EURO Journal on Computational Optimization 10 (2022) 100029

Contents lists available at ScienceDirect

EURO Journal on Computational Optimization

journal homepage: www.elsevier.com/locate/ejco

# New neighborhoods and an iterated local search algorithm for the generalized traveling salesman problem

Jeanette Schmidt *, Stefan Irnich

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

ARTICLE INFO

*Keywords:*
Traveling salesman
Generalized traveling salesman problem
Iterated local search
Variable neighborhood descent

ABSTRACT

For a given graph with a vertex set that is partitioned into clusters, the generalized traveling salesman problem (GTSP) is the problem of finding a cost-minimal cycle that contains exactly one vertex of every cluster. 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 an effective iterated local search (ILS) for the GTSP. The ILS performs a straightforward random neighborhood selection within the local search and applies an ordinary record-to-record ILS acceptance criterion. The computational experiments on four symmetric standard GTSP libraries show that, with some purposeful refinements, the ILS can compete with state-of-the-art GTSP algorithms.



* Corresponding author.
*E-mail addresses:* sjeanett@uni-mainz.de (J. Schmidt), irnich@uni-mainz.de (S. Irnich).

https://doi.org/10.1016/j.ejco.2022.100029

## 1. Introduction

For a given graph with a vertex set that is partitioned into clusters, the generalized traveling salesman problem (GTSP) is the problem of finding a cost-minimal cycle that contains exactly one vertex of every cluster. 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 $i \in I = \{1, 2, \ldots, N\}$, such that $V = \bigcup_{i \in I} V_i$. Note that clusters do not necessarily result from a geometric (i.e. distance or proximity based) grouping. Depending on the application, clusters may contain vertices that are 'far away' from each other.

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 (unordered) 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 $i$th cluster, i.e., $m_i = |V_i|$ for all $i \in I$. A feasible solution to the GTSP is a cycle $x = (x_1, x_2, \ldots, x_N, x_1)$ 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 this paper, we present an effective metaheuristic for the GTSP. The metaheuristic follows the principles of a clean *iterated local search* [ILS, 35]. ILS is a stochastic local search [in the sense of 22]: Iteratively, a current solution is randomly perturbed, the perturbed solution undergoes a local search, and the resulting solution is tested to be accepted as the new current solution by comparing current and resulting solution (possibly taking into account the search history). These simple steps are repeated until a termination condition is met. The best seen solution is the output of the ILS.

The local search part of our ILS combines known and three new GTSP neighborhoods in a *variable neighborhood descent* [VND, 17] fashion. Our ILS performs a straightforward random neighborhood prioritization within the VND and applies an ordinary record-to-record acceptance criterion [8].

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. We show that 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 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, the use of the GTSP-adapted Balas-Simonetti neighborhood [2] for improving high-quality solutions,

and a modified cooling schedule for the record-to-record acceptance criterion. Moreover, we present a straightforward grouping scheme for GTSP instances that allows us to control which algorithmic refinements to apply to which type of GTSP instance. In computational experiments on standard GTSP benchmark instances, we show that the resulting refined ILS produces very reasonable solutions.

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 [32] as well as the routing of clients through welfare agencies [44], and computer file sequencing [21].

Since its introduction in the late sixties/early seventies [21,44] 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 [39], Laporte and Semet [31], Ben-Arieh *et al.* [3]. The resulting TSP instances can then be solved with a TSP solver, either heuristically or, if not too large, exactly. For example, Helsgaun [20] combined the Noon-Bean reduction with the Lin-Kernighan-Helsgaun algorithm [LKH, 19] 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.* [10] 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 [38]. 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., 37] 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 40]. A first approach, called RP2 and nowadays known as *cluster optimization* (CO), was invented by Fischetti *et al.* [10]. CO determines a globally best vertex selection for a

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

| (Meta)heuristic | Reference(s) |
|---|---|
| Tour construction heuristic | [37]; [10] |
| Composite heuristic | [43] |
| Adaptive Large Neighborhood Search | [47] |
| Ant Colony Optimization | [52]; [41]; [40] |
| Genetic/memetic Algorithm | [48]; [46]; [16]; [14]; [6] |
| Lin-Kerninghan adaption | [26] |
| Multi-start method | [7] |
| Particle Swarm Optimization | [50] |
| Variable Neighborhood Search | [23] |

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 GTSP algorithms, e.g., of Cacchiani *et al.* [7], Reihaneh and Karapetyan [41], Smith and Imeson [47].

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 [33]. Such improvement procedures have been used in Yang *et al.* [52] and Bontoux *et al.* [6]. 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.* [10] based on 2-Opt and 3-Opt exchanges. Renaud and Boctor [43] 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 [23] 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 [13].

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 [16,14]. Adapted relocation moves, also known as insert or node shift moves, are used in [48,46,50,16,14,47]. Bontoux *et al.* [6] search for best relocation moves with a special dynamic-programming algorithm that is inspired by the work of Feillet *et al.* [9] 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.* [6] use the original version within their memetic algorithm, while Karapetyan and Gutin [26] 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 [27] provides such a synopsis. Furthermore, they explain efficient neighborhood exploration algorithms for all neighborhoods.

The GTSP remains an active research object with recent publications by Krari *et al.* [29,30], and Ren *et al.* [42]. After all, the memetic algorithm *GK* [14], the Lin-Kernighan-Helsgaun algorithm *GLKH* [20] and the adaptive large neighborhood search *GLNS* [47] are still the best-performing state-of-the-art GTSP solvers. Since we will evaluate our algorithm against these GTSP solvers, we briefly outline their main algorithmic components in comparison to our ILS in Table 2.

## 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 traditional TSP neighborhoods (Section 3.1), polynomially-sized GTSP neighborhoods (Section 3.2), and exponentially-sized GTSP neighborhoods (Section 3.3). We introduce three exponentially-sized GTSP neighborhoods that have not yet been considered in other works. Moreover, we present effective neighborhood exploration algorithms for them. For a GTSP-tailored version of the Balas-Simonetti neighborhood and an adapted version of the Gutin neighborhood, the algorithms are polynominaly bounded but guarantee that a best improving neighbor is found. The third neighborhood is String Relocation+, for which we develop an effective neighborhood exploration heuristic.

### *3.1. TSP neighborhoods*

We first consider (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 [33]. 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 [34]. This technique is known under different names as fixed radius near neighbor search [4], fixed radius search [22, p. 373f], or sequential search [24]. 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

**Table 2**
Main algorithmic components of the state-of-the-art GTSP solvers.

| Algorithm | Metaheuristic paradigm | Neighborhoods used in local search part | Acceptance criterion | Specific components |
|---|---|---|---|---|
| GK | Genetic Algorithm (GA) | ($k$-neighbor) Swap, (direct) 2-Opt, CO, Relocation+ | only improving solutions | well-fitted genetic operators, efficient termination condition |
| GLKH | Local Search | LKH, CO | only improving solutions | Noon and Bean [39] reduction, LKH algorithm |
| GLNS | Adaptive Large Neighborhood Search (ALNS) | Relocation+, CO | Metropolis | unified insertion/removal mechanism, increased randomness during insertions and removals, warm starts, additive noise on insertion costs, improved adaptive weights |
| our algorithm | Iterated Local Search (ILS) | 2-Opt, 3-Opt, Double Bridge, Relocation+, Swap+, CO, Balas-Simonetti for GTSP, adapted Gutin neighborhood, String Relocation+ | record-to-record criterion | efficient local search part with new GTSP neighborhoods, refined ILS tailored to specific groups of GTSP instances |

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 [22, p. 373f].

*Double-Bridge.* The double-bridge move [25] 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 [12] 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.

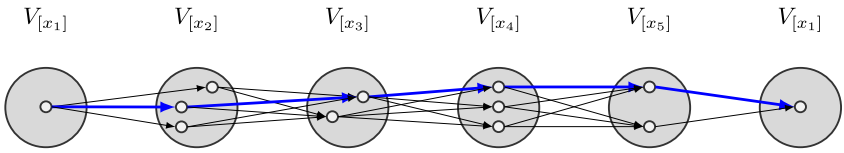


**Fig. 1.** Layered Network for CO.

*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 $i \in I$ [10]. Hence, the sequence of the clusters is kept, however, 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 Fig. 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 Fig. 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.

