the standard deviations, and the average times $\overline{t_{\text{best}}}$ until the best solution of each run is found. The best average values are printed bold. Columns $\gamma_{A,B}$ list Type I error probabilities of one-sided Wilcoxon tests [22] for the assumption that from VNS variants A and B, the one with the observed lower average solution value is significantly better than the other variant.

For TSPlib instances in Table 6, we observe a consistent and obvious trend: Among all VNS variants, SAVNS performs best. Results are more ambiguous for the random instances in Table 7. Though SAVNS is still the best strategy on group Euclidean and non-Euclidean instances, its performance is significantly worse than those of VNS2 on random Euclidean instances. By analyzing the log files, we suspect that this is due to CRAN, which is very efficient on these instances, but also rather time consuming. In the test runs, this neighborhood structure is moved to the front and therefore examined very often. This slowed down the search process. Nevertheless, in almost all cases the variants utilizing the more sophisticated neighborhood structures based on graph reduction outperformed the simpler VNS1.

In Table 8, we summarize the relative differences between all three VNS variants. Listed are the best and average solution qualities with corresponding standard deviations and the average times until the best solutions were found, expressed in relation to the results of VNS1 from Tables 6 and 7.

Table 6: Results on TSPlib instances with geographical clustering, variable CPU time.

		VNS1 (I)				VNS2 (II)				SAVNS (A)				Wilcoxon tests		
Instance	time	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$\gamma_{\text{I,II}}$	$\gamma_{\text{II,A}}$	$\gamma_{\text{I,A}}$
gr137	150s	440.0	440.8	2.4	6.4s	440.0	**440.0**	0.0	31.7s	440.0	**440.0**	0.0	16.2s	0.12	NA	0.12
kroa150	150s	11532.0	11532.8	2.4	21.8s	11532.0	11533.4	4.0	56.1s	11532.0	**11532.0**	0.0	28.4s	0.79	0.06	0.12
krob200	300s	13177.0	13309.4	79.9	75.5s	13177.0	**13201.2**	22.2	137.6s	13177.0	13206.4	23.8	120.4s	<0.01	0.14	<0.01
ts225	300s	68346.0	68769.7	305.9	72.4s	68346.0	68658.1	212.2	140.7s	68346.0	**68563.7**	166.3	123.8s	0.08	0.07	<0.01
gil262	300s	1078.0	1157.1	54.9	146.5s	1059.0	1076.3	15.9	192.7s	1059.0	**1070.8**	10.1	168.3s	<0.01	0.27	<0.01
pr264	300s	29948.0	31639.6	1449.3	138.0s	29810.0	30434.8	725.5	238.3s	29810.0	**29879.3**	56.0	226.4s	<0.01	<0.01	<0.01
pr299	450s	22853.0	23953.9	792.4	331.1s	22644.0	22829.6	215.9	307.7s	22644.0	**22659.1**	12.7	300.7s	<0.01	<0.01	<0.01
lin318	450s	21506.0	23101.1	840.3	409.4s	21028.0	21750.4	444.4	338.5s	20795.0	**21321.0**	228.7	299.4s	<0.01	<0.01	<0.01
rd400	600s	6846.0	7275.4	238.0	414.3s	6755.0	6961.7	103.2	425.8s	6745.0	**6833.2**	42.0	427.6s	<0.01	<0.01	<0.01
fl417	600s	10234.0	10636.4	286.8	198.7s	10085.0	10547.9	234.2	329.9s	9708.0	**9881.9**	160.8	431.2s	0.08	<0.01	<0.01
gr431	600s	1337.0	1408.2	50.8	434.3s	1303.0	1346.2	27.0	346.0s	1284.0	**1312.3**	18.0	467.3s	<0.01	<0.01	<0.01
pr439	600s	65830.0	72752.7	3857.2	497.2s	60972.0	67727.8	3302.3	460.1s	60642.0	**62276.9**	1710.7	471.7s	<0.01	<0.01	<0.01
pcb442	600s	25545.0	26051.2	197.2	570.5s	22439.0	23882.8	947.0	372.2s	22148.0	**22612.6**	325.9	468.4s	<0.01	<0.01	<0.01

Table 7: Results on instance sets from [7] and corresponding created ones, 600s CPU time. Three different instances are considered for each set.

