Gutenberg School of Management and Economics

& Research Unit “Interdisciplinary Public Policy”

Discussion Paper Series

Inter-Depot Moves and Dynamic-Radius Search for Multi-Depot Vehicle Routing Problems

Jean Bertrand Gauthier, Stefan Irnich

February 25, 2020

Discussion paper number 2004

Johannes Gutenberg University Mainz
Gutenberg School of Management and Economics
Jakob-Welder-Weg 9
55128 Mainz
Germany
wiwi.uni-mainz.de

Contact Details:

Jean Bertrand Gauthier
Logistikmanagement
Johannes Gutenberg University Mainz
Jakob-Welder-Weg 9
55128 Mainz
Germany

jgauthie@uni-mainz.de

Stefan Irnich
Logistikmanagement
Johannes Gutenberg University Mainz
Jakob-Welder-Weg 9
55128 Mainz
Germany

irnich@uni-mainz.de

Inter-Depot Moves and Dynamic-Radius Search for Multi-Depot Vehicle Routing Problems

Jean Bertrand Gauthier*,[a], Stefan Irnich[a]

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

Abstract

Radius search is an effective neighborhood exploration technique for standard edge-exchange neighborhoods such as 2-opt, 2-opt*, swap, relocation, Or-opt, string exchange, etc. Up to now, it has only been used for vehicle routing problems with a homogeneous fleet and in the single-depot context. In this work, we extend dynamic-radius search to the multi-depot vehicle routing problem, in which 2-opt and 2-opt* moves may involve routes from different depots. To this end, we equip dynamic-radius search with a modified pruning criterion that still guarantees identifying a best-improving move, either intra-depot or inter-depot, with little additional computational effort. We experimentally confirm that substantial speedups of factors of 100 and more are observed compared to an also optimized implementation of lexicographic search, another effective neighborhood exploration technique using a feasibility-based pruning criterion. Moreover, the computational results show that depot swapping strongly favors heuristic solution quality, especially for multi-depot configurations where depots are not located close to each other.

Key words: Vehicle routing, Local search, Sequential search, Dynamic-radius search, Inter-depot

1. Introduction

This paper extends the realm of application of dynamic-radius search, an effective neighborhood exploration technique, to the *multi-depot vehicle routing problem* (MDVRP). The two fundamental neighborhoods, 2-opt and 2-opt*, are extended to the multi-depot environment so that inter-depot moves of the affected routes are explicitly considered. Since these moves have been studied before by Escobar et al. (2014) only in a limited fashion, our contributions are threefold. First, we formalize the various inter-depot cases that arise in 2-opt and 2-opt*. Second, we show how to equip dynamic-radius search with a modified pruning criterion so that best-improving moves which allow inter-depot edges can be found with little additional computational effort. This is indeed not a simple exercise and we would as such like to stress that *dynamic-radius search* is not a heuristic way to explore a neighborhood as it guarantees that improving moves are found so long as no local minimum is reached. Third, we prove with rigorous statistical tests that the incorporation of such moves is essential for solution quality.

Dynamic-radius search builds on several classical works on the symmetric *traveling salesman problem* (TSP). In the context of the TSP, Hoos and Stützle (2005) use the expression *fixed-radius search* to collectively describe the idea of Steiglitz and Weiner (1968) and numerous extensions such as Bentley (1992); Martin et al. (1992); Reinelt (1994); Johnson and McGeoch (1997). For each vertex i, the predecessor p_i and successor s_i in the current TSP tour must be known. Then, for finding improving 2-opt and 3-opt moves, the neighborhood exploration procedures first loop over all vertices i to determine a first deleted edge (p_i, i) or (i, s_i). The first inserted edge $e = (i, j)$, replacing the deleted edge, must now be shorter, i.e., $c_{ij} < c_{p_i,i}$

*Corresponding author.
Email addresses: jgauthie@uni-mainz.de (Jean Bertrand Gauthier), irnich@uni-mainz.de (Stefan Irnich)

or $c_{ij} < c_{i,s_i}$, respectively. This can be interpreted as choosing j as a neighbor of i within the known radius given by $c_{p_i,i}$ or c_{i,s_i}. For 3-opt, the second decision about the second edge to insert must again produce a positive accumulated gain. This selection criterion for inserted edges is know as the *gain criterion*. Lin and Kernighan (1973) always choose edges according to the gain criterion in their famous Lin-Kernighan variable-depth neighborhood. Irnich et al. (2006) have generalized this idea of using the gain criterion for the CVRP and to VRP-specific moves such as node/vertex relocation, Or-opt, string exchange, etc. Moreover, Irnich (2008b) has shown that these techniques can be further generalized to handle more constrained versions of the VRP such as the VRP with time windows, the periodic VRP, pickup-and-delivery problems, etc.

Irnich et al. (2006) have introduced dynamic-radius search for VRPs under the name *sequential search*. This naming was inspired by the fact that moves are decomposed into partial moves and complete moves result from *sequentially* selecting the constituting partial moves. Note that cyclic independent move decomposition as coined in Irnich et al. (2006) relies on the fact that all 2-opt and 3-opt moves can be characterized by a single alternating cycle of deleted and added edges (see Funke et al. 2005; for a deeper analysis of single alternating cycle TSP neighborhoods). However, the term sequential search might nowadays be confused with single-threaded algorithms opposed to parallel/multi-threaded algorithms. Moreover, we think that the term *radius* better captures the idea of pruning the search tree using the gain criterion whereas *dynamic* follows in the footstep of the aforementioned *fixed* version. Certainly not taken lightly, we finally opt for this name change decision.

A formal definition of the MDVRP is in order. It can be defined over a complete undirected graph $G = (V, E)$ where the vertex set V consists of customers N and depots D. Each customer $i \in N$ has a positive demand q_i and we define $q_d = 0$ for every depot $d \in D$. Moreover, a fleet of m homogeneous vehicles characterized by a capacity Q and a maximal tour duration T (one of these limits may be omitted) are stationed at each depot $d \in D$. A global fleet-size limit $\kappa \leq m\,|D|$ on the number of vehicles may also be given. The edge set E models direct connections between customers as well as depots and customers. For each edge $e \in E$, we assume routing costs c_e and a travel time t_e. A route for depot $d \in D$ is a directed cycle $r = (d = i_0, i_1, i_2, \ldots, i_p, i_{p+1} = d)$ in which all vertices $i_1, i_2, \ldots, i_p \in N$ are different customers. A route r is feasible if it is capacity feasible, i.e., $\sum_{i \in V(r)} q_i \leq Q$, and duration feasible, i.e., $\sum_{e \in E(r)} t_e \leq T$, where $V(r)$ denotes the set of vertices and $E(r)$ the set of edges in r. The cost of a route r is $c_r = \sum_{e \in E(r)} c_e$. Let R_d denote the subset of all feasible, non-empty routes associated with depot $d \in D$. A set R of feasible routes is a solution to the MDVRP if (1) the local and global fleet-size limits are not exceeded, i.e., $|R \cap R_d| \leq m$ for all $d \in D$ and $\sum_{d \in D} |R \cap R_d| \leq \kappa$, and (2) all customers are serviced exactly once, i.e., $N = \bigcup_{r \in R} (V(r) \cap N)$ and $V(r_1) \cap V(r_2) \cap N = \emptyset$ for all $r_1 \neq r_2 \in R$. Such a solution is optimal if it minimizes the *(total) routing costs* $c(R) = \sum_{r \in R} c_r$.

Our overall experimental setup uses a rather simple multi-start neighborhood-based local search approach detailed in Section 5. We sketch it now in order to specify precisely the terminology. A neighborhood (e.g., 2-opt) uses moves to map a solution to alternative solutions called *neighbors*. We assume that the available neighborhoods are given by a set Ψ. Hence, for a given neighborhood $\mathcal{N} \in \Psi$ and current solution R, a *move* $\mu \in \mathcal{N}$ produces from R the neighbor solution $R' = \mu(R)$. Its marginal *gain* is defined as $g(\mu, R) = c(R) - c(\mu(R))$ (or simply g when contextually clear). The move is improving if $g > 0$, and a solution is a *local optimum* w.r.t. $\mathcal{N}$ if no improving move $\mu \in \mathcal{N}$ exists. *Neighborhood exploration* is the systematic search for an improving move and associated improving solution. We assume here that the exploration filters out infeasible solutions. Moreover, the *pivoting rule* (such as first improvement or best improvement) controls whether and (if so) when neighborhood exploration is stopped before all possible moves are considered. Finally, each constructed starting solution R is improved by iterative neighborhood explorations. One of the possible neighborhood exploration technique is dynamic-radius search and we discuss its relationship to other techniques in Section 3.

We combine the neighborhoods $\mathcal{N} \in \Psi$ in a *variable neighborhood descent* (VND) fashion, i.e., the neighborhoods $\mathcal{N} \in \Psi$ are parameterized and the choice of a next neighborhood to explore is determined according to a priority parameter of each neighborhood. For each starting solution R, we finally settle on a heuristic solution $\tilde{R}$ which is locally optimal with respect to all $\mathcal{N} \in \Psi$. This solution $\tilde{R}$ depends on

the starting solution R, the available Ψ, and for each neighborhood $\mathcal{N} \in \Psi$, its prioritization in VND, its pivoting rule, and its neighborhood exploration strategy. We study the trade-off between solution quality and the time it takes to find local optima $\tilde{R}$.

Our computational experiments presented in Section 6 show that this rather simple metaheuristic approach is competitive regarding solution quality and computation time compared to the state-of-the art fully-fledged metaheuristics for the MDVRP.

The remainder of this paper is structured as follows: In Section 2, we briefly recall 2-opt and 2-opt* moves for VRPs distinguishing intra/inter-tour moves as well as intra/inter-depot moves. Section 3 discusses neighborhood exploration techniques. The new radius search algorithm for efficiently exploring the extended neighborhoods with inter-depot 2-opt and 2-opt* moves is presented in Section 4. The multi-start neighborhood-based local search approach that we use as a simple metaheuristic is explained in Section 5. Computational results follow in Section 6. Final conclusions are drawn in Section 7.

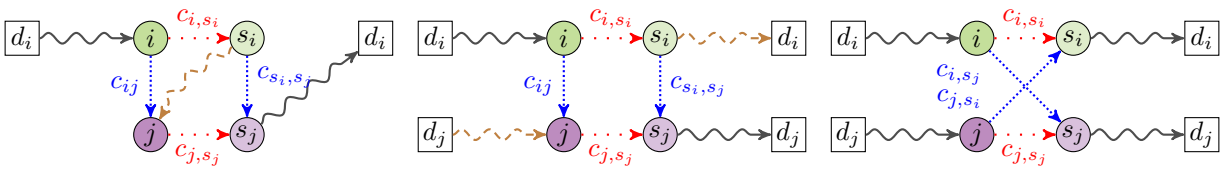
2. 2-Opt and 2-Opt* Moves in Multi-Depot Vehicle Routing Problems

Notation. A convenient representation of a solution R is obtained by concatenating all routes, in any order and orientation, as one long sequence of vertices. This is known as the *giant route* or *giant tour* representation (Bellmore and Hong 1974). In the following, the resulting sequence, denoted $\mathcal{V}$, allows us to loop over relevant vertices by writing $i \in \mathcal{V}$. In order to have a unique predecessor p_i and a unique successor s_i for each vertex $i \in \mathcal{V}$, the sequence $\mathcal{V}$ must include different *copies* of the depots, two for each route in order to also distinguish between its source and sink depots (which must represent the identical physical depot). For any route $r \in R$, let the first and last visited customers be denoted by f_r and $l_r \in N$, respectively. Moreover, a reference to the original depot is given by $d_r \in D$. In particular, any two routes r and r' are associated with the same depot if and only if $d_r = d_{r'}$. For any vertex $i \in V$, its *associated route* is denoted by $r_i \in R$. Finally, we define the corresponding depot d_i of vertex i as d_{r_i}, the first customer f_i as f_{r_i}, and the last customer l_i as l_{r_i}.

Legend. A consequence of having chosen an orientation for each route is that we can depict and write solutions with directed arcs instead of undirected edges. Indicating the direction of traversal makes reading and understanding the following figures easier. However, the underlying MDVRP is still assumed to be symmetric. In the following figures, a snake-shaped link between vertices w and v forms a directed path of arbitrary length (written as $w \rightsquigarrow v$ or $v \overset{\leftarrow}{\rightsquigarrow} w$ when it is an inversion of a given path) whereas a straight link indicates a single arc. A solid connection keeps its orientation once the move is completed while a dashed one is inverted as a consequence of the move. An arc is marked for deletion with a loosely dotted pattern while a densely dotted one indicates an insertion. Affected vertices are filled in solid color when they are chosen and a lighter shade when they are implied by a choice. This shading rule also applies when deleted and inserted arcs are implied to repair an otherwise infeasible move. Finally, customers take on circle-shaped vertices whereas different depots are explicitly distinguished by different polygon-shaped vertices (squares and pentagons).

