New Neighborhoods and an Iterated Local Search Algorithm for the Generalized Traveling Salesman Problem

Jeanette Schmidt, Stefan Irnich

September 25, 2020

Discussion paper number 2020

Contact Details:

Jeanette Schmidt
Chair of Logistics Management
Johannes Gutenberg University
Jakob-Welder-Weg 9
55128 Mainz, Germany
sjeanett@uni-mainz.de

Stefan Irnich
Chair of Logistics Management
Johannes Gutenberg University
Jakob-Welder-Weg 9
55128 Mainz, Germany
irnich@uni-mainz.de

New Neighborhoods and an Iterated Local Search Algorithm for the Generalized Traveling Salesman Problem

Jeanette Schmidt[*,a], Stefan Irnich[a]

[a] *Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany.*

Abstract

The generalized traveling salesman problem (GTSP) is the problem of finding a cost-minimal cycle in a clustered graph so that exactly one vertex of every cluster is contained in the cycle. We introduce three new GTSP neighborhoods that allow the simultaneous permutation of the sequence of the clusters and the selection of vertices from each cluster. The three neighborhoods and some known neighborhoods from the literature are combined into a simple but effective iterated local search (ILS) for the GTSP. The simplicity of the ILS consists in its straightforward random neighborhood selection within the local search and an ordinary record-to-record ILS acceptance criterion. The computational experiments on four symmetric standard GTSP libraries show that, with some small refinements, the ILS can compete with state-of-the-art algorithms, although it is simple in structure and less involved to code compared to many other metaheuristics.

Key words: traveling salesman, generalized traveling salesman problem, iterated local search, variable neighborhood descent

1. Introduction

The *generalized traveling salesman problem* (GTSP) is the problem of finding a cost-minimal cycle in a clustered graph so that exactly one vertex of every cluster is contained in the cycle. Formally, an instance of the symmetric GTSP is defined by an edge-weighted complete undirected graph $G = (V, E)$, where V denotes the set of vertices and E the set of edges. The vertices are partitioned into N non-empty disjoint subsets, denoted as *clusters* and indexed by $i \in I = \{1, 2, \dots, N\}$, such that $V = \bigcup_{i \in I} V_i$. For a vertex $v \in V$, let $i = [v] \in I$ be the index of the cluster to which the vertex belongs, i.e., $v \in V_{[v]}$. The edge set E comprises (ordered) pairs vw of vertices $v, w \in V$ for $[v] \neq [w]$, implying $vw \equiv wv$ and symmetric edge weights $c_{vw} = c_{wv}$. We use the symbols n for the cardinality of the vertex set and m_i for the size of the ith cluster, i.e., $m_i = |V_i|$ for all $i \in I$. A feasible solution to the GTSP is a cycle $x = (x_1, x_2, \dots, x_N, x_1)$ with exactly one vertex per cluster, i.e., $[x_i] \neq [x_j]$ for all $i, j \in I, i \neq j$. Such a cycle is also denoted as a *G-tour*. The cost of the G-tour x is defined as $c(x) = \sum_{i \in I} c_{x_i x_{i+1}}$ (assuming $x_{N+1} = x_1$). The objective of the GTSP is to find a minimum-cost G-tour.

In the paper at hand, we present a simple but effective metaheuristic for the GTSP. The metaheuristic follows the principles of a clean *iterated local search* (ILS, Lourenço *et al.*, 2003). The local search part of the ILS combines a small subset of known and three new GTSP neighborhoods in a *variable neighborhood descent* (VND, Hansen and Mladenović, 2001) fashion. The simplicity of the ILS consists in its straightforward random neighborhood selection within the VND and an ordinary record-to-record ILS acceptance criterion (Dueck, 1993).

[*]Corresponding author.
Email addresses: `sjeanett@uni-mainz.de` (Jeanette Schmidt), `irnich@uni-mainz.de` (Stefan Irnich)

One contribution of our work is the introduction of three new neighborhoods for the GTSP that allow the simultaneous permutation of the sequence of the clusters and the selection of vertices from each cluster. Despite the exponential size of the new neighborhoods, they can be explored efficiently: We present three neighborhood exploration algorithms, all with polynomial worst-case complexity, where two algorithms allow the complete exploration of the respective neighborhood so that a provably best improving neighbor (if existent) is found. For the third new neighborhood, a polynomial time heuristic neighborhood exploration is presented. With these components, our basic ILS is already competitive with GTSP metaheuristics from the literature.

The second contribution is the refinement of the basic ILS regarding a reset mechanism to the best found solution, the neighborhood prioritization in the VND, and the use of the GTSP-adapted Balas-Simonetti neighborhood (Balas and Simonetti, 2001) for improving high-quality solutions. Moreover, we present a straightforward grouping scheme for GTSP instances that allow us to control which algorithmic refinements are to be applied to which type of GTSP instance. In computational experiments on standard GTSP benchmark instances we show that the resulting refined ILS produces excellent solutions, e.g., outperforming the GLNS metaheuristic of Smith and Imeson (2017, reviewed in the next section) on the `GTSP_LIB` (Fischetti *et al.*, 1997; Silberholz and Golden, 2007) regarding average gaps. Finally, we find two new best-known solutions, one in the `GTSP_LIB` and one in the `LARGE_LIB` (Helsgaun, 2015).

The remainder of the paper is structured as follows. In Section 2, we review the pertinent GTSP literature. We describe GTSP neighborhoods and corresponding efficient neighborhood exploration algorithms used in our ILS in Section 3. The overall ILS is presented in Section 4 together with computational results obtained with the basic ILS implementation. Refinements of the ILS tailored to specific groups of GTSP instances are presented in Section 5. Here, we introduce measures that decide which ILS variant to apply for a given GTSP instance. The paper closes with final conclusions drawn in Section 6.

2. Literature Review

A variety of combinatorial optimization problems can be modeled as GTSPs, or contain the GTSP as a subproblem. Among them are location routing problems, material-flow system design problems, post-box collection problems (Laporte *et al.*, 1996) as well as the routing of clients through welfare agencies (Saksena, 1970), and computer file sequencing (Henry-Labordere, 1969).

Since its introduction in the late sixties/early seventies (Henry-Labordere, 1969; Saksena, 1970) a lot of attention has been paid to solving the GTSP. One group of solution approaches relies on the fact that every GTSP instance can be transformed into an equivalent *traveling salesman problem* (TSP) instance. Thus, different reduction algorithms were developed, e.g., by Noon and Bean (1993); Laporte and Semet (1999); Ben-Arieh *et al.* (2003). The resulting TSP instances can then be solved with a TSP solver, either heuristically or, if not too large, exactly. For example, Helsgaun (2015) combined the Noon-Bean reduction with the Lin-Kernighan-Helsgaun algorithm (LKH, Helsgaun, 2000) into a powerful GTSP solver.

Several exact approaches for the GTSP have shown very good results: The branch-and-cut algorithm of Fischetti *et al.* (1997) solves symmetric GTSP instances with up to 89 clusters and 442 vertices to optimality. A Lagrangian based approach to solve asymmetric GTSP instances was developed by Noon and Bean (1991). Their results show success on a range of randomly generated instances with up to 100 vertices. Solving larger instances to proven optimality can still be a very hard task nowadays.

Certainly, large-scale GTSP instances require heuristic solution approaches. Many different approaches have been published, ranging from simple tour construction heuristics (e.g., Noon, 1988) to more involved and rather effective metaheuristics. Table 1 provides an overview of the pertinent GTSP publications. Almost all listed metaheuristics have at least one thing in common: They contain local optimization techniques that run one or more local/neighborhood-search heuristics to improve a given solution (an exception is Pintea *et al.*, 2017). A first approach, called RP2 and nowadays known as *cluster optimization* (CO), was invented by Fischetti *et al.* (1997). CO determines a globally best vertex selection according to a given and fixed cluster sequence. This is done by constructing and solving a shortest-path problem in a layered network (a detailed description follows in Section 3.3). Because of its efficiency, CO can be found in many different

Table 1: Selected Literature on (Meta)Heuristics for the GTSP.

(Meta)heuristic	Reference(s)
Tour construction heuristic	Noon (1988); Fischetti *et al.* (1997)
Composite heuristic	Renaud and Boctor (1998)
Adaptive Large Neighborhood Search	Smith and Imeson (2017)
Ant Colony Optimization	Yang *et al.* (2008); Reihaneh and Karapetyan (2012); Pintea *et al.* (2017)
Genetic/memetic Algorithm	Snyder and Daskin (2006); Silberholz and Golden (2007); Gutin *et al.* (2008); Gutin and Karapetyan (2010); Bontoux *et al.* (2010)
Lin-Kerninghan adaption	Karapetyan and Gutin (2011)
Multi-start method	Cacchiani *et al.* (2011)
Particle Swarm Optimization	Tasgetiren *et al.* (2007)
Variable Neighborhood Search	Hu and Raidl (2008)

GTSP algorithms, e.g., of Cacchiani *et al.* (2011); Reihaneh and Karapetyan (2012); Smith and Imeson (2017).

Another straightforward approach is to use a standard TSP improvement procedure, such as the 2-Opt, 3-Opt, and k-Opt (for $k \geq 4$) heuristics (Lin, 1965). Such improvement procedures have been used in Yang *et al.* (2008) and Bontoux *et al.* (2010). CO and k-Opt are two extremes of GTSP improvement procedures, where the first only optimizes the vertex selection per cluster and the second only optimizes the sequence in which the clusters are visited. The other decision remains fully fixed in both cases.

The disadvantage of the approaches that work only on one type of improvement method is that they are too myopic. Note that often a local improvement requires a modification in both the vertex selection and cluster sequence. In order to overcome this disadvantage, several researchers have developed GTSP-tailored neighborhoods so that both types of decisions can change within one move. One of them is the RP1 procedure by Fischetti *et al.* (1997) based on 2-Opt and 3-Opt exchanges. Renaud and Boctor (1998) introduced the G2-Opt, G3-Opt, and G-Opt heuristics, which reverse tour segments and determine an optimal vertex selection via the CO algorithm. Finally, Renaud and Boctor combine G-Opt and G2-Opt to a powerful improvement part of their composite heuristic. Some years later, Hu and Raidl (2008) revisited the G2-Opt heuristic and refined the vertex selection process for a given cluster sequence. In detail, they apply an incremental bidirectional shortest-path calculation to save computation time. Another interesting method is to adapt the classical TSP 2-Opt as suggested by Gutin and Karapetyan (2009).