		VNS1 (I)				VNS2 (II)				SAVNS (A)				Wilcoxon tests		
Instance	time	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$\gamma_{\text{I,II}}$	$\gamma_{\text{II,A}}$	$\gamma_{\text{I,A}}$
Group Eucl 125	600s	159.5	**159.5**	0.0	29.8s	159.5	**159.5**	0.0	93.1s	159.5	**159.5**	0.0	38.2s	NA	NA	NA
	600s	163.5	**163.5**	0.0	25.1s	163.5	**163.5**	0.0	35.9s	163.5	**163.5**	0.0	14.4s	NA	NA	NA
	600s	166.1	**166.1**	0.0	9.8s	166.1	**166.1**	0.0	27.0s	166.1	**166.1**	0.0	6.5s	NA	NA	NA
Group Eucl 500	600s	684.4	717.6	25.8	432.8s	640.4	678.5	17.8	379.9s	629.8	**644.8**	9.1	460.2s	<0.01	<0.01	<0.01
	600s	765.1	799.0	35.4	505.4s	657.4	698.9	24.2	438.2s	645.3	**663.8**	11.9	527.0s	<0.01	<0.01	<0.01
	600s	747.3	761.7	37.2	332.8s	651.7	693.7	25.4	486.3s	643.0	**655.6**	8.9	505.8s	<0.01	<0.01	<0.01
Group Eucl 600	600s	105.1	**105.1**	0.0	44.9s	105.1	105.5	1.9	315.3s	105.1	**105.1**	0.0	40.7s	0.50	0.50	NA
	600s	105.2	**105.2**	0.0	10.7s	105.2	105.9	2.7	398.4s	105.2	**105.2**	0.0	8.3s	0.01	0.01	NA
	600s	107.5	**107.5**	0.0	3.3s	107.5	**107.5**	0.0	48.0s	107.5	**107.5**	0.0	1.5s	NA	NA	NA
Group Eucl 1280	600s	399.0	436.8	36.5	461.9s	364.6	385.7	11.0	408.2s	342.5	**358.2**	11.5	522.7s	<0.01	<0.01	<0.01
	600s	376.2	404.6	24.3	497.6s	354.5	390.8	15.0	437.2s	345.5	**351.4**	6.3	520.2s	0.02	<0.01	<0.01
	600s	379.4	433.3	35.8	491.3s	377.5	406.7	17.5	445.7s	357.9	**371.4**	8.2	508.0s	<0.01	<0.01	<0.01
Random Eucl 250	600s	3571.1	3746.6	123.2	156.2s	2995.1	**3226.6**	116.9	503.4s	3235.2	3449.3	145.7	546.2s	<0.01	<0.01	<0.01
	600s	3309.5	3661.8	166.9	156.1s	2662.1	**2968.7**	189.4	491.2s	2750.2	3299.3	161.4	550.2s	<0.01	<0.01	<0.01
	600s	3051.9	3403.2	270.1	126.9s	2698.6	**2888.1**	98.4	500.1s	2793.2	3111.9	255.0	560.1s	<0.01	<0.01	<0.01
Random Eucl 400	600s	943.8	1027.2	71.5	99.9s	841.0	**926.9**	61.2	401.9s	943.8	1028.4	48.7	59.9s	<0.01	<0.01	0.28
	600s	813.3	1059.5	138.3	193.0s	813.3	**875.7**	56.8	310.9s	813.3	1043.2	109.0	89.1s	<0.01	<0.01	0.20
	600s	814.9	858.7	40.5	53.7s	794.4	**836.1**	39.8	385.2s	814.9	860.7	27.8	100.0s	<0.01	<0.01	0.15
Random Eucl 600	600s	599.8	725.1	103.9	183.5s	599.8	**605.1**	7.5	371.2s	599.8	699.2	90.4	77.3s	<0.01	<0.01	0.47
	600s	785.4	823.7	55.3	226.7s	643.1	**762.0**	50.6	428.7s	643.1	813.4	53.9	151.4s	<0.01	<0.01	0.40
	600s	695.3	778.9	60.1	256.1s	596.5	**674.3**	52.3	399.4s	695.3	743.7	94.2	88.3s	<0.01	<0.01	<0.01
Non-Eucl 200	600s	180.1	244.9	34.4	335.7s	172.6	224.5	23.4	323.2s	179.8	**219.0**	31.0	239.8s	0.01	0.06	<0.01
	600s	133.8	216.7	33.8	189.5s	140.2	**198.3**	31.1	251.2s	179.3	215.9	19.0	139.7s	0.02	0.01	0.48
	600s	156.0	179.8	28.4	124.8s	135.5	190.6	29.0	272.0s	154.4	**175.6**	20.1	98.3s	0.04	<0.01	0.38
Non-Eucl 500	600s	902.6	1121.3	123.3	206.1s	830.5	996.7	88.5	243.0s	771.0	**960.2**	99.3	387.8s	<0.01	0.04	<0.01
	600s	897.5	1008.2	128.9	175.2s	860.7	1016.4	87.2	230.7s	734.3	**921.2**	106.9	340.7s	0.27	<0.01	0.01
	600s	785.3	1025.8	109.5	242.0s	844.0	990.8	83.0	186.4s	710.4	**929.0**	120.8	382.4s	0.10	<0.01	<0.01
Non-Eucl 600	600s	102.9	138.6	25.1	330.8s	93.0	129.9	14.2	315.2s	91.6	**125.0**	20.7	301.0s	0.09	0.02	<0.01
	600s	107.3	132.0	19.0	277.3s	102.3	130.2	13.6	332.8s	94.6	**118.5**	19.9	275.7s	0.45	<0.01	<0.01
	600s	85.4	119.4	19.0	300.4s	94.8	133.1	17.8	404.7s	79.5	**113.4**	18.5	232.6s	<0.01	<0.01	0.13

Table 8: Comparison of the VNS variants: relative values for VNS2 over VNS1 and SAVNS over VNS1, absolute time limits.