Developed with the TSP in mind, Croes (1958) has devised an algorithm which performs so-called *inversions*. These are essentially 2-opt exchanges in the sense Lin (1965) has generally coined λ-opt. The 2-opt neighborhood has been generalized to single-depot vehicle routing problems in a straightforward manner. Potvin and Rousseau (1995) have introduced the 2-opt* neighborhood to tackle the vehicle routing problem with time windows (VRPTW, Desaulniers et al. 2014). The leading observation is that a 2-opt inter-tour move induces two segment inversions in the affected routes (see Figure 1b). A 2-opt* move differs from the latter in that it maintains the general ordering of the customers in the current solution and is thus more likely to produce an alternative solution feasible with respect to time windows (see Figure 1c).

The canonical description of both 2-opt and 2-opt* can be done with two deleted edges (i, s_i) and (j, s_j) as well as two inserted edges. In the 2-opt case, the inserted edges are (i, j) and (s_i, s_j) whereas in the 2-opt* case they are (i, s_j) and (j, s_i). The gain of these moves can then be computed for 2-opt as $g = c_{i,s_i} + c_{j,s_j} - c_{ij} - c_{s_i,s_j}$ and for 2-opt* as $g = c_{i,s_i} + c_{j,s_j} - c_{i,s_j} - c_{j,s_i}$. This description however



(a) **2-opt** intra-tour. (b) **2-opt** inter-tour, $d_i = d_j$. (c) **2-opt***, $d_i = d_j$.

Figure 1: Intra-depot 2-opt and 2-opt* moves.

only holds if we face an *intra-depot* move, that is, a move that affects a single route (*intra-tour*) or two routes (*inter-tour*) related to the same depot. Figure 1, which uses our multi-depot notation, therefore only captures intra-depot 2-opt and 2-opt* moves.

In the next two subsections, we show that multi-depot considerations can nevertheless be taken into account during the analysis of an *inter-depot* move. Observe that an inter-depot move must necessarily be inter-tour. In both neighborhoods, 2-opt and 2-opt*, we break down the possible cases that arise and must be covered by an exhaustive neighborhood exploration. In particular, an edge exchange infeasibly connecting two different depots can be repaired in either of two different ways. Our various visual aids give the final interpretation of the traditional neighborhood move together with a *repair operation* that must be performed to ensure each route has matching source and sink depots. Moreover, we show that all final move configurations fall under well-defined cases (standard, exception, and rejection, see below). The gain computation of the final move is of course affected as a byproduct of the repair operation. It is opportune at this point to separate the presentation of 2-opt and 2-opt*.

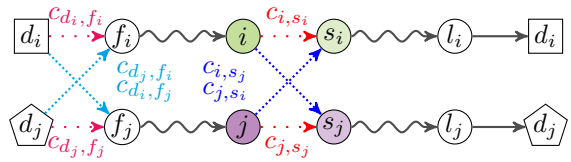
*2.1. Inter-Depot 2-Opt**

An inter-depot 2-opt* move happens when the two deleted arcs (i, s_i) and (j, s_j) belong to two different routes $r_i \neq r_j$ of two different depots $d_i \neq d_j$. As illustrated in Figure 2, eight cases are sufficient to exhaustively cover repair options. Figures 2a and 2b display the *standard case* where both deleted arcs (i, s_i) and (j, s_j) are either not the two first arcs or not the two last arcs in their respective routes r_i and r_j. This offers two possibilities of swapping the depots at this end, i.e., swapping at the source (Figure 2a) and at the sink (Figure 2b). Note that both cases allow the deleted arcs (i, s_i) and (j, s_j) to be at the opposite end of the two routes, so that the new tours are then $(d_i, f_j \rightsquigarrow l_j, d_i)$ and $(d_j, f_i \rightsquigarrow l_i, d_j)$. This is the special case of an exchange of two complete routes between two different depots.

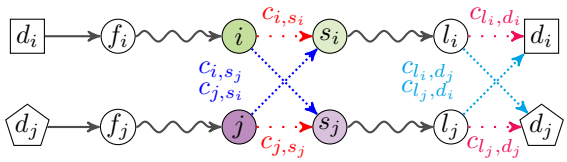
The next four *exception cases* depicted in Figures 2c–2f happen if exactly one of the deleted arcs is the first (last) and the other one is not the first (not the last). In the exception cases, only three arcs instead of four are finally exchanged, one of the arcs inserted in the standard case is absent (arc (i, s_j) or (j, s_i)). Note that the four cases result from the inherent symmetry, on the one hand between the two routes r_i and r_j (swapping the indices i and j), and on the other hand, between source and sink (reversal of the routes' orientation).

The first six cases shown in Figures 2a–2f cover all feasible inter-depot 2-opt* moves. There exist two more cases visualized in Figures 2g and 2h: (left) depot swap of the source depots with $i = d_i$ and $j = d_j$ and (right) depot swap of the sink depots with $s_i = d_i$ and $s_j = d_j$. These cases are however infeasible because the initially chosen arcs for deletion coincide with the two arcs that one wants to delete to perform the source (sink) depot swap.

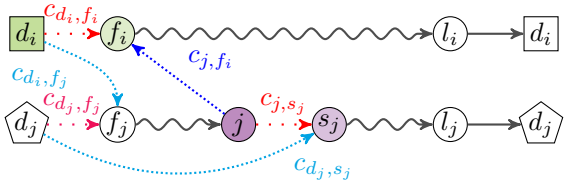
Ultimately, every 2-opt* inter-route move with different depots $d_i \neq d_j$ is evaluated with two repair operations (one source and one sink) based on the expressed conditions yielding up to two distinct inter-depot 2-opt* moves. Note that the conditions on i and j shown on the left-hand side of Figure 2 are independent from the conditions on s_i and s_j shown on the right-hand side. For example, if $i = d_i$ and $j \neq d_j$ holds (case 2c), exactly one of the four cases 2b, 2d, 2f, or 2h is true for s_i and s_j.



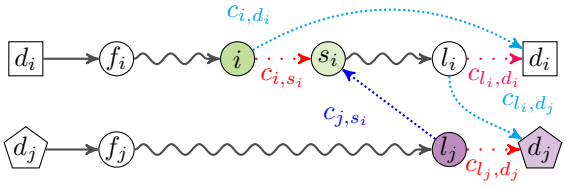
(a) Standard case: $i \neq d_i$, $j \neq d_j$.



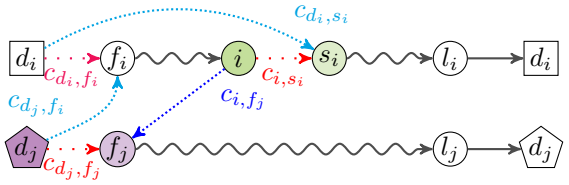
(b) Standard case: $s_i \neq d_i$, $s_j \neq d_j$



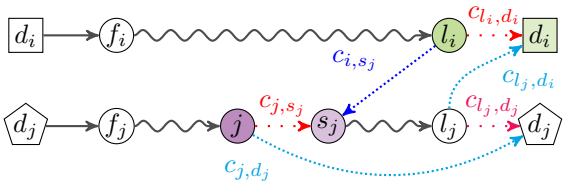
(c) Exception case: $i = d_i, j \neq d_j$, arc (i, s_j) is absent.



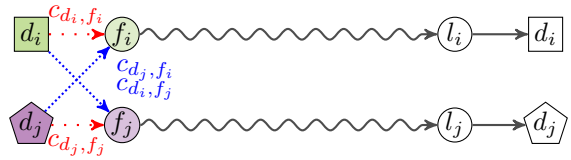
(d) Exception case: $s_i \neq d_i, s_j = d_j$, arc (i, s_j) is absent.



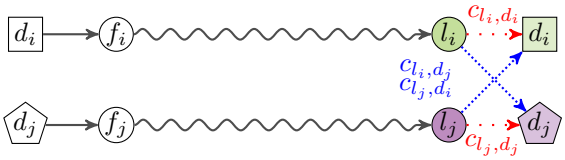
(e) Exception case: $i \neq d_i, j = d_j$, arc (j, s_i) is absent.



(f) Exception case: $s_i = d_i, s_j \neq d_j$, arc (j, s_i) is absent.



(g) Rejection case: $i = d_i, j = d_j$.



(h) Rejection case: $s_i = d_i, s_j = d_j$.

Figure 2: Inter-depot **2-opt*** moves, swap of source (left) or sink (right) depots.

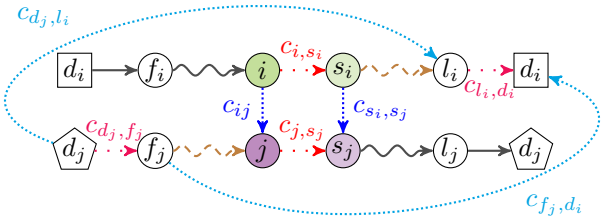
2.2. Inter-Depot 2-Opt

For the 2-opt neighborhood, we also raise standard and exception cases as presented in Figure 3, but omit rejection cases for the sake of brevity. The repair operation must always swap sink and source depots of the two different routes. As a mnemonic device, depots finally always remain attached to their current route. In the swap cases 3a, 3c, and 3e, the source/sink roles are preserved, whereas the roles are interchanged in the swap cases 3b, 3d, and 3f.

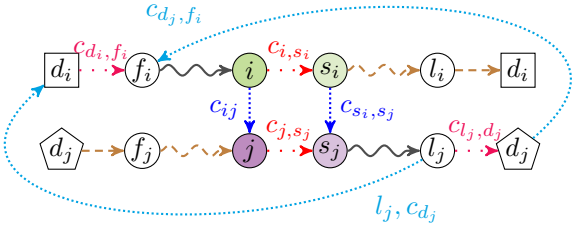
3. Neighborhood Exploration Techniques

In the following, we describe and compare the three fundamental neighborhood exploration techniques in local search: lexicographic search, radius search, and granular search. The organization follows this order and we complete these descriptions in Section 3.4 with a comparison and summarizing remarks.

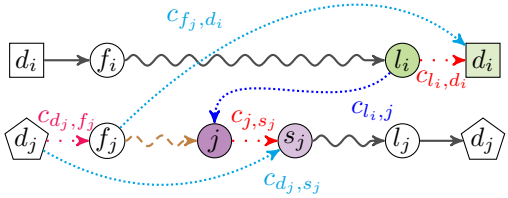
Regardless of the way a neighborhood is explored, the two bottleneck operations of testing moves are: the gain computation and the feasibility check. For the gain computation, it can be straightforward as is the case in the studied variant or involved, e.g., needed when computing route durations in the presence of time-dependent travel times as proposed by Visser and Spliet (2017). For the feasibility check, it is almost never a trivial task as per the inclusion of non-additive resource constraints. In accordance with Savelsbergh (1990); Irnich (2008b); Vidal et al. (2014b), several resource constraints can nevertheless be tested in *constant time*



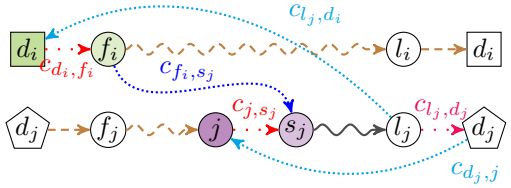
(a) Standard case: $s_i \neq d_i$, $j \neq d_j$.



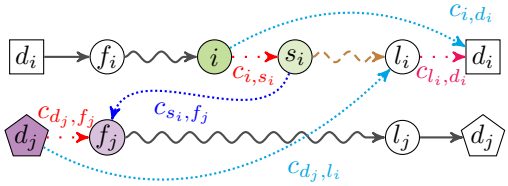
(b) Standard case: $i \neq d_i$, $s_j \neq d_j$.



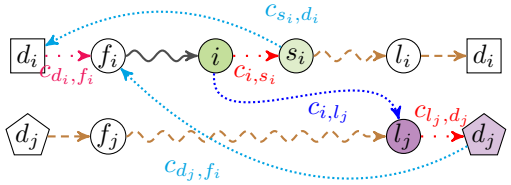
(c) Exception case: $s_i = d_i, j \neq d_j$, arc (s_i, s_j) is absent.



(d) Exception case: $i = d_i, s_j \neq d_j$, arc (i, j) is absent.



(e) Exception case: $s_i \neq d_i, j = d_j$, arc (i, j) is absent.



(f) Exception case: $i \neq d_i, s_j = d_j$, arc (s_i, s_j) is absent.

