Effective Neighborhood Structures for the Generalized Traveling Salesman Problem

Bin Hu and Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
{hu|raidl}@ads.tuwien.ac.at

Abstract. We consider the generalized traveling salesman problem in which a graph with nodes partitioned into clusters is given. The goal is to identify a minimum cost round trip visiting exactly one node from each cluster. For solving difficult instances of this problem heuristically, we present a new Variable Neighborhood Search (VNS) approach that utilizes two complementary, large neighborhood structures. One of them is the already known generalized 2-opt neighborhood for which we propose a new incremental evaluation technique to speed up the search significantly. The second structure is based on node exchanges and the application of the chained Lin-Kernighan heuristic. A comparison with other recently published metaheuristics on TSPlib instances with geographical clustering indicates that our VNS, though requiring more time than two genetic algorithms, is able to find substantially better solutions.

Key words: Network Design, Generalized Traveling Salesman Problem, Variable Neighborhood Search

1 Introduction

The Generalized Traveling Salesman Problem (GTSP) extends the classical Traveling Salesman Problem (TSP) and is defined as follows. We consider an undirected weighted complete 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 $V_1, V_2, \ldots, V_r,\ \bigcup_{i=1}^{r} V_i = V,\ V_i \cap V_j = \emptyset,\ i, j = 1, \ldots, r,\ i \neq j$.

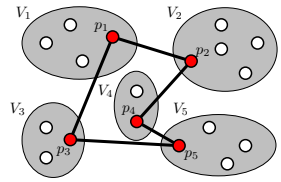

Fig. 1. Example for a GTSP solution.

A solution to the GTSP defined on G is a subgraph $S = \langle P, T \rangle$ with $P = \{p_1, p_2, \ldots, p_r\} \subseteq V$ connecting exactly one node from each cluster, i.e. $p_i \in V_i$ for all $1 \leq i \leq r$, and $T \subseteq E$ being a round trip, see Fig. 1. The costs of such a round trip 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. When edge costs satisfy the triangle inequality, even if we allow more than one node per cluster to be connected, an optimal solution of the GTSP always contains only one node from each cluster [11]. Obviously, the GTSP is NP-hard since it contains the classical TSP as the special case in which each cluster consists of a single node only.

The GTSP finds practical application particularly in many variants of routing problems, e.g. when some good can be delivered to multiple alternative addresses of customers. Occasionally, such applications can be directly modeled as the GTSP, but more often the GTSP appears as a subproblem [9].

In this paper, we present a general Variable Neighborhood Search (VNS) approach [6] for heuristically solving this problem. As local improvement within VNS, we use Variable Neighborhood Descent (VND) based on two different types of exponentially large neighborhoods, which can be seen as dual to each other. One neighborhood structure is the generalized 2-opt, which has been introduced in [19]; for it, we propose a new incremental evaluation scheme leading to a substantial speed-up. As second neighborhood structure we investigate a new approach: the nodes to be spanned from each cluster are fixed and TSP tours are derived via the chained Lin-Kernighan algorithm.

Section 2 gives an overview on research done on the GTSP so far. In section 3, we describe the initialization procedures, followed by section 4 explaining the neighborhood structures in detail. Section 5 shows the VNS settings, and experimental results are discussed section 6. Finally, we conclude in section 7.

2 Previous Work

The GTSP was introduced independently by Henry-Labordere [7], Srivastava et al. [22], and Saskena [20]. Laporte et al. [11, 10] provided integer programming formulations for the symmetrical and asymmetrical GTSP, respectively. The formulation for the symmetrical case was later enhanced by Fischetti et al. [4] who proposed several classes of facet defining inequalities and corresponding separation procedures. Based on these, they developed a branch-and-cut algorithm [5] which could solve instances with up to 442 nodes to optimality.

Several approaches exist which transform the GTSP into the classical TSP. They have been studied by Noon and Bean [16], Lien et al. [13], Dimitrijevic and Saric [2], Laporte and Semet [12], and Behzad and Modarres [1]. Unfortunately, many transformations substantially increase the numbers of nodes and edges and are therefore of limited practical value. Furthermore, some transformations even require additional constraints, thus making general algorithms for the classical TSP inapplicable. Among the more efficient approaches, Dimitrijevic and Saric [2] proposed a transformation of the GTSP into the TSP on a digraph containing twice the number of nodes compared to the original graph. This technique was

further improved by Behzad and Modarres [1] where the transformed graph has the same number of nodes as the original graph. However, the transformation increases edge costs significantly, what may lead to problems in some cases.