TSPlib instances	VNS2 over VNS1				SAVNS over VNS1			
Instance	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$
gr137	100.0%	99.8%	0.0%	495.3%	100.0%	99.8%	0.0%	253.1%
kroa150	100.0%	100.0%	166.7%	257.3%	100.0%	100.0%	0.0%	130.3%
krob200	100.0%	99.2%	27.8%	182.3%	100.0%	99.2%	29.8%	159.5%
ts225	100.0%	99.8%	69.4%	194.3%	100.0%	99.7%	54.4%	171.0%
gil262	98.2%	93.0%	29.0%	131.5%	98.2%	92.5%	18.4%	114.9%
pr264	99.5%	96.2%	50.1%	172.7%	99.5%	94.4%	3.9%	164.1%
pr299	99.1%	95.3%	27.2%	92.9%	99.1%	94.6%	1.6%	90.8%
lin318	97.8%	94.2%	52.9%	82.7%	96.7%	92.3%	27.2%	73.1%
rd400	98.7%	95.7%	43.4%	102.8%	98.5%	93.9%	17.6%	103.2%
fl417	98.5%	99.2%	81.7%	166.0%	94.9%	92.9%	56.1%	217.0%
gr431	97.5%	95.6%	53.1%	79.7%	96.0%	93.2%	35.4%	107.6%
pr439	92.6%	93.1%	85.6%	92.5%	92.1%	85.6%	44.4%	94.9%
pcb442	87.8%	91.7%	480.2%	65.2%	86.7%	86.8%	165.3%	82.1%
Random instance sets	VNS2 over VNS1				SAVNS over VNS1			
Instance	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$
Group Eucl 125	100.0%	100.0%	–	312.4%	100.0%	100.0%	–	128.2%
	100.0%	100.0%	–	143.4%	100.0%	100.0%	–	57.4%
	100.0%	100.0%	–	275.5%	100.0%	100.0%	–	66.3%
Group Eucl 500	93.6%	94.6%	69.0%	87.8%	92.0%	89.9%	35.3%	106.3%
	85.9%	87.5%	68.4%	86.7%	84.3%	83.1%	33.6%	104.3%
	87.2%	91.1%	68.3%	146.1%	86.0%	86.1%	23.9%	152.0%
Group Eucl 600	100.0%	100.4%	–	702.2%	100.0%	100.0%	–	90.6%
	100.0%	100.7%	–	3723.4%	100.0%	100.0%	–	77.6%
	100.0%	100.0%	–	1454.5%	100.0%	100.0%	–	45.5%
Group Eucl 1280	91.4%	88.3%	30.1%	88.4%	85.8%	82.0%	31.5%	113.2%
	94.2%	96.6%	61.7%	87.9%	91.8%	86.9%	25.9%	104.5%
	99.5%	93.9%	48.9%	90.7%	94.3%	85.7%	22.9%	103.4%
Random Eucl 250	83.9%	86.1%	94.9%	322.3%	90.6%	92.1%	118.3%	349.7%
	80.4%	81.1%	113.5%	314.7%	83.1%	90.1%	96.7%	352.5%
	88.4%	84.9%	36.4%	394.1%	91.5%	91.4%	94.4%	441.4%
Random Eucl 400	89.1%	90.2%	85.6%	402.3%	100.0%	100.1%	68.1%	60.0%
	100.0%	82.7%	41.1%	161.1%	100.0%	98.5%	78.8%	46.2%
	97.5%	97.4%	98.3%	717.3%	100.0%	100.2%	68.6%	186.2%
Random Eucl 600	100.0%	83.5%	7.3%	202.3%	100.0%	96.4%	87.8%	42.1%
	81.9%	92.5%	91.5%	189.1%	81.9%	98.7%	97.5%	66.8%
	85.8%	86.6%	87.0%	156.0%	100.0%	95.5%	156.7%	34.5%
Non-Eucl 200	95.8%	91.7%	68.0%	96.3%	99.8%	89.4%	90.1%	71.4%
	104.8%	91.5%	92.0%	132.6%	134.0%	99.6%	56.2%	73.7%
	86.9%	106.0%	102.1%	217.9%	99.0%	97.7%	70.8%	78.8%
Non-Eucl 500	92.0%	88.9%	71.8%	117.9%	85.4%	85.6%	80.5%	188.2%
	95.9%	100.8%	67.6%	131.7%	81.8%	91.4%	82.9%	194.5%
	107.5%	96.6%	75.8%	77.0%	90.5%	90.6%	110.3%	158.0%
Non-Eucl 600	90.4%	93.7%	56.6%	95.3%	89.0%	90.2%	82.5%	91.0%
	95.3%	98.6%	71.6%	120.0%	88.2%	89.8%	104.7%	99.4%
	111.0%	111.5%	93.7%	134.7%	93.1%	95.0%	97.4%	77.4%