Figure 3: Inter-depot **2-opt** moves, source/sink depot roles are preserved (left) or interchanged (right).

by doing precomputations on the current solution and some segments of arcs (consecutive arcs in the current solution). For the latter computations, segments have to be built-up in a vertex-by-vertex fashion, leading to the lexicographic search paradigm. Constant time tests are well established for capacity, time-window, and pickup-and-delivery constraints. In the following, we use the 2-opt intra-tour move (see Figure 1a) to explain the different search paradigms. A synopsis of the three search principles is shown in Figure 4.

3.1. Lexicographic Search

The *lexicographic search* as presented by Savelsbergh (1990) consists in an elegant and systematic way to explore a neighborhood using the customer order observed in the current solution. It is especially intuitive for k-edge exchange moves. For exploring the 2-opt neighborhood, lexicographic search explores the vertices i and j using the order given by $\mathcal{V}$ in two nested loops, see Algorithm 1. The first loop iterates over $i \in \mathcal{V}$ whereas the second loop steps over $j > i \in \mathcal{V}$. Hence, the inner loop iterator is always greater than the preceding outer loop while still covering all possibilities for the 2-opt moves.

The key observation, see also Figure 1a, is that in the inner loop the new route of vertex i must contain the path $P = (d_i \rightsquigarrow i, j \overleftsquigarrow s_i)$, where the original orientation of path $(s_i \rightsquigarrow j)$ has been inverted. Since path P grows by one vertex in every inner-loop iteration, the idea of Savelsbergh (1990) is to prune the search based on the *local infeasibility* of P (see first if-condition in Algorithm 1), that is, *break* the inner loop if it can be shown that any further vertex in P always leads to a resource-infeasible move. Since this is a necessary but not sufficient condition for the feasibility of the overall move, a *global feasibility* check (see second if-condition in Algorithm 1) must be performed. Note that in an inter-tour move, the inner loop would *skip* the remaining vertices $s_j, \ldots, d_j$ of route r_j rather than *break*.

For capacity constraints, the local feasibility of P and any path constructed in later iterations of the inner loop amounts to checking $\sum_{i \in V(P)} q_i \leq Q$ (always fulfilled in the intra-tour case). Assuming that the

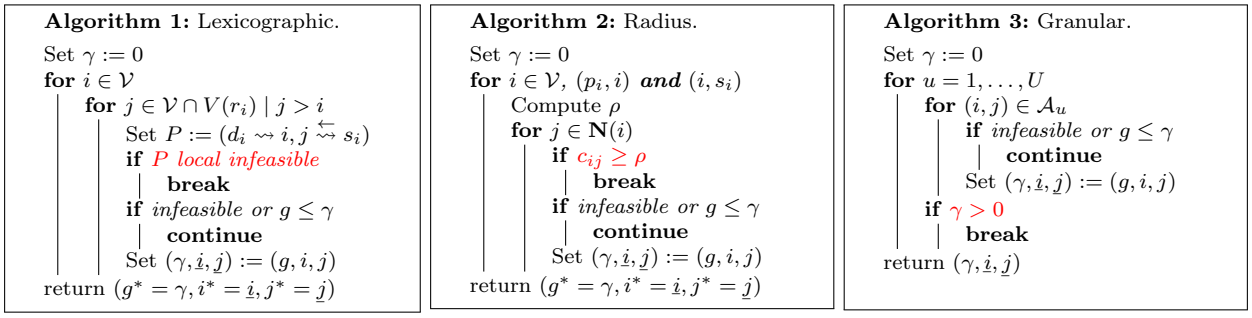


Figure 4: Synopsis of the three neighborhood exploration methods for the intra-depot 2-opt neighborhood. It is assumed that both the feasibility and gain of a move are evaluated directly before the if-condition "*infeasible or* $g \leq \gamma$" is reached.

triangle inequality holds for travel times, route duration constraints necessarily require $\sum_{e \in E(P)} t_e \leq T$. For the MDVRP, all this can be tested in $\mathcal{O}(1)$ by summing up demand and travel times for segment $(d_i \rightsquigarrow i)$ (outer loop over i) and segment $(j \overleftsquigarrow s_i)$ (inner loop over j).

More intricate feasibility conditions such as time windows, pairing, precedence constraints, and many more can be checked in $\mathcal{O}(1)$ as well. Recall that the 2-opt intra-tour move results in the new route $(d_i \rightsquigarrow i, j \overleftsquigarrow s_i, s_j \rightsquigarrow d_i) = (P, s_j \rightsquigarrow i)$. As a preparatory step, one must compute an upper bound on the resource consumption when arriving at the last segment $(s_j \rightsquigarrow d_i)$. This requires a $\mathcal{O}(n)$ preparation before the exploration is started. Then, the outer loop computes the resource consumption at the end of the first segment $(d_i \rightsquigarrow i)$, while the inner loop computes the *resource extension function* (Irnich 2008a) for the second segment $(j \overleftsquigarrow s_i)$ so that both the resources at the end of P and its local feasibility are determined in $\mathcal{O}(1)$. To check global feasibility, the latter resource values are then propagated along the arc (s_i, s_j) and compared against the respective resource upper bounds that were computed in the preparatory step.

Summarizing, the effectiveness of lexicographic search stems from its feasibility-based pruning. It is particularly well suited for a VRP with intricate or very constraining feasibility constraints. If both checking its resource consumption and propagating resource levels over entire segments can be done in constant time, there is no extra effort in the worst-case time complexity when exploring a neighborhood. For 2-opt, the result is a $\mathcal{O}(n^2)$ neighborhood exploration.

3.2. Radius Search

The idea of accelerating the neighborhood exploration based on the length of the inserted arc can done with *a priori* computed bounded candidate lists. This idea can be combined with pruning using the gain criterion. Finally, Irnich et al. (2006) sharpened the gain criterion by incorporating the quality of already detected improving solutions. Accordingly, we present candidate-lists based search (Section 3.2.1), fixed-radius search (Section 3.2.2), and dynamic-radius search (Section 3.2.3). In all variants, neighbor lists are preemptively sorted according to arc costs. For the special case of costs defined by Euclidian distances, one can imagine the search being conducted starting from the closest neighbor and spiraling outwards until some *break* condition is met. Note that this break condition is identical for 2-opt intra- and inter-tour moves so that implementations of radius search naturally consider both types of moves together.

3.2.1. Bounded Candidate-Lists based Search

Bounded candidate-lists based search follows the idea that *good arcs to be inserted should have a small cost*. It initially builds, for each vertex i, a bounded length *candidate list* $\mathbf{N}(i)$ of neighbor vertices j in close proximity of i. The neighbor $j \in \mathbf{N}(i)$ represents the arc (i, j), and only moves inserting the arc (i, j) with $j \in \mathbf{N}(i)$ are considered as possible moves. In a naive implementation, a fixed size σ for the neighborhoods is chosen first (e.g., 50 neighbors) and $\mathbf{N}(i)$ is then filled with the σ closest vertices.

In Algorithm 2, the first if-condition for breaking the inner loop is never fulfilled for a non-constraining radius such as $\rho = \infty$. A speedup solely results from the *bounded* candidate lists, because it reduces the number of arcs (i, j) to be tested. Indeed, for a fixed size σ the considered search space becomes linear in $|\mathcal{V}|$. Obviously, it is however not guaranteed that a true local optimum is found as long as $\sigma = |\mathbf{N}(i)| < |\delta(i)|$.

Another way to interpret bounded candidate-lists based search is to see it as a radius search, as depicted in Algorithm 2, where the radius ρ is not computed inside the outer loop. Instead, ρ is *a priori* chosen (for each i possibly in a different way) so that the inner for loop and if-condition can be implemented by filling $\mathbf{N}(i)$ appropriately.

For the TSP, bounded candidate lists have also been constructed on the basis of other criteria. Helsgaun (2000), for example, uses a modified edge weight/cost obtained from an approximation of the Held-Karp lower bound. For the tested instances, all edges of an optimal TSP solution were shown to be contained in candidate lists of smaller size σ. No general guarantee can be given for arbitrary instances. In contrast, the fixed-radius search presented next follows a different line of thought in order to achieve provably local optimal solutions.

3.2.2. Fixed-Radius Search

In routing problems, a move μ comprises the deletion of some arcs and the insertion of the same number of different arcs. It can therefore be decomposed into a number k of partial moves $p_1, \ldots, p_k$, i.e., $\mu = p_1 \circ p_2 \circ \cdots \circ p_k$, each of which deletes and inserts some of the arcs.

Definition 1. (Irnich et al. 2006) A move μ is respectively *cyclic-independent* and *cost-independent* if $\mu = p_{\pi(1)} \circ p_{\pi(2)} \circ \cdots \circ p_{\pi(k)}$ for all cyclic permutations of $\{1, 2, \ldots, k\}$ and $g(\mu) = \sum_{i=1}^{k} g(p_i)$, where $g(p_i)$ is a *partial gain* associated with p_i.

A neighborhood with moves that can be decomposed with respect to Definition 1 can be searched with the gain criterion in light of the following theorem:

Theorem 1. *(Lin and Kernighan 1973) If a sequence of k numbers $(g_i)_{i=\{1,\ldots,k\}}$ has a positive sum, i.e., $\sum_{i=1}^{k} g_i > 0$, then there exists a cyclic permutation π of these numbers such that every partial sum is positive, i.e., $\sum_{i=1}^{\ell} g_{\pi(i)} > 0$ for all $1 \leq \ell \leq k$.*

Note that neither Definition 1 nor Theorem 1 claim that for a given neighborhood the move decomposition is unique. Let us exemplify the gain criterion for intra-depot 2-opt moves as depicted in Figures 1a and 1b. First, an improving 2-opt move μ has a gain $g = g(\mu) = c_{i,s_i} - c_{ij} + c_{j,s_j} - c_{s_i,s_j} > 0$. Second, to satisfy Definition 1, one can decompose it into $\mu = p_1 \circ p_2 = p_2 \circ p_1$, where one partial move p_1 is deleting arc (i, s_i) plus inserting arc (i, j) and another partial move p_2 is deleting arc (j, s_j) plus inserting arc (s_i, s_j). From Theorem 1, we obtain that $g = g(p_1) + g(p_2) > 0$ is improving only if

$$g(p_1) = c_{i,s_i} - c_{ij} > 0 \quad \textbf{or} \quad g(p_2) = c_{j,s_j} - c_{s_i,s_j} > 0. \tag{1}$$

This is a *necessary* condition for move μ to be improving. By handling both options of the compound condition (1), we obtain independent *radius conditions* based on the cost of two different deleted arcs (i, s_i) or (j, s_j). This can be exploited algorithmically as done with the break condition in Algorithm 2 which makes fixed-radius search effective: given a vertex i, only those neighbors $j \in \mathbf{N}(i)$ which fulfill the first radius condition $c_{ij} < \rho$ with $\rho = c_{i,s_i}$ need to be inspected.

We have to underline that one must ensure that a move rejected from one partial move can be recovered when the other is analyzed. This can be easy to overlook when simplifying the loop design. Indeed, when testing the deleted arc (i, s_i) with $j \in \mathbf{N}(i)$ and later interchanging the roles of the vertices as (j, s_j) with $i \in \mathbf{N}(j)$, we can observe two facts: we may evaluate the same move twice, and we have never evaluated the second radius condition. The former point is a nuisance, but the latter point implies that this is an incomplete examination of the compound condition (1). Additionally testing over neighbors $s_j \in \mathbf{N}(s_i)$ such that $c_{s_j,s_i} < \rho$ however makes the search exhaustive. Indeed, one can verify that once again interchanging the roles of i and j yields a deleted arc (j, s_j) with $s_i \in \mathbf{N}(s_j)$. For the reader keeping count, we are

evaluating in the worst case the same move four times. Among these overlapping move evaluations, two are mandatory to ascertain a complete examination of the neighborhood space and two are redundant by cost symmetry. Since it is impossible to know in advance which of the two partial gains, if any, could fulfill its radius condition, the redundancy is in principle unavoidable by the conservative nature of the gain criterion. As supported by our computational results, we however claim that this redundancy is in practice more limited than what transpires by this quadruple factor (see also the example presented in the next section).

Finally, two equivalent nested loop constructions are possible: an outer loop $i \in \mathcal{V}$ followed by two inner loops $j \in \mathbf{N}(i)$ and $s_j \in \mathbf{N}(s_i)$, or alternatively an outer loop $i \in \mathcal{V}$ for which both deleted arcs (p_i, i) and (i, s_i) are tested followed by a single inner loop $j \in \mathbf{N}(i)$. We retain the latter presentation in Algorithm 2 for aesthetics reasons but advice the former for low-level efficiency. Indeed, looking at the finer details of the implementation, measurable speedup comes from tailoring neighbor lists to the specific inner loops $j \in \mathbf{N}(i)$ and $s_j \in \mathbf{N}(s_i)$ to include or exclude depot arcs, (i.e., arcs having a depot copy as an end vertex). Moreover, for 2-opt we see in Section 4.2 that the evaluation of the so-called multi-depot threshold can be simplified by using the same base term which only depends on the deleted arc (i, s_i).