To approach larger GTSP instances, various metaheuristics have been suggested. Renaud et al. [19] developed a complex composite heuristic whose components can be used for other (meta-)heuristics as well. They introduced generalized k-opt heuristics which are derived from Lin's classical 2-opt and 3-opt local search for the TSP [14]. Snyder and Daskin [21] describe a Genetic Algorithm (GA) that achieves relatively good results in short running times. It uses random keys to encode solutions and a parameterized uniform crossover operator including local improvement based on the 2-opt heuristic to boost solution quality. Wu et al. [23] also proposed a GA using a direct representation in which the spanned nodes from each cluster and the sequence in which they are visited in the tour are stored. This approach has further been enhanced by Huang et al. [8] who apply a so-called hybrid chromosome encoding. However, reported results are on average inferior when compared to those of the GA from [21].

3 Solution Representation and Initialization

In our VNS, we represent a solution $S = \langle P, T \rangle$ in a direct way by storing the spanned nodes of each cluster $P = \{p_1, p_2, \ldots, p_r\}$ with $p_i \in V_i$, $i = 1, \ldots, r$, and additionally the visiting order in the round trip as circular permutation $\pi = \langle \pi_1, \ldots, \pi_r \rangle$ of the cluster indices $\{1, \ldots, r\}$.

Depending on the instance type we use two different procedures to compute feasible initial solutions for the VNS. Both are extensions of well-known greedy strategies for the classical TSP. The first algorithm is the (generalized) Nearest Neighbor Heuristic (NNH), and it can in principle be applied to all kinds of instances. The second procedure is specifically targeted to Euclidean instances where the clustering is based on geographical proximity. It exploits Euclidean coordinates of nodes and is called Generalized Insertion Heuristic (GIH). The following paragraphs describe these algorithms in detail.

3.1 Nearest Neighbor Heuristic for the GTSP (NNH)

Noon [17] suggested this approach, which computes a feasible solution as follows. We begin to construct a tour S_v from an arbitrarily chosen starting node $v \in V$. Iteratively, the algorithm always continues to the closest node belonging to a cluster that has not been visited yet and includes the corresponding edge. When nodes of all clusters have been reached, the tour is closed by including a final edge back to node v. This process is carried out once for each node in V as starting node, and the best tour is retained. See also Algorithm 1.

3.2 Generalized Insertion Heuristic for the GTSP (GIH)

This heuristic is inspired by the composite heuristic GI^3 from Renaud et al. [18]. In a first phase, it determines the set of spanned nodes P by calculating for

Algorithm 1: Nearest Neighbor Heuristic

```
for v ∈ |V| do
    S_v = ∅
    W = V
    add v to S_v
    v' = v
    for i = 1, ..., r − 1 do
        remove from W all nodes belonging to the same cluster as v'
        u = node in W nearest to v'
        add u to the partial tour S_v
        v' = u
return tour S = S_{v*} with v* = argmin_{v∈V} C(S_v)
```

each cluster V_i the node p_i having the lowest sum of distances to all other nodes in other clusters. After fixing these nodes, the CLOCK heuristic from [19] is performed to construct a tour containing many but not necessarily all nodes of P.

Recall that GIH only works on Euclidean instances where the nodes' coordinates are given. The CLOCK heuristic begins a partial tour S at the northernmost node from P. In case of a tie, the easternmost node among the northernmost nodes is chosen. This initial insertion is followed by four loops: In the first loop the procedure appends to S the northernmost node to the east of the last inserted node. In case of a tie, the easternmost node among these is chosen again. The process is repeated until there are no nodes to the east of the last appended node. The second, third, and forth loops work in the same way by appending to S the easternmost node to the south, the southernmost node to the west, and the westernmost node to the north of the last inserted node, respectively.

When the CLOCK heuristic terminates, there are in general some nodes from P left which are not yet included in the tour S. In contrast to the more complex GI^3 heuristic [18], we simply choose for each of these remaining nodes $p_j \in P \setminus H$ greedily the "cheapest" insertion position k so that $c(p_{\pi_{k-1}}, p_j) + c(p_j, p_{\pi_k}) - c(p_{\pi_{k-1}}, p_{\pi_k}) \leq c(p_{\pi_{i-1}}, p_j) + c(p_j, p_{\pi_i}) - c(p_{\pi_{i-1}}, p_{\pi_i})\ \forall i = 1, \ldots, |H|$ with $\pi_0 = \pi_{|H|}$.