Short runs: In a second set of experiments, we tested our algorithms by using an alternative termination criterion. Tables 9 and 10 show results of the same VNS algorithms when the runs are terminated after 30 consecutive iterations without obtaining improvements. Compared to the results in Tables 6 and 7, where absolute time limits were used, the runs of VNS2 and SAVNS terminate much earlier at the expense of obtaining slightly inferior results. However, for some large instances like pcb442 or from the set Group Eucl 500, VNS1 takes longer to converge. In these cases, the advantage of the neighborhood structures based on graph reduction that are used in VNS2 and SAVNS are more obvious. They keep the evaluation time low and thus the overall convergence speed becomes more robust, even for large instances.

When using this termination criterion, the solution quality of VNS2 and SAVNS become more similar than when using absolute time limits. This is understandable since both variants use the same set of neighborhood structures, but potentially in a different order. As concluded in [14], by using a dynamic ordering of the neighborhood structures, the solution quality cannot be expected to be increased in general. However, the computational time can decrease since the more useful neighborhood structures are called more often. Hence when using the same time limits as the termination criterion, SAVNS performs better overall. From another point of view, in order to obtain solutions of equal quality, SAVNS requires less time overall.

Table 11 lists the relative differences between all three VNS variants for this set of experiments.

6.3 Contributions of Neighborhood Structures

In order to analyze the individual contributions of the different neighborhood structures to the whole success, we logged how often each neighborhood structure was able to improve a current solution in Tables 6 and 7. We then determined the ratios of successful improvements over how often each neighborhood structure was evaluated and normalized these values over all neighborhood structures, yielding percentage values describing their relative efficiencies. For VNS2 and all considered large test instances, these efficiencies are listed in Tables 12 and 13. For SAVNS, these values are similar since the same neighborhood structures are used.

In general, each neighborhood structure contributes substantially to the whole success. In Table 12, we observe that by increasing the size of the instances, the more complex neighborhood structures like EAN and NEN become more efficient while the improvement rates of the simpler ones decrease. In Table 13, we combined the data for each set of three identically parameterized instances for simplicity. The dependency on the instance size is less obvious, but the neighborhoods' relative efficiencies strongly depend on the structure of the instance. In particular, NRAN is less successful on non-Euclidean instances but second best for random Euclidean instances. All in all, CRAN is able to improve solutions most often. NON performs best on instances with many nodes per cluster, while EAN performs best on instances containing many relatively small clusters.

Table 9: Results on TSPlib instances with geographical clustering, termination after 30 iterations without improvement.

	VNS1 (I)				VNS2 (II)				SAVNS (A)				Wilcoxon tests		
Instance	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$\gamma_{\text{I,II}}$	$\gamma_{\text{II,A}}$	$\gamma_{\text{I,A}}$
gr137	447.0	468.6	14.3	1.7s	441.0	467.9	17.2	0.7s	444.0	**464.0**	14.7	0.9s	0.37	0.18	0.10
kroa150	11532.0	11878.7	397.2	1.3s	11532.0	11885.0	271.7	1.1s	11532.0	**11873.6**	304.8	0.8s	0.22	0.28	0.32
krob200	13342.0	**13628.8**	236.6	24.0s	13233.0	13632.6	183.6	7.2s	13383.0	13688.9	199.8	6.1s	0.30	0.22	0.11
ts225	68992.0	70928.7	1987.9	16.1s	68827.0	**70724.1**	1608.0	6.2s	68583.0	71062.5	2181.2	5.2s	0.37	0.33	0.41
gil262	1061.0	1125.7	49.7	330.0s	1059.0	**1109.3**	50.3	32.0s	1074.0	1126.5	50.8	25.6s	0.04	0.05	0.44
pr264	30060.0	32017.6	1426.8	240.0s	29881.0	**31279.9**	1140.0	30.3s	30096.0	31517.4	1319.3	24.1s	0.03	0.33	0.10
pr299	22844.0	23647.8	554.6	471.9s	22644.0	23293.5	463.4	32.2s	22709.0	**23228.1**	410.8	28.4s	<0.01	0.33	<0.01
lin318	21591.0	22971.2	941.4	451.1s	21316.0	**21989.0**	557.4	57.9s	21243.0	22046.0	564.3	49.0s	<0.01	0.35	<0.01
rd400	6811.0	7167.8	228.5	416.4s	6789.0	6998.0	108.3	74.5s	6801.0	**6992.1**	117.4	60.7s	<0.01	0.36	<0.01
fl417	9893.0	10674.5	301.0	282.5s	9784.0	**10500.4**	345.4	36.3s	9984.0	10506.1	302.3	28.1s	0.03	0.49	0.02
gr431	1351.0	1412.2	56.3	419.3s	1313.0	1364.7	31.8	61.0s	1286.0	**1361.9**	30.3	52.0s	<0.01	0.28	<0.01
pr439	63535.0	72118.0	4587.0	424.1s	61368.0	66835.1	3500.3	70.3s	61067.0	**66805.8**	3461.8	59.1s	<0.01	0.49	<0.01
pcb442	25472.0	25873.9	302.3	800.8s	22402.0	23890.7	999.1	109.4s	22665.0	**23957.2**	1058.8	88.0s	<0.01	0.42	<0.01