At this point, we would also like to mention that the 2-opt move can also be decomposed in a different way. If the one partial move is deleting (i, s_i) and inserting (s_i, s_j), the other partial move is deleting (j, s_j) and inserting (s_j, i). Anyway, this decomposition is also asymmetric and therefore also requires the distinction of two cases.

Finally, fixed-radius search can use the complete candidate list $\mathbf{N}(i)$ for every vertex i, i.e., $\sigma = |V| - 1$ as long as computer memory permits the storage of $\mathcal{O}(n^2)$ elements. Complete candidate lists require a $\mathcal{O}(n^2 \log n)$ preprocessing, where each candidate list is sorted. Using complete candidate lists ensures that fixed-radius search terminates in a local optimum.

3.2.3. Dynamic-Radius Search

As first discussed by Irnich et al. (2006), the gain criterion can be sharpened if a lower bound γ on the best gain g^* is known, e.g., because an improving move has already been found. In this case, the search effort for further improving moves can be potentially lightened by reducing the search radius ρ. The theoretical foundation is the following corollary of Theorem 1.

Corollary 1. *(Irnich et al. 2006; p. 2411) If a sequence of k numbers $(g_i)_{i=\{1,\dots,k\}}$ has a sum greater than g, i.e., $\sum_{i=1}^{k} g_i > g$, then there exists a cyclic permutation π of these numbers for which every partial sum fulfills $\sum_{i=1}^{\ell} g_{\pi(i)} > \ell/k\, g$ for all $1 \le \ell \le k$.*

For a move μ with k partial moves, it means that at the first level the radius can be reduced from ρ^1 to $\rho^1 - \gamma/k$, and at the second level from ρ^2 to $\rho^2 - 2\gamma/k$, etc. For the 2-opt move and its decomposition discussed above, the gain criterion improving condition (1) becomes

$$c_{i,s_i} - c_{ij} > \gamma/2 \quad \textbf{or} \quad c_{j,s_j} - c_{s_i,s_j} > \gamma/2 \tag{2}$$

with the corresponding radius conditions

$$c_{ij} < \rho \quad \text{with} \quad \rho = c_{i,s_i} - \gamma/2 \tag{3a}$$

$$\text{and} \quad c_{s_i,s_j} < \rho \quad \text{with} \quad \rho = c_{j,s_j} - \gamma/2. \tag{3b}$$

Comparing (1) and (2) for the deleted arc (i, s_i), the initial radius, i.e., the one used in fixed-radius search, is $\rho^0 = c_{i,s_i}$ (the superscript 0 corresponds to $\gamma = 0$). The radius that results when the lower bound is exact, i.e., $\gamma = g^*$, is $\rho^* = c_{i,s_i} - g^*/2$. Depending on the previously found improving moves, the radius ρ actually used in inner loop of Algorithm 2 is between ρ^* and ρ^0. Therefore, it is always at least as sharp as fixed-radius search. In the following, we call a radius search that is based on the sharpened gain criterion a *dynamic-radius search.*

Figure 5 depicts part of an MDVRP solution in which we find an improving intra-depot 2-opt inter-tour move as seen in Figure 1b. It deletes arcs $(4, 15)$ and $(12, 44)$, inserts arcs $(4, 12)$ and $(15, 44)$, and inverts the segments $(15, 37, 17, d)$ and $(d, 12)$. Let us assume the deleted arc being tested is $(i, s_i) = (4, 15)$. The

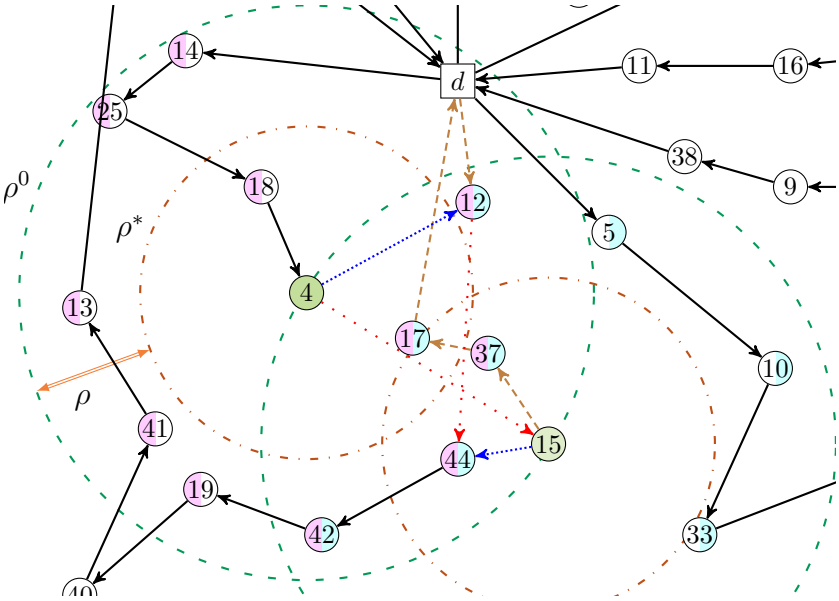


Figure 5: Dynamic-radius search for **2-opt** where the first deleted arc is $(4, 15)$, neighbors in $\mathbf{N}(4)$ and $\mathbf{N}(15)$ inspected with respect to threshold ρ^0 are highlighted. The selected move is an intra-depot (because $d_4 = d_{12} = d$) and inter-tour move. It deletes $(12, 44)$ and inserts $(4, 12)$ and $(15, 44)$. At any time of the search, we have $\rho^* \leq \rho \leq \rho^0$ anywhere within the double arrow.

concentric circles indicate which of the neighbors $\mathbf{N}(4)$ and $\mathbf{N}(15)$ must be respectively evaluated for the radii ρ^0 and ρ^*. The neighbors are color shaded in one or even two colors if they appear in the area of the circles with radius ρ^0. As $12 \in \mathbf{N}(4)$ and $44 \in \mathbf{N}(15)$ within radius ρ^0, the depicted 2-opt move is found twice in fixed-radius search, once with the inserted arc $(4, 12)$ as a neighbor of $i = 4$ and once more with the inserted arc $(15, 44)$ as a neighbor of $s_i = 15$. In fact, we evaluate it another two times when looking at deleted arc $(12, 44)$ and its cost $c_{12,44}$ used for the threshold. It is however likely that the two different deleted arcs have quite different costs which already gives a first explanation for a smaller observed redundancy than the aforementioned factor of four.

Dynamic-radius search can reduce the observed redundancy even further although it is not as direct. Imagine this particular move yields the best gain g^*, the threshold being used at any given time in the exploration is then anywhere in the interval $[\rho^*, \rho^0]$ depending on the value γ. It is tempting to look at the smaller overlapping area of the circles but this is mostly irrelevant since moves evaluated with vertices in this area are not comparable, e.g., inserted arcs $(4, 17)$ and $(15, 17)$ do not lead to the same move. The reduced observed redundancy simply comes from the smaller radius which potentially contains much less vertices to evaluate. In the most optimistic scenario, testing another deleted arc (i, s_i) with a larger cost $c_{i,s_i} \geq \gamma/2$, the threshold ρ becomes zero or even negative, thus implying that no neighbors must be analyzed at all. The consequence of this is that the reduced observed redundancy is very tangible although unpredictable as it depends on the loop construction and the observed gain.

We stress again the conservative nature of criterion (2) and take a look at when redundancy is maximal. For fixed-radius search, it occurs when $c_{i,s_i} = c_{j,s_j}$. A lot more conditions need to align for maximal redundancy in dynamic-radius search, that is, it occurs when $g^* = \varepsilon$ is rather small and the arc cost of all four arcs is almost identical, e.g., $c_{i,s_i} = c_{j,s_j} = c_{ij} = c_{s_i,s_j} + \epsilon$ which leads to $\rho^0 = \rho^* + \varepsilon/2$. These observations are independent of how one decomposes the move into partial moves or how one implements the inner loops.

Finally, we close with a geometric interpretation of dynamic-radius search and an open question for future research. Every unit reduction of the radius ρ reduces the area of the admissible neighbors quadratically by definition of a circle. Any radius $\rho > c_{15,44}$ would suffice for $44 \in \mathbf{N}(15)$ to qualify for the gain criterion.

Note that such a radius is smaller than ρ^* except when the deleted edges are of the same length. The question is therefore: *Is there a way to predict the smallest but sufficiently large radius ensuring that a move with maximum gain is identified?*

3.3. Granular Search

We briefly review *granular search* to clarify its relationship to radius search. Granular search is a neighborhood search and exploration technique that has been introduced via *granular tabu search* implementations for VRPs (Toth and Vigo 2003; Escobar et al. 2014; Schneider et al. 2017). The idea is an extension of bounded candidate-lists, as explained in Section 3.2.1, where arcs to be inserted are still ordered but now stored in a single global list, instead of one candidate list per vertex. This global list, denoted $\mathcal{A}$, is called the *generator-arc list*. Granular search explores the given neighborhood by considering only those moves where one specific inserted arc, the *generator arc*, is in $\mathcal{A}$. In case of the 2-opt neighborhood, the generator arc can be defined as the arc (i, j). Since (i, j) completely determines the 2-opt move, neighborhood exploration boils down to loop over $(i, j) \in \mathcal{A}$ and to implicitly construct and evaluate the feasibility and gain of the associated neighbor solution (the latter is possible in constant time for the MDVRP). Note that for a 2-opt move and the second inserted arc (s_i, s_j) it is *not* required that $(s_i, s_j) \in \mathcal{A}$ holds. In the same vein, redundancy occurs if both arcs (i, j) and (s_i, s_j) are present in $\mathcal{A}$.

Only if the generator-arc list comprises all feasible arcs can granular search guarantee that an improving move is found whenever one exists. Typically, $\mathcal{A}$ is a heavily truncated list so that granular search only explores heuristically. Moreover, in the tabu search context, for which granular search was invented, one is interested in a best but not necessarily improving move. A generator-arc list is well-suited for this task.

The *granularity* aspect of granular search comes from a partitioning of the generator-arc list, that is, $\mathcal{A} = \bigcup_{u=1}^{U} \mathcal{A}_u$, in which the sorted order of the arcs in $\mathcal{A}$ is also maintained. If no improving move is found with $\mathcal{A}_u$, the exploration is pursued in $\mathcal{A}_{u+1}$ for all $1 \leq u \leq U - 1$, see Algorithm 3.

The speedup of granular search also results from the increased flexibility of maintaining the generator arcs in any order. Indeed, the above mentioned implementations for VRPs exploit that a better selection of generator arcs (i.e., the choice of $\mathcal{A}$) and ordering of generator arcs (i.e., their assignment to $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_U$) often results when pseudo-costs are used instead of the given routing costs c_{ij}.

3.4. Comparison and Remarks

The synopsis of lexicographic, radius, and granular search provided by Figure 4 and the above discussion highlights the different ideas behind the neighborhood exploration techniques: Lexicographic search primarily prunes the search tree on the basis of local feasibility and is therefore well suited for strongly constrained VRPs. In contrast, radius search primarily prunes on the basis of the gain criterion and can be expected to be less effective for strongly constrained VRPs, because here many infeasible moves look promising from a gain's perspective. Later results will however show that for loosely constrained VRPs like the MDVRP, radius search typically outperforms lexicographic search. Granular search prunes on the basis of a heuristic preselection of generator arcs, which can either based on feasibility or cost criteria or a mix of both.

In all cases, we have the freedom to decide which comes first: the feasibility test or the gain computation. One should decide this by comparing the computational effort and effectiveness of both tasks.

We have presented lexicographic and radius search as *best improvement* exploration strategies. Even though both can be prematurely stopped when any improving and feasible solution has been found (this is *first improvement*), the idea of dynamic-radius search is to not stop but to explicitly exploit previously found improving solutions that lead to reduced search radii. By design, granular search follows a best improvement strategy per generator-arc list subset $\mathcal{A}_u$. First improvement would only really make sense if $U = 1$ in which case the initial sorting of the arcs is even more crucial.

4. Dynamic-Radius Search for Inter-Depot 2-Opt and 2-Opt* Moves

Using dynamic-radius search in the multi-depot environment, one must realize that the radii ρ^0 and ρ^*, as defined in (3) for the intra-depot 2-opt cases in Figures 1a and 1b, do not account for the additional

source/sink depot swap cost that may occur. Using these radii, we are no longer guaranteed to find the remaining improving moves, let alone provide the best gain available, unless we find a way to correctly consider this otherwise neglected cost prior to the radius breakpoint, that is, before testing neighbor vertices in an inner loop. For this purpose, we introduce a *correction term* τ on the standard threshold ρ which bounds from above the potential gain any source or sink depot swap can produce with respect to the current solution R and the explored neighborhood.