As a final step, we try to improve the obtained feasible tour S by calling the shortest path algorithm, which will be introduced in Sect. 4.1. This procedure may replace nodes by other nodes of the same cluster, but it does not modify the visiting order π anymore. See Algorithm 2 for more details of the whole GIH.

This construction heuristic is much faster than the original GI^3, mainly because the latter uses a more sophisticated local improvement. Nevertheless, solutions obtained by GIH are typically only slightly inferior, and it usually takes just a few VNS iterations to catch up with or exceed the quality of GI^3's solutions.

While NNH has time complexity $\Theta(r \cdot |V|^2)$, GIH can be implemented in time $\Theta(|V|^2)$ and usually finds significantly better solutions to Euclidean instances with geographical clustering. However, GIH's applicability is far more limited.

Algorithm 2: Generalized Insertion Heuristic

for $i = 1, \ldots, r$ **do**
  p_i = node in V_i with the least sum of costs to all other nodes in other clusters
partial tour S = CLOCK heuristic($\{p_1, \ldots, p_r\}$)
for $j = 1, \ldots, r$ **do**
  if $p_j \notin S$ **then**
    $k = \text{minarg}_{i=1,\ldots,|S|}\ (c(p_{\pi_{i-1}}, p_j) + c(p_j, p_{\pi_i}) - c(p_{\pi_{i-1}}, p_{\pi_i}))$, $\pi_0 = \pi_{|S|}$
    insert p_j at position k in S
apply shortest path algorithm on S
return S

4 Neighborhood Structures

In our VNS, we use two complementary neighborhood structures. On the one hand, we approach the GTSP from the *global view* by first deciding in which order the clusters are to be visited and then computing an optimal selection of spanned nodes. On the other hand, we may start from the opposite direction and define a set of nodes for which we derive an appropriate tour.

4.1 Generalized 2-opt Neighborhood (G2-opt)

Renaud et al. [18] introduced the generalized 2-opt heuristic, which is based on the well known 2-opt heuristic for the classical TSP [14]. G2-opt is defined on a circular permutation $\pi = \langle \pi_1, \ldots, \pi_r \rangle$ indicating the visiting order of the clusters $\langle V_{\pi_1}, \ldots, V_{\pi_r} \rangle$, see Fig. 2(a). A particular permutation π thus represents the set of all feasible round trips $\langle p_{\pi_1}, p_{\pi_2}, \ldots, p_{\pi_r} \rangle$ with $p_{\pi_i} \in V_{\pi_i}$, $i = 1, \ldots, r$, and this set is in general exponentially large with respect to the number of nodes. However, the minimum cost round trip can be determined via a shortest path algorithm in polynomial time.

Given the visiting order of clusters, we can construct a graph containing edges only between nodes of consecutive clusters and a clone of the starting cluster attached to the last cluster, as it is shown in Fig. 2(b).

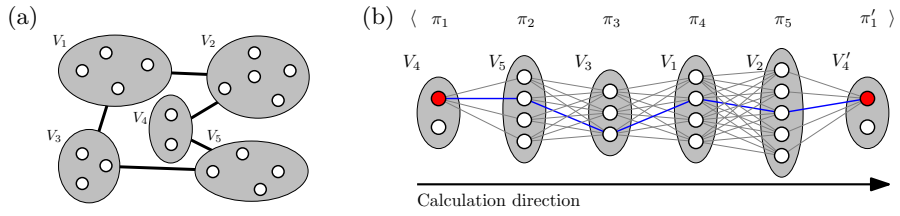

Fig. 2. (a) Visiting order of clusters characterized by permutation $\pi = \langle 4, 5, 3, 1, 2 \rangle$ and (b) corresponding graph on which the shortest path algorithm is applied, starting at the first node of cluster V_4 and ending at its clone in cluster V'_4.