Karapetyan and Gutin [27] 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 [27] 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 $\mathcal{O}\left(n \cdot \min_i m_i \cdot \max_i m_i\right)$ time. As the (practical) impact of the two last techniques is relatively minor in our context, 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 [1,2]. 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 $\pi$ 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 $G_k$. Fig. 2 shows the auxiliary network $G_k$ for $k = 2$, where layers run horizontally and states vertically. In general, the auxiliary network $G_k$ is a layered

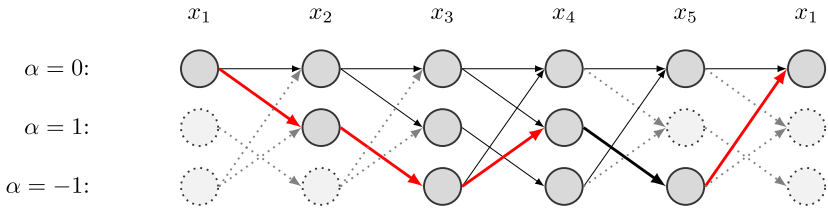


**Fig. 2.** Auxiliary Network $G_k$ for $k = 2$.

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 per layer for $k = 2$; $N + 1 = 5 + 1 = 6$ layers) partially describing the permutation $\pi$. To this end, the states in each layer have an associated value $\alpha$ depicted left to the states of a row in Fig. 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 $i \in \{1, 2, \ldots, N\}$ (for $k = 2$, each subgraph comprises six $[= 3 \cdot 2]$ vertices). 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 Fig. 2, there are five $(\leq 2 \cdot 3 \cdot 2^0 = 6)$ arcs connecting two consecutive layers.

The point is now that every 0-0'-path in $G_k$ describes exactly one permutation $\pi$ 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 Fig. 2, associated with $\alpha = 0$). A state with value $\alpha$ in $i$th 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 Fig. 2) represents the neighbor $x' = x = (x_1, x_2, \ldots, x_5, x_1)$. 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 $x$.

Accordingly, if arcs $(v, w) \in G_k[L_i \cup L_{i+1}]$ with values $\alpha_v$ for the state $v \in L_i$ and $\alpha_w$ 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., $\mathcal{O}(N\, k(k + 1)2^{k-2})$. 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 no difficulty.

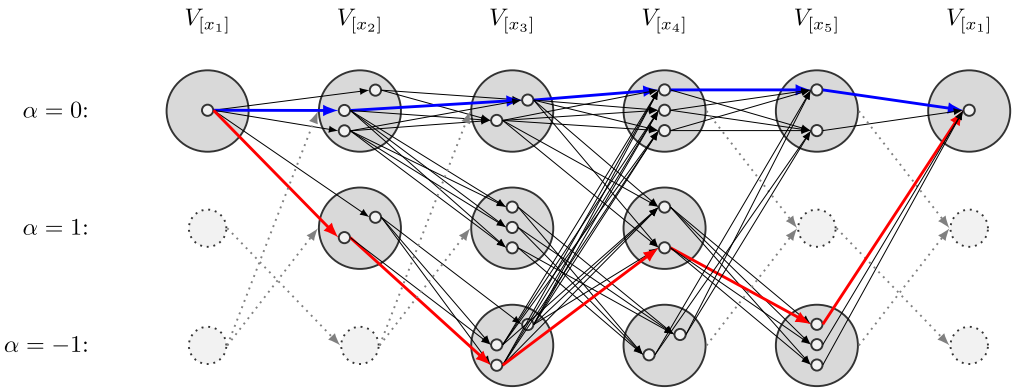


**Fig. 3.** Auxiliary GTSP Network $G_k$ for $k = 2$.

We now combine CO and BS into a new neighborhood that simultaneously permutes the order of the clusters (via BS) and allows to select alternative vertices from the permuted clusters. Fig. 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 $(V_{[x]_1}, V_{[x_2]}, \ldots, V_{[x]_N}, V_{[x_{N+1}]})$ (assuming $V_{[x_{N+1}]} = V_{[x_1]}$). Note the similarity between the states in Fig. 2 and meta-states in Fig. 3. However, meta-states are not part of the neighborhood's network $G_k$ but are depicted only for the purpose of explanation.

In Fig. 3, the different number of states and their graphical positioning helps to distinguish between different clusters. For example, $V_{[x_1]}$ comprises one state, $V_{[x_2]}$ three states, and $V_{[x_3]}$ two states. Both $V_{[x_2]}$ and $V_{[x_4]}$ contain three states, but are depicted differently.

All states of a meta-state at the $i$th 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 Fig. 1 is identical to $G_1$. Note that the upper row of meta-states in Fig. 3 is identical to the layered network for CO depicted in Fig. 1. Indeed, all states in the upper row have $\alpha = 0$, i.e., the sequence of these states is not permuted at all. Any path through the meta-states with $\alpha = 0$ in Fig. 3 stands for a selection of vertices from the currently given sequence of clusters (the path depicted in blue is one example). These are exactly the decision made by choosing an neighbor from the CO neighborhood. On the contrary, any 0-0'-path in Fig. 3 through at least one meta-state with $\alpha \neq 0$ permutes the given sequence of clusters. The 0-0'-path depicted in red induces the new sequence $(V_{[x_1]}, V_{[x_3]}, V_{[x_2]}, V_{[x_5]}, V_{[x_4]}, V_{[x_1]})$ of the clusters.

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, 15, assuming $N \geq k(k+1)$] and the second term comes from the CO analysis.

*Gutin Neighborhood.* The assignment neighborhood of the TSP [15] 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 example, the subset $Z = \{2, 4, 7\}$ allows for the tour $x = (x_1, x_2, \ldots, x_7, x_1)$ 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 of the Gutin neighborhood for the GTSP works as follows: Let a G-tour $x = (x_1, x_2, \ldots, x_N, x_1)$ be given. First, we determine the set $Z$ with a randomized heuristic. Initially we set $Z = \varnothing$. Iterating over $i = 1, 2, \ldots, N$, we first test whether $i-1$ has been chosen. If $i - 1 \in Z$, we skip $i$ (for $i = N$ we also test whether $1 \in Z$) and iterate. Otherwise, we toss a coin to decide whether $i-1$ 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, $1+1/2+1/4+1/8+\cdots = 2$. 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 computation of the insertion cost and the second from the exact solution of the assignment problem over $(a_{i,j})_{i,j \in Z}$.

*String Relocation+.* We describe now the new string relocation neighborhood $\mathcal{N}_L^{SR}$ for $L \geq 1$, which 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 $x = (x_1, x_2, \ldots, x_N, x_1)$ is given. The string $(x_i, x_{i+1}, \ldots, x_{i+k})$ to be removed 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 $(x_i', x_{i+1}', \ldots, x_{i+k}')$ with $[x_i'] = [x_i], [x_{i+1}'] = [x_{i+1}], \ldots, [x_{i+k}'] = [x_{i+k}]$. 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:

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

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

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

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

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 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 $\mathcal{O}\left(n^2\right)$ time.

The exploration heuristic for $\mathcal{N}_L^{SR}(x)$ is presented with the pseudo-code in Algorithm 1. In Step 1, we determine the first vertex $x_i$ and the starting position $i$ of the string that is relocated. With the loop in Step 2, we consider all possible replacements of $x_i$ by an $x'_i$ of the same cluster. The loop in Step 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 $x'_{i+k-1}$ (Step 5). Thanks to the lookup table, Step 5 takes only constant time $\mathcal{O}(1)$. The result is that the string $(x_i, x_{i+1}, \ldots, x_{i+k})$ of length $k$ is possibly replaced by the string $(x'_i, x'_{i+1}, \ldots, x'_{i+k})$. The insertion position $j$ is determined in the loop in Step 7. Finally, the resulting move is completely determined now so that the cost of the move and possible improvement can be checked (Step 8).

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

## 4. Basic iterated local search

In this section, we propose our basic metaheuristic algorithm based on ILS [35] using a random VND [49] 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 previous local minimum or locally optimal solution $x^*$ is accepted as the new current solution. Hence, the acceptance criterion controls the balance between intensification and diversification.

In the following we will discuss the implementation of the main functions considered in the basic ILS (Section 4.1), introduce the GTSP benchmark set (Section 4.2), and evaluate the basic ILS by presenting computational results (Section 4.3).

### *4.1. Implementation of the main ILS functions*

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.

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

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

*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.* [10] adapted the well-known TSP insertion heuristics to obtain a feasible G-tour. Based on these, Smith and Imeson [47] 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 [10]. 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., $y = \arg\min_{x \in V_{[i]}, 1 \le j \le k} \{c_{x_j x} + c_{x x_{j+1}} - c_{x_j x_{j+1}}\}$. 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 traditional 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 $k \le 3$, 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 $L = 4$.