Table 10: Results on instance sets from [7] and corresponding created ones, termination after 30 iterations without improvement. Three different instances are considered for each set.

	VNS1 (I)				VNS2 (II)				SAVNS (A)				Wilcoxon tests		
Instance	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$\gamma_{\text{I,II}}$	$\gamma_{\text{II,A}}$	$\gamma_{\text{I,A}}$
Group Eucl 125	159.8	168.3	7.8	0.5s	159.8	165.5	6.4	0.3s	159.5	**164.5**	6.5	0.3s	0.07	0.06	<0.01
	163.5	170.5	5.1	0.6s	165.2	**169.4**	2.6	0.2s	163.5	170.2	4.3	0.3s	0.38	0.23	0.49
	166.4	183.5	7.9	0.3s	167.5	180.8	7.1	0.2s	166.1	**178.9**	8.5	0.3s	0.03	0.16	<0.01
Group Eucl 500	685.3	715.8	21.1	1314.7s	629.1	**663.3**	20.0	142.9s	644.9	669.9	15.9	131.6s	<0.01	0.05	<0.01
	713.3	797.3	40.9	824.2s	661.4	**693.1**	24.5	154.6s	650.9	696.7	28.7	137.0s	<0.01	0.25	<0.01
	685.1	760.4	39.4	553.4s	649.9	677.2	23.5	134.0s	650.3	**676.7**	18.3	125.1s	<0.01	0.44	<0.01
Group Eucl 600	105.1	112.1	7.0	4.0s	105.1	**109.1**	6.0	1.4s	105.1	109.7	7.3	1.3s	0.05	0.47	0.05
	105.2	107.5	4.8	3.7s	105.2	107.1	4.1	1.5s	105.2	**106.7**	3.6	1.3s	0.05	0.25	0.14
	107.5	107.7	0.2	2.9s	107.5	**107.7**	0.3	1.0s	107.5	**107.7**	0.3	0.9s	0.44	0.15	0.11
Group Eucl 1280	377.3	425.4	30.0	821.3s	357.1	378.5	13.8	155.4s	350.5	**377.8**	15.0	195.3s	<0.01	0.43	<0.01
	365.0	407.1	28.3	464.3s	347.6	**364.4**	11.6	154.9s	348.0	364.7	12.1	151.5s	<0.01	0.47	<0.01
	384.9	436.4	32.9	556.7s	363.0	389.2	15.0	160.7s	363.6	**386.5**	13.6	118.5s	<0.01	0.20	<0.01
Random Eucl 250	3720.3	3859.0	83.8	216.0s	3689.5	3845.0	79.6	25.5s	3627.6	**3823.4**	83.8	31.5s	0.33	0.15	0.12
	3541.7	4043.7	271.9	116.5s	3556.9	**3955.5**	212.6	28.4s	3363.7	3967.3	271.9	32.7s	0.07	0.49	0.16
	3078.8	3813.0	291.9	131.6s	3209.8	3791.3	249.1	23.3s	3160.0	**3741.7**	297.3	31.4s	0.31	0.21	0.16
Random Eucl 400	1107.6	1412.1	224.2	2.8s	1098.4	**1361.1**	165.9	1.2s	1065.3	1367.1	177.9	1.4s	0.23	0.41	0.24
	1184.7	1468.3	153.9	2.2s	1089.9	1415.1	206.6	1.1s	930.8	**1371.7**	230.3	1.3s	0.07	0.29	0.03
	957.4	1224.9	197.9	2.8s	933.3	**1150.4**	169.2	1.3s	894.4	1156.9	177.6	1.4s	0.04	0.44	0.08
Random Eucl 600	667.0	1275.3	366.2	8.6s	647.2	1163.0	347.8	3.0s	689.7	**1068.4**	228.3	3.0s	0.13	0.21	0.01
	697.2	1145.5	243.9	6.7s	852.5	1163.8	269.9	2.5s	781.7	**1089.2**	203.5	2.6s	0.49	0.23	0.26
	743.0	1124.5	262.8	7.1s	774.0	**1038.7**	180.4	3.1s	742.1	1048.9	279.9	3.0s	0.14	0.27	0.10
Non-Eucl 200	314.6	470.3	93.0	0.4s	345.3	**457.9**	79.0	0.3s	347.5	466.3	78.2	0.4s	0.30	0.29	0.47
	332.8	395.0	36.4	0.4s	232.6	**368.0**	43.6	0.3s	231.5	388.6	60.4	0.2s	<0.01	0.07	0.18
	183.3	347.2	83.9	0.3s	174.4	**329.4**	55.0	0.2s	220.8	334.6	42.3	0.2s	0.29	0.16	0.34
Non-Eucl 500	840.3	1119.4	104.5	142.7s	882.5	1115.9	131.7	54.2s	984.8	**1115.6**	85.2	41.7s	0.46	0.39	0.23
	865.7	1087.8	89.3	87.3s	645.5	1039.7	141.8	20.2s	747.3	**1021.7**	106.4	14.9s	0.06	0.23	<0.01
	914.0	1075.8	90.9	93.6s	806.8	**1019.4**	86.5	21.2s	835.4	1042.4	99.7	18.4s	0.01	0.22	0.10
Non-Eucl 600	209.9	354.6	74.5	3.6s	188.8	**320.7**	83.1	1.4s	196.8	326.5	92.9	1.1s	0.04	0.50	0.05
	180.7	252.2	35.3	1.9s	170.6	247.2	36.2	0.8s	136.0	**237.4**	51.8	0.8s	0.20	0.07	0.02
	161.3	284.7	57.8	2.9s	141.2	270.1	67.0	1.6s	158.5	**252.6**	52.8	1.5s	0.23	0.14	0.01