On this graph, we calculate shortest paths starting from each node of the starting cluster and ending at its clone. To ensure that at most one node is included from each cluster, we may simply assume the edges to be directed according to π. The overall cheapest path represents the optimal tour for this cluster order. Formally speaking, let L_{uv} denote the length of the shortest path from node $u \in V_{\pi_k}$ to node $v \in V_{\pi_l}$, $k < l$. Let L_u be the length of the shortest path containing r-edges, starting from $u \in V_{\pi_1}$, and ending at its clone in V'_{π_1}. The length L of the overall shortest tour respecting visiting order π is:

$$
\begin{array}{ll}
L = \min_{u \in V_{\pi_1}} L_u & \\
L_u = \min_{v \in V_{\pi_r}} (L_{uv} + c(v,u)) & \forall u \in V_{\pi_1} \\
L_{uv} = c(u,v) & \forall u \in V_{\pi_1},\ \forall v \in V_{\pi_2} \\
L_{uv} = \min_{w \in V_{\pi_{k-1}}} (L_{uw} + c(w,v)) & \forall u \in V_{\pi_1},\ \forall v \in V_{\pi_k},\ k = 3, \ldots, r
\end{array}
$$

To reduce the computational effort, we exploit the fact that π is rotation-invariant and choose V_{π_1} so that it is a cluster of smallest cardinality. The complexity of this dynamic programming algorithm is bounded by $O(|V_{\pi_1}| \cdot n^2/r)$.

Our generalized 2-opt neighborhood of a current solution S having cluster ordering π can now be defined as the set of all feasible round trips induced by any cluster ordering π' that differs from π by precisely one inversion I_{ij}, i.e. $\pi' = \langle \pi_1, \ldots, \pi_{i-1}, \pi_j, \ldots, \pi_i, \pi_{j+1}, \ldots, \pi_r \rangle,\ 1 \leq i < j \leq r$.

Incremental bidirectional shortest path calculation. Instead of determining the shortest path L always from V_{π_1} (a cluster with the smallest number of nodes) to the cloned cluster V'_{π_1}, we can partition this task into three parts:

1. Perform shortest path calculations in forward direction from $u \in V_{\pi_1}$ to each node of a cluster V_{π_m} where m may be chosen arbitrarily.
2. Perform shortest path calculations in backward direction starting from $u' \in V'_{\pi_1}$ to each node of cluster $V_{\pi_{m+1}}$ where u' is the clone of node u.
3. Consider all edges in $E^m = \{(a,b) \in E \mid a \in V_{\pi_m} \wedge b \in V_{\pi_{m+1}}\}$ and the corresponding complete paths from u to u' including the above determined shortest paths to nodes in V_{π_m} and $V_{\pi_{m+1}}$, respectively. Take a $(a^*, b^*) \in E^m$ yielding an overall shortest path, i.e. $L_{ua^*} + c(a^*, b^*) + L_{b^*u'} \leq L_{ua} + c(a,b) + L_{bu'}\ \forall (a,b) \in E^m$.

This procedure, illustrated in Fig. 3, is in practice almost equally efficient as the simple one-way dynamic programming algorithm. When considering that we want to search the general 2-opt neighborhood, however, it provides the advantage of allowing for a substantially faster incremental evaluation scheme: If π' differs from π by an inversion I_{ij} with $i \leq m \leq j$, we do not have to recalculate the distances and predecessors of the nodes in clusters $V_{\pi_1}, \ldots, V_{\pi_{i-1}}$ and $V_{\pi_{j+1}}, \ldots, V'_{\pi_1}$, assuming we have stored these values in steps 1 and 2 before.

As a matter of course, m is always chosen to lie within the inversion interval. Clusters from V_{π_i} to V_{π_j} are marked "invalid" for both calculation directions after performing the inversion. Whenever we apply the shortest path algorithm

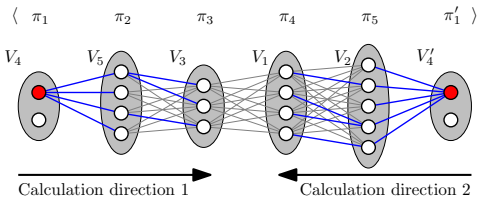

Fig. 3. Example for a bidirectional shortest path calculation with $m = 3$.

in a particular direction, the evaluation is skipped for all clusters which are still valid, and the actual computation starts at the first invalid cluster. When processing these clusters, their "invalid" flags are removed.