Every time the local search (Step 5 in Algorithm 2) is invoked, the random VND randomly selects new priorities for all available neighborhoods $\mathcal{N}^1, \mathcal{N}^2, \ldots, \mathcal{N}^\ell$. With chosen priorities (we assume that neighborhoods are reshuffled accordingly), it works like any other standard VND, i.e., it always starts the neighborhood exploration with $\mathcal{N}^1$, continues with $\mathcal{N}^2$ etc., and finally ends with $\mathcal{N}^\ell$. More precisely: If, for the current solution $x'$, an improving neighbor $x^* \in \mathcal{N}^1(x')$ is found, this neighbor $x^*$ is returned, and it undergoes the acceptance criterion (Step 6). Otherwise, $\mathcal{N}^2(x')$ is explored next. In general, if an improving neighbor $x^* \in \mathcal{N}^p(x')$ is found in the $p$th neighborhood, it is returned, and afterwards the VND starts from the beginning with the first neighborhood $\mathcal{N}^1$.

We ensure that a neighborhood exploration is however only started if an improvement is possible. Hence, if neighborhoods are included into one another, i.e., $\mathcal{N}^i \subset \mathcal{N}^j$ for $i \neq j$, we make sure that the smaller neighborhood $\mathcal{N}^i$ is prioritized before the larger neighborhood $\mathcal{N}^j$, i.e., $i < j$. We exploit $\mathcal{N}^{CO} = \mathcal{N}_1^{BS} \subset \mathcal{N}_2^{BS} \subset \mathcal{N}_3^{BS}$. Finally, we allow that two neighborhoods receive identical random priorities, in which case we alternatingly explore them.

*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 guide 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 [25] 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 the objective function 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 randomized GTSP construction heuristic.

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

We balance intensification and diversification with the deterministic record-to-record travel [8] 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. Step 6 in Algorithm 2). Initially, we set $\epsilon$ 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 [28], we systematically lower $\epsilon$ in the course of the ILS. The cooling update takes place every $N$ iterations of the main loop. The update lowers $\epsilon$ by the factor $h = 0.8$ (Step 10). That means, e.g., that $\epsilon$ is at approximately a tenth of its initial value after $10N$ iterations.

### *4.2. 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 [14],
- the Lin-Kernighan-Helsgaun *GLKH* algorithm of Helsgaun [20], and
- the large neighborhood search *GLNS* of Smith and Imeson [47].

Note that *GK*, *GLKH*, and *GLNS* are the best-performing state-of-the-art GTSP solvers published in the literature (see Section 2). Smith and Imeson [47] 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.* [10] and extended by Silberholz and Golden [46]. The vertex clustering simulates geographical regions. The 45 largest instances of the library have been used in the `GK`, `GLKH`, and `GLNS` comparison presented by Smith and Imeson [47]. We omit the five asymmetric instances.

- The `BAF_LIB` was introduced by Bontoux [5]. The pseudo-random clustering scheme implies that there are *no* geographical regions. The standard comparison uses the 45 largest instances.
- The `MOM_LIB` was introduced by Mestria *et al.* [36] and contains instances with six different clustering schemes. Here, the 45 largest instances are used in the *GK*, *GLKH*, and *GLNS* comparison of Smith and Imeson [47].
- The `LARGE_LIB` was introduced by Helsgaun [20] and contains very large instances. The clusters were also generated with the clustering scheme of Fischetti *et al.* [10], i.e., they simulate geographical regions. The 26 smallest instances (referred to as `LARGE_LIB(I)` in the following) have been used to benchmark the *GK*, *GLKH*, and *GLNS* algorithms. Seven even larger instances (referred to as `LARGE_LIB(II)`) have been used to benchmark the *GLKH* against the *GLNS* algorithm, while no results for GK have been reported.

### *4.3. 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™ 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 [47] did: Smith and Imeson set the computation time limit to

- 300 seconds for all instances of the `GTSP_LIB`, `MOM_LIB`, and `BAF_LIB`, and
- 1200 seconds for all instances of the `LARGE_LIB`.

According to PassMark (https://www.cpubenchmark.net/singleCompare.php), our processor is slightly slower in the single-thread performance than the Intel® Core™ i7-6700 clocked at 3.40 GHz used by Smith and Imeson (2062 points versus 2303 points). We therefore grant 335 seconds for instances of `GTSP_LIB`, `MOM_LIB`, and `BAF_LIB` and 1341 seconds for instances of `LARGE_LIB`.

Table 3 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.
- `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 [47] (available at https://ece.uwaterloo.ca/~sl2smith/GLNS/) and [20] (available at http://akira.ruc.dk/~keld/research/GLKH/).
- `#best` is the number of runs, out of ten, in which a BKS was obtained.
- $\Delta(\%)$ 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 $e = 100 \cdot (c(x_{ILS}) - c(x_{BKS}))/c(x_{BKS})$.

We give some side notes: The basic ILS finds a BKS at least once for 134 of the 163 instances, while for 98 instances it was determined in 10 of 10 runs.

For the GTSP_LIB, a BKS is found in 8.88 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 10 with an average of 5.

For the MOM_LIB, the basic ILS finds a BKS in at least one run for 44 of the 45 instances. For 27 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 4.6, with an average percentage error of 0.15%.

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 0.56%. On the downside, for two instances, a BKS was never obtained in all 10 runs. For both of them, the average percentage error is rather high with 1.39% and 2.40%.

For the LARGE_LIB, we summarize the results as follows: From 26 of the subset LARGE_LIB(I) instances, only 6 instances can be solved with the BKS in 10 of 10 runs, two 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 1.22%. In particular for instances with more than 400 clusters, the average percentage error varies between 1.05% and 6.72%. On the second subset LARGE_LIB, no BKS has been obtained. Also here, the percentage error of 5.49% is relatively large.

Finally, we compare the basic ILS with the three algorithms *GK*, *GLKH*, and *GLNS*. Table 4 presents average values of *#best* and $\Delta(\%)$ on the five 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 [47]. Moreover, the *GK* algorithm could not be compared on the LARGE_LIB(II) instances, because these instances were not considered in [14]. 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.

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 iterated 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 straightforward but powerful algorithm and well-reproducible results. In Section 5.1, we describe the implementation of refinements regarding a *reset component*, *prioritization*