Table 11: Comparison of the VNS variants: relative values for VNS2 over VNS1 and SAVNS over VNS1, termination after 30 iterations without improvement.

TSPlib instances	VNS2 over VNS1				SAVNS over VNS1			
Instance	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$
gr137	98.7%	99.9%	120.3%	41.2%	99.3%	99.0%	102.8%	52.9%
kroa150	100.0%	100.1%	68.4%	84.6%	100.0%	100.0%	76.7%	61.5%
krob200	99.2%	100.0%	77.6%	30.0%	100.3%	100.4%	84.4%	25.4%
ts225	99.8%	99.7%	80.9%	38.5%	99.4%	100.2%	109.7%	32.3%
gil262	99.8%	98.5%	101.2%	9.7%	101.2%	100.1%	102.2%	7.8%
pr264	99.4%	97.7%	79.9%	12.6%	100.1%	98.4%	92.5%	10.0%
pr299	99.1%	98.5%	83.6%	6.8%	99.4%	98.2%	74.1%	6.0%
lin318	98.7%	95.7%	59.2%	12.8%	98.4%	96.0%	59.9%	10.9%
rd400	99.7%	97.6%	47.4%	17.9%	99.9%	97.5%	51.4%	14.6%
fl417	98.9%	98.4%	114.8%	12.8%	100.9%	98.4%	100.4%	9.9%
gr431	97.2%	96.6%	56.5%	14.5%	95.2%	96.4%	53.8%	12.4%
pr439	96.6%	92.7%	76.3%	16.6%	96.1%	92.6%	75.5%	13.9%
pcb442	87.9%	92.3%	330.5%	13.7%	89.0%	92.6%	350.2%	11.0%
Random instance sets	VNS2 over VNS1				SAVNS over VNS1			
Instance	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$	$C(T_{\text{best}})$	$\overline{C(T)}$	std dev	$\overline{t_{\text{best}}}$
	100.0%	98.3%	82.1%	60.0%	99.8%	97.7%	83.3%	60.0%
Group Eucl 125	101.0%	99.4%	51.0%	33.3%	100.0%	99.8%	84.3%	50.0%
	100.7%	98.5%	89.9%	66.7%	99.8%	97.5%	107.6%	100.0%
	91.8%	92.7%	94.8%	10.9%	94.1%	93.6%	75.4%	10.0%
Group Eucl 500	92.7%	86.9%	59.9%	18.8%	91.3%	87.4%	70.2%	16.6%
	94.9%	89.1%	59.6%	24.2%	94.9%	89.0%	46.4%	22.6%
	100.0%	97.3%	85.7%	35.0%	100.0%	97.9%	104.3%	32.5%
Group Eucl 600	100.0%	99.6%	85.4%	40.5%	100.0%	99.3%	75.0%	35.1%
	100.0%	100.0%	150.0%	34.5%	100.0%	100.0%	150.0%	31.0%
	94.6%	89.0%	46.0%	18.9%	92.9%	88.8%	50.0%	23.8%
Group Eucl 1280	95.2%	89.5%	41.0%	33.4%	95.3%	89.6%	42.8%	32.6%
	94.3%	89.2%	45.6%	28.9%	94.5%	88.6%	41.3%	21.3%
	99.2%	99.6%	95.0%	11.8%	97.5%	99.1%	100.0%	14.6%
Random Eucl 250	100.4%	97.8%	78.2%	24.4%	95.0%	98.1%	100.0%	28.1%
	104.3%	99.4%	85.3%	17.7%	102.6%	98.1%	101.8%	23.9%
	99.2%	96.4%	74.0%	42.9%	96.2%	96.8%	79.3%	50.0%
Random Eucl 400	92.0%	96.4%	134.2%	50.0%	78.6%	93.4%	149.6%	59.1%
	97.5%	93.9%	85.5%	46.4%	93.4%	94.4%	89.7%	50.0%
	97.0%	91.2%	95.0%	34.9%	103.4%	83.8%	62.3%	34.9%
Random Eucl 600	122.3%	101.6%	110.7%	37.3%	112.1%	95.1%	83.4%	38.8%
	104.2%	92.4%	68.6%	43.7%	99.9%	93.3%	106.5%	42.3%
	109.8%	97.4%	84.9%	75.0%	110.5%	99.1%	84.1%	100.0%
Non-Eucl 200	69.9%	93.2%	119.8%	75.0%	69.6%	98.4%	165.9%	50.0%
	95.1%	94.9%	65.6%	66.7%	120.5%	96.4%	50.4%	66.7%
	105.0%	99.7%	126.0%	38.0%	117.2%	99.7%	81.5%	29.2%
Non-Eucl 500	74.6%	95.6%	158.8%	23.1%	86.3%	93.9%	119.1%	17.1%
	88.3%	94.8%	95.2%	22.6%	91.4%	96.9%	109.7%	19.7%
	89.9%	90.4%	111.5%	38.9%	93.8%	92.1%	124.7%	30.6%
Non-Eucl 600	94.4%	98.0%	102.5%	42.1%	75.3%	94.1%	146.7%	42.1%
	87.5%	94.9%	115.9%	55.2%	98.3%	88.7%	91.3%	51.7%