To fully exploit this incremental evaluation, we further enumerate the possible inversions of π in a specific way: First, all inversions of pairs of two adjacent clusters are considered from left to right, then the inversions of all triplets in the reverse direction from right to left, next all 4-cluster inversions from left to right again, etc. Hereby, π_1 (and its clone in the corresponding graph for the shortest path calculation) remain fixed. See also Fig. 4. This strategy allows the largest data-reuse and minimizes the total number of clusters for which computations are necessary. It is in particular advantageous when we use a *next improvement* strategy in the local search, since we start with inversions of smallest size yielding the largest time savings; see Algorithm 3.

In the worst case, when we have to evaluate the whole neighborhood, $O(r^2)$ inversions must be considered. A naive complete enumeration would require time $O(r^2 \cdot |V_{\pi_1}| \cdot n^2/r) = O(|V_{\pi_1}| \cdot n^2 \cdot r)$. To be more precise, we have $(r - l + 1)$ possibilities for inversions of length l, $l = 2, \ldots, r-2$. For each of them, the classical shortest path algorithm would have to consider all r clusters. However, with the incremental bidirectional shortest path calculation, we only have to consider $l + 1$ clusters after the first iteration. Hence, the classical algorithm evaluates $\sum_{l=2}^{r-2} r \cdot (r - l + 1) = \frac{r^3 - r^2 - 6r}{2}$ clusters while the incremental scheme

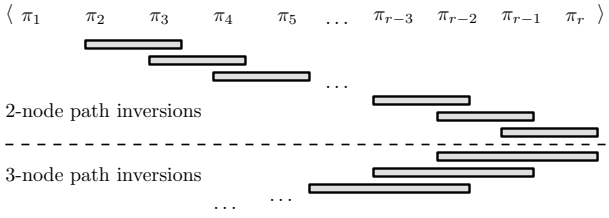

Fig. 4. Enumeration order of the inversions on π for making best use of the incremental bidirectional shortest path calculations.

Algorithm 3: Search Generalized 2-opt Neighborhood (S)

```
for l = 2, ..., r − 2 do
    if l is even then
        for i = 2, ..., r − l + 1 do
            π' = ⟨π_1, ..., π_{i−1}, π_{i+l−1}, ..., π_i, π_{i+l}, ..., π_r⟩
            Apply incremental bidirectional shortest path calculation on π'
            if obtained solution S' is better than original solution S then
                return solution S'
    else
        for i = r − l + 1, ..., 2 do
            π' = ⟨π_1, ..., π_{i−1}, π_{i+l−1}, ..., π_i, π_{i+l}, ..., π_r⟩
            Apply incremental bidirectional shortest path calculation on π'
            if obtained solution S' is better than original solution S then
                return solution S'
return: no better solution found, i.e. S is a local optimum w.r.t. G2-opt
```

only processes $\sum_{l=2}^{r-2}(l+1)\cdot(r-l+1) = \frac{r^3+6r^2-25r-6}{6}$ clusters for the whole neighborhood. Asymptotically, the latter is faster by factor 3.

4.2 Node Exchange Neighborhood (NEN)

With this new neighborhood structure, the search focuses on the set of spanned nodes $P = \{p_1, \ldots, p_r\}$. The node exchange neighborhood of a current solution S with spanned nodes P includes all feasible tours S' for each node set P' that can be derived from P by replacing one spanned node $p_i \in V_i$, $i \in \{1, \ldots, r\}$, by another node v of the same cluster V_i. This neighborhood therefore is induced by $\sum_{i=1}^{r}(|V_i|-1) = O(|V|)$ different node sets resulting in a total of $O(|V|\cdot r!)$ round trips.

Unfortunately, determining the minimum cost round trip for a given node set P' is NP-hard since this subproblem corresponds to the classical TSP. Hence, instead of calculating the optimal round trip, we use the well known Chained Lin-Kernighan (CLK) algorithm [15] implemented in the Concorde library[1] to find a good but not necessarily optimal tour S' for a certain P'.

Though the size of this TSP is relatively small ($|P'| = r$), a complete evaluation of NEN is relatively time-demanding – even when using CLK – since we have to solve $O(|V|)$ different TSPs. To further speed up the neighborhood search, we restrict CLK to consider edges of the k-nearest-neighbor graph induced by P' only. For Euclidean instances and available point positions, this k-nearest-neighbor graph is efficiently derived using a KD-tree data structure. Tuning the parameter k, we can balance between speed and thoroughness of the search process. For the actual tests, we set k to 10. Algorithm 4 summarizes the steps of evaluating NEN.