**Table 3**
Results of the Basic ILS on 163 symmetric GTSP Instances.

| GTSP_LIB | | | | MDM_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 | 4,420 | 10 | — | 50kroA100 | 15,944 | 10 | — | baf20rd100 | 1,747 | 10 | — | 200C1k.0 | 6,375,154 | 10 | — |
| 39rat195 | 854 | 10 | — | 50kroB100 | 15,842 | 10 | — | baf21eil101 | 105 | 10 | — | 200E1k.0 | 9,662,857 | 0 | 0.34 |
| 40d198 | 10,557 | 10 | — | 50lin105 | 11,294 | 10 | — | baf21lin105 | 2,758 | 10 | — | 235pcb1173 | 23,399 | 1 | 0.78 |
| 40kroa200 | 13,406 | 10 | — | 50lin318 | 18,163 | 10 | — | baf22pr107 | 6,849 | 10 | — | 259d1291 | 28,400 | 4 | 0.11 |
| 40krob200 | 13,111 | 10 | — | 50nrw1379 | 7,449 | 10 | — | baf24gr120 | 1,377 | 10 | — | 261rl1304 | 150,468 | 0 | 0.22 |
| 41gr202 | 23,301 | 10 | — | 50pcb1173 | 9,385 | 10 | — | baf25pr124 | 10,745 | 10 | — | 265rl1323 | 154,023 | 0 | 0.35 |
| 45ts225 | 68,340 | 10 | — | 50pcb442 | 14,430 | 10 | — | baf26bier127 | 11,740 | 10 | — | 276nrw1379 | 20,050 | 0 | 0.32 |
| 45tsp225 | 1,612 | 10 | — | 50pr1002 | 54,583 | 10 | — | baf28pr136 | 17,824 | 10 | — | 280fl1400 | 15,316 | 10 | — |
| 46gr229 | 71,972 | 10 | — | 50pr439 | 45,253 | 10 | — | baf29pr144 | 14,070 | 10 | — | 287u1432 | 54,469 | 0 | 0.28 |
| 46pr226 | 64,007 | 10 | — | 50rat783 | 1,626 | 10 | — | baf30kroA150 | 7,005 | 10 | — | 316fl1577 | 14,182 | 10 | — |
| 53gil262 | 1,013 | 10 | — | 50rat99 | 814 | 10 | — | baf30kroB150 | 5,855 | 10 | — | 331d1655 | 29,443 | 0 | 0.49 |
| 53pr264 | 29,549 | 10 | — | 50vm1084 | 54,156 | 10 | — | baf31pr152 | 13,002 | 10 | — | 350vm1748 | 185,459 | 0 | 0.38 |
| 56a280 | 1,079 | 10 | — | 72vm1084-8x9 | 64,647 | 10 | — | baf32u159 | 7,301 | 10 | — | 364u1817 | 25,530 | 0 | 0.46 |
| 60pr299 | 22,615 | 10 | — | 75lin105 | 13,134 | 10 | — | baf39rat195 | 477 | 10 | — | 378rl1889 | 184,034 | 0 | 0.36 |
| 64lin318 | 20,765 | 10 | — | 81vm1084-9x9 | 69,659 | 10 | — | baf40d198 | 1,466 | 10 | — | 421d2103 | 40,049 | 0 | 1.02 |
| 80rd400 | 6,361 | 10 | — | 100i1000-410 | 5,481 | 10 | — | baf40kroA200 | 7,113 | 10 | — | 431u2152 | 27,614 | 0 | 1.04 |
| 84u417 | 9,651 | 10 | — | 100i1500-506 | 5,088 | 8 | 0.06 | baf40kroB200 | 7,126 | 10 | — | 464u2319 | 65,758 | 0 | 2.57 |
| 87gr431 | 101,946 | 10 | — | 100i2000-604 | 5,316 | 7 | 0.08 | baf41gr202 | 3,531 | 10 | — | 479pr2392 | 169,874 | 0 | 2.42 |
| 88pr439 | 60,099 | 10 | — | 100i2500-708 | 5,297 | 10 | — | baf45ts225 | 25,697 | 10 | — | 608pcb3038 | 52,416 | 0 | 5.07 |
| 89pcb442 | 21,657 | 10 | — | 100i3000-803 | 5,458 | 3 | 0.04 | baf46pr226 | 13,555 | 10 | — | 633C3k.0 | 10,255,031 | 0 | 2.28 |
| 99d493 | 20,023 | 10 | — | 100nrw1379 | 10,566 | 10 | — | baf53gil262 | 571 | 4 | 0.32 | 633E3k.0 | 16,197,552 | 0 | 5.37 |
| 107ali535 | 128,639 | 10 | — | 100pcb1173 | 13,901 | 4 | 0.20 | baf53pr264 | 7,716 | 10 | — | 759fl3795 | 18,662 | 0 | 1.05 |
| 107att532 | 13,464 | 10 | — | 100pr1002 | 74,269 | 9 | 0.00 | baf60pr299 | 10,047 | 10 | — | 893fnl4461 | 63,163 | 0 | 6.72 |
| 107si535 | 13,502 | 10 | — | 100prb1173-10x10 | 12,644 | 10 | — | baf64lin318 | 7,489 | 10 | — | Average for LARGE_LIB(I): | | **2.50** | **1.22** |
| 113pa561 | 1,038 | 10 | — | 100rat783-10x10 | 2,216 | 10 | — | baf80rd400 | 3,254 | 3 | 1.07 | | | | |
| 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 | 0.22 | baf88pr439 | 13,882 | 10 | — | 1183rl5915 | 309,243 | 0 | 5.71 |
| 132d657 | 22,498 | 8 | 0.02 | 144rat783-12x12 | 2,813 | 5 | 0.02 | baf89pcb442 | 8,749 | 10 | — | 1187rl5934 | 295,767 | 0 | 5.52 |
| 134gr666 | 163,028 | 7 | 0.17 | 150i1000-411 | 6,296 | 9 | 0.05 | baf99d493 | 3,081 | 10 | — | 1480pla7397 | 12,732,870 | 0 | 4.41 |
| 145u724 | 17,272 | 10 | — | 150i1500-507 | 6,085 | 10 | — | baf107att532 | 3,880 | 8 | 0.06 | 100C10k.0 | 6,158,999 | 0 | 1.17 |
| 157rat783 | 3,262 | 2 | 0.10 | 150i2000-605 | 5,940 | 9 | 0.00 | baf107si535 | 8,912 | 8 | 0.32 | 2000C10k.0 | 18,044,846 | 0 | 4.86 |
| 200dsj1000 | 9,187,884 | 4 | 0.10 | 150i2500-709 | 6,158 | 6 | 0.10 | baf113pa561 | 431 | 0 | 1.39 | 2000E10k.0 | 28,769,011 | 0 | 8.41 |
| 201pr1002 | 114,311 | 5 | 0.03 | 150i3000-804 | 6,551 | 0 | 0.41 | baf115rat575 | 1,330 | 9 | 0.15 | 2370rl11849 | 427,996 | 0 | 8.36 |
| 207si1032 | 22,306 | 0 | 0.07 | 150nrw1379 | 13,370 | 3 | 0.15 | baf131p654 | 5,824 | 4 | 0.03 | Average for LARGE_LIB(II): | | **0.00** | **5.49** |
| 212u1060 | 106,007 | 1 | 0.23 | 150pcb1173 | 17,082 | 3 | 0.35 | baf132d657 | 8,132 | 10 | — | | | | |
| 217vm1084 | 130,704 | 8 | 0.07 | 150pr1002 | 92,969 | 7 | 0.03 | baf145u724 | 7,354 | 1 | 0.56 | | | | |
| | | | | 150rat783 | 3,131 | 2 | 0.31 | baf157rat783 | 1,700 | 9 | 0.25 | | | | |
| | | | | 150vm1084 | 95,922 | 10 | — | baf201pr1002 | 48,400 | 0 | 2.40 | | | | |
| | | | | 200i2000-606 | 7,272 | 1 | 0.28 | baf207si1032 | 18,836 | 9 | 0.00 | | | | |
| | | | | 200i2500-710 | 7,191 | 4 | 0.15 | baf212u1060 | 38,639 | 5 | 0.16 | | | | |
| | | | | 200i3000-805 | 6,902 | 2 | 0.32 | baf217vm1084 | 44,681 | 10 | — | | | | |
| Average | | **8.88** | **0.02** | | | **7.84** | **0.06** | | | **8.67** | **0.15** | | | **1.97** | **2.12** |

**Table 4**
Comparison of the basic ILS with Algorithms from the Literature.

| | GTSP_LIB | | MOM_LIB | | BAF_LIB | | LARGE_LIB(I) | | LARGE_LIB(II) | |
|---|---|---|---|---|---|---|---|---|---|---|
| Algorithm | #best | Δ(%) | #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 | 0.00 | **2.56** |
| *GLNS* | 8.73 | **0.01** | **9.18** | **0.02** | **8.91** | **0.07** | **3.31** | **0.50** | 0.00 | 6.46 |
| Basic ILS | 8.88 | 0.02 | 7.84 | 0.06 | 8.67 | 0.15 | 2.50 | 1.22 | 0.00 | 5.49 |

*of the neighborhoods in VND*, the use of the *Balas-Simonetti neighborhood for detecting high-quality solutions*, and a modified *acceptance criterion*. Section 5.2 introduces an instance-based selection of an ILS setup, while Section 5.3 discusses the final results.

### *5.1. 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 [18], 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

$$\begin{aligned}\mathcal{N}^{all} = \{&\mathcal{N}^{2Opt}_{best}, \mathcal{N}^{2Opt}_{first}, \mathcal{N}^{3Opt}_{best}, \mathcal{N}^{3Opt}_{first}, \mathcal{N}^{dbl-brdg}, \dots, \\ &\mathcal{N}^{BS}_2, \mathcal{N}^{BS}_3, \mathcal{N}^{BS}_4, \mathcal{N}^{BS}_5, \dots, \mathcal{N}^{SR}_4, \dots, \mathcal{N}^{Gutin}\}\end{aligned}$$

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 163 GTSP instances.

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 $\mathcal{N}^{VND}$. Each addition of a neighborhood defines one iteration. Initialization and iterations are performing the following steps:
  - The neighborhood selection process is initialized with $\mathcal{N}_0^{VND} = \varnothing$.
  - In the $p$th iteration ($p = 1, 2, \dots$), i.e., when $|\mathcal{N}_{p-1}^{VND}| = p-1$, the next parameterized neighborhood $\mathcal{N}^p \in \mathcal{N}^{all} \setminus \mathcal{N}_{p-1}^{VND}$ is the one producing the best improvement over the ILS using $\mathcal{N}_{p-1}^{VND}$, when added to $\mathcal{N}_{p-1}^{VND}$. For each candidate $\mathcal{N}^p$, we consider the tentative subset $\mathcal{N}_p^{VND} = \{\mathcal{N}^p\} \cup \mathcal{N}_{p-1}^{VND}$ and evaluate the ILS with the instances $\mathcal{I}$.
  - We measure the improvement (relative to the subset $\mathcal{I}$) as the value

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

    where $c(x_{ILS}^I)$ is the cost of a best solution of instance $I$ obtained with the refined ILS using $\mathcal{N}_p^{VND}$, and $c(x_{BKS}^I)$ 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., $\Delta(I, \mathcal{N}_p^{VND}) \geq \Delta(I, \mathcal{N}_{p-1}^{VND})$ for all $\mathcal{N}^p \in \mathcal{N}^{all} \setminus \mathcal{N}_{p-1}^{VND}$, the process of adding neighborhoods stops.
    The final set of neighborhoods is then chosen as $\mathcal{N}^{VND} = \mathcal{N}_{p-1}^{VND}$.