Table 12: Relative improvement rates of NON, NRAN, CRAN, EAN, and NEN for TSPlib instances.

Instance	$\|V\|$	r	$\|V\|/r$	NON	NRAN	CRAN	EAN	NEN
gr137	137	28	4.89	22.37%	20.50%	25.43%	22.29%	9.40%
kroa150	150	30	5.00	22.24%	17.75%	25.70%	21.94%	12.36%
krob200	200	40	5.00	17.87%	18.17%	23.97%	25.94%	14.05%
ts225	225	45	5.00	16.35%	19.82%	21.32%	25.68%	16.83%
gil262	262	53	4.94	15.03%	17.34%	21.56%	27.81%	18.26%
pr264	264	54	4.89	14.43%	20.27%	23.00%	26.49%	15.80%
pr299	299	60	4.98	15.07%	17.72%	21.75%	27.39%	18.08%
lin318	318	64	4.97	15.02%	18.42%	21.00%	27.62%	17.94%
rd400	400	80	5.00	13.92%	14.66%	18.40%	27.38%	25.64%
fl417	417	84	4.96	12.69%	21.43%	17.39%	29.69%	18.80%
gr431	431	87	4.95	11.39%	17.52%	19.79%	30.85%	20.45%
pr439	439	88	4.99	14.75%	16.52%	20.81%	27.54%	20.38%
pcb442	442	89	4.97	13.61%	15.21%	20.73%	28.02%	22.42%

Table 13: Relative improvement rates of NON, NRAN, CRAN, EAN, and NEN for random instances.

Instance	$\|V\|$	r	$\|V\|/r$	NON	NRAN	CRAN	EAN	NEN
Group Eucl 125	125	25	5	33.79%	13.56%	32.65%	12.43%	7.56%
Group Eucl 500	500	100	5	22.84%	12.92%	24.83%	20.36%	19.03%
Group Eucl 600	600	20	30	30.68%	6.40%	27.52%	4.75%	30.66%
Group Eucl 1280	1280	64	20	24.95%	10.92%	22.97%	14.09%	27.07%
Random Eucl 250	250	50	5	13.90%	24.94%	28.08%	22.04%	11.04%
Random Eucl 400	400	20	20	26.38%	24.94%	33.44%	8.11%	7.12%
Random Eucl 600	600	20	30	25.23%	27.24%	33.37%	8.13%	6.04%
Non-Eucl 200	200	20	10	31.76%	3.43%	35.99%	15.17%	13.65%
Non-Eucl 500	500	100	5	13.55%	5.10%	23.22%	33.95%	24.16%
Non-Eucl 600	600	20	30	35.83%	1.53%	45.57%	7.91%	9.16%

7 Conclusions

In this article, we proposed three different Variable Neighborhood Search approaches for the Generalized Minimum Edge-Biconnected Network Problem (GMEBCNP). They are based on four neighborhood structures addressing particular properties such as spanned nodes and/or edges between them. For two of them there exist simple and more advanced variants. The latter utilize a graph reduction technique which allows us to efficiently determine the optimal spanned nodes for the majority of clusters once the connections between clusters are fixed.