Let us further clarify the need for a correction term by presenting some facts regarding the inter-depot 2-opt* and 2-opt moves as broken down in the cases of Figures 2a–2f and 3a–3f. Note that we shorthand the latter expression to 'case xy' and even only use the index 'y' in mathematical formulas of a given neighborhood.

Table 1 summarizes all cases by listing their corresponding gains. The headers (standard, source, sink, and exception) refer, respectively, to move-defining, exchanged arcs of the *standard* cases 1c, 2a and 2b of 2-opt* and *standard* cases 1b, 3a and 3b of 2-opt, the arcs exchanged at the *source* and the *sink* to repair the otherwise infeasible depot assignment, and the added arcs of the *exception* cases that do not fall into one of the former categories. It is obvious that the gain computations differ significantly from one another and especially compared to the standard ones.

2-opt*	Fig.	Gain g from edges			
Move type		standard	source	sink	exception
Intra-depot	1c	$c_{i,s_i} - c_{j,s_i} + c_{j,s_j} - c_{i,s_j}$			
Inter-depot	2a	$c_{i,s_i} - c_{j,s_i} + c_{j,s_j} - c_{i,s_j}$	$+c_{d_i,f_i} - c_{d_j,f_i} + c_{d_j,f_j} - c_{d_i,f_j}$		
	2b	$c_{i,s_i} - c_{j,s_i} + c_{j,s_j} - c_{i,s_j}$		$+c_{l_i,d_i} - c_{l_i,d_j} + c_{l_j,d_j} - c_{l_j,d_i}$	
	2c	$-c_{j,s_i} + c_{j,s_j}$	$+c_{d_i,f_i} + c_{d_j,f_j} - c_{d_i,f_j}$		$-c_{d_j,s_j}$
	2d	$c_{i,s_i} - c_{j,s_i}$		$+c_{l_i,d_i} - c_{l_i,d_j} + c_{l_j,d_j}$	$-c_{i,d_i}$
	2e	$c_{i,s_i} - c_{i,s_j}$	$+c_{d_i,f_i} - c_{d_j,f_i} + c_{d_j,f_j}$		$-c_{d_i,s_i}$
	2f	$c_{i,s_i} + c_{j,s_j} - c_{i,s_j}$		$+c_{l_j,d_j} - c_{l_j,d_i}$	$-c_{j,d_j}$

(a) The seven different cases of **2-opt***.

2-opt	Fig.	Gain g from edges			
Move type		standard	source	sink	exception
Intra-tour	1a	$c_{i,s_i} + c_{j,s_j} - c_{ij} - c_{s_i,s_j}$			
Intra-depot	1b	$c_{i,s_i} + c_{j,s_j} - c_{ij} - c_{s_i,s_j}$			
Inter-depot	3a	$c_{i,s_i} + c_{j,s_j} - c_{ij} - c_{s_i,s_j}$	$+c_{d_j,f_j} - c_{d_j,l_i}$	$+c_{l_i,d_i} - c_{f_j,d_i}$	
	3b	$c_{i,s_i} + c_{j,s_j} - c_{ij} - c_{s_i,s_j}$	$+c_{d_i,f_i} - c_{l_j,d_i}$	$+c_{l_j,d_j} - c_{d_j,f_i}$	
	3c	$c_{i,s_i} + c_{j,s_j} - c_{ij}$	$+c_{d_j,f_j}$	$-c_{f_j,d_i}$	$-c_{d_j,s_j}$
	3d	$c_{i,s_i} + c_{j,s_j} - c_{s_i,s_j}$	$-c_{l_j,d_i}$	$+c_{l_j,d_j}$	$-c_{d_j,j}$
	3e	$c_{i,s_i} + c_{j,s_j} - c_{s_i,s_j}$	$-c_{d_j,l_i}$	$+c_{l_i,d_i}$	$-c_{i,d_i}$
	3f	$c_{i,s_i} + c_{j,s_j} - c_{ij}$	$+c_{d_i,f_i}$	$-c_{d_j,f_i}$	$-c_{s_i,d_i}$

(b) The eight different cases of **2-opt**.

Table 1: Gains of the **2-opt*** and **2-opt** moves.

In Table 2, the various cases that may arise are conditioned based on the known deleted arc (i, s_i) given by the outer loop and whose cost is central to the threshold computation. For each neighborhood, we have an *explicit* and an *implicit* column. This distinction is explained and utilized in the upcoming multi-depot threshold analysis. In order to correctly apply dynamic-radius search, we must match the multi-depot threshold with inner loops that are based on neighbor lists $\mathbf{N}(i)$ and $\mathbf{N}(s_i)$.

Finally, the symbol ρ is reserved for the threshold given by the standard cases displayed in Figure 1a–1c. The latter appears as a common term (and therefore the lower bound) in all our thresholds. Each case 'y' indeed gives rise to a local threshold $\rho + \tau_y$ where τ_y is a correction term. We make the multi-depot threshold clear by expressing it as $\rho_{MD} = \rho + \tau$, where τ is a case-dependent expression contributing to the

Conditions on i and s_i	2-opt* Figure 2		2-opt Figure 3	
	Explicit	Implicit	Explicit	Implicit
$i \neq d_i$ and $s_i \neq d_i$	a, b, d	e	a, b, e	f
$i = d_i$ and $s_i \neq d_i$	b, c, d		a, e	d
$i \neq d_i$ and $s_i = d_i$	a	e, f	b, c	f
$i = d_i$ and $s_i = d_i$	c	f	c	d

Table 2: Arising cases conditioned on the known deleted arc (i, s_i).

correction term. We now distinguish the 2-opt* and 2-opt neighborhoods.

4.1. 2-Opt Moves*

Recall that Figures 1c and 2a–2f display the seven cases to handle for the 2-opt* neighborhood. Following the same line of arguments as explained in Section 3.2.3 for 2-opt, the radius for the standard intra-depot 2-opt* case 1c is given by

$$c_{j,s_i} < \rho \quad \text{with} \quad \rho = c_{i,s_i} - \gamma/2 \tag{4a}$$

$$\text{and} \quad c_{i,s_j} < \rho \quad \text{with} \quad \rho = c_{j,s_j} - \gamma/2. \tag{4b}$$

Since dynamic-radius search exploits that the 2-opt* is completely symmetric with respect to i and j, there is a single inner loop say on $j \in \mathbf{N}(s_i)$ for the known deleted arc (i, s_i). Observe that in cases 2e and 2f, arc (j, s_i) is absent from the final move. This implies that its cost is irrelevant and cannot be subjected to a threshold. We sidestep this by observing that case 2c is symmetric in the affected routes to case 2e and likewise 2d to 2f. The idea is then that we must define a threshold in such a way that if an improving move with $g > \gamma$ exists in cases 2e or 2f, it is found by their symmetric counterpart. Our case-by-case analysis works as follows:

In the standard source case 2a, the arcs affected by the 2-opt* move are distinct from those needed to fix the depots. A new best gain is established by such a move if $g = [c_{i,s_i} - c_{j,s_i} + c_{d_i,f_i} - c_{d_j,f_i}] + [c_{j,s_j} - c_{i,s_j} + c_{d_j,f_j} - c_{d_i,f_j}] > \gamma$. As before, we break down the gain into two expressions according to the bracketed parts. This implies that $c_{j,s_i} < c_{i,s_i} + c_{d_i,f_i} - c_{d_j,f_i} - \gamma/2$. The point here is that the term c_{d_j,f_i} (depending on j) is unknown at the moment when (i, s_i) is deleted and the threshold ρ must be computed. We can however replace c_{d_j,f_i} by a lower bound over any depot reconnection yielding a radius large enough:

$$c_{j,s_i} < \rho + \tau_{\mathrm{a}} \qquad \text{with} \qquad \tau_{\mathrm{a}} = c_{d_i,f_i} - \min_{d \in D} c_{d,f_i}\,. \tag{5a}$$

We find the interpretation of this correction term quite elegant because it goes in line with intuitive expectations of the multi-depot environment: If customer f_i is already attached to the nearest depot, then the right side simplifies to the original ρ value given by (4a). Otherwise, the radius takes into account the potential for swapping source depots as the cost difference between the current depot assignment and one of the actual nearest depot.

In the standard sink case 2b, the gain can be decomposed into $g = [c_{i,s_i} - c_{j,s_i} + c_{l_i,d_i} - c_{l_i,d_j}] + [c_{j,s_j} - c_{i,s_j} + c_{l_j,d_j} - c_{l_j,d_i}]$ so that the resulting radius is given by

$$c_{j,s_i} < \rho + \tau_{\mathrm{b}} \qquad \text{with} \qquad \tau_{\mathrm{b}} = c_{l_i,d_i} - \min_{d \in D} c_{l_i,d}\,, \tag{5b}$$

where we note the similarity with the standard source case 2a that comes with exchanged roles of source and sink depot swap in the gain formula and the nearest-depot interpretation

In the exception case 2c, the gain is computed as $g = c_{i,s_i} - c_{j,s_i} + [c_{d_j,f_j} - c_{d_i,f_j}] + [c_{j,s_j} - c_{d_j,s_j}]$. Observe that arc (i, s_j) is omitted from the final move (see Table 1a), since it would otherwise be inserted

and removed upon swapping source depots. It turns out that we cannot reasonably decompose the gain's components so that the gain criterion can be applied. Moreover, remember that we must ensure a move from case 2e can be found anyway. The following threshold bounds from above the gain seen in case 2c thus fulfilling the latter wish:

$$c_{j,s_i} < \rho + \tau_\mathrm{c} \qquad \text{with} \qquad \tau_\mathrm{c} = \max_{r \in R} \left[c_{d_r,f_r} - c_{d_i,f_r} + \max_{j \in N(r)} \left(c_{j,s_j} - c_{d_r,s_j} \right) \right] - \gamma/2, \tag{5c}$$

where $N(r)$ is the set of customers in route r and we have a subtraction of $\gamma/2$. As we did not decompose the gain (into two independent parts) to test condition $g > \gamma$, the whole γ can be subtracted from the computed radius. With $-\gamma = -2 \cdot \gamma/2$ and the first half being already present in ρ (4), the validity of (5c) becomes clear. Fortunately, since the terms are consciously organized, intuition still answers the call. The first bracket describes whether there is some route r for which the first customer f_j would be closer to the source depot d_i than to the currently assigned source depot d_j, whereas the second bracket describes whether there is some route r for which short-cutting from d_j to s_j is favorable.

The exception case 2d slightly differs from the former case 2c because of the different arc costs that are unknown in j and known in i. The gain $g = c_{i,s_i} - c_{j,s_i} + [c_{l_i,d_i} - c_{i,d_i}] + [c_{l_j,d_j} - c_{l_i,d_j}]$ still offers no acceptable decomposition and we also want to cover case 2f. The resulting radius condition becomes

$$c_{j,s_i} < \rho + \tau_\mathrm{d} \qquad \text{with} \qquad \tau_\mathrm{d} = [c_{l_i,d_i} - c_{i,d_i}] + \max_{r \in R} \left(c_{l_r,d_r} - c_{l_i,d_r} \right) - \gamma/2. \tag{5d}$$

The difference compared to (5c) is that now there is no inner max-term over $j \in N(r)$.

In summary, the case-by-case analysis has led to four different case-dependent correction terms given by the equations (5a)–(5d). We still face the complication that the threshold must be computed when deleting arc (i, s_i) and before knowing which correction term to apply. However, we do know whether $i = d_i$ or $i \neq d_i$ as well as whether $s_i = d_i$ or $s_i \neq d_i$. Depending on these four possibilities, we can filter out which of the seven cases may happen (using Table 2). Accordingly, we define a final radius, tailored to the first deleted arc (i, s_i), as the maximum of the corresponding radii.

A careful examination of the conditioned cases in Table 2 allows the nodes i and s_i to be treated independently in the final formula. For example, the term τ_c occurs only when $i = d_i$ and the term τ_a only when $i \neq d_i$ which is indeed irrespective of s_i. Judiciously collecting all terms results in an elegant convoluted threshold expressed with respect to the various correction terms (5a)–(5d):

$$\rho_{MD} = \rho + \max \left[\begin{cases} \tau_\mathrm{c} & \text{if } i = d_i \\ \tau_\mathrm{a} & \text{if } i \neq d_i \end{cases} , \begin{cases} 0 & \text{if } s_i = d_i \\ \max\{\ \tau_\mathrm{b}\ ,\ \tau_\mathrm{d}\ \} & \text{if } s_i \neq d_i \end{cases} \right]. \tag{6}$$

We can summarize that this radius definition, which depends on the type of the first deleted arc (i, s_i), covers all cases of 2-opt*. The different correction terms added to ρ were precisely highlighted. The test $c_{s_i,j} < \rho_{MD}$ is clearly a relaxed radius condition compared to the standard case, but it allows us to exactly explore the 2-opt* neighborhood.