Special k-Opt heuristics, like swap moves (special 4-Opt) and relocation moves (special 3-Opt) are well known from the TSP. They were also adapted for the GTSP, e.g., different types of swap moves can be found in (Gutin *et al.*, 2008; Gutin and Karapetyan, 2010). Adapted relocation moves, also known as insert or node shift moves, are used in (Snyder and Daskin, 2006; Silberholz and Golden, 2007; Tasgetiren *et al.*, 2007; Gutin *et al.*, 2008; Gutin and Karapetyan, 2010; Smith and Imeson, 2017). Bontoux *et al.* (2010) search for best relocation moves with a special dynamic-programming algorithm that is inspired by the work of Feillet *et al.* (2004) on the elementary shortest-path problem with resource constraints.

As the LKH algorithm is still the state-of-the-art heuristic for the classical TSP, it can also be used in the GTSP context. For example, Bontoux *et al.* (2010) use the original version within their memetic algorithm, while Karapetyan and Gutin (2011) developed different Lin-Kernighan adaptations for the GTSP.

It is beyond the scope of this paper to provide a comprehensive classification of all known and new neighborhoods. The article by Karapetyan and Gutin (2012) provides such a synopsis. Furthermore, they explain efficient neighborhood exploration algorithms for all neighborhoods.

3. Neighborhoods and Efficient Neighborhood Exploration

In this section, we describe the neighborhoods used in our ILS and efficient neighborhood exploration algorithms. We distinguish between pure TSP neighborhoods (Section 3.1), polynomially-sized GTSP neighborhoods (Section 3.2), and exponentially-sized GTSP neighborhoods (Section 3.3). For the latter, we introduce three neighborhoods that have not yet been considered in other works.

3.1. TSP Neighborhoods

We consider first (symmetric) TSP neighborhoods for the GTSP. Moves of this type do not change the selection of vertices from each cluster.

2-Opt and 3-Opt. The k-Opt neighborhoods have been made popular by the work of Lin (1965). The current TSP solution x is divided into $k \geq 2$ segments that can be inverted and permuted. The result is a neighborhood of size $\mathcal{O}(N^k)$.

We apply a cost-based pruning to accelerate the neighborhood exploration based on the gain criterion of Lin and Kernighan (1973). This technique is known under different names as fixed radius near neighbor search (Bentley, 1992), fixed radius search (Hoos and Stützle, 2005, p. 373f), or sequential search (Irnich *et al.*, 2006). For the GTSP, a prerequisite is to have, for each vertex $v \in V$, an ordered *neighbor list* of all other vertices sorted by increasing distance. For a G-tour $x = (x_1, x_2, \ldots, x_N, x_1)$, the complete neighbor lists of the vertices $x_1, x_2, \ldots, x_N$ should be thinned out so that they comprise only vertices from $\{x_1, x_2, \ldots, x_N\}$ and no vertices from $V \setminus \{x_1, x_2, \ldots, x_N\}$. This preparatory step takes $\mathcal{O}(nN)$ time and space. The actual neighborhood exploration is then drastically accelerated on average (Hoos and Stützle, 2005, p. 373f).

Double-Bridge. The double-bridge move (Johnson and McGeoch, 1997) is a special 4-Opt move that divides the given tour into four non-empty segments $x = (A, B, C, D)$ that are permuted into $x' = (A, D, C, B)$. Glover (1996) has shown that, for a given TSP tour x, the double-bridge neighborhood, which comprises $\mathcal{O}(N^4)$ tours, can be explored completely and efficiently in $\mathcal{O}(N^2)$ time and space, with the help of a dynamic program. We also use this indirect neighborhood exploration technique.

3.2. Polynomial GTSP Neighborhoods

Next we consider neighborhoods that are inspired by TSP neighborhoods but allow to modify the selection of a vertex for at least one cluster.

Relocation+. The classical TSP relocation move selects a vertex x_i in the given tour x, removes it from its current position, and inserts it between two other consecutive vertices x_j and x_{j+1}. For a G-tour x, it is now allowed to replace x_i by a vertex x'_i of the cluster $V_{[x_i]}$. Hence,

$$x = (x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_j, x_{j+1}, \ldots, x_N, x_1)$$

is altered into

$$x' = (x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_j, x'_i, x_{j+1}, \ldots, x_N, x_1)$$

with $[x'_i] = [x_i]$. The size of the neighborhood is $\mathcal{O}(nN)$.

Swap+. The swap neighborhood of a TSP tour selects two non-neighboring vertices and swaps them. For a G-tour x, the two swapped vertices x_i and x_j can be replaced by other vertices $x'_i \in V_{[x_i]}$ and $x'_j \in V_{[x_j]}$. Hence, the given

$$x = (x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_N, x_1)$$

is modified into

$$x = (x_1, x_2, \ldots, x_{i-1}, x'_j, x_{i+1}, \ldots, x_{j-1}, x'_i, x_{j+1}, \ldots, x_N, x_1)$$

with $[x'_i] = [x_i]$ and $[x'_j] = [x_j]$. The size of this GTSP neighborhood is $\mathcal{O}(n^2)$.

3.3. Exponential GTSP Neighborhoods

The following exponential GTSP neighborhoods can all be explored with an indirect search method such that the computational effort is polynomially bounded.

Cluster Optimization. CO takes a given G-tour $x = (x_1, x_2, \ldots, x_N, x_1)$ and replaces it by a shortest G-tour $x' = (x'_1, x'_2, \ldots, x'_N, x'_1)$ with $[x'_i] = [x_i]$ for all $i \in I$ (Fischetti *et al.*, 1997). Hence, the sequence of the clusters is kept, but different vertices in all clusters can be selected. This is a neighborhood of size $\mathcal{O}\left(\prod_{i \in I} m_i\right)$.

A best neighbor solution results from solving one or several shortest-path problems in the layered graph with clusters as layers, see Figure 1. Consecutive layers connect all vertices $x'_i \in V_{[x_i]}$ with all vertices $x'_{i+1} \in V_{[x_{i+1}]}$ for $i \in I$ with arc weights $c_{x'_i, x'_{i+1}}$. While in Figure 1 the first cluster is a singleton set, the general case requires the solution of one shortest-path problem for each possible $x'_1 \in V_{[x_1]}$ as a start and end vertex.

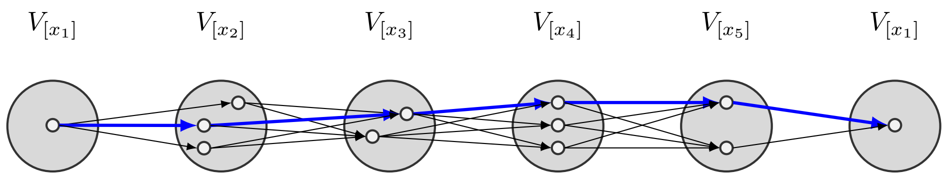

Figure 1: Layered Network for CO.

Karapetyan and Gutin (2012) suggest several techniques that aim at lowering the worst-case time complexity of the computation. First, every G-tour can be rotated so that w.l.o.g. $m_1 = \min_{i \in I} m_i$, i.e., the first cluster has minimum cardinality. Karapetyan and Gutin describe further implementation improvements such as the reduction of the first cluster (exploiting that the clusters $V_{[x_2]}$ and $V_{[x_N]}$ are known) and optimizing the dynamic-programming calculation order. They prove that CO can be searched efficiently in $\mathcal{O}(n \cdot \min_i m_i \cdot \max_i m_i)$ time. As the (practical) impact of the two last techniques is relatively minor, we only rotate the G-tour and use a first cluster of minimum cardinality.

Balas-Simonetti for the GTSP. In this section, we present a new neighborhood for the GTSP that is the synthesis of CO and the Balas-Simonetti neighborhood originally introduced for the TSP and TSP with time windows (Balas, 1999; Balas and Simonetti, 2001). We describe the TSP case first before we provide details about the synthesis.

For a given integer $k \geq 2$, the Balas-Simonetti (BS) neighborhood $\mathcal{N}_k^{BS}$ allows the restricted permutation of the vertices relative to a given tour or path. More precisely, for a given tour $x = (x_1, x_2, \ldots, x_N, x_1)$ another tour $x' = (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(N)}, x_{\pi(1)})$ defined by a permutation π on $\{1, 2, \ldots, N\}$ is in the neighborhood, i.e., $x' \in \mathcal{N}_k^{BS}(x)$, if

$$i + k \leq j \qquad \text{implies} \qquad \pi(i) < \pi(j) \qquad\qquad \text{for all } i, j \in \{1, 2, \ldots, N\}.$$

A larger value of k offers more flexibility so that the neighborhoods are nested, i.e., $\mathcal{N}_k^{BS} \subset \mathcal{N}_{k+1}^{BS}$ for all $k \geq 2$.

A least-cost neighbor can be determined by solving a shortest-path problem in an auxiliary network G_k. Figure 2 shows the auxiliary network G_k for $k = 2$. In general, the auxiliary network G_k is a layered network with $N+1$ layers $L_1, L_2, \ldots, L_N, L_{N+1}$ (assuming $L_{N+1} = L_1$), one for each position $(1, 2, \ldots, N, 1)$ of the given tour. All layers are identical. Each layer comprises exactly $(k+1)2^{k-2}$ states (three states for $k = 2$) partially describing the permutation π. To this end, states have an associated value α depicted left to the states of a row in Figure 2. Moreover, also two consecutive layers L_i and L_{i+1} induce identical subgraphs $G_k[L_i \cup L_{i+1}]$ for all $i \in \{1, 2, \ldots, N\}$. The number of arcs of $G_k[L_i \cup L_{i+1}]$ is bounded by $k(k+1)2^{k-2}$. For $k = 2$ in Figure 2, there are five ($\leq 2 \cdot 3 \cdot 2^0 = 6$) arcs connecting two layers.