Experiments were performed on TSPlib based instances with geographical clustering, Euclidean instances with grid and random clustering, and non-Euclidean instances. We used a multi-commodity flow mixed integer programming formulation to solve smaller instances of the GMEBCNP with up to 60 nodes in reasonable time to proven optimality. In comparison, the VNS variants are in most of their runs also able to identify optimal solutions for those small instances, but in substantially shorter time (fractions of a second).

Comparing the results of our VNS variants for medium and large instances, we conclude that the used neighborhood structures are effective and their combination within the VNS scales well to large instances. In particular, we observed that the graph reduction technique applied in the more sophisticated neighborhood structures of VNS2 and SAVNS is a major improvement. The self-adaptive ordering of neighborhoods, as it is done in SAVNS, turned out to be helpful in the majority of the experiments.

8 Future Work

In the future, we want to consider further large neighborhood structures that are evaluated by means of integer linear programming. The described multi-commodity flow formulation provides a basis for this. Another plan is to extend the graph reduction technique by including further shrinking strategies.

Since all instances we used for testing were complete graphs, we did not implement special mechanisms for handling incomplete graphs. Though it is easy to remove all unneeded variables from the MIP model, it is less straightforward to adapt the neighborhood structures for the VNS adequately. We want to investigate possibilities like narrowing the search space or repairing solutions due to non-existing edges in the future.

The *At-least* variant of the GMEBCNP, where at least one node of each cluster must be connected, is also worth studying. Though related, many techniques in this paper, e.g., the graph reduction technique, are not directly applicable to this variant. We would like to investigate this problem in the future and adapt our proposed neighborhood structures and techniques for it. Furthermore, there are other related problems, such as the prize-collecting variants [9], which are also important in practice, and for which our basic approach also looks promising.

References

[1] N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Symp New Directions Recent Results in Algorithms Complexity, New York, NY, Academic Press, 1976, pp. 441.

[2] A. Czumaj and A. Lingas, On approximability of the minimum-cost k-connected spanning subgraph problem, Proc 10th Ann ACM-SIAM Symp Discr Algorithms (SODA), Society for Industrial and Applied Mathematics, 1999, pp. 281–290.

[3] C. Feremans, Generalized spanning trees and extensions, Ph.D. Thesis, Universite Libre de Bruxelles, Brussels, Belgium, 2001.

[4] C. Feremans, M. Labbe, and G. Laporte, Generalized network design problems, Eur J Oper Res 148 (2003), 1–13.

[5] M. Fischetti, J.J. Salazar, and P. Toth, A branch-and-cut algorithm for the symmetric generalized traveling salesman problem, Oper Res 45 (1997), 378–394.

[6] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of $\mathcal{NP}$-completeness, W. H. Freeman & Co., New York, NY, 1990.

[7] 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.

[8] B. Golden, S. Raghavan, and D. Stanojevic, Heuristic search for the generalized minimum spanning tree problem, INFORMS J Comput 17 (2005), 290–304.

[9] B. Golden, S. Raghavan, and D. Stanojevic, The prize-collecting generalized minimum spanning tree problem, J Heuristics 14 (2008), 69–93.

[10] P. Hansen and N. Mladenovic, "An introduction to variable neighborhood search," Meta-heuristics: Advances and trends in local search paradigms for optimization, S. Voss, S. Martello, I.H. Osman, and C. Roucairol (Editors), Kluwer Academic Publishers, 1999, pp. 433–458.

[11] 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.

[12] A.L. Henry-Labordere, The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem, RAIRO B2 (1969), 43–49.

[13] B. Hu, M. Leitner, and G.R. Raidl, Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem, J Heuristics 14 (2008), 473–499.

[14] B. Hu and G.R. Raidl, Variable neighborhood descent with self-adaptive neighborhood-ordering, Proc 7th EU/Meeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics, Malaga, Spain, 2006.

[15] D. Huygens, Version generalisee du probleme de conception de reseau 2-arete-connexe, Master's Thesis, Universite Libre de Bruxelles, 2002.

[16] S. Khuller and U. Vishkin, Biconnectivity approximations and graph carvings, J ACM 41 (1994), 214–235.

[17] M. Leitner, Solving two generalized network design problems with exact and heuristic methods, Master's Thesis, Vienna University of Technology, Vienna, Austria, 2006.

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

[19] P.C. Pop, The generalized minimum spanning tree problem, Ph.D. Thesis, University of Twente, The Netherlands, 2002.

[20] J.P. Saskena, Mathematical model of scheduling clients through welfare agencies, J Can Oper Res Soc 8 (1970), 185–200.

[21] S.S. Srivastava, S. Kumar, R.C. Garg, and P. Sen, Generalized traveling salesman problem through n sets of nodes, CORS J 7 (1969), 97–101.

[22] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics 1 (1945), 80–83.