4.2. 2-Opt Moves

Deriving the correction term for 2-opt is slightly more intricate yet we find very similar expressions. Let us again be supported by the broken down cases as depicted in Figure 3 together with their respective gains in Table 1b. Exception cases 3c and 3f can indeed be merged into a one-sided test because the arc costs are symmetric (same for 3e and 3d). The following correction terms cover the relevant cases 3a, 3b, 3c, and 3e:

$$\tau_\mathrm{a} = c_{l_i,d_i} - \min_{d \in D} c_{d,l_i} \tag{7a}$$

$$\tau_\mathrm{b} = c_{d_i,f_i} - \min_{d \in D} c_{d,f_i} \tag{7b}$$

$$\tau_\mathrm{c} = \max_{r \in R} \left[(c_{d_r,f_r} - c_{f_r,d_i}) + \max_{j \in N(r)} \left(c_{j,s_j} - c_{d_r,s_j} \right) \right] - \gamma/2 \tag{7c}$$

$$\tau_\mathrm{e} = c_{l_i,d_i} - c_{i,d_i} + \max_{r \in R} \left(c_{d_r,f_r} - c_{d_r,l_i} \right) - \gamma/2 \tag{7e}$$

Using Table 2, the final radius is obtained by collecting the terms with respect to the deleted arc (i, s_i) and independently treating the vertices i and s_i similarly to 2-opt* as

$$\rho_{MD} = \rho + \max \left[\begin{cases} 0 & \text{if } i = d_i \\ \tau_{\mathrm{b}} & \text{if } i \neq d_i \end{cases} , \begin{cases} \tau_{\mathrm{c}} & \text{if } s_i = d_i \\ \max\{\ \tau_{\mathrm{a}}\ ,\ \tau_{\mathrm{e}}\ \} & \text{if } s_i \neq d_i \end{cases} \right]. \tag{8}$$

5. Iterated Local Search

In this section, we describe the algorithmic details of our metaheuristic. We have designed it with simplicity in mind so that local search is the fundamental building block. A good pick in this respect is *iterated local search* (ILS, Lourenço et al. 2002) as it combines local search with a *perturbation* mechanism. Local optima are permuted into new solutions so that local search can be applied repeatedly.

In our ILS, capacity constraints and duration constraints (if any) are handled as hard constraints such that feasibility of all routes is maintained starting from the construction heuristic to the perturbation and throughout the local search. The vehicle fleet-size limit however is construed as a soft constraint. It is relaxed in the construction procedure and in the perturbation. Our local search is able to ensure that a given solution does not degrade in its fleet-size feasibility. In order to reach solutions that are perfectly fleet-size feasible, we use a *fleet-reduction operation*.

The main complication of the perturbation step is indeed to construct solutions that are fleet-size feasible. In pretests, we found that for some instances (in particular those that do not have a distance constraint), constructing overall feasible solutions is rather simple. However, for some other instances, ensuring feasibility after perturbation is a delicate task. This explains why we had to design the ILS with a somewhat more involved perturbation mechanism.

We describe the construction heuristic in Section 5.1, the local search in Section 5.2, the perturbation and fleet-reduction operations in Section 5.3, and we provide an overview and pseudo-code of the entire ILS in Section 5.4.

5.1. Construction Heuristic

Our construction procedure is based on the *savings* heuristic of Clarke and Wright (1964). The core idea remains to process an arc list sorted decreasingly by their potential saving, but the multi-depot aspect and the additional duration constraints are accounted for as follows. First, we draw uniformly distributed parameters $\zeta \in [0, 2]$ and $\lambda \in [\zeta - 0.25, \zeta + 1.75]$. Then, we compute the saving of every arc (i, j) with respect to each depot $d \in D$ as $\sigma^d_{ij} = -\lambda c_{ij} + c_{di} + c_{jd} + \zeta(c_{di} - c_{jd})$, where λ and ζ influence the comparison between the cost of arc (i, j) and of connecting i and j to d. We of course reject those combinations where (d, i, j, d) is an infeasible route. To further randomize the procedure, for each arc (i, j), we take from these $|D|$ depot-specific savings values $(\sigma^d_{ij})_{d \in D}$ an arbitrary one that is non-negative, denoted σ_{ij}. Next, we sort these savings $(\sigma_{ij})_{(i,j)}$ decreasingly.

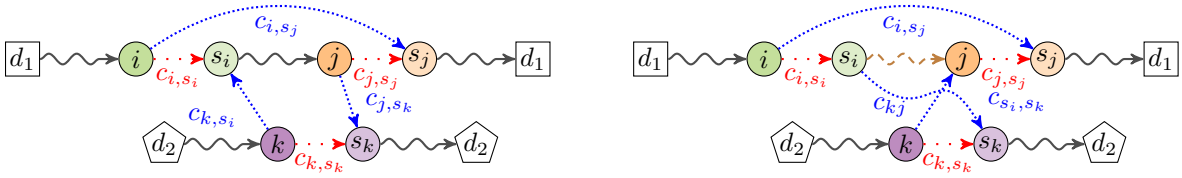
At the start, all customers $i \in N$ form separate segments (i). After computing and sorting the savings values $(\sigma_{ij})_{(i,j)}$, the main loop considers the associated arcs (i, j) one by one. If the vertices i and j are the last/first of their segments and the concatenation of their segments $(v \rightsquigarrow i)$ and $(j \rightsquigarrow w)$ gives a feasible route $(d, v \rightsquigarrow i, j \rightsquigarrow w, d)$ for some depot $d \in D$, we join the segments together. At the end, when no more segments can be joined, each segment is finally assigned to the depot d that leads to the cheapest feasible route. Note that this type of depot assignment may lead to a solution R that is infeasible regarding the fleet-size constraints. We accept slightly infeasible solutions R if $|R \cap R_d| \leq \delta m$, where δ is a parameter. We call solutions R that respect the relaxed fleet-size constraints *δ-fleet feasible* solutions.

The above procedure is repeated with $\delta = 1.5$ and new random parameters ζ and λ until the constructed solution R is δ-fleet feasible.

5.2. Local Search

A reasonable local search procedure must use additional neighborhoods besides 2-opt and 2-opt*. For the purpose of this study, we complement them with six other neighborhoods *relocation*, *swap*, *string exchange*

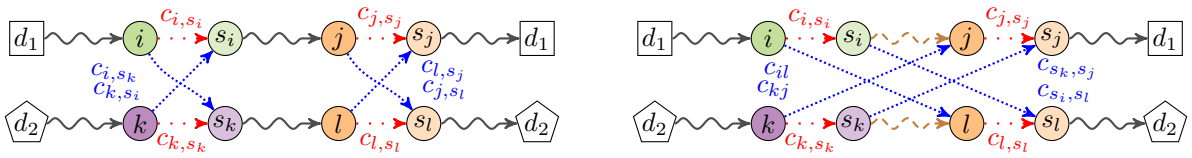
(ordered and inverted), and *Or-opt* (ordered and inverted) as commonly defined in Aarts and Lenstra (1997); Funke et al. (2005). Figures 6 and 7 describe the general composition of Or-opt and string-exchange moves in ordered and inverted variants. Both neighborhoods restrict the length of their relocated chains to a length parameter L. Note that a relocation move is an Or-opt move with $L = 1$, and likewise a swap move is a string exchange with $L = 1$. We nevertheless implemented independent relocation- and swap-neighborhood exploration algorithms to benefit from the specialization, because for relocation and swap the distinction between ordered and inverted chains is irrelevant. In the following, we use $L = 5$ for Or-opt and string exchange unless stated otherwise.



(a) Ordered chain.

(b) Inverted chain.

Figure 6: Inter-depot **Or-opt**, $|s_i \rightsquigarrow j| \leq L$.



(a) Ordered chains.

(b) Inverted chains.

Figure 7: Inter-depot **string exchange**, $|s_i \rightsquigarrow j| \leq L$ and $|s_k \rightsquigarrow l| \leq L$.

Relocation, swap, string exchange, and Or-opt are naturally compliant with the multi-depot environment because all these inter-depot moves result in routes that have matching source and sink depots.

Our local search is kept as simple as possible: All eight neighborhoods (we consider inverted and ordered Or-opt and string exchange as different neighborhoods) are explored in a cyclic fashion. The exploration is done with a best-improvement strategy. Whenever an improving move is returned from the neighborhood exploration, it is performed and we move to the next neighborhood. Local search terminates when all neighborhoods are explored without success, so that the solution R returned is always a local optimum with regards to all eight neighborhoods.

Note finally that δ-fleet feasibility can easily be maintained in the local search if the starting solution fulfills it. We must only use a feasible number of copies of the depots. More precisely, exactly $2\lfloor \delta m \rfloor$ copies are needed per depot $d \in D$ (two for each route, see Section 2).

5.3. Perturbation and Fleet-Reduction

A local optimum R is perturbed by a multi-phase re-clustering. We first permute the routes randomly in the giant tour and then apply a circular shift on a random position. From this new customer sequence, routes are filled in order while satisfying resource consumption. This process is repeated 2 to 5 times (uniformly random). If the re-clustering fails to produce a suitable customer assignment, i.e., a δ-fleet feasible solution, a new solution R is constructed with the construction heuristic.

The purpose of the fleet-reduction operation is to transform a given solution R that is δ-fleet feasible into one that is perfectly fleet-size feasible. As this is an NP-hard and sometimes practically difficult task, the fleet-reduction operation may terminate with a solution that is only partly improved regarding fleet-size feasibility. Note that in any case such an improvement comes at the cost of worsening the objective value.

The fleet-reduction operation tries to patch up the given solution by moving chains of customers from overused to underused depots and their routes. We reuse the exploration of 2-opt* and Or-opt neighborhoods to find a chain inside a route belonging to an overused depot that can be moved to another depot at minimal cost. 2-opt* and Or-opt moves are repeated until a fleet-size feasible solution is constructed or the search for such a feasible 2-opt* or Or-opt move fails. The modified solution is returned irrespective of its feasibility status.

5.4. Iterated Local Search

The general design is an ILS with a limit of n_{ILS} local search iterations as summarized by the pseudo-code in Algorithm 4. The current solution is denoted by R and it is initialized by the result of the savings heuristic in Step 1. The counter *count* (initialized at zero in Step 2) keeps track of the number of consecutive iterations for which the fleet-reduction operation fails to produce a fleet-size feasible solution.

Algorithm 4: Iterated Local Search (ILS)

```
   // Initialization
1  Savings(R)                                                    // Section 5.1
2  count := 0
   // Main Loop
3  for n_ILS iterations do
       // Local Search Phase
4      LocalSearch(R, δ = 1.25)                                  // Section 5.2
5      if not fleet-size feasible then
6          Fleet-reduction(R)                                    // Section 5.3
7          if fleet-size feasible then
8              LocalSearch(R, δ = 1.0)                           // Section 5.2
       // Perturbation
9      if not fleet-size feasible then
10         count := count + 1
11         if random() < 0.7^count then
12             Perturbation(R)                                   // Section 5.3
13         else
14             Savings(R)                                        // Section 5.1
15     else
16         count := 0
17         Perturbation(R)                                       // Section 5.3
```

We perform up to two passes in each local descent (Steps 4 and 8). In the first pass (Step 4), the limit on the number of vehicles is relaxed. When a local optimum is reached, if said limit is satisfied we move on to the perturbation operation. Otherwise, the fleet-reduction operation tries to make the solution fleet-size feasible (Step 6) and, if so, the second local search pass is performed, for which the strict fleet-size limit is imposed (Step 8).

The perturbation mechanism (Steps 9 to 17) uses the savings heuristic as a fallback whenever the actual perturbation procedure described in Section 5.3 fails to produce a δ-fleet feasible solution.

6. Computational Results

The implementation of the ILS algorithm is written in `C++` and compiled in 64-bit release mode under `Microsoft Visual Studio 2015`. The experiments are conducted on a `Microsoft Windows 10` standard personal computer equipped with an `Intel i7-6700` CPU clocked at 3.40 GHz and 16 GB of RAM. A single thread is allocated to each run.

Section 6.1 describes the benchmark instances used in this study. A comparison of the previous dynamic-radius search implementation follows in Section 6.2. The impact of the correction term is analyzed in

Section 6.3 by evaluating its contribution under various usage scenarios. Finally, Section 6.4 compares the results of our algorithm to best known solutions available in the literature.

6.1. Instances

We start our analysis of dynamic-radius neighborhood exploration techniques by reproducing a comparative assessment with lexicographic search using

- 560 CVRP instances [10 (seed) × 4 (density) × 14 (size)] from Irnich et al. (2006).

We then put the proposed ILS to test by tackling the following commonly used MDVRP instances from the literature:

- 33 MDVRP instances [p01–pr10] from Cordeau et al. (1997);
- 14 MDVRPTW instances [pr11a–pr24a with neglected time windows (TW)] from Vidal et al. (2013);
- 10 MDVRP instances [Belgium-] from De Smet et al. (2006).