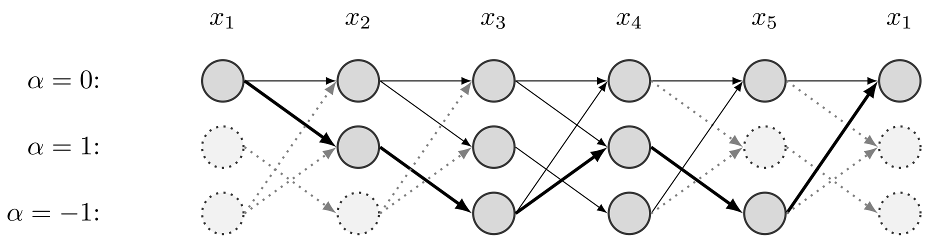

Figure 2: Auxiliary Network G_k for $k = 2$.

The point is now that every 0-0'-path in G_k describes exactly one permutation π and therefore the neighbor x', where $0 \in L_1$ and $0' \in L_{N+1}$ are specific null states in the layers (depicted on top in each layer in Figure 2, associated with $\alpha = 0$). A state with value α in ith layer refers to vertex $x_{i+\alpha}$. For example, the path depicted on top passing all vertices with values $\alpha = 0$ (using only the horizontally drawn arcs in Figure 2) represents the neighbor $x' = x$. The path drawn in bold represents the neighbor $x' = (x_{1+0}, x_{2+1}, x_{3+(-1)}, x_{4+1}, x_{5+(-1)}, x_{1+0}) = (x_1, x_3, x_2, x_5, x_4, x_1)$ of $x = (x_1, x_2, \ldots, x_5, x_1)$.

Accordingly, if arcs $(v, w) \in G_k[L_i \cup L_{i+1}]$ with values α_v for the state $v \in L_i$ and α_w for the state $w \in L_{i+1}$ are equipped with the cost $c_{x_{i+\alpha_v}, x_{x+1+\alpha_w}}$, the length of any 0-0'-path is identical to the length of the represented tour x'. Hence, solving a shortest-path problem between 0 and $0'$ in G_k provides a least-cost neighbor of x.

Some remarks are due:

- Since the network G_k is acyclic, a pulling or reaching type of dynamic-programming (DP) labeling approach for solving the shortest-path problem has a complexity proportional to the total number of arcs, i.e., $\mathcal{O}\left(N\, k(k+1)2^{k-2}\right)$. In particular, for a fixed k, the complexity is linear in the tour length N.
- Our sparse representation of the auxiliary network in the DP stores distance labels for all states. For the arcs, it suffices to only store the structure of $G_k[L_1 \cup L_2]$, because all consecutive layers are connected in the same way. Arc costs are computed on the fly.
- Some states in the first layers and in the last layers are irrelevant, because they are unreachable from 0 and 0' (backwardly). For the above-sketched implementation of the DP, the unreachable states pose to difficulty.

We can now combine CO and BS into a neighborhood that simultaneously permutes the order of the clusters (via BS) and allows to select alternative vertices from the permuted clusters. Figure 3 visualizes the idea for a given G-tour $x = (x_1, x_2, \ldots, x_N, x_1)$. Meta-states (big circles) represent the clusters that are initially sorted into the sequence $(V_{[1]}, V_{[2]}, \ldots, V_{[N]}, V_{[N+1]})$ (assuming $V_{[N+1]} = V_{[1]}$; meta-states are depicted only for the purpose of explanation). In Figure 3, the different number of states and their graphical positioning helps to distinguish between different clusters. For example, $V_{[1]}$ comprises one state, $V_{[2]}$ three states, and $V_{[3]}$ two states. Both $V_{[2]}$ and $V_{[4]}$ contain three states, but are depicted differently.

All states of a meta-state at the ith layer are connected with all states of a meta-state at the $(i + 1)$th layer (for $i \in \{1, 2, \ldots, N\}$) if and only if there is a corresponding arc in the original auxiliary network G_k. These complete connections between meta-states of consecutive layers are similar to the arcs in the CO network. Indeed, for $k = 1$ (note that in this case the BS neighborhood of the TSP degenerates to $\{x\} = \mathcal{N}_1^{BS}(x)$) the CO network shown in Figure 1 is identical to G_1. This network can also be found at the top of Figure 3 if only meta-states with $\alpha = 0$ are considered.

The new GTSP neighborhood $\mathcal{N}_k^{BS}(x)$ is huge having (at least) $(k/e)^{N-1} \cdot \prod_{i \in I} m_i$ elements, where the first term bounds the number of different permutations from below (Theorem 7, Gutin *et al.*, 2007, assuming $N \geq k(k + 1)$) and the second term comes from the CO analysis.

Gutin Neighborhood. The assignment neighborhood of the TSP (Gutin *et al.*, 2007) first chooses a set $Z \subset \{1, 2, \ldots, N\}$ such that x_i and x_j for $i, j \in Z, i \neq j$ are non-adjacent in the given tour $x = (x_1, x_2, \ldots, x_N, x_1)$. The vertices $\{x_i : i \in Z\}$ can now be removed from x and be reinserted into the void positions one-to-one. For

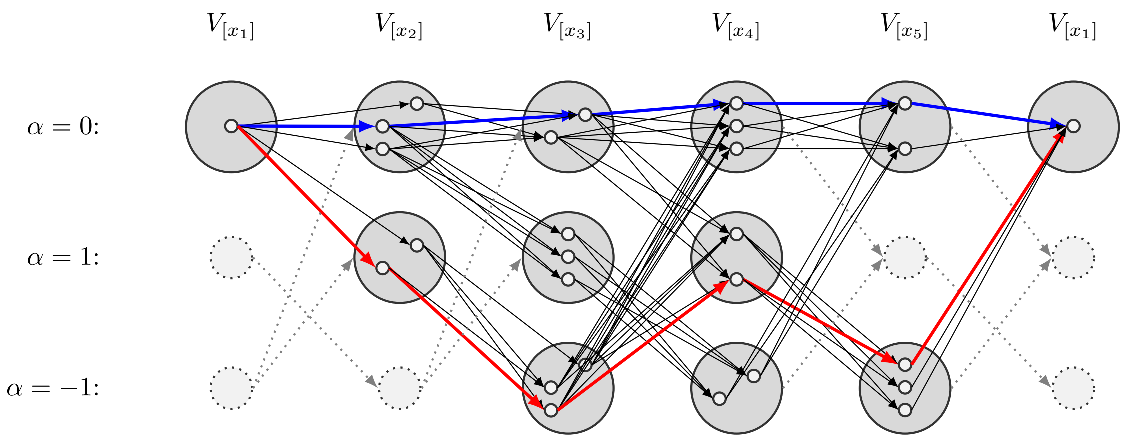

Figure 3: Auxiliary GTSP Network G_k for $k = 2$.

example, the subset $Z = \{2, 4, 7\}$ allows for the tour $x = (x_1, x_2, \dots, x_7, x_1)$ exactly six reinsertions, where one of the resulting neighbors is $x' = (x_1, x_4, x_3, x_7, x_5, x_6, x_2, x_1)$. In general, the neighborhood $\mathcal{N}_Z^{Gutin}(x)$ comprises $|Z|!$ elements and a best neighbor can be identified in $\mathcal{O}(|Z|^3)$ by solving an assignment problem.

Our adaptation for the GTSP works as follows: Let a G-tour $x = (x_1, x_2, \dots, x_N, x_1)$ be given. First, we determine the set Z with a randomized heuristic. Initially we set $Z = \varnothing$. Iterating over $i = 1, 2, \dots, N$, we first test whether x_{i-1} has been chosen. If $x_{i-1} \in Z$, we skip x_i (for $i = n$ we also test whether $x_1 \in Z$) and iterate. Otherwise, we toss a coin to decide whether x_i should be included into Z (with probability 0.5) or not, and iterate. Note that the expected length of a non-movable segment (between two consecutive elements of Z) is, therefore, $1 + 1/2 + 1/4 + 1/8 + \dots = 2$. Hence, Z comprises $N/3$ elements on average.

Second, when computing the insertion cost for moving vertex $x_i, i \in Z$, into the void position $j \in Z$, we allow that x_i is replaced by a best $x'_i \in V_{[x_i]}$. Accordingly, we compute the insertion cost as

$$a_{i,j} := \min_{x'_i \in V_{[x_i]}} \left(c_{x_{j-1}, x'_i} + c_{x'_i, x_{j+1}} \right).$$

with the convention $x_0 = x_N$ and $x_{N+1} = x_1$. In this way, the new GTSP neighborhood $\mathcal{N}_Z^{Gutin}(x)$ can simultaneously change the ordering of the clusters and the choice of cluster representatives.

The computational effort for determining a best neighbor in $\mathcal{N}_Z^{Gutin}(x)$ is bounded by $\mathcal{O}(nN + N^3)$, where the first term results from the insertion cost computation and the second from the exact solution of the assignment problem over $(a_{i,j})_{i,j \in Z}$.

String Relocation+. The string relocation neighborhood $\mathcal{N}_L^{SR}$ for $L \geq 1$ is a generalization of Relocation+ described in Section 3.2. Instead of moving a single vertex, a string of length up to L is removed from its current position and inserted into another position allowing different cluster representatives. Formally, a G-tour $x = (x_1, x_2, \dots, x_N, x_1)$ is given. The string $(x_i, x_{i+1}, \dots, x_{i+k})$ to remove is defined by $i \in I$ and an integer k, $1 \leq k \leq L$ (assuming that $x_{i+p} = x_{i+p-N}$ for $2N > i + p > N$). It can be replaced by a string $(x'_i, x'_{i+1}, \dots, x'_{i+k})$ with $[x'_i] = [x_i], [x'_{i+1}] = [x_{i+1}], \dots, [x'_{i+k}] = [x_{i+k}]$. The new string is then inserted between x_j and x_{j+1} for some $j \in I$ in the given G-tour. The neighbor solution is:

$$x' = (x_1, x_2, \dots, x_{i-1}, x_{i+k+1}, \dots, x_j, x'_i, x'_{i+1}, \dots, x'_{i+k}, x_{j+1}, \dots, x_N, x_1)$$

The neighborhood $\mathcal{N}_L^{SR}(x)$ comprises $\mathcal{O}(N^2 L m_{\max}^{L+1})$ elements, where $m_{\max}^{-} = \max_i m_i$.