The sequence in which the neighborhoods are added to the subset $\mathcal{N}^{VND}$ defines the priorities. Note that the VND is deterministic and that the selection of the instances $I \in \mathcal{I}$ is the random component of the approach to design refined ILS setups.

By repeating the random selection of instances $\mathcal{I}$, we have obtained several versions of a nested VND. Three rather well-performing ones give rise to refined versions of ILS in which the random VND is replaced by nested VND I, VND II, or VND III summarized in Table 5. For example, VND I uses five neighborhoods with prioritization $\mathcal{N}^1$ = Relocation+, $\mathcal{N}^2$ = BS($k = 4$), $\mathcal{N}^3$ = Double Bridge, $\mathcal{N}^4$ = 3-Opt, and $\mathcal{N}^5$ = Gutin. For all five neighborhoods, the versions with best-improvement pivoting rule

**Table 5**
Selection, Priorities, and Pivoting Strategies of Neighborhoods.

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

*Note:* The maximum string length for String Relocation+ is $L = 4$ for VND II and VND III. The neighborhoods Swap+, CO, and BS for $k = 3$ are not used in VND I, VND II, and VND III, opposed to the basic ILS described in the previous section.

are selected. None of the three VND setups uses the neighborhoods Swap+, CO, or BS($k = 3$).

We interpret the outcome of the parameter tuning for the three VNDs with neighborhood prioritization as follows:

- The neighborhoods Swap+ and CO are redundant. This is not surprising given that CO in contained in BS and Swap+ is contained in the Gutin neighborhood.
- All other neighborhoods are beneficial, at least in some combination.
- A high priority is either assigned to Relocation+ (in VND I) or to Gutin (in VND II and III). These neighborhoods have in common that they are able to simultaneously relocate a vertex and change the cluster representative.
- BS for $k = 3$ is not beneficial when supersets of the neighborhood (for $k \geq 4$) are available alternatives.

*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 Step 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.

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

**Table 6**
Performance of the four ILS Setups on four GTSP Libraries.

| | GTSP_LIB | | MOM_LIB | | BAF_LIB | | LARGE_LIB(I) | | LARGE_LIB(II) | |
|---|---|---|---|---|---|---|---|---|---|---|
| ILS with | #best | Δ(%) | #best | Δ(%) | #best | Δ(%) | #best | Δ(%) | #best | Δ(%) |
| random VND | 8.88 | 0.02 | **7.84** | **0.06** | **8.67** | **0.15** | 2.50 | 1.22 | 0.00 | 5.49 |
| VND I | **9.20** | **0.01** | 7.42 | 0.11 | 8.22 | 0.43 | 2.81 | 0.96 | 0.00 | 6.44 |
| VND II | 8.90 | 0.02 | 7.44 | 0.10 | 7.44 | 0.95 | **2.92** | **0.71** | 0.00 | 5.11 |
| VND III | 7.73 | 0.09 | 4.71 | 0.42 | 7.24 | 0.38 | 2.23 | 0.77 | 0.00 | **3.47** |

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 5). 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., $c(x) \leq b$.

*Modified Acceptance Criterion.* The rather poor performance on the LARGE_LIB partly results from the fact that only very few ILS iterations can be performed within the given overall time budget. As a consequence, the cooling scheme used in the record-to record acceptance criterion is not appropriate for these instances.

Therefore, we replace the naive cooling schedule (Steps 10 and 11 in Algorithm 2, i.e., every $N$ iterations the value of $\epsilon$ is multiplied by $h = 0.8$) by a scheme that first tries to predict the overall number of ILS iterations that can be performed within the given time budget $t^{max}$ (the equivalent of 300 or 1200 seconds, see Section 4.3). To this end, we always run the ILS for 50 iterations with the initial value of $\epsilon$. Let time $t_{50}$ denote the computation time consumed up to this point. Then,

$$iter^{remain} = 50 \cdot (t^{max} - t_{50})/t_{50}$$

is an estimate for the number of ILS iterations that can be performed in the remaining computation time. As explained in Section 4, we would like to perform at least 10 cooling down steps so that with the cooling rate $h = 0.8$ the initial temperature $\epsilon$ is divided by 10 (note that $10 \approx 1/0.8^{10}$). In order to make all results reproducible, i.e., less dependent of small deviations in computation time, we round down $iter^{remain}/10$ to the next smaller power of 2. Let $N^{cool}$ this number. The only modification to the cooling scheme is that a cooling down step (triggered in Step 10 of Algorithm 2) performs an update of $\epsilon$ after every $N^{cool}$ ILS iterations (instead of after every $N$ iterations). For the large-scale instances, it means that more cooling down steps are performed after fewer ILS iterations. Moreover, the initial value of $\epsilon$ is set to 0.01 (it was defined as 0.03 in the basic ILS), so that a slightly more restrictive regime for accepting deteriorating solution is established.

Table 6 summarizes the comparison of the basic ILS and the three new refined ILS versions using VND I, VND II, and VND III. 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(%)).

The refined ILS with VND II outperforms the other VND versions on the `LARGE_LIB(I)`. Likewise, the refined ILS with VND III performs best on `LARGE_LIB(II)`. For `MOM_LIB` and `BAF_LIB`, the basic ILS is still the best algorithm.

We interpret the outcome of the experiments as follows:

- The ILS with VND I is the clear winning setup for the `GTSP_LIB`.
- No refined ILS setups are successful for the `MOM_LIB` and `BAF_LIB`.
- ILS with VND II and VND III lead to substantial improvements for the `LARGE_LIB`.
- VND II and III are very similar (see Table 5). The different performance on the two subsets `(I)` and `(II)` of the `LARGE_LIB` are surprising. It seems that the very largest instances (subset II) benefit more from the fast 3-Opt neighborhood than the more time-consuming Gutin neighborhood. Indeed, when G-tours are very long the routing aspect becomes increasingly important.
- The comparison of the four ILS setups on the entire benchmark shows that an instance-based selection of a setup has a high potential to improve the overall performance. This idea is elaborated and detailed in the next section.

### *5.2. Instance-based selection of an ILS setup*

With four alternative ILS setups at hand (basic ILS, refined ILS with VND I, VND II or VND III), 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

$$\gamma(I) = \frac{\bar{c}_{in}}{\bar{c}}$$

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

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 $250 < N < 500$. Extra large instances, i.e., instances with $N \geq 500$, can be solved best with the refined ILS with VND III.

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

- If $\gamma(I) < 0.5$, use the refined ILS with VND I;
- If $\gamma(I) \geq 0.5$ and $250 < N < 500$, use the refined ILS with VND II;

- If $\gamma(I) \geq 0.5$ and $N \geq 500$, use the refined ILS with VND III;
- 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.

### *5.3. Computational results of the refined ILS*

The computational setup (time budget etc.) is the same as for the computational experiments presented in Section 4.3. Table 7 shows the results obtained with the refined ILS on the 163 symmetric GTSP instances. The meaning of the columns is the same as in Table 3.

Again, we give some side notes. The refined ILS finds a BKS at least once for 136 of the 163 instances (83.4%). While this number increases by two instances in comparison to the basic ILS, the number of instances for which a BKS was not found decreases from 29 to 27.

For the `GTSP_LIB`, a BKS is found in 9.20 of 10 runs on average.

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 7.33. Their corresponding average deviation could be reduced from 0.06% to 0.04% compared to the basic ILS.

For the `LARGE_LIB(I)`, 6 of 26 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 (2 less compared to the basic ILS). For instances with more than 400 clusters, the average percentage error is relatively large and varies between 0.62% and 3.53%, with an average of 1.39% (3.06% for basic ILS). During pre-tests, a new BKS is found for instance 287u1432 (See Appendix for more information). For the second part of `LARGE_LIB`, `LARGE_LIB(II)`, still no BKS is found, but the average percentage error was decreased from 5.49% to 3.26%.