Moreover, we decided to newly generate

- 1,120 large-scale MDVRP instances [10(depot configuration) × 4(density) × 14(size) × 2(R vs. RC)]

in which the multi-depot characteristics are systematically varied in order to have a sufficiently large benchmark allowing rigorous statistical tests. Such tests are hardly possible with the limited sets [pr-] and [Belgium-]. Note also that the [pr-] instances are well studied but relatively small (up to 360 customers), while the [Belgium-] instances are less studied but the largest instance comprises 2,750 customers. The CVRP instances from the previous work (Irnich et al. 2006) and the new ones are available at `https://logistik.bwl.uni-mainz.de/research/benchmarks/`.

6.2. Improved Implementation

We mention for the sake of scientific rigor that, not only have we reproduced the results of Irnich et al. (2006) for the single-depot (=CVRP) environment, we exceeded expectations. Dynamic-radius search has been re-implemented with a greater focus on low-level efficiency but we also gave due attention to lexicographic search. The reader may compare the previous results (Irnich et al. 2006; Figures 7 and 9) with Figures 8 and 9 referring to the 560 CVRP instances. Irnich et al. (2006) systematically compared lexicographic and radius search analyzing two indicators: the acceleration factor describing the average ratio of computing times needed with lexicographic compared to dynamic-radius search (ratios are dimensionless) as well as average neighborhood-exploration times (in milliseconds [ms]). To understand what is depicted, one must know that the effectiveness of (dynamic-)radius search for the CVRP strongly depends on how many customers a route contains on average, denoted as the *load factor*. Therefore, the results presented in Figures 8 and 9 are exactly grouped by both number $|N|$ of customers and load factor f.

Figure 8 shows the average acceleration factor. Comparing relative performance to lexicographic search with the 2-opt neighborhood (old factor 35 versus new factor 20) and the string-exchange neighborhood (900 versus 600) for the largest instances with $|N| = 2{,}500$ customers, it appears that the benefit of dynamic-radius search has slightly decreased. This is due to significant improvements in the lexicographic search implementation but also the fact that the increased maximum string length from $L = 3$ to 5 favors the lexicographic paradigm. Contrarily to the previous computational study in (Irnich et al. 2006), the trend lines also do not show any significant negative slope when increasing the number of customers. This is particularly striking on the string-exchange neighborhood and can be explained by the fact that we have incorporated feasibility pruning inside the string examinations of dynamic-radius search. Some neighborhoods such as relocation now also exploit the threshold whenever a new best gain is identified rather than only in the inner loop head.

Figure 9 depicts the average computation time for a single neighborhood exploration. We achieve a reduction in absolute computation time by at least a factor of 10 for every neighborhood including those with larger maximal string lengths L. We believe that technological progress since 2006 cannot single-handedly explain this improvement as evidenced by the average search times of the swap neighborhood (70 milliseconds [ms] versus 4 ms) and string-exchange neighborhood (350 ms versus 9 ms) for instances with $|N| = 2{,}500$ customers. We point here to the aforementioned transition to an explicit double inner-loop design which allows finer treatment of depot arcs, see Section 3.2.2.

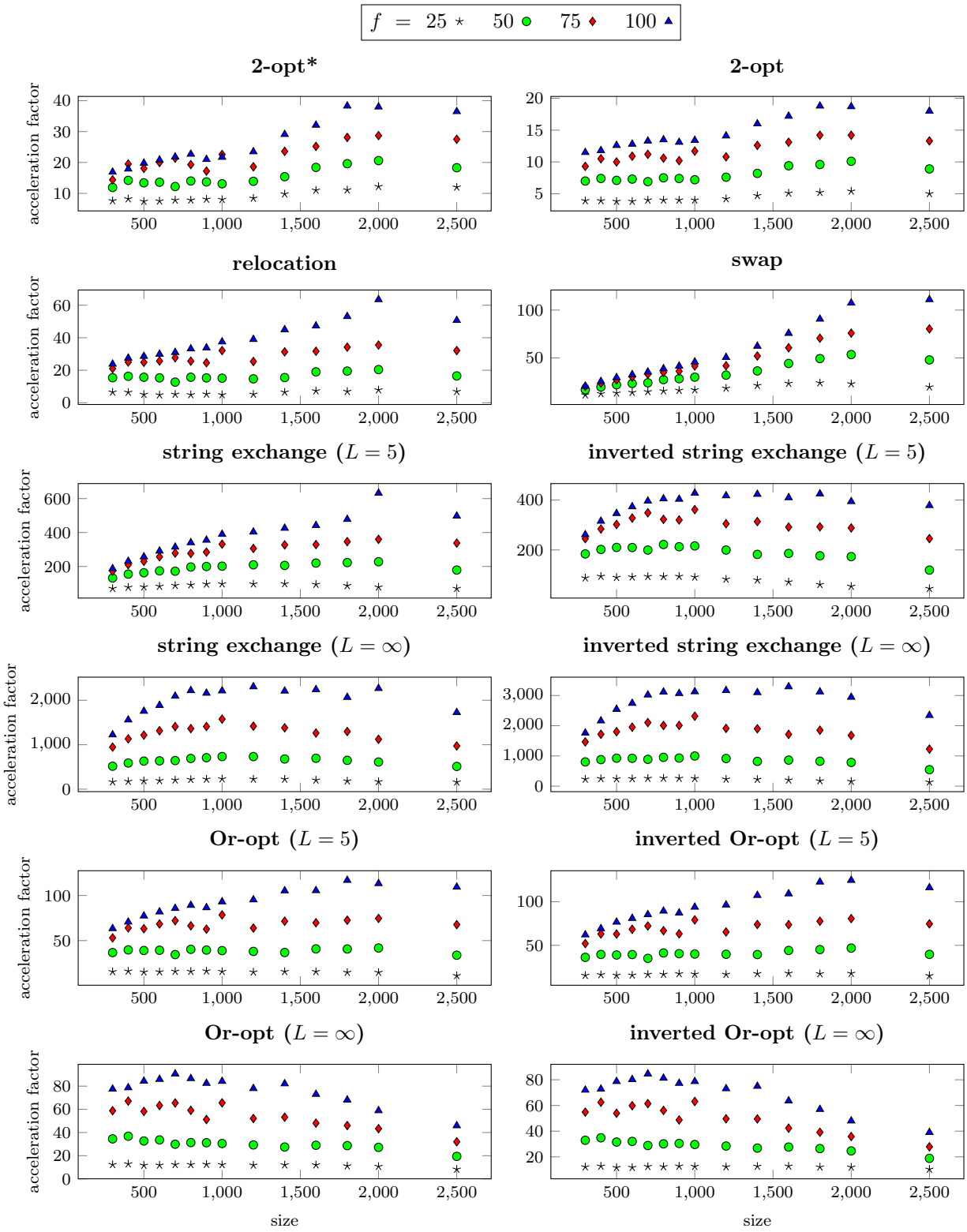


Figure 8: [CVRP, Irnich et al. (2006)] *Average acceleration factor* of dynamic-radius search over lexicographic search for various neighborhood operators and instances ranging from 300 to 2,500 in customer size.

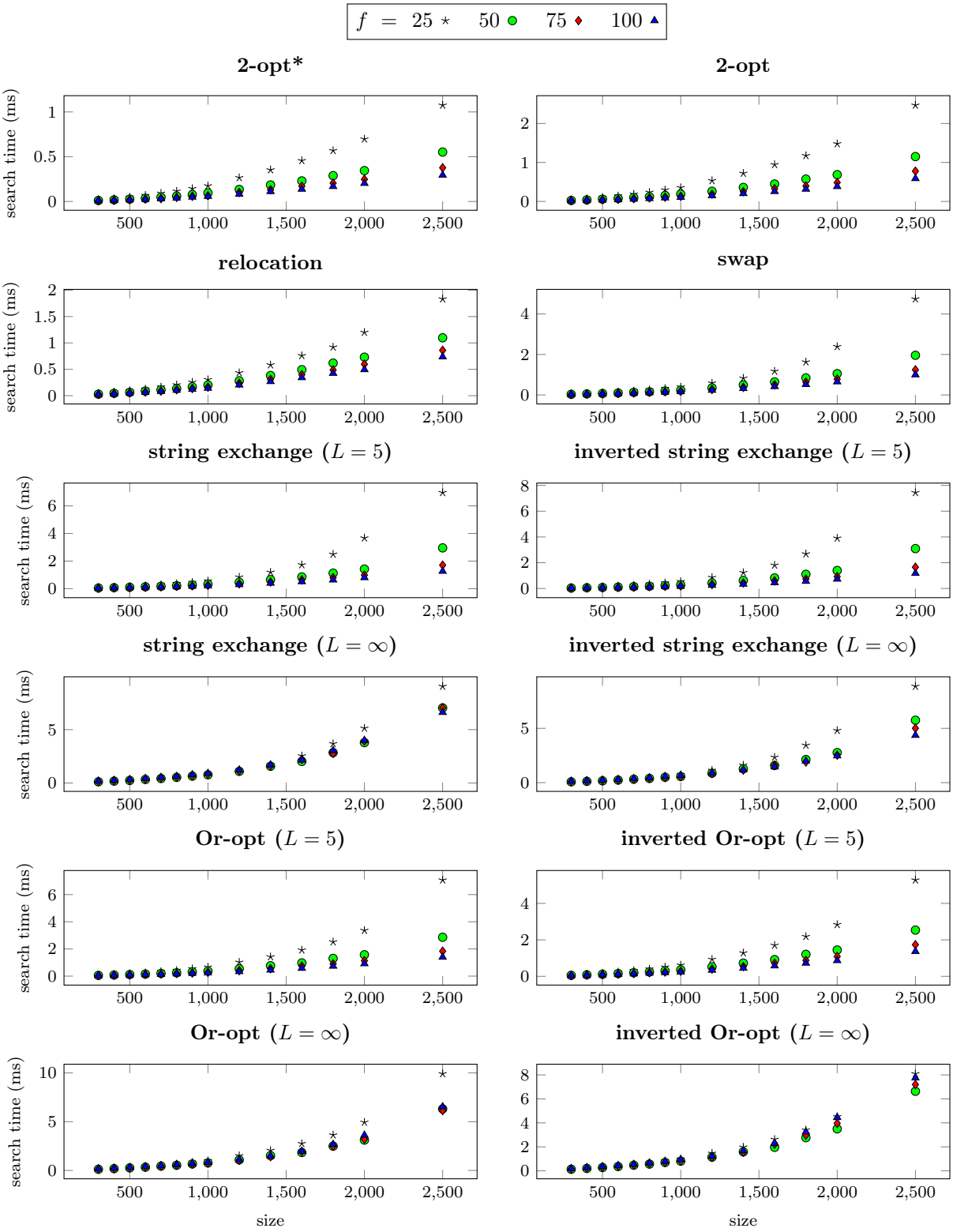


Figure 9: [CVRP, Irnich et al. (2006)] *Average neighborhood-exploration time* (in milliseconds) of dynamic-radius search for various neighborhood operators and instances ranging from 300 to 2,500 in customer size.

As a last note, we rectify a mistake in the pseudo-code (Irnich et al. 2006; Algorithm 8, Line 6) of the swap neighborhood. Making abstraction of the different nomenclature, it should read as "LET $B_1 = (c_{v_1,t_1} + c_{t_1,w_1})/2 - G^*/\mathbf{4}$" instead of $\mathbf{2}$ in the last denominator.

6.3. Inter-Depot Moves and the Correction Term

In Section 6.3.1, we analyze three options for implementing the neighborhood exploration of 2-opt and 2-opt*: allowing inter-depot moves by either using the correction term (*With*) or not (*Without*), and forbidding inter-depot moves altogether (*Forbid*). Option *Without* uses the simple radii of the inner-depot cases, i.e., (3) and (4), but still requires the whole machinery of the depot repair operation. It does not guarantee that a move with maximum gain is identified. In Section 6.3.2, we provide insights regarding the depot configurations by analyzing the same three options on the newly created instance set. Finally, we take a look in Section 6.3.3 at an alternative way to recover inter-depot 2-opt and 2-opt* moves that foregoes both the challenging implementation and the correction term.

6.3.1. Comparison on commonly used Benchmark Instances

We examine how the three options *With*, *Without*, and *Forbid* perform against each other by comparing the respective relative gap measures. Given an option o and an MDVRP instance b, we compute the sampled relative gaps over each local descent l as $\zeta_{blo} = (z_{blo} - \bar{z}_b)/\bar{z}_b \times 100$, where z_{blo} is the value obtained in run l and $\bar{z}_b$ is the best known objective value for the instance b. Moreover, we denote per instance and option the average relative gap by $\bar{\zeta}_{bo}$ and the first order statistic by $\tilde{\zeta}_{bo}$. Figure 10 displays aggregated results of the relative gaps on 1,000 local descents for the three options. For each instance, we find the corresponding legend marking of an option at three heights, the middle one (filled symbols) is the average, the one above (unfilled symbols) is one standard deviation away whereas the one below (unfilled symbols) is the first order statistic.