To keep the computational effort manageable, we explore $\mathcal{N}_L^{SR}(x)$ with the following heuristic. Beforehand, only once per GTSP instance, we compute and store in a lookup table the following information for each vertex w and each $i \in I$ with $i \neq [w]$: The element $u = MDE(w, i) \in V_{[i]}$ is the vertex with minimum

Algorithm 1: Heuristic to explore $\mathcal{N}_L^{SR}(x)$.

```
Input: $x$, $L$
1  for $i \in I$ do
2      for $x'_i \in V_{[x_i]}$ do
3          Let $string \leftarrow (x'_i)$
4          for $k = 1, 2, \dots, L$ do
5              Let $x'_{i+k} \leftarrow MDE(x'_{i+k-1}, [x_{i+k}])$
6              Let $string \leftarrow (string, x'_{i+k})$
7              for $j \in I \setminus \{i-1, i, i+1, \dots, i+k, i+k+1\}$ do
8                  if $Improving$ then
9                      Let $i^* \leftarrow i$, $j^* \leftarrow j$, $string^* \leftarrow string$
Output: $i^*$, $j^*$, $string^*$
```

distance to w (MDE, minimum distance element), i.e., $c_{wu} = \min_{v \in V_{[i]}} c_{wv}$ (ties are broken arbitrarily). The precomputation of the MDE values takes $\mathcal{O}(n^2)$ time.

The actual exploration heuristic for $\mathcal{N}_L^{SR}(x)$ is presented with the pseudo-code in Algorithm 1. Line 1, we determine the first vertex x_i and the starting position i of the string that is relocated. With the loop in Line 2, we consider all possible replacements of x_i by an x'_i of the same cluster. The loop in Line 4 determines the length k of the string. The following replacements of x_{i+k} by x'_{i+k} are then looked up with the help of the auxiliary function MDE, i.e., the following replacements are heuristically chosen as close as possible to the preceding and last chosen vertex x'_{i+k-1} (Line 5). Thanks to the lookup table, Line 5 takes only constant time $\mathcal{O}(1)$. The result is that the string $(x_i, x_{i+1}, \dots, x_{i+k})$ of length k is possibly replaced by the string $(x'_i, x'_{i+1}, \dots, x'_{i+k})$. The insertion position j is determined in the loop in Line 7. Finally, the resulting move is completely determined now so that the cost of the move and possible improvement can be checked (Line 8).

The overall time complexity of the exploration heuristic is bounded by $\mathcal{O}(NLn)$.

4. Basic Iterated Local Search

In this section, we propose our simple metaheuristic algorithm based on ILS (Lourenço *et al.*, 2003) using a random VND (Subramanian *et al.*, 2010) as a local-search component. More precisely, given a current feasible G-tour x, the algorithm alternates between a local search starting from x using multiple neighborhoods and ending at a joint local minimum x^*, and a perturbation step. So every time the local search is trapped in a local minimum x^*, ILS perturbs it and starts a new local search based on this modified solution x'. As a consequence, ILS does a randomized walk in the space of all joint local minima.

Moreover, to better guide the search to more promising solutions, an acceptance criterion can be used. It decides whether the local minimum or locally optimal solution x^* is accepted as the new current solution or the previous current solution will be randomly perturbed again. Hence, the acceptance criterion controls the balance between intensification and diversification.

To achieve high performance, the four main functions *Initial Solution Construction*, *Local Search*, *Perturbation*, and *Acceptance Criterion* have to be tailored to the needs of the GTSP. In the following, we discuss how we implemented these main functions. Furthermore, Algorithm 2 shows how they are finally combined into the basic ILS metaheuristic that we apply and evaluate later in this section.

Initial Solution Construction. Several heuristics can be used to obtain a feasible starting solution for the GTSP. Most of them are derived by simple tour construction heuristics for the classical TSP. As an example, Fischetti *et al.* (1997) adapted the well-known TSP insertion heuristics to obtain a feasible G-tour. Based on

Algorithm 2: Iterated Local Search (ILS) with Record-to-Record Travel Acceptance Criterion

```
   Input: acceptance parameter $\epsilon > 0$, cooling rate $0 < h < 1$)
1  $x_{init} \leftarrow$ Initial Solution Construction()
2  $x \leftarrow x_{ILS} \leftarrow$ Local Search($x_{init}$)
3  repeat
4      $x' \leftarrow$ Perturbation($x$)
5      $x^* \leftarrow$ Local Search($x'$)
       /* Test Acceptance Criterion */
6      if $c(x^*) < c(x)$   or   $c(x^*) \leq (1+\epsilon) \cdot c(x_{ILS})$ then
7          $x \leftarrow x^*$
8      if $c(x^*) < c(x_{ILS})$ then
9          $x_{ILS} \leftarrow x^*$
10     if cooling update condition fulfilled then
11         $\epsilon \leftarrow \epsilon \cdot h$
12 until time limit reached
   Output: $x_{ILS}$
```

these, Smith and Imeson (2017) provide a unified insertion procedure that contains three insertion procedures as special cases.

We create an initial solution with a random insertion procedure similar to the insertion heuristics presented in (Fischetti *et al.*, 1997). Initially, we randomly pick a vertex $x_1 \in V$ and add it as the start and end vertex of the subtour (x_1, x_1). In each iteration, the subtour $(x_1, x_2, \ldots, x_k, x_{k+1} = x_1)$ (with $k < N$) is then enlarged by randomly choosing one of the non-visited clusters $V_{[i]}$ and inserting the vertex $y \in V_{[i]}$ into the subtour that minimizes the insertion cost, i.e., $y = \arg\min_{x \in V_{[i]}, 1 \leq j \leq k} \{c_{x_j x} + c_{x x_{j+1}} - c_{x_j x_{j+1}}\}$. The procedure stops when the G-tour visits all N clusters.

Local Search. We apply a random VND with all neighborhoods described in Section 3. The idea is that the resulting VND deeply explores the solution space with the combination of pure TSP neighborhoods (Section 3.1), TSP-inspired GTSP neighborhoods (Section 3.2), and exponentially-sized GTSP neighborhoods (Section 3.3). We choose a first improvement pivoting strategy for all neighborhoods except those explored implicitly with an optimization algorithm (double-bridge, CO, BS, Gutin, and SR neighborhood). Moreover, pre-tests have shown that the BS neighborhood should be used with $k \leq 3$, because the state space for larger k grows considerably, making labeling prohibitively slow (compared to the other exploration algorithms). Note that we consider BS neighborhoods with different k values as different neighborhoods. Finally, the maximum string length for the SR neighborhood is set to $L = 4$.

The random VND chooses one of the available neighborhoods at random. The neighborhood exploration is however only started if an improvement is possible. Hence, if the same neighborhood is chosen two times in a row and the first exploration has confirmed local optimality, the second exploration is omitted. Moreover, recall that the CO neighborhood is identical to $\mathcal{N}_1^{BS}$ and that $\mathcal{N}_1^{BS} \subset \mathcal{N}_2^{BS} \subset \mathcal{N}_3^{BS}$ etc. Therefore, if $\mathcal{N}_3^{BS}$ fails to find an improvement, CO and $\mathcal{N}_2^{BS}$ are omitted. Likewise, if $\mathcal{N}_2^{BS}$ fails, CO is omitted.

Perturbation. When the VND terminates, a local minimum w.r.t. all neighborhoods has been found. To escape from such a local minimum and to lead the search towards a region of the solution space not yet explored, ILS applies a perturbation step. The design of this perturbation step is delicate: If the perturbation step is too strong, ILS behaves like a random multi-start algorithm with a relatively high computational burden for the VND. In this case, an average iteration (of the main loop of Algorithm 2) takes more time. On the other hand, if the perturbation is too weak, the VND tends to fall back into the known local optimum just found in the iteration before. The diversification of the search is rather limited then.

For the sake of simplicity, we use the random double-bridge move for the perturbation of the current G-tour x. It is known from the TSP (Johnson and McGeoch, 1997) that the double-bridge move gives an effective perturbation. In comparison to the double-bridge move used within the local search (Section 3.1), the edges to be removed are chosen randomly. Moreover, if the VND falls back into same the local optimum three times in a row (as a necessary condition we test whether objective values are identical), we do not perturb with a random double-bridge move. Instead the current solution is set to a fresh starting solution, computed with the above-described GTSP construction heuristic.

Acceptance Criterion. The acceptance criterion controls the balance between intensification and diversification of the search. In case no acceptance criterion is used, the ILS performs a randomized walk in the space of all local optima, i.e., a strong diversification is achieved. On the opposite, if only better solutions are accepted, intensification is very strong.

We balance intensification and diversification with the deterministic record-to-record travel (Dueck, 1993) acceptance criterion as follows: Every solution x^* improving the current solution x is always accepted. Moreover, non-improving solutions x^* (those with $c(x^*) > c(x)$) are accepted if the deviation from the cost of the best observed solution (= record, x_{ILS}) so far is smaller than a predefined threshold. To this end, we test $c(x^*) \leq (1 + \epsilon) \cdot c(x_{ILS})$ for a (small) value $\epsilon > 0$ (cf. Line 6 in Algorithm 2). Initially, we set ϵ to 0.03 so that a deviation of 3 % from the record is allowed. With a straightforward geometric cooling schedule, commonly used in simulated annealing, we systematically lower ϵ in the course of the ILS. The cooling update takes place every N iterations of the main loop. The update lowers ϵ by the factor $h = 0.8$ (Line 10). That means, e.g., that ϵ is at approximately a tenth of its initial value after $10N$ iterations.

4.1. GTSP Instances

We evaluate the performance of the basic ILS on commonly used symmetric GTSP problem libraries that have also been used to compare

- the memetic algorithm *GK* of Gutin and Karapetyan (2010),
- the Lin-Kernighan-Helsgaun *GLKH* algorithm of Helsgaun (2015), and
- the large neighborhood search *GLNS* of Smith and Imeson (2017).