[1] www.tsp.gatech.edu/concorde.html

Algorithm 4: Search Node Exchange Neighborhood (S)

```
for i = 1, ..., r do
    forall v ∈ V_i \ {p_i} do
        P' = P \ {p_i} ∪ {v}
        determine k-nearest-neighbor graph G^k induced by P'
        apply CLK on G^k to obtain round trip S'
        if current solution S' better than so far best then
            save S' as so far best
restore and return best solution found
```

5 Variable Neighborhood Search Framework

We use the general variable neighborhood search (VNS) scheme with embedded variable neighborhood descent (VND) as proposed in [6].

Arrangement of the neighborhoods in VND: We alternate between G2-opt and NEN in this order. G2-opt is always considered first since its evaluation has a lower computational complexity.

Shaking in VNS: To perform shaking, we randomly exchange s spanned nodes by other nodes of the corresponding clusters and apply s random swap moves on the cluster ordering π. A swap move exchanges two positions in π. Parameter s depends on the number of clusters in the input graph and varies from 1 to $\frac{r}{7}$. We obtained the best results with these settings for s during our tests.

6 Computational Results

We tested the VNS on TSPlib[2] instances with geographical clustering which is done as follows [3]. 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, the clustering is done by assigning each of the remaining nodes to its nearest center node. We consider the largest of such TSPlib instances with up to 442 nodes, 97461 edges, and 89 clusters.

Our experiments were performed on a Pentium 4 2.6 GHz PC. In order to compute average values and standard deviations, we performed 30 runs for each instance. The VNS terminated after 10 consecutive outer iterations without finding a new best solution.

Table 1 presents results of our VNS and compares them to those of Fischetti et al's exact branch-and-cut algorithm (B&C) [5], the GI^3 heuristic [19], the random key GA (rk-GA) [21], and the hybrid chromosome GA (hc-GA) [8].

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

Listed are for each instance its name, the numbers of nodes and clusters, the optimal solution value and run-time of B&C, and average percentage gaps of the heuristics' final objective values to the optimum solution value, as well as corresponding CPU-times. Best results are printed bold.

Since the B&C algorithm ran on a HP 9000/720, the GI^3 heuristic on a Sun Sparc Station LX, the rk-GA on a Pentium 4, 3.2 GHz PC, and the hc-GA on a 1.2 GHz PC, it is hard to compare the CPU-times directly. Nevertheless, it is obvious that in particular the rk-GA is very fast and computes high quality results within a few seconds. Comparing VNS with both GAs, VNS requires significantly more time, but it is often able to find superior solutions.

Especially on the larger instances with ≥ 299 nodes, VNS benefits from the sophisticated large neighborhood search and its average gaps of are consistently substantially smaller than those of all other considered heuristics. For 18 out of the 28 instances, VNS was even able to obtain optimal solutions in all of the 30 performed runs; its total average gap is only 0.05%, and no average gap exceeds

Table 1. Results on TSPlib instances with geographical clustering.