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

*Effectiveness of the Balas-Simonetti Neighborhood for High-Quality Solutions.* We analyze now the impact that the Balas-Simonetti Neighborhood $\mathcal{N}_8^{BS}$ has on the overall effectiveness of the refined ILS. Note that the instance-based selection of an ILS setup (see Section 5.2) chooses the refined version only for the libraries `GTSP_LIB` and `LARGE_LIB`. For these instances, Table 8 shows the number of neighborhood explorations (*#calls*) of $\mathcal{N}_8^{BS}$, the percentage of neighborhood explorations that found an improving neighbor (*success*), and the percentage of the overall runtime (time) that is consumed by the explorations of $\mathcal{N}_8^{BS}$. For all three indicators, the table includes the minimum (*min*), the average (*avg*), and the maximum (*max*) over the ten runs per instance and all instances of the respective library. Note that whenever an improving neighbor has been found, the

**Table 7**
Results of the Refined ILS on the 163 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 | 4,420 | 10 | — | 50kroA100 | 15,944 | 10 | — | baf20rd100 | 1,747 | 10 | — | 200C1k.0 | 6,375,154 | 10 | — |
| 39rat195 | 854 | 10 | — | 50kroB100 | 15,842 | 10 | — | baf21eil101 | 105 | 10 | — | 200E1k.0 | 9,662,857 | 2 | 0.18 |
| 40d198 | 10,557 | 10 | — | 50lin105 | 11,294 | 10 | — | baf21lin105 | 2,758 | 10 | — | 235pcb1173 | 23,399 | 0 | 0.74 |
| 40kroa200 | 13,406 | 10 | — | 50lin318 | 18,163 | 10 | — | baf22pr107 | 6,849 | 10 | — | 259d1291 | 28,400 | 8 | 0.04 |
| 40krob200 | 13,111 | 10 | — | 50nrw1379 | 7,449 | 10 | — | baf24gr120 | 1,377 | 10 | — | 261rl1304 | 150,468 | 6 | 0.06 |
| 41gr202 | 23,301 | 10 | — | 50pcb1173 | 9,385 | 10 | — | baf25pr124 | 10,745 | 10 | — | 265rl1323 | 154,023 | 0 | 0.12 |
| 45ts225 | 68,340 | 10 | — | 50pcb442 | 14,430 | 10 | — | baf26bier127 | 11,740 | 10 | — | 276nrw1379 | 20,050 | 2 | 0.46 |
| 45tsp225 | 1,612 | 10 | — | 50pr1002 | 54,583 | 10 | — | baf28pr136 | 17,824 | 10 | — | 280fl1400 | 15,316 | 10 | — |
| 46gr229 | 71,972 | 10 | — | 50pr439 | 45,253 | 10 | — | baf29pr144 | 14,070 | 10 | — | 287u1432 | 54,469 | 0 | 0.31 |
| 46pr226 | 64,007 | 10 | — | 50rat783 | 1,626 | 10 | — | baf30kroA150 | 7,005 | 10 | — | 316fl1577 | 14,182 | 10 | — |
| 53gil262 | 1,013 | 10 | — | 50rat99 | 814 | 10 | — | baf30kroB150 | 5,855 | 10 | — | 331d1655 | 29,443 | 0 | 0.33 |
| 53pr264 | 29,549 | 10 | — | 50vm1084 | 54,156 | 10 | — | baf31pr152 | 13,002 | 10 | — | 350vm1748 | 185,459 | 0 | 0.20 |
| 56a280 | 1,079 | 10 | — | 72vm1084-8x9 | 64,647 | 10 | — | baf32u159 | 7,301 | 10 | — | 364u1817 | 25,530 | 0 | 0.32 |
| 60pr299 | 22,615 | 10 | — | 75lin105 | 13,134 | 10 | — | baf39rat195 | 477 | 10 | — | 378rl1889 | 184,034 | 0 | 0.26 |
| 64lin318 | 20,765 | 10 | — | 81vm1084-9x9 | 69,659 | 10 | — | baf40d198 | 1,466 | 10 | — | 421d2103 | 40,049 | 0 | 0.62 |
| 80rd400 | 6,361 | 10 | — | 100i1000-410 | 5,481 | 10 | — | baf40kroA200 | 7,113 | 10 | — | 431u2152 | 27,614 | 0 | 0.74 |
| 84u417 | 9,651 | 10 | — | 100i1500-506 | 5,088 | 8 | 0.06 | baf40kroB200 | 7,126 | 10 | — | 464u2319 | 65,758 | 0 | 1.72 |
| 87gr431 | 101,946 | 10 | — | 100i2000-604 | 5,316 | 7 | 0.08 | baf41gr202 | 3,531 | 10 | — | 479pr2392 | 169,874 | 0 | 1.20 |
| 88pr439 | 60,099 | 10 | — | 100i2500-708 | 5,297 | 10 | — | baf45ts225 | 25,697 | 10 | — | 608pcb3038 | 52,416 | 0 | 1.72 |
| 89pcb442 | 21,657 | 10 | — | 100i3000-803 | 5,458 | 3 | 0.04 | baf46pr226 | 13,555 | 10 | — | 633C3k.0 | 10,255,031 | 0 | 0.92 |
| 99d493 | 20,023 | 10 | — | 100nrw1379 | 10,566 | 10 | — | baf53gil262 | 571 | 4 | 0.32 | 633E3k.0 | 16,197,552 | 0 | 1.71 |
| 107ali535 | 128,639 | 10 | — | 100pcb1173 | 13,901 | 4 | 0.20 | baf53pr264 | 7,716 | 10 | — | 759fl3795 | 18,662 | 0 | 0.35 |
| 107att532 | 13,464 | 10 | — | 100pr1002 | 74,269 | 9 | 0.00 | baf60pr299 | 10,047 | 10 | — | 893fnl4461 | 63,163 | 0 | 3.53 |
| 107si535 | 13,502 | 10 | — | 100prb1173-10x10 | 12,644 | 10 | — | baf64lin318 | 7,489 | 10 | — | Average for LARGE_LIB(I): | | **3.00** | **0.60** |
| 113pa561 | 1,038 | 10 | — | 100rat783-10x10 | 2,216 | 10 | — | baf80rd400 | 3,254 | 3 | 1.07 | | | | |
| 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 | 0.22 | baf88pr439 | 13,882 | 10 | — | 1183rl5915 | 309,243 | 0 | 2.46 |
| 132d657 | 22,498 | 10 | — | 144rat783-12x12 | 2,813 | 5 | 0.02 | baf89pcb442 | 8,749 | 10 | — | 1187rl5934 | 295,767 | 0 | 2.39 |
| 134gr666 | 163,028 | 9 | 0.05 | 150i1000-411 | 6,296 | 9 | 0.05 | baf99d493 | 3,081 | 10 | — | 1480pla7397 | 12,732,870 | 0 | 1.95 |
| 145u724 | 17,272 | 9 | 0.02 | 150i1500-507 | 6,085 | 10 | — | baf107att532 | 3,880 | 8 | 0.06 | 100C10k.0 | 6,158,999 | 0 | 1.17 |
| 157rat783 | 3,262 | 4 | 0.08 | 150i2000-605 | 5,940 | 9 | 0.00 | baf107si535 | 8,912 | 8 | 0.32 | 2000C10k.0 | 18,044,846 | 0 | 3.44 |
| 200dsj1000 | 9,187,884 | 8 | 0.02 | 150i2500-709 | 6,158 | 6 | 0.10 | baf113pa561 | 431 | 0 | 1.39 | 2000E10k.0 | 28,769,011 | 0 | 5.74 |
| 201pr1002 | 114,311 | 8 | 0.01 | 150i3000-804 | 6,551 | 0 | 0.41 | baf115rat575 | 1,330 | 9 | 0.15 | 2370rl11849 | 427,996 | 0 | 5.67 |
| 207si1032 | 22,306 | 0 | 0.06 | 150nrw1379 | 13,370 | 3 | 0.15 | baf131p654 | 5,824 | 4 | 0.03 | Average for LARGE_LIB(II): | | **0.00** | **3.26** |
| 212u1060 | 106,007 | 3 | 0.20 | 150pcb1173 | 17,082 | 3 | 0.35 | baf132d657 | 8,132 | 10 | — | | | | |
| 217vm1084 | 130,704 | 9 | 0.03 | 150pr1002 | 92,969 | 7 | 0.03 | baf145u724 | 7,354 | 1 | 0.56 | | | | |
| | | | | 150rat783 | 3,131 | 2 | 0.31 | baf157rat783 | 1,700 | 9 | 0.25 | | | | |
| | | | | 150vm1084 | 95,922 | 10 | — | baf201pr1002 | 48,400 | 0 | 2.40 | | | | |
| | | | | 200i2000-606 | 7,272 | 1 | 0.28 | baf207si1032 | 18,836 | 9 | 0.00 | | | | |
| | | | | 200i2500-710 | 7,191 | 4 | 0.15 | baf212u1060 | 38,639 | 5 | 0.16 | | | | |
| | | | | 200i3000-805 | 6,902 | 2 | 0.32 | baf217vm1084 | 44,681 | 10 | — | | | | |
| Average | | **9.20** | **0.01** | | | **7.84** | **0.06** | | | **8.67** | **0.15** | | | **2.36** | **1.16** |