Note that *GK*, *GLK*, and *GLNS* are the best-performing state-of-the-art GTSP solvers published in the literature (see Section 2). Smith and Imeson (2017) made the three GTSP algorithms well comparable by running all of them for the same amount of computational time (we provide details below). They were tested on four libraries:

- The `GTSP_LIB` was introduced by Fischetti *et al.* (1997) and later extended by Silberholz and Golden (2007). It consists of 88 symmetric and asymmetric instances with up to 1084 vertices and 217 clusters taken from the `TSP_LIB` (Reinelt, 1991). The vertex clustering simulates geographical regions. The 45 largest instances of the library have been used in the `GK`, `GLK`, and `GLNS` comparison presented by Smith and Imeson (2017). We omit the five asymmetric instances.
- The `BAF_LIB` was introduced by Bontoux (2008) and was also derived from the `TSP_LIB`. The pseudo-random clustering scheme implies that there are *no* geographical regions. The library comprises 56 symmetric instances with up to 1084 vertices and 217 clusters. The standard comparison uses the 45 largest instances.
- The `MOM_LIB` was introduced by Mestria *et al.* (2013) and contains 249 symmetric instances with six different clustering schemes. The largest instances have up to 3000 vertices and up to 200 clusters. All instances are adapted either from `TSP_LIB` or the Concorde project (Applegate *et al.*, 1999). Here, the 45 largest instances are used in the *GK*, *GLK*, and *GLNS* comparison of Smith and Imeson (2017).
- The `LARGE_LIB` was introduced by Helsgaun (2015) and contains 44 very large symmetric instances ranging from 1,000 to 85,900 vertices. Instances originally stem from the `TSP_LIB`, the 8th DIMACS Implementation Challenge (Johnson *et al.*, 2000), and the National TSP benchmark library (Applegate *et al.*, 2015). The clusters were also generated with the clustering scheme of Fischetti *et al.* (1997). Only the 27 smallest instances have been used to benchmark the *GK*, *GLK*, and *GLNS* algorithms.

4.2. Computational Results of the Basic ILS

The basic ILS was coded in C++ and compiled with MS Visual Studio 2015 in release mode. All computations were performed on a standard PC with MS Windows 10 running on an Intel® Core^{TM} i7-

5930K CPU clocked at 3.5 GHz and with 64 GB RAM.

We designed the experiments in the same way as Smith and Imeson (2017) did: The computation time limit was 300 seconds for all instances of the `GTSP-LIB`, `MOM-LIB`, and `BAF-LIB`, and 1200 seconds for all instances of the `LARGE-LIB`. Table 2 shows the computational results, where columns have the following meaning:

- `instance` is the name of the GTSP instance. The name's prefix is the number N of clusters and the suffix is number n of vertices. For all instances marked with `*`, a new best solution was found.
- `best known` shows the cost of the best-known solution x_{BKS} (BKS), i.e., $c(x_{BKS})$, known from the literature for the instance. This information is taken from Smith and Imeson (2017) (available at `https://ece.uwaterloo.ca/~sl2smith/GLNS/`) and Helsgaun (2015) (available at `http://akira.ruc.dk/~keld/research/GLKH/`).
- `#best` is the number of runs, out of ten, in which a BKS was obtained or a better solution was found.
- $\Delta(\%)$ shows the average percentage error between the cost $c(x_{BKS})$ of a BKS and the cost $c(x_{ILS})$ obtained by our ILS over 10 runs. The percentage error e is calculated as $e = 100 \cdot (c(x_{ILS}) - c(x_{BKS}))/c(x_{BKS})$. If a new solution could be found, e is negative. Accordingly, this negative e is also included in the calculation of the average percentage error $\Delta(\%)$.
  For example: A new best solution with cost 1600 was computed for the instance 45tsp225 of the `GTSP_LIB` in 10 of 10 runs. The previous BKS was $c(x_{BKS}) = 1612$. Hence, $e = -0.74\,\%$ for all runs so that also $\Delta = -0.74\,\%$.

Table 2: Results of the Basic ILS on 157 symmetric GTSP Instances.

GTSP_LIB				MOM_LIB				BAF_LIB				LARGE_LIB			
instance	best	#best	Δ(%)	instance	best	#best	Δ(%)	instance	best	#best	Δ(%)	instance	best	#best	Δ(%)
31pr152	51,576	10	—	50i2000-603	4,325	10	—	baf20kroD100	5,266	10	—	10C1k.0	2,522,585	10	—
32u159	22,664	10	—	50i2500-707	3,961	10	—	baf20kroE100	5,449	10	—	31C3k.0	3,553,142	10	—
35si175	5,564	10	—	50i3000-802	4,070	10	—	baf20rat99	230	10	—	49usa1097	10,337	10	—
36brg180	442	10	—	50kroA100	15,944	10	—	baf20rd100	1,747	10	—	100C10k.0	6,158,999	0	1.16
39rat195	854	10	—	50kroB100	15,842	10	—	baf21eil101	105	10	—	200C1k.0	6,375,154	10	—
40d198	10,557	10	—	50lin105	11,294	10	—	baf21lin105	2,758	10	—	200E1k.0	9,662,857	0	0.38
40kroa200	13,406	10	—	50lin318	18,163	10	—	baf22pr107	6,849	10	—	235pcb1173	23,399	1	0.80
40krob200	13,111	10	—	50nrw1379	7,449	10	—	baf24gr120	1,377	10	—	259d1291	28,400	3	0.13
41gr202	23,301	10	—	50pcb1173	9,385	10	—	baf25pr124	10,745	10	—	261rl1304	150,468	0	0.23
45ts225	68,340	10	—	50pcb442	14,430	10	—	baf26bier127	11,740	10	—	265rl1323	154,023	0	0.36
45tsp225 *	1,612	10	−0.74	50pr1002	54,583	10	—	baf28pr136	17,824	10	—	276nrw1379	20,050	0	0.41
46gr229	71,972	10	—	50pr439	45,253	10	—	baf29pr144	14,070	10	—	280fl1400	15,316	10	—
46pr226	64,007	10	—	50rat783	1,626	10	—	baf30kroA150	7,005	10	—	287u1432	54,469	0	0.31
53gil262	1,013	10	—	50rat99	814	10	—	baf30kroB150	5,855	10	—	316fl1577	14,182	10	—
53pr264	29,549	10	—	50vm1084	54,156	10	—	baf31pr152	13,002	10	—	331d1655	29,443	0	0.52
56a280	1,079	10	—	72vm1084-8x9	64,647	10	—	baf32u159	7,301	10	—	350vm1748	185,459	0	0.55
60pr299	22,615	10	—	75lin105	13,134	10	—	baf39rat195	477	10	—	364u1817	25,530	0	0.59
64lin318	20,765	10	—	81vm1084-9x9	69,659	10	—	baf40d198	1,466	10	—	378rl1889	184,034	0	0.63
80rd400	6,361	10	—	100i1000-410	5,481	10	—	baf40kroA200	7,113	10	—	421d2103	40,049	0	1.59
84u417	9,651	10	—	100i1500-506	5,088	8	0.06	baf40kroB200	7,126	10	—	431u2152	27,614	0	1.63
87gr431	101,946	10	—	100i2000-604	5,316	6	0.17	baf41gr202	3,531	10	—	464u2319	65,758	0	3.37
88pr439	60,099	10	—	100i2500-708	5,297	10	—	baf45ts225	25,697	10	—	479pr2392	169,874	0	3.27
89pcb442	21,657	10	—	100i3000-803	5,458	2	0.04	baf46pr226	13,555	10	—	608pcb3038	52,416	0	5.39
99d493	20,023	10	—	100nrw1379	10,566	10	—	baf53gil262	571	3	0.40	633C3k.0	10,255,031	0	2.49
107ali535	128,639	10	—	100pcb1173	13,901	4	0.20	baf53pr264	7,716	10	—	633E3k.0	16,197,552	0	5.80
107att532	13,464	10	—	100pr1002	74,269	9	0.00	baf60pr299	10,047	10	—	759fl3795	18,662	0	1.19
107si535	13,502	10	—	100prb1173-10x10	12,644	10	—	baf64lin318	7,489	10	—	893fnl4461	63,163	0	6.84
113pa561	1,038	10	—	100rat783-10x10	2,216	10	—	baf80rd400	3,254	2	1.38				
115rat575	2,388	10	—	100rat783	2,496	10	—	baf84fl417	2,226	10	—				
115u574	16,689	10	—	100vm1084	78,440	10	—	baf87gr431	10,569	10	—				
131p654	27,428	10	—	144pcb1173-12x12	16,412	1	0.28	baf88pr439	13,882	10	—				
132d657	22,498	8	0.02	144rat783-12x12	2,813	4	0.03	baf89pcb442	8,749	10	—				
134gr666	163,028	7	0.17	150i1000-411	6,296	9	0.05	baf99d493	3,081	10	—				
145u724	17,272	9	0.02	150i1500-507	6,085	9	0.00	baf107att532	3,880	8	0.06				
157rat783	3,262	1	0.14	150i2000-605	5,940	9	0.00	baf107si535	8,912	7	0.47				
200dsj1000	9,187,884	4	0.10	150i2500-709	6,158	5	0.17	baf113pa561	431	0	1.44				
201pr1002	114,311	2	0.07	150i3000-804	6,569	0	0.44	baf115rat575	1,330	9	0.15				
207si1032	22,306	0	0.07	150nrw1379	13,370	3	0.21	baf131p654	5,824	4	0.03				
212u1060	106,007	1	0.31	150pcb1173	17,082	2	0.50	baf132d657	8,132	10	—				
217vm1084	130,704	6	0.11	150pr1002	92,969	7	0.03	baf145u724	7,354	0	0.63				
				150rat783	3,131	2	0.34	baf157rat783	1,700	8	0.56				
				150vm1084	95,922	10	—	baf201pr1002	48,400	0	2.40				
				200i2000-606	7,274	0	0.39	baf207si1032	18,836	9	0.00				
				200i2500-710	7,191	3	0.28	baf212u1060	38,639	5	0.24				
				200i3000-805	6,909	0	0.65	baf217vm1084	44,681	10	—				
Average		**8.70**	**0.01**			**7.62**	**0.09**			**8.56**	**0.17**			**2.37**	**1.39**

We give some side notes: The basic ILS finds a BKS at least once for 131 of the 157 instances, while for 95 instances it was determined in 10 of 10 runs. In addition, for the instance 45tsp225 of the `GTSP_LIB`, a new best solution was found.

For `GTSP_LIB`, a BKS (or a better solution) is found in 8.70 of 10 runs on average. For instances with up to 131 clusters, a BKS is found or undercut in 10 of 10 runs, while for instances with more than 131 clusters, the number of runs in which a BKS is obtained varies between 0 and 9 with an average of 4.