Instance			B&C		GI^3		rk-GA		hc-GA		VNS		
Name	$\|V\|$	r	C_{opt}	*time*	*gap*	*time*	$\overline{gap}$	$\overline{time}$	$\overline{gap}$	$\overline{time}$	$\overline{gap}$	σ_{gap}	$\overline{time}$
kroa100	100	20	9711	18.4s	**0.00%**	6.8s	**0.00%**	0.4s	-	-	**0.00%**	0.00	2.5s
krob100	100	20	10328	22.2s	**0.00%**	6.4s	**0.00%**	0.4s	-	-	**0.00%**	0.00	0.4s
rd100	100	20	3650	16.6s	**0.00%**	7.3s	**0.00%**	0.5s	-	-	**0.00%**	0.00	0.9s
eil101	101	21	249	25.6s	0.40%	5.2s	**0.00%**	0.4s	-	-	0.04%	0.12	16.3s
lin105	105	21	8213	16.4s	**0.00%**	14.4s	**0.00%**	0.5s	-	-	**0.00%**	0.00	0.6s
pr107	107	22	27898	7.4s	**0.00%**	8.7s	**0.00%**	0.4s	-	-	**0.00%**	0.00	0.5s
pr124	124	25	36605	25.9s	0.43%	12.2s	**0.00%**	0.6s	-	-	**0.00%**	0.00	26.6s
bier127	127	26	72418	23.6s	5.55%	36.1s	**0.00%**	0.4s	-	-	**0.00%**	0.00	1.4s
pr136	136	28	42570	43.0s	1.28%	12.5s	**0.00%**	0.5s	-	-	**0.00%**	0.00	48.1s
pr144	144	29	45886	8.2s	0.00%	16.3s	**0.00%**	1.0s	-	-	**0.00%**	0.00	4.0s
kroa150	150	30	11018	100.3s	0.00%	17.8s	**0.00%**	0.7s	**0.00%**	0.4s	**0.00%**	0.00	1.2s
krob150	150	30	12196	60.6s	0.00%	14.2s	**0.00%**	0.9s	**0.00%**	0.9s	**0.00%**	0.00	3.7s
pr152	152	31	51576	94.8s	0.47%	17.6s	**0.00%**	1.2s	**0.00%**	0.6s	**0.00%**	0.00	7.6s
u159	159	32	22664	146.4s	2.60%	18.5s	**0.00%**	0.8s	**0.00%**	1.0s	**0.00%**	0.00	22.6s
rat195	195	39	854	245.9s	**0.00%**	37.2s	**0.00%**	1.0s	-	-	0.01%	0.04	105.6s
d198	198	40	10557	763.1s	0.60%	60.4s	**0.00%**	1.6s	-	-	0.02%	0.05	141.3s
kroa200	200	40	13406	187.4s	**0.00%**	29.7s	**0.00%**	1.8s	0.01%	1.8s	**0.00%**	0.00	16.9s
krob200	200	40	13111	268.5s	**0.00%**	35.8s	**0.00%**	1.9s	0.06%	8.0s	**0.00%**	0.00	18.8s
ts225	225	45	68340	37875.9s	0.61%	89.0s	**0.02%**	2.1s	0.13%	19.0s	0.03%	0.07	274.4s
pr226	226	46	64007	106.9s	**0.00%**	25.5s	**0.00%**	1.5s	**0.00%**	0.6s	**0.00%**	0.00	1.7s
gil262	262	53	1013	6624.1s	5.03%	115.4s	0.79%	1.9s	**0.00%**	41.2s	0.05%	0.16	372.5s
pr264	264	53	29549	337.0s	0.36%	64.4s	**0.00%**	2.1s	**0.00%**	3.1s	0.01%	0.04	268.2s
pr299	299	60	22615	812.8s	2.23%	90.3s	0.11%	3.2s	0.10%	68.6s	**0.00%**	0.01	220.5s
lin318	318	64	20765	1671.9s	4.59%	206.8s	0.62%	3.5s	0.72%	18.3s	**0.30%**	0.61	320.1s
rd400	400	80	6361	7021.4s	1.23%	403.5s	1.18%	5.9s	2.15%	17.4s	**0.74%**	0.51	502.0s
fl417	417	84	9651	16719.4s	0.48%	427.1s	0.05%	5.3s	0.12%	19.4s	**0.00%**	0.00	92.4s
pr439	439	88	60099	5422.8s	3.52%	611.0s	0.26%	9.5s	0.76%	10.9s	**0.12%**	0.11	519.0s
pcb442	442	89	21657	58770.5s	5.91%	567.7s	1.70%	9.0s	0.94%	31.8s	**0.08%**	0.08	596.6s
Average gaps					1.26%		0.17%		0.30%		0.05%		

0.75%. From all considered heuristics, GI^3 was the weakest, with worst average results and running times in the same order of magnitude as our VNS.

In particular for the two large instances pr226 and fl417, already VND was able to directly identify the optimal solutions, i.e. merely alternating between G2-opt and NEN was sufficient to get to the global optima, and no VNS iterations were required. This documents how well these neighborhood structures complement each other.

7 Conclusions and Future Work

In this article, we proposed a variable neighborhood search approach for the generalized traveling salesman problem utilizing two large neighborhood structures. They can be seen as dual to each other: While G2-opt predefines the possible cluster orderings and uses a relatively sophisticated but efficient procedure for augmenting these partial solutions with appropriate selections of nodes, the situation is vice versa in the newly proposed NEN.

Considering in particular G2-opt, the described incremental evaluation scheme turned out to be a major speed-up factor in comparison to the previously used evaluation via independent standard shortest path calculations.