**Table 8**
Indicators for the use of the Balas-Simonetti neighborhood $\mathcal{N}_8^{BS}$ in the refined ILS.

| | GTSP_LIB | | | LARGE_LIB (I) | | | LARGE_LIB (II) | | | LARGE_LIB Total | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #calls | success (%) | time (%) | #calls | success (%) | time (%) | #calls | success (%) | time (%) | #calls | success (%) | time (%) |
| min | 0 | 0 | 0 | 8.90 | 2.05 | 0.06 | 0 | 0 | 0 | 0 | 0 | 0 |
| avg | 24.34 | 17.58 | 0.31 | 137.08 | 13.57 | 1.34 | 64.99 | 3.96 | 1.53 | 124.40 | 11.72 | 1.43 |
| max | 270.90 | 36.36 | 4.12 | 594.80 | 27.20 | 7.58 | 212.50 | 10.10 | 4.79 | 594.80 | 27.20 | 7.58 |

**Table 9**
Comparison of the Refined ILS with and without the Balas-Simonetti neighborhood for high-quality solutions.

| | GTSP_LIB | | LARGE_LIB (I) | | LARGE_LIB (II) | | LARGE_LIB Total | |
|---|---|---|---|---|---|---|---|---|
| Algorithm | #best | Δ(%) | #best | Δ(%) | #best | Δ(%) | #best | Δ(%) |
| Refined ILS | 9.20 | 0.01 | 3.00 | 0.60 | 0.00 | 3.26 | 2.36 | 1.16 |
| Refined ILS w/o $\mathcal{N}_8^{BS}$ | 9.10 | 0.02 | 2.96 | 0.62 | 0.00 | 3.29 | 2.33 | 1.19 |

VND continues the current ILS iteration up to the point when a joint local minimum of all neighborhoods including $\mathcal{N}_8^{BS}$ is reached.

The 0 values for #calls occur for rather small instances with only a few vertices of GTSP_LIB as well as for very large instances of LARGE_LIB (II). In the latter case, the total number of ILS iterations is smaller than 200 (see Section 5.1), i.e., the exploration of the other neighborhoods is so time-consuming that the high-quality component is never invoked. Anyway, the average number of explorations is between 24 and 137 with an average success percentage between approximately 4 and 17 percent. The percentage of run time consumed by the $\mathcal{N}_8^{BS}$ neighborhood exploration is always below 8 percent, and only 0.31 and 1.43 percent on average for the libraries GTSP_LIB and LARGE_LIB, respectively.

The impact that the use of the $\mathcal{N}_8^{BS}$ neighborhood exploration has on solution quality is summarized in Table 9. The entries *#best* and Δ(%) have the same meaning as those of Table 6. The comparison between the refined ILS and the one without the component that fosters high-quality solutions (via $\mathcal{N}_8^{BS}$ neighborhood exploration) is consistent and in favor of the additional component: All *#best*-values are larger and all Δ(%)-values are smaller in the fully-equipped algorithm. The strongest result is that the $\mathcal{N}_8^{BS}$ neighborhood exploration halves the average Δ(%)-value for the GTSP_LIB.

*Comparison against GK, GLKH, and GLNS.* Finally, we compare the refined ILS with the basic ILS and the best-performing algorithms *GK*, *GLKH* and *GLNS* from the literature. Table 10 provides an overview with all entries as defined before for Table 6. The refined ILS clearly outperforms the basic ILS on GTSP_LIB and both parts of the 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(I), where clusters are generated so that they do *not* comprise sets of vertices that are mutually close. For the GTSP_LIB, however, *GLKH* and the refined

**Table 10**
Comparison of the Basic and Refined ILS with best-performing Algorithms from the Literature.

| | GTSP_LIB | | MOM_LIB | | BAF_LIB | | LARGE_LIB (I) | | LARGE_LIB (II) | | LARGE_LIB Total | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Algorithm | #best | Δ(%) | #best | Δ(%) | #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 | 0.00 | **2.56** | 2.39 | **0.97** |
| GLNS | 8.73 | **0.01** | **9.18** | **0.02** | **8.91** | **0.07** | **3.31** | **0.50** | 0.00 | 6.46 | **2.61** | 1.76 |
| Basic ILS | 8.88 | 0.02 | 7.84 | 0.06 | 8.67 | 0.15 | 2.50 | 1.22 | 0.00 | 5.49 | 1.97 | 2.12 |
| Refined ILS | **9.20** | **0.01** | 7.84 | 0.06 | 8.67 | 0.15 | 3.00 | 0.60 | 0.00 | 3.26 | 2.36 | 1.16 |
| Rank | 1 | 1 | 3 | 3 | 2 | 2 | 3 | 3 | | 2 | 3 | 2 |

ILS are best and on par regarding #best and Δ(%). No algorithm finds a BKS on the LARGE_LIB(II), however, *GLKH* is best regarding Δ(%), followed by the refined ILS. It seems that the refined ILS can cope well with instances where the clusters consist of relatively close vertices.

## 6. Conclusions

The paper has introduced an 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 (GTSP-tailored Balas-Simonetti neighborhood, adapted Gutin neighborhood, and String Relocation+) that allow simultaneous modifications of the sequence of the clusters and the selection of vertices per cluster. Effective neighborhood exploration algorithms for these neighborhoods have been developed and analyzed. Unexpectedly, this straightforward 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. Moreover, when a strict time budget for the ILS is imposed, the cooling schedule for the record-to-record-based acceptance criterion should be adapted. Estimating the number of ILS iterations and choosing a properly aligned cooling schedule turned out to be simple but effective.

With these refinements, the computational results also show that the new GTSP neighborhoods contribute significantly to the success of the 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 (we have found one new BKS during experimentation with preliminary versions of the ILS, see Appendix). For the latter library and the largest instances therein, the refined ILS performs slightly better than the GLNS algorithm, which, in turn, outperforms all other algorithms from the literature on the MOM_LIB and BAF_LIB. Overall, there is no clear winner algorithm that outperforms all others on all the available libraries that comprise GTSP instances with rather different characteristics. Indeed, the algorithms GK, GLKH, GLNS, and our ILS follow completely different solution paradigms. An overall superior algorithm may require a clever combination of these paradigms. What we offer with our work is three new exponentially-sized neighborhoods (Balas-Simonetti, String Relocation+, and an adaptation of the Gutin neighborhood) that may contribute to the attempt to design an overall superior algorithm.

For the future, further accelerating neighborhood explorations with established techniques like granular search [51,45], don't look bits [22, p. 375], and dynamic sequential/radius search [24,11] as well as with new ideas to be worked out is a promising research path.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgement

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

## Appendix A

During experimentation with different ILS setups, we have found a new best-known solution (BKS) for the instance 287u1432 of the LARGE_LIB with cost of 54,433:

**G-tour:** (295 296 292 291 1000 1001 286 281 262 260 239 265 267 270 268 187 233 227 224 223 244 99 103 8 2 1 1425 1423 1414 1421 115 111 114 215 206 153 154 199 198 177 196 183 184 362 358 354 357 400 402 436 437 567 574 576 426 425 418 319 416 412 322 325 329 330 331 303 302 311 312 788 783 782 585 586 779 775 771 768 805 808 802 801 799 909 915 921 923 895 893 867 868 891 888 882 945 949 952 956 957 954 876 877 879 870 854 853 849 843 832 830 829 824 823 821 859 818 816 752 758 759 747 739 740 734 728 723 717 716 710 702 700 698 687 672 671 666 658 656 652 646 650 632 628 627 623 592 596 604 610 547 540 526 521 514 512 509 508 504 499 485 486 487 488 441 455 458 459 467 391 392 381 380 373 367 371 370 168 164 161 160 148 146 141 137 124 126 130 1391 1388 1378 1377 1374 1381 1382 1364 1353 1335 1333 1316 1329 1328 1326 1323 1321 1309 1302 1299 1283 1280 1278 1274 1273 1184 1254 1255 1244 1247 1235 1234 1223 1226 1220 1217 1213 1212 1211 1106 1105 1110 1095 1092 1086 1082 967 970 1078 976 977 1072 1069 1066 1064 1130 1121 1120 1200 1198 1195 1125 1137 1177 1173 1169 1168 67 68 48 37 39 31 28 25 89 88 90 248 83 82 254 256 1010 1019 1023 1024 1034 1148 1147 1041 1038 1045 987 986 939 938 937 929 926 295)