For `MOM_LIB`, the basic ILS finds a BKS in at least one run for 42 of the 45 instances. For 26 instances, all ten runs find it. For the instances for which a BKS was not obtained in all ten runs, the average number of achieved BKS is 4, with an average percentage error of 0.2%.

For the `BAF_LIB`, the basic ILS finds a BKS in all ten runs for 33 instances. For the other instances, the average number of BKSs found is 5 with an average percentage error of 0.65 %. On the downside, for three instances, a BKS was never obtained in all 10 runs. For two of them, the average percentage error is rather high with 1.44 % and 2.40 %.

For the `LARGE_LIB`, we summarize the results as follows: From 27 instances, only 6 instances can be solved with the BKS in 10 of 10 runs, 2 instances can be solved at least one time, whereas for 19 instances a BKS is never obtained. Thus, the average percentage error is relatively large with 1.79 %. In particular for instances with more than 400 clusters, the average percentage error varies between 1.19 % and 6.84 %.

Finally, we compare the basic ILS with the three algorithms *GK*, *GLKH*, and *GLNS*. Table 3 presents average values of *#best* and $\Delta(\%)$ on the four groups of instances and for the four algorithms. Note that only the symmetric instances are taken into account to be comparable with the analysis of Smith and Imeson (2017). In spite of its simplicity, the basic ILS produces reasonable results on the first three libraries, while results for the `LARGE_LIB` are clearly behind.

Table 3: Comparison of the basic ILS with Algorithms from the Literature

	`GTSP_LIB`		`MOM_LIB`		`BAF_LIB`		`LARGE_LIB`	
Algorithm	#best	$\Delta(\%)$	#best	$\Delta(\%)$	#best	$\Delta(\%)$	#best	$\Delta(\%)$
GK	9.10	**0.01**	8.44	0.03	8.11	0.29	2.77	0.76
GLKH	**9.20**	**0.01**	5.40	0.82	5.04	6.51	3.04	0.52
GLNS	8.73	**0.01**	**9.18**	**0.02**	**8.91**	**0.07**	**3.31**	**0.50**
Basic ILS	8.70	**0.01**	7.62	0.09	8.56	0.17	2.37	1.39

We can provide some reasons why the basic ILS is not convincing on the `LARGE_LIB`: In particular for instances with a large number N of clusters, the ILS performs only a relative small number of iterations, because some neighborhoods are very time-consuming. It is clear that, in these cases, the more time-consuming neighborhoods should either be completely omitted or should be explored less often, e.g., only when elite solutions are found.

5. Refined Iteratated Local Search

Section 4 has shown that the basic ILS already achieves reasonable results. It is however not yet competitive with the currently best algorithm *GLNS* and does not consistently outperform *GK* and *GLKH*. Our overall goal for the ILS still remains to design a simple but powerful algorithm and well-reproducible results. We implement the three following refinements:

Reset. We observed that, in particular for some more difficult instances, the best found solution x_{ILS} is identified only once in a run (if ever). It seems that intensifying the search in the solution space around x_{ILS} could help to identify other very good solutions, hopefully better ones.

Hence, if no improvement takes place for a pre-defined number of iterations, the *reset component* resets the current solution x to the best found solution x_{ILS}. Pre-tests have shown a reset after every 50 iterations is a good compromise balancing intensification and diversification.

VND with Neighborhood Prioritization. The classical VND, as proposed by Hansen and Mladenović (2005), orders the neighborhoods according to their size and expected computational effort. The first neighborhood is then the one with the smallest computational effort. The second neighborhood is only explored when a local optimum in the first is reached. Moreover, after an improving move in the second neighborhood, one returns to the exploration of the first neighborhood. More than two neighborhoods are "prioritized" in the same fashion.

We want to find out whether a prioritization is also beneficial for the local-search component of our ILS. This includes the possibility to completely disregard neighborhoods and to choose a pivoting strategy (first improvement or best improvement) per neighborhood. Moreover, for the BS neighborhood, a value for the parameter k has to be set. In contrast, the maximum string length in the SR neighborhood is fixed to $L = 4$, because larger values lead to unacceptably long computation times.

To this end, we consider the set of all parameterized neighborhoods

$$\mathcal{N}^{all} = \{\mathcal{N}^{2Opt}_{best}, \mathcal{N}^{2Opt}_{first}, \mathcal{N}^{3Opt}_{best}, \mathcal{N}^{3Opt}_{first}, \mathcal{N}^{dbl-brdg}, \dots, \mathcal{N}^{BS}_{2}, \mathcal{N}^{BS}_{3}, \mathcal{N}^{BS}_{4}, \mathcal{N}^{BS}_{5}, \dots, \mathcal{N}^{SR}_{4}, \dots, \mathcal{N}^{Gutin}\}$$

resulting from the description given in Section 3.

Of course, we do not perform a full factorial parameter study over $\mathcal{N}^{all}$, simply because the number of possible VND variants and resulting ILS setups is too large. Furthermore, there is the danger of overfitting the possible setups to the set of 157 GTSP instances that is considered.

Our pragmatic approach to design refined ILS setups with different prioritized neighborhoods in the VND can be summarized as follows:

- Initially, we randomly select one of the four GTSP libraries and a subset $\mathcal{I}$ of $10 = |\mathcal{I}|$ instances from it.
- For this subset $\mathcal{I}$, we determine the subset $\mathcal{N}^{VND} \subset \mathcal{N}^{all}$ of useful neighborhoods by adding, one by one, a not yet selected parameterized neighborhood $\mathcal{N} \in \mathcal{N}^{all} \setminus \mathcal{N}^{VND}$ to $\mathcal{N}^{VND}$. This is done in the following way:
  - The neighborhood selection process is initialized with $\mathcal{N}^{VND}_0 = \varnothing$.
  - In the pth iteration ($p = 1, 2, \dots$), i.e., when $|\mathcal{N}^{VND}_{p-1}| = p - 1$, the next parameterized neighborhood $\mathcal{N}^p \in \mathcal{N}^{all} \setminus \mathcal{N}^{VND}_{p-1}$ is the one producing the best improvement over the ILS using $\mathcal{N}^{VND}_{p-1}$, when added to $\mathcal{N}^{VND}_{p-1}$, i.e., for the tentative subset $\mathcal{N}^{VND}_p = \{\mathcal{N}^p\} \cup \mathcal{N}^{VND}_{p-1}$.
  - We measure the improvement (relative to the subset $\mathcal{I}$) as the value

$$\Delta(\mathcal{I}, \mathcal{N}^{VND}_p) = \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \left(c(x^I_{ILS}) - c(x^I_{BKS}) \right) / c(x^I_{BKS}),$$

    where $c(x^I_{ILS})$ is the cost of a best solution of instance I obtained with the refined ILS using $\mathcal{N}^{VND}_p$, and $c(x^I_{BKS})$ the cost of a BKS for instance I known from the literature.
  - We perform a single run per instance $I \in \mathcal{I}$ limited to 120 seconds of computation time.
  - If there is no improvement, i.e., $\Delta(I, \mathcal{N}^{VND}_p) \geq \Delta(I, \mathcal{N}^{VND}_{p-1})$ for all $\mathcal{N}^p \in \mathcal{N}^{all} \setminus \mathcal{N}^{VND}_{p-1}$, the process of adding neighborhoods stops.
    The final set of neighborhoods is chosen as $\mathcal{N}^{VND} = \mathcal{N}^{VND}_{p-1}$.

By repeating the random selection of instances $\mathcal{I}$, we have obtained several versions of a nested VND. Two rather well-performing but different ones give rise to two refined versions of ILS in which the random VND is replaced by nested VND I or VND II summarized in Table 4.

Balas-Simonetti for High-Quality Solutions. When a high-quality solution is found, we try to further improve it with the Balas-Simonetti neighborhood with a high k-value (Section 3.3). We define high-quality solutions as G-tours with a cost falling into the lower 1 %-fractile of all local optimal solutions x^* found so far (see Line 5 of Algorithm 2). Therefore, the ILS regularly computes and updates a bound b on the cost. More precisely, we set the first bound b after 200 iterations of the main loop and subsequently update b after every 50 iterations, but only if the newly computed bound is lower than the old bound. With this latter condition, the bound b never increases.

Table 4: Selection, Priorities, and Pivoting Strategies of Neighborhoods.

	VND I		VND II	
Neighborhood	priority	pivoting	priority	pivoting
2-Opt	—	—	6	first
3-Opt	4	best	2	best
Double Bridge	3	best	4	best
Relocation+	1	best	—	—
Swap+	—	—	—	—
CO	—	—	—	—
BS $k = 3$	—	—	—	—
BS $k = 4$	2	best	—	—
BS $k = 5$	—	—	3	best
String Relocation+	—	—	5	best
Gutin Neighborhood	5	best	1	best

Note: The maximum string length for String Relocation+ is $L = 4$ for VND II.

If a solution x^* is a high-quality solution, the BS neighborhood is explored. We have experimented with different values of k. Obviously, larger values of k offer a higher potential for an improvement but also come at a large computational effort. As explained for CO, we rotated the G-tour x so that the first cluster has minimum cardinality. Additionally, we run the corresponding shortest-path problem only for the vertex x_1 that is currently selected in $x = (x_1, x_2, \ldots, x_N, x_1)$. With these accelerations, pre-tests have shown that $k = 8$ is still possible and offers a good trade-off between solution quality and computational time.

Technically, the Balas-Simonetti neighborhood with $k = 8$ is added to the VND with the lowest possible priority, i.e., highest value *priority* (cf. Table 4). Every time the VND calls this new BS neighborhood, it checks if the cost of the current solution x belongs to the best 1 % fractile, i.e., $c(x) \leq b$.

Table 5: Performance of the three ILS Setups on four GTSP Libraries.