It further turned out that the VNS slightly benefits from a good starting solution. Therefore, we described the more generally applicable nearest neighbor heuristic and particularly for Euclidean instances with given point positions the generalized insertion heuristic. Both are reasonably fast and provide solutions of appropriate quality.

We tested the VNS on TSPlib instances with geographical clustering consisting of up to 442 nodes. Compared to two recent genetic algorithms, the VNS performs slower, but it is able to generate remarkably better solutions, in particular for larger instances.

Future work will in particular include tests on other types of instances, e.g. with non-Euclidean distances and incomplete graphs. An incremental evaluation scheme for NEN seems to be a challenging task but might further speed up the algorithm. Promising is also the combination of these neighborhood structures with others, and to investigate their application in other types of metaheuristics.

References

1. A. Behzad and M. Modarres. A new efficient transformation of the generalized traveling salesman problem into traveling salesman problem. In *Proceedings of the 15th International Conference of Systems Engineering*, pages 6–8, 2002.
2. V. Dimitrijevic and Z. Saric. An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs. *Information Science*, 102(1-4):105–110, 1997.
3. C. Feremans. *Generalized Spanning Trees and Extensions*. PhD thesis, Universite Libre de Bruxelles, 2001.
4. M. Fischetti, J. J. Salazar, and P. Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26:113–123, 1995.

5. M. Fischetti, J. J. Salazar, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45:378–394, 1997.
6. P. Hansen and N. Mladenovic. An introduction to variable neighborhood search. In S. Voss et al., editors, *Meta-heuristics, Advances and trends in local search paradigms for optimization*, pages 433–458. Kluwer Academic Publishers, 1999.
7. Henry-Labordere. The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem. *RAIRO Operations Research*, B2:43–49, 1969.
8. H. Huang, X. Yang, Z. Hao, C. Wu, Y. Liang, and X. Zhao. Hybrid chromosome genetic algorithm for generalized traveling salesman problems. *Advances in Natural Computation*, 3612/2005:137–140, 2005.
9. G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah. Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society*, 47(12):1461–1467, 1996.
10. G. Laporte, H. Mercure, and Y. Nobert. Generalized traveling salesman problem through n sets of nodes: The asymmetric case. *Discrete Applied Mathematics*, 18:185–197, 1987.
11. G. Laporte and Y. Nobert. Generalized traveling salesman problem through n sets of nodes: An integer programming approach. *INFOR*, 21, issue 1:61–75, 1983.
12. G. Laporte and F. Semet. Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. *INFOR*, 37(2):114–120, 1999.
13. Y. N. Lien, E. Ma, and B. W. S. Wah. Transformation of the generalized traveling salesman problem into the standard traveling salesman problem. *Information Sciences*, 74(1–2):177–189, 1993.
14. S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Computer Journal*, 44:2245–2269, 1965.
15. O. Martin, S. W. Otto, and E. W. Felten. Large-step Markov chains for the traveling salesman problem. *Complex Systems*, 5:299–326, 1991.
16. C. Noon and J. C. Bean. An efficient transformation of the generalized traveling salesman problem. *INFOR*, 31(1):39–44, 1993.
17. C. E. Noon. *The Generalized Traveling Salesman Problem.* PhD thesis, University of Michigan, 1988.
18. J. Renaud and F. F. Boctor. An efficient composite heuristic for the symmetric generalized traveling salesman problem. *European Journal of Operational Research*, 108:571–584, 1998.
19. J. Renaud, F. F. Boctor, and G. Laporte. A fast composite heuristic for the symmetric traveling salesman problem. *INFORMS Journal on Computing*, 8, issue 2:134–143, 1996.
20. J. P. Saskena. Mathematical model of scheduling clients through welfare agencies. *Journal of the Canadian Operational Research Society*, 8:185–200, 1970.
21. L. V. Snyder and M. S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. Technical Report 04T-018, Dept. of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA, 2004.
22. Srivastava, S. S. S. Kumar, R. C. Garg, and P. Sen. Generalized traveling salesman problem through n sets of nodes. *CORS Journal*, 7:97–101, 1969.
23. C. Wu, Y. Liang, H. P. Lee, and C. Lu. Generalized chromosome genetic algorithm for generalized traveling salesman problems and its applications for machining. *Physical Review E*, 70, issue 1, 2004.