## References


[1] E. Balas, New classes of efficiently solvable generalized traveling salesman problems, Ann. Oper. Res. 86 (1999) 529–558.
[2] E. Balas, N. Simonetti, Linear time dynamic-programming algorithms for new classes of restricted TSPs: a computational study, INFORMS J. Comput. 13 (1) (2001) 56–75.
[3] D. Ben-Arieh, G. Gutin, M. Penn, A. Yeo, A. Zverovitch, Transformations of generalized ATSP into ATSP, Oper. Res. Lett. 31 (5) (2003) 357–365.
[4] J.J. Bentley, Fast algorithms for geometric traveling salesman problems, ORSA J. Comput. 4 (4) (1992) 387–411.
[5] B. Bontoux, Techniques hybrides de recherche exacte et approchée: application à des problèmes de transport, Ph.D. thesis, Université d'Avignon, Français, 2008.
[6] B. Bontoux, C. Artigues, D. Feillet, A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem, Comput. Oper. Res. 37 (11) (2010) 1844–1852.
[7] V. Cacchiani, A. Muritiba, M. Negreiros, P. Toth, A multistart heuristic for the equality generalized traveling salesman problem, Networks 57 (2011) 231–239.
[8] G. Dueck, New optimization heuristics, J. Comput. Phys. 104 (1) (1993) 86–92.
[9] D. Feillet, P. Dejax, M. Gendreau, C. Gueguen, An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems, Networks 44 (3) (2004) 216–229.
[10] M. Fischetti, J. Gonzáles, P. Toth, A branch-and-cut algorithm for the symmetric generalized traveling salesman problem, Oper. Res. 45 (3) (1997) 378–394.
[11] J.B. Gauthier, S. Irnich, 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, 2020.
[12] F. Glover, Finding a best traveling salesman 4-opt move in the same time as a best 2-opt move, J. Heuristics 2 (1996) 169–179.
[13] G. Gutin, D. Karapetyan, Generalized traveling salesman problem reduction algorithms, arXiv: 0804.0735v2, 2009.
[14] G. Gutin, D. Karapetyan, A memetic algorithm for the generalized traveling salesman problem, Nat. Comput. 9 (1) (2010) 47–60.
[15] G. Gutin, A. Yeo, A. Zverovitch, Exponential neighborhoods and domination analysis for the TSP, in: The Traveling Salesman Problem and Its Variations, Combinatorial Optimization, Springer US, 2007, pp. 223–256.
[16] G. Gutin, D. Karapetyan, N. Krasnogor, Memetic algorithm for the generalized asymmetric traveling salesman problem, in: Nature Inspired Cooperative Strategies for Optimization (NICSO 2007), Springer, Berlin, Heidelberg, 2008, pp. 199–210.
[17] P. Hansen, N. Mladenović, Variable neighborhood search: principles and applications, Eur. J. Oper. Res. 130 (3) (2001) 449–467.
[18] P. Hansen, N. Mladenović, Variable neighborhood search, in: E.K. Burke, G. Kendall (Eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer US, 2005, pp. 211–238, chapter 8.
[19] K. Helsgaun, An effective implementation of the Lin-Kernighan traveling salesman heuristic, Eur. J. Oper. Res. 126 (2000) 106–130.
[20] K. Helsgaun, Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun Algorithm, Math. Program. Comput. 7 (3) (2015) 269–287.
[21] A. Henry-Labordere, The record balancing problem: a dynamic programming solution of a generalized travelling salesman problem, RIRO B-2 (1969) 43–49.
[22] H.H. Hoos, T. Stützle, Stochastic Local Search, Morgan Kaufmann, Amsterdam, 2005.
[23] B. Hu, G.R. Raidl, Effective neighborhood structures for the generalized traveling salesman problem, in: Evolutionary Computation in Combinatorial Optimization, Springer, Berlin, Heidelberg, 2008, pp. 36–47.
[24] S. Irnich, B. Funke, T. Grünert, Sequential search and its application to vehicle-routing problems, Comput. Oper. Res. 33 (8) (2006) 2405–2429.
[25] D.S. Johnson, L.A. McGeoch, The traveling salesman problem: a case study in local optimization, in: E. Aarts, J. Lenstra (Eds.), Local Search in Combinatorial Optimization, Wiley, Chichester, 1997, pp. 215–310, chapter 8.
[26] D. Karapetyan, G. Gutin, Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem, Eur. J. Oper. Res. 208 (3) (2011) 221–232.

[27] D. Karapetyan, G. Gutin, Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem, Eur. J. Oper. Res. 219 (2) (2012) 234–251.
[28] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, in: Readings in Computer Vision, Elsevier, 1987, pp. 606–615.
[29] M.E. Krari, B. Ahiod, Y.B.E. Benani, A pre-processing reduction method for the generalized travelling salesman problem, Oper. Res. 21 (4) (2019) 2543–2591.
[30] M.E. Krari, B. Ahiod, B.E. Benani, A memetic algorithm based on breakout local search for the generalized traveling salesman problem, Appl. Artif. Intell. 34 (7) (2020) 537–549.
[31] G. Laporte, F. Semet, Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem, Inf. Syst. Oper. Res. 37 (2) (1999) 114–120.
[32] G. Laporte, A. Asef-Vaziri, C. Sriskandarajah, Some applications of the generalized travelling salesman problem, J. Oper. Res. Soc. 47 (12) (1996) 1461–1467.
[33] S. Lin, Computer solutions of the traveling salesman problem, Bell Syst. Tech. J. 44 (1965) 2245–2269.
[34] S. Lin, B.W. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Oper. Res. 21 (2) (1973) 498–516.
[35] H.R. Lourenço, O.C. Martin, T. Stützle, Iterated local search, in: F. Glover, G.A. Kochenberger (Eds.), Handbook of Metaheuristics, Springer, 2003, pp. 320–353.
[36] M. Mestria, L.S. Ochi, S. de Lima Martins, GRASP with path relinking for the symmetric Euclidean clustered traveling salesman problem, Comput. Oper. Res. 40 (12) (2013) 3218–3229.
[37] C.E. Noon, The generalized traveling salesman problem, Ph.D. thesis, University of Michigan, 1988.
[38] C.E. Noon, J.C. Bean, A Lagrangian based approach for the asymmetric generalized traveling salesman problem, Oper. Res. 39 (4) (1991) 623–632.
[39] C.E. Noon, J.C. Bean, An efficient transformation of the generalized traveling salesman problem, Inf. Syst. Oper. Res. 31 (1) (1993) 39–44.
[40] C.M. Pintea, P.C. Pop, C. Chira, The generalized traveling salesman problem solved with ant algorithms, Complex Adapt. Syst. Model. 5 (1) (2017).
[41] M. Reihaneh, D. Karapetyan, An efficient hybrid ant colony system for the generalized traveling salesman problem, Algorithmic Oper. Res. 7 (2012) 22–29.
[42] X. Ren, X. Wang, Z. Wang, T. Wu, Parallel DNA algorithms of generalized traveling salesman problem-based bioinspired computing model, Int. J. Comput. Intell. Syst. 14 (1) (2020) 228.
[43] J. Renaud, F.F. Boctor, An efficient composite heuristic for the symmetric generalized traveling salesman problem, Eur. J. Oper. Res. 108 (3) (1998) 571–584.
[44] J.P. Saksena, Mathematical model of scheduling clients through welfare agencies, J. Canad. Oper. Res. Soc. 8 (1970) 185–200.
[45] C. Schröder, J.B. Gauthier, T. Gschwind, M. Schneider, 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, 2020.
[46] J. Silberholz, B. Golden, The generalized traveling salesman problem: a new genetic algorithm approach, in: Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies, Springer US, 2007, pp. 165–181.
[47] S.L. Smith, F. Imeson, GLNS: an effective large neighborhood search heuristic for the generalized traveling salesman problem, Comput. Oper. Res. 87 (2017) 1–19.
[48] L. Snyder, M. Daskin, A random-key genetic algorithm for the generalized traveling salesman problem, Eur. J. Oper. Res. 174 (1) (2006) 38–53.
[49] A. Subramanian, L.M.A. Drummond, C. Bentes, L.S. Ochi, R. Farias, A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery, Comput. Oper. Res. 37 (11) (2010) 1899–1911.
[50] M.F. Tasgetiren, P.N. Suganthan, Q.Q. Pan, 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, 2007, pp. 158–167.
[51] P. Toth, D. Vigo, The granular tabu search and its application to the vehicle-routing problem, INFORMS J. Comput. 15 (4) (2003) 333–346.
[52] J. Yang, X. Shi, M. Marchese, Y. Liang, An ant colony optimization method for generalized TSP problem, Prog. Nat. Sci. 18 (11) (2008) 1417–1422.