	`GTSP_LIB`		`MOM_LIB`		`BAF_LIB`		`LARGE_LIB`	
ILS with	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)
random VND	8.70	0.01	**7.62**	**0.09**	**8.56**	**0.17**	2.37	1.39
VND I	**9.15**	**−0.004**	7.16	0.14	8.16	0.44	2.70	1.15
VND II	8.73	0.01	7.27	0.10	7.29	1.05	**2.81**	**1.06**

Table 5 summarizes the comparison of the basic ILS and the two new refined ILS versions using VND I and VND II. Note that the first line repeats the results presented before for the basic ILS, which uses the random VND and no reset to x_{ILS}. The refined ILS with VND I performs very well on the `GTSP_LIB` both regarding the average number of BKSs found (*#best*) and the average deviation to the BKS (Δ(%)). Recall that a negative deviation is possible: it results from the new BKS that was computed for one instance. The refined ILS with VND II outperforms the one with VND I and the basic ILS on the `LARGE_LIB`. For `MOM_LIB` and `BAF_LIB`, the basis ILS is still the best algorithm.

5.1. Instance-Based Selection of an ILS Setup

With three alternative ILS setups at hand (basic ILS, refined ILS with VND I or VND II), the question is now whether one can estimate the performance beforehand. Looking into instance-by-instance results for the three refined ILS setups, the refined ILS with VND I outperforms the other version on instances that are *not* geometrically clustered. Accordingly, we define for a GTSP instance, the *average relative inner-cluster* *distance* as

$$\alpha(I) = \frac{\bar{c}_{in}}{\bar{c}} \qquad \text{with} \quad \bar{c}_{in} = \left(\sum_{i \in I} \sum_{x < x' \in V_i} c_{x,x'} \right) \Big/ \left(\sum_{i \in I} \binom{|V_i|}{2} \right) \quad \text{and} \quad \bar{c} = \left(\sum_{x < x' \in V} c_{x,x'} \right) \Big/ \binom{|V|}{2},$$

where $\bar{c}_{in}$ is the average distance between vertices of the same cluster and $\bar{c}$ the average distance between two arbitrary vertices. Moreover, the refined ILS with VND II works well for large instances, where we define large instance as one with $N > 250$.

Having defined this, the following simple rules assign an instance I to one of the three refined ILS versions:

- If $\alpha(I) < 0.5$, use the refined ILS with VND I;
- If $\alpha(I) \geq 0.5$ and $N > 250$, use the refined ILS with VND II;
- In all other cases, use the basic ILS (with random VND, without reset).

The resulting metaheuristic with the instance-based selection of the ILS setup is denoted as *the refined ILS* from now on.

5.2. Computational Results of the Refined ILS

Table 6 shows the results obtained with the refined ILS on the 157 symmetric GTSP instances. The meaning of the columns is the same as in Table 2.

Table 6: Results of the Refined ILS on the 157 symmetric GTSP Instances

GTSP_LIB				MOM_LIB				BAF_LIB				LARGE_LIB			
instance	best	#best	Δ(%)	instance	best	#best	Δ(%)	instance	best	#best	Δ(%)	instance	best	#best	Δ(%)
31pr152	51,576	10	—	50i2000-603	4,325	10	—	baf20kroD100	5,266	10	—	10C1k.0	2,522,585	10	—
32u159	22,664	10	—	50i2500-707	3,961	10	—	baf20kroE100	5,449	10	—	31C3k.0	3,553,142	10	—
35si175	5,564	10	—	50i3000-802	4,070	10	—	baf20rat99	230	10	—	49usa1097	10,337	10	—
36brg180	442	10	—	50kroA100	15,944	10	—	baf20rd100	1,747	10	—	100C10k.0	6,158,999	0	1.16
39rat195	854	10	—	50kroB100	15,842	10	—	baf21eil101	105	10	—	200C1k.0	6,375,154	10	—
40d198	10,557	10	—	50lin105	11,294	10	—	baf21lin105	2,758	10	—	200E1k.0	9,662,857	2	0.21
40kroa200	13,406	10	—	50lin318	18,163	10	—	baf22pr107	6,849	10	—	235pcb1173	23,399	0	0.74
40krob200	13,111	10	—	50nrw1379	7,449	10	—	baf24gr120	1,377	10	—	259d1291	28,400	10	—
41gr202	23,301	10	—	50pcb1173	9,385	10	—	baf25pr124	10,745	10	—	261rl1304	150,468	3	0.17
45ts225	68,340	10	—	50pcb442	14,430	10	—	baf26bier127	11,740	10	—	265rl1323	154,023	0	0.18
45tsp225 *	1,612	10	−0.74	50pr1002	54,583	10	—	baf28pr136	17,824	10	—	276nrw1379	20,050	0	0.45
46gr229	71,972	10	—	50pr439	45,253	10	—	baf29pr144	14,070	10	—	280fl1400	15,316	10	—
46pr226	64,007	10	—	50rat783	1,626	10	—	baf30kroA150	7,005	10	—	287u1432 *	54,437	1	0.24
53gil262	1,013	10	—	50rat99	814	10	—	baf30kroB150	5,855	10	—	316fl1577	14,182	10	—
53pr264	29,549	10	—	50vm1084	54,156	10	—	baf31pr152	13,002	10	—	331d1655	29,443	0	0.28
56a280	1,079	10	—	72vm1084-8x9	64,647	10	—	baf32u159	7,301	10	—	350vm1748	185,459	0	0.22
60pr299	22,615	10	—	75lin105	13,134	10	—	baf39rat195	477	10	—	364u1817	25,530	1	0.28
64lin318	20,765	10	—	81vm1084-9x9	69,659	10	—	baf40d198	1,466	10	—	378rl1889	184,034	0	0.24
80rd400	6,361	10	—	100i1000-410	5,481	10	—	baf40kroA200	7,113	10	—	421d2103	40,049	0	0.74
84u417	9,651	10	—	100i1500-506	5,088	8	0.06	baf40kroB200	7,126	10	—	431u2152	27,614	0	1.09
87gr431	101,946	10	—	100i2000-604	5,316	6	0.17	baf41gr202	3,531	10	—	464u2319	65,758	0	1.67
88pr439	60,099	10	—	100i2500-708	5,297	10	—	baf45ts225	25,697	10	—	479pr2392	169,874	0	1.34
89pcb442	21,657	10	—	100i3000-803	5,458	2	0.04	baf46pr226	13,555	10	—	608pcb3038	52,416	0	3.89
99d493	20,023	10	—	100nrw1379	10,566	10	—	baf53gil262	571	3	0.40	633C3k.0	10,255,031	0	1.80
107ali535	128,639	10	—	100pcb1173	13,901	4	0.20	baf53pr264	7,716	10	—	633E3k.0	16,197,552	0	4.39
107att532	13,464	10	—	100pr1002	74,269	9	0.00	baf60pr299	10,047	10	—	759fl3795	18,662	0	0.53
107si535	13,502	10	—	100prb1173-10x10	12,644	10	—	baf64lin318	7,489	10	—	893fnl4461	63,163	0	6.77
113pa561	1,038	10	—	100rat783-10x10	2,216	10	—	baf80rd400	3,254	2	1.38				
115rat575	2,388	10	—	100rat783	2,496	10	—	baf84fl417	2,226	10	—				
115u574	16,689	8	0.04	100vm1084	78,440	10	—	baf87gr431	10,569	10	—				
131p654	27,428	10	—	144pcb1173-12x12	16,412	1	0.28	baf88pr439	13,882	10	—				
132d657	22,498	10	—	144rat783-12x12	2,813	4	0.03	baf89pcb442	8,749	10	—				
134gr666	163,028	9	0.05	150i1000-411	6,296	9	0.05	baf99d493	3,081	10	—				
145u724	17,272	8	0.04	150i1500-507	6,085	9	0.00	baf107att532	3,880	8	0.06				
157rat783	3,262	4	0.08	150i2000-605	5,940	9	0.00	baf107si535	8,912	7	0.47				
200dsj1000	9,187,884	8	0.02	150i2500-709	6,158	5	0.17	baf113pa561	431	0	1.44				
201pr1002	114,311	8	0.01	150i3000-804	6,551	0	0.44	baf115rat575	1,330	9	0.15				
207si1032	22,306	0	0.07	150nrw1379	13,370	3	0.21	baf131p654	5,824	4	0.03				
212u1060	106,007	2	0.25	150pcb1173	17,082	2	0.50	baf132d657	8,132	10	—				
217vm1084	130,704	9	0.03	150pr1002	92,969	7	0.03	baf145u724	7,354	0	0.63				
				150rat783	3,131	2	0.34	baf157rat783	1,700	8	0.56				
				150vm1084	95,922	10	—	baf201pr1002	48,400	0	2.40				
				200i2000-606	7,272	0	0.39	baf207si1032	18,836	9	0.00				
				200i2500-710	7,191	3	0.28	baf212u1060	38,639	5	0.24				
				200i3000-805	6,902	0	0.65	baf217vm1084	44,681	10	—				
Average		**9.15**	**−0.004**			**7.62**	**0.09**			**8.56**	**0.17**			**2.85**	**0.98**

Again, we give some side notes. The refined ILS finds a BKS at least once for 134 of the 157 instances (85 %). While this number remains unchanged in comparison to the basic ILS, the number of instances for which a BKS was not found decreases from 26 to 23. The new BKS for instance 45tsp225 of the `GTSP_LIB` can be confirmed and another new one for the instance 287u1432 of the `LARGE_LIB` is found.

For the `GTSP_LIB`, a BKS (or a better one) is found in 9.15 of 10 runs on average. Due to the negative deviation for instance 45tsp225 and the relatively small deviations for all other instances the average deviation to the BKS decreases to $-0.004\,\%$ over all instances. For instances with up to 113 clusters, a BKS is found or undercut in 10 of 10 runs, while for instances with more than 113 clusters, the number of runs in which a BKS is obtained varies between 0 and 10 with an average of 7. Their corresponding average deviation could be reduced from 0.11% to 0.06% compared to the basic ILS.

For the `LARGE_LIB`, 7 of 27 instances are solved with a BKS in 10 of 10 runs, 4 instances are solved with a BKS at least one time, and for 16 instances the BKS is never obtained (3 less compared to the basic ILS). For instances with more than 400 clusters, the average percentage error is relatively large and varies between 0.74 % and 6.77 %, with an average of 2.47 % (3.51 % for basic ILS). The new BKS with costs of $54,437$ is found for instance 287u1432 in 1 of 10 runs, leading to a reduction of the total G-tour costs by 0.06 %.

The results for `MOM_LIB` and `BAF_LIB` remain unchanged because all these instances are still solved with the basic ILS.

Finally, we compare the refined ILS with the basic ILS and the best-performing algorithms *GK*, *GLKH* and *GLNS* from the literature. Table 7 provides an overview with all entries as defined before for Table 5. The refined ILS clearly outperforms the basic ILS on `GTSP_LIB` and `LARGE_LIB`. In comparison to the literature, results are not clear-cut and there is no unique winner. The *GLNS* outperforms all other algorithms on the `MOM_LIB`, `BAF_LIB`, and `LARGE_LIB`, where clusters are generated so that they do *not* comprise sets of vertices that are mutually close. For the `GTSP_LIB`, however, *GLKH* is best regarding #best (directly followed by the refined ILS) and the refined ILS is best regarding Δ(%). It seems that the refined ILS can cope very well with instances where the clusters consist of relatively close vertices.

Table 7: Comparison of the Basic and Refined ILS with best-performing Algorithms from the Literature.

	`GTSP_LIB`		`MOM_LIB`		`BAF_LIB`		`LARGE_LIB`	
Algorithm	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)
GK	9.10	0.01	8.44	0.03	8.11	0.29	2.77	0.76
GLKH	**9.20**	0.01	5.40	0.82	5.04	6.51	3.04	0.52
GLNS	8.73	0.01	**9.18**	**0.02**	**8.91**	**0.07**	**3.31**	**0.50**
Basic ILS	8.70	0.01	7.62	0.09	8.56	0.17	2.37	1.39
Refined ILS	9.15	$\mathbf{-0.004}$	7.62	0.09	8.56	0.17	2.85	0.98

6. Conclusions

The paper has introduced a simple but effective ILS for solving symmetric instances of the GTSP. The basic ILS combines several neighborhoods into a random VND that are then used as the local-search component of the ILS. For the VND, we introduced three new neighborhoods that allow simultaneous modifications of the sequence of the clusters and the selection of vertices per cluster. Unexpectedly, this truly simple design of an ILS already gives reasonable results on standard benchmarks.

One finding of our experimental studies is that a single ILS setup that is not tailored to characteristics of the GTSP instance at hand can be easily improved. Indeed, while some instances have geometrically defined clusters that comprise mutually close vertices, other instances systematically spread the vertices of all clusters. The distinction of these cases is crucial, and we have defined a refined ILS that chooses priorities for the neighborhoods in the VND according to the size N and the average relative inner-cluster distance.

With these refinements, the computational results also show that the new GTSP neighborhoods contribute significantly to the success of both ILS versions. This can be seen, for example, from the fact that the Balas-Simonetti neighborhood with $k = 8$ further improves elite solutions. Moreover, the good performance of the adapted Gutin neighborhood becomes evident within the refined ILS, where this neighborhood is used with highest priority/first in the nested VND II.

The refined ILS is particularly powerful on the `GTSP_LIB` and improves the results of the basic ILS on the `LARGE_LIB` (finding one new best solution in both libraries). For the latter library, the refined ILS performs slightly below the best algorithms from the literature. This can be attributed to prohibitively increasing computation times for exploring some neighborhoods for large-scale GTSP instances. A strong point in favor of the new ILS is, however, its simple design making it much less involved to code compared to many other metaheuristics.

For the future, further accelerating neighborhood explorations with established techniques like granular search (Toth and Vigo, 2003; Schröder *et al.*, 2020), don't look bits (Hoos and Stützle, 2005, p. 375), and dynamic sequential/radius search (Irnich *et al.*, 2006; Gauthier and Irnich, 2020) as well as with new ideas to be worked out is a promising research path.

Acknowledgement

This research was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant IR 122/7-2. This support is gratefully acknowledged.

References

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (1999). CONCORDE. Available at `http://www.math.uwaterloo.ca/tsp/concorde/index.html`.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2015). National traveling salesman problems. Available at `http://www.math.uwaterloo.ca/tsp/world/countries.html`.

Balas, E. (1999). New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research*, **86**, 529–558.

Balas, E. and Simonetti, N. (2001). Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS Journal on Computing*, **13**(1), 56–75.

Ben-Arieh, D., Gutin, G., Penn, M., Yeo, A., and Zverovitch, A. (2003). Transformations of generalized ATSP into ATSP. *Operations Research Letters*, **31**(5), 357–365.

Bentley, J. J. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, **4**(4), 387–411.

Bontoux, B. (2008). *Techniques hybrides de recherche exacte et approchée : application à des problèmes de transport*. Ph.D. thesis, Université d'Avignon. Français.

Bontoux, B., Artigues, C., and Feillet, D. (2010). A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem. *Computers & Operations Research*, **37**(11), 1844–1852.

Cacchiani, V., Muritiba, A., Negreiros, M., and Toth, P. (2011). A multistart heuristic for the equality generalized traveling salesman problem. *Networks*, **57**, 231–239.

Dueck, G. (1993). New optimization heuristics. *Journal of Computational Physics*, **104**(1), 86–92.

Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, **44**(3), 216–229.

Fischetti, M., Gonzáles, J., and Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, **45**(3), 378–394.

Gauthier, J. B. and Irnich, S. (2020). Inter-depot moves and dynamic-radius search for multi-depot vehicle routing problems. Technical Report LM-2020-3, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany.

Glover, F. (1996). Finding a best traveling salesman 4-opt move in the same time as a best 2-opt move. *Journal of Heuristics*, **2**, 169–179.

Gutin, G. and Karapetyan, D. (2009). Generalized traveling salesman problem reduction algorithms. arXiv 0804.0735v2.

Gutin, G. and Karapetyan, D. (2010). A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, **9**(1), 47–60.

Gutin, G., Yeo, A., and Zverovitch, A. (2007). Exponential neighborhoods and domination analysis for the TSP. In *The Traveling Salesman Problem and Its Variations*, Combinatorial Optimization, pages 223–256. Springer US.

Gutin, G., Karapetyan, D., and Krasnogor, N. (2008). Memetic algorithm for the generalized asymmetric traveling salesman problem. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, pages 199–210. Springer Berlin Heidelberg.

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, **130**(3), 449–467.

Hansen, P. and Mladenović, N. (2005). Variable neighborhood search. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 8, pages 211–238. Springer US.

Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, **126**, 106–130.

Helsgaun, K. (2015). Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun Algorithm. *Mathematical Programming Computation*, **7**(3), 269–287.

Henry-Labordere, A. (1969). The record balancing problem: A dynamic programming solution of a generalized travelling salesman problem. *RIRO B-2*, pages 43–49.

Hoos, H. H. and Stützle, T. (2005). *Stochastic Local Search*. Morgan Kaufmann, Amsterdam.

Hu, B. and Raidl, G. R. (2008). Effective neighborhood structures for the generalized traveling salesman problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 36–47. Springer Berlin Heidelberg.

Irnich, S., Funke, B., and Grünert, T. (2006). Sequential search and its application to vehicle-routing problems. *Computers & Operations Research*, **33**(8), 2405–2429.

Johnson, D. S. and McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 8, pages 215–310. Wiley, Chichester.

Johnson, D. S., McGeoch, L. A., Glover, F., and Rego, C. (2000). 8th dimacs implementation challenge: The traveling salesman problem. Available at http://dimacs.rutgers.edu/archive/Challenges/TSP/.

Karapetyan, D. and Gutin, G. (2011). Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem. *European Journal of Operational Research*, **208**(3), 221–232.

Karapetyan, D. and Gutin, G. (2012). Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem. *European Journal of Operational Research*, **219**(2), 234–251.

Laporte, G. and Semet, F. (1999). Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, **37**(2), 114–120.

Laporte, G., Asef-Vaziri, A., and Sriskandarajah, C. (1996). Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society*, **47**(12), 1461–1467.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, **44**, 2245–2269.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, **21**(2), 498–516.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 320–353. Springer.

Mestria, M., Ochi, L. S., and de Lima Martins, S. (2013). GRASP with path relinking for the symmetric Euclidean clustered traveling salesman problem. *Computers & Operations Research*, **40**(12), 3218–3229.

Noon, C. E. (1988). *The generalized traveling salesman problem*. Ph.D. thesis, University of Michigan.

Noon, C. E. and Bean, J. C. (1991). A Lagrangian based approach for the asymmetric generalized traveling salesman problem. *Operations Research*, **39**(4), 623–632.

Noon, C. E. and Bean, J. C. (1993). An efficient transformation of the generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, **31**(1), 39–44.

Pintea, C. M., Pop, P. C., and Chira, C. (2017). The generalized traveling salesman problem solved with ant algorithms. *Complex Adaptive Systems Modeling*, **5**(1).

Reihaneh, M. and Karapetyan, D. (2012). An efficient hybrid ant colony system for the generalized traveling salesman problem. *Algorithmic Operations Research*, **7**, 22–29.

Reinelt, G. (1991). TSPLIB - A traveling salesman problem library. *ORSA Journal on Computing*, **3**(4), 376–384.

Renaud, J. and Boctor, F. F. (1998). An efficient composite heuristic for the symmetric generalized traveling salesman problem. *European Journal of Operational Research*, **108**(3), 571–584.

Saksena, J. P. (1970). Mathematical model of scheduling clients through welfare agencies. *Journal of the Canadian Operational Research Society*, **8**, 185–200.

Schröder, C., Gauthier, J. B., Gschwind, T., and Schneider, M. (2020). In-depth analysis of granular local search for capacitated vehicle routing. Working Paper DPO-2020-03, Deutsche Post Chair – Optimization of Distribution Networks, RWTH Aachen University, Aachen, Germany.

Silberholz, J. and Golden, B. (2007). The generalized traveling salesman problem: A new genetic algorithm approach. In *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, pages 165–181. Springer US.

Smith, S. L. and Imeson, F. (2017). GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, **87**, 1–19.

Snyder, L. and Daskin, M. (2006). A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, **174**(1), 38–53.

Subramanian, A., Drummond, L. M. A., Bentes, C., Ochi, L. S., and Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, **37**(11), 1899–1911.

Tasgetiren, M. F., Suganthan, P. N., and Pan, Q. Q. (2007). A discrete particle swarm optimization algorithm for the generalized traveling salesman problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO 2007*, pages 158–167.

Toth, P. and Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, **15**(4), 333–346.

Yang, J., Shi, X., Marchese, M., and Liang, Y. (2008). An ant colony optimization method for generalized TSP problem. *Progress in Natural Science*, **18**(11), 1417–1422.