It is visible that option *Forbid* (blue triangle) performs more inconsistently than the other two despite reaching similar first order statistics. In order to take a more objective stand than what is visually available, Table 3 reports the findings using hypothesis testing on the average relative gap $\bar{\zeta}_{bo}$. For each option o, we additionally list proportions of instances where the *incumbent solution* $\min_{l,o} z_{blo}$ is found and the relative computational times with respect to the fastest option. Let $\mathbf{t}_{blo}$ denote the computational time for solve local descent l. The *overall time* of option o is then $\mathbf{t}_o = \sum_{b,l} \mathbf{t}_{blo}$. The fastest option is denoted $o_{\min} = \arg\min_o(\mathbf{t}_o)$. For the geometric measurement of option o, we compute the geometric mean of $\sum_l \mathbf{t}_{blo} / \sum_l \mathbf{t}_{bl,o_{\min}}$ over the instances b.

	z	z	Incumbent (%)		Relative time	
Option	Without	Forbid	Inclusive	Exclusive	Overall	Geometric
With	−1.503	−4.127	53.2	25.5	1.95	1.36
Without		−4.419	51.1	23.4	1.06	1.02
Forbid			48.9	21.3	1.00	1.00

Table 3: [MDVRP, Cordeau et al. (1997); Vidal et al. (2013)] Pairwise Wilcoxon signed-ranked test z-scores, inclusive/exclusive incumbent proportions, and relative computing times from 1,000 starting solutions on three options for implementing the neighborhood exploration of 2-opt and 2-opt* (all available neighborhood operators are utilized).

We use the Wilcoxon signed-ranked test to compare the options pairwise. For example, the null hypothesis is that option *With* has no added value compared to option *Without*, i.e., H_0 : instance paired differences $\bar{\zeta}_{b,With} - \bar{\zeta}_{b,Without}$ are distributed around zero, whereas the alternative hypothesis H_1 is that the differences are significantly signed, i.e., the options impact solution quality.

Intermediate measures of the opposing options *With* and *Without* are $W = \sum_{i=1}^{N_r} \text{sgn}(\Delta_i) \cdot R_i = -284$, where $N_r = 47$ is the number of non-zero paired differences, $\text{sgn}(\Delta_i)$ denotes the sign of their respective

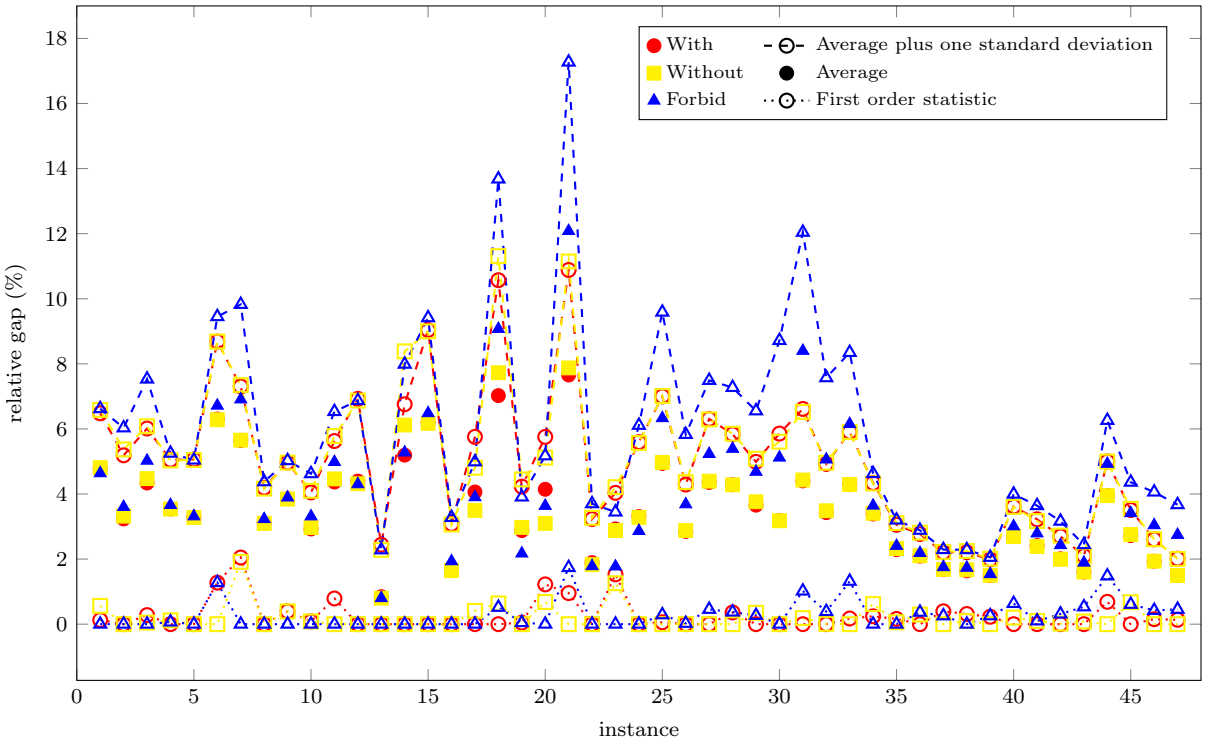


Figure 10: [MDVRP, Cordeau et al. (1997); Vidal et al. (2013)] Sampled average, standard deviation, and first order statistic of relative gaps in percentage from 1,000 starting solutions on three options for implementing the neighborhood exploration of 2-opt and 2-opt* (all available neighborhood operators are utilized).

values, and R_i their ranks. We then evaluate $\sigma_W^2 = \frac{N_r(N_r+1)(2N_r+1)}{6} = 35{,}720$. The resulting z-score is $W/\sigma_W \approx -1.503$ which corresponds to a p-value of 0.066. As a result, we can reject the null hypothesis at around 95 % confidence level in the unilateral test.

With respect to the other columns of Table 3, the *inclusive* column states that option *With* finds the incumbent 53.2 % of the time whereas the *exclusive* column that option *Forbid* finds an incumbent that no other option has identified 21.3 % of the time. Option *Forbid* is the fastest such that option *With* takes overall an arithmetic factor of 1.95 longer than *Forbid*, i.e., $\mathbf{t}_{With}/\mathbf{t}_{Forbid} = 1.95$. For the geometric mean, it is 1.36.

Finally, performing an hypothesis test over the first order statistics $\tilde{\zeta}_{b,o}$ renders pairwise *non*-significant results even over option *Forbid* which matches what we see visually. However, the above presented hypothesis testing over averages speaks volumes about the consistency of the various options: the correction term incurs a computational cost but indeed seems to provide added value whether we look at the average test scores or the found incumbent percentages.

As a side note, we also experimented with another option *Last resort* which only uses the correction term after option *Without* reaches a local optimum. In line with expectations, it performs almost equally as option *Without*. Including this additional option *Last resort* in the comparison would bias the table especially with respect to the exclusive incumbent count.

6.3.2. Comparison on new Large-Scale MDVRP Instances

The new instances are generated across 14 groups increasing in the number of customers from $|N| = 300$ to 2,500 (in steps of 100, 200, and 500). Each group contains 80 instances which systematically vary several key characteristics: customer distribution, load factor, and depot configuration. The customers are

distributed either randomly over a square or using a mixture of random and clustered coordinates. Moreover, four different load factors $f = 25, 50, 75$, and 100 for the average ratio of customers per route are considered (as in Irnich et al. (2006), the value of f is controlled by choosing $Q \approx f \cdot \sum_{i \in N} q_i / |N|$ which strongly impacts several types of results). Finally, we have also predetermined ten configurations of depot locations ranging in the number of depots ($2 \leq |D| \leq 6$) and differing in depot proximity from close to remote from one another.

Table 4 lists these configurations with a geometric qualifier, the number $|D|$ of depots, and the degree of proximity (*near* or *far*). Moreover, we report the z-score of option *With* opposed to option *Without* while omitting those of option *Forbid* because of its repeated poor performance as in the previous experiment. In the remaining columns, the table displays relative time ratios of option *With* compared to option *Without* as well as the difference in exclusive incumbents. For the first configuration 'stack', option *With* takes overall 1.53 times longer than *Without* to complete and finds 1 more exclusive incumbent. The takeaway here is that we can claim that *far* depot configurations benefit more from using the correction term. Furthermore, using the correction term in near depot configurations is relatively more computationally expensive than in far depot configurations. This makes sense, since depots far away from each other give greater arbitrage possibility in route optimization and therefore render the depot association an even more important question. With respect to the relative time, recall that it is empirically observed that customers are often assigned to their nearest depot. We venture that in a near depot configuration (such as configurations stack (identical) or cluster, and all those with near proximity, i.e., ID = 1, 2, 3, 5, 7, and 9 in Table 4) the depot assignment might make less difference and therefore options *With* and *Without* perform comparably (except for ID = 9, where the reason remains unclear to us). Indeed, recall that the correction terms of the standard cases are zero if customers are already assigned to a closest depot. Unfortunately, this is not necessarily true for the exception cases: In 2-opt* exception case 2c, the term (5c) could be positive. For 2-opt, both exception cases could yield positive correction terms for (7c) and (7e). In a depot configuration with multiple depots at the same location (case stack, ID = 1 in Table 4), we may have positive correction terms that yet reduce the pruning potential even though they do not provide any better moves. This also explains why we can end up in different local optima when comparing options *With* and *Without*.

				With vs. Without			
Instance group					Relative time		Exclusive
ID	Configuration	$\|D\|$	Proximity	z-score	Overall	Geometric	Difference
1	stack	2	identical	0.183	1.53	1.30	1
2	cluster	3	near	−0.851	1.50	1.29	−1
3	diagonal	2	near	−1.338	1.37	1.21	4
4	diagonal	2	far	−2.119	1.15	1.08	8
5	diamond	4	near	0.479	1.56	1.32	4
6	diamond	4	far	−3.989	1.24	1.12	13
7	cross	5	near	0.160	1.66	1.41	−1
8	cross	5	far	−6.645	1.30	1.16	−15
9	circle	6	near	−3.536	1.54	1.29	17
10	circle	6	far	−5.377	1.15	1.10	−5

Table 4: Characteristics of the ten depot configurations and comparison of options *With* and *Without*.

6.3.3. Inter-Depot Moves via String Exchange and Or-Opt

In this section, we exploit that all inter-depot 2-opt* and 2-opt moves are specific string exchange and Or-opt moves on the giant tour. For example, 2-opt* moves (Figure 2) can be reproduced by ordered variants of the string exchange (standard cases 2a, 2b) and Or-opt (exception cases 2c, 2d, 2e, 2f) moves. In particular, one can see that the string $f_i \rightsquigarrow i$ in Figure 2e (2-opt*) can have any length and corresponds to

the string $s_i \rightsquigarrow j$ in Figure 6a (ordered Or-opt). With respect to 2-opt moves (Figure 3), inverted variants of the string exchange and Or-opt respectively reproduce standard and exception cases.

Accounting for inter-depot 2-opt* and 2-opt moves in dynamic-radius search implies that two algorithmic tasks have to be fulfilled: the computation of correction terms and obviously the implementation of the actual inter-depot moves. The exception correction terms are likely to be larger than their standard counterparts to account for worst-case scenarios in unknown customer j. Fortunately, exceptions occur only sporadically, see Table 2. With this in mind, we discuss three alternatives to cope with exhaustively testing for inter-depot moves.

First, it is possible to duplicate the 2-opt* and 2-opt methods and specialize these copies to account for specific inter-depot cases. In this fashion, we inevitably face redundant move tests and therefore have an overall slower method. Moreover, it is a cumbersome implementation for which one must indeed implement inter-depot cases.

Second, specializing the string exchange and Or-opt neighborhoods to test for specific depot cases is even more cumbersome (number and complexity of loop blocks) and slower, since we must also pay for the overhead of these richer neighborhoods.

Third, we can forbid inter-depot 2-opt and 2-opt* moves and herewith get rid of all the correction terms as well as the repair operations. Instead, we allow arbitrary string lengths, i.e., $L = \infty$ in the string exchange and Or-opt neighborhood exploration.

We have implemented this third alternative and tested how well string exchange and Or-opt scale with this length increase. Obviously, the time of neighborhood exploration increases with an unbounded length $L = \infty$ but it is limited by the longest route of the candidate solution. The results of the comparison between the maximum string length $L = \infty$ and $L = 5$ are shown in Figure 11. We ultimately observe a factor around 6 (up to 10) on instances with load factor $f = 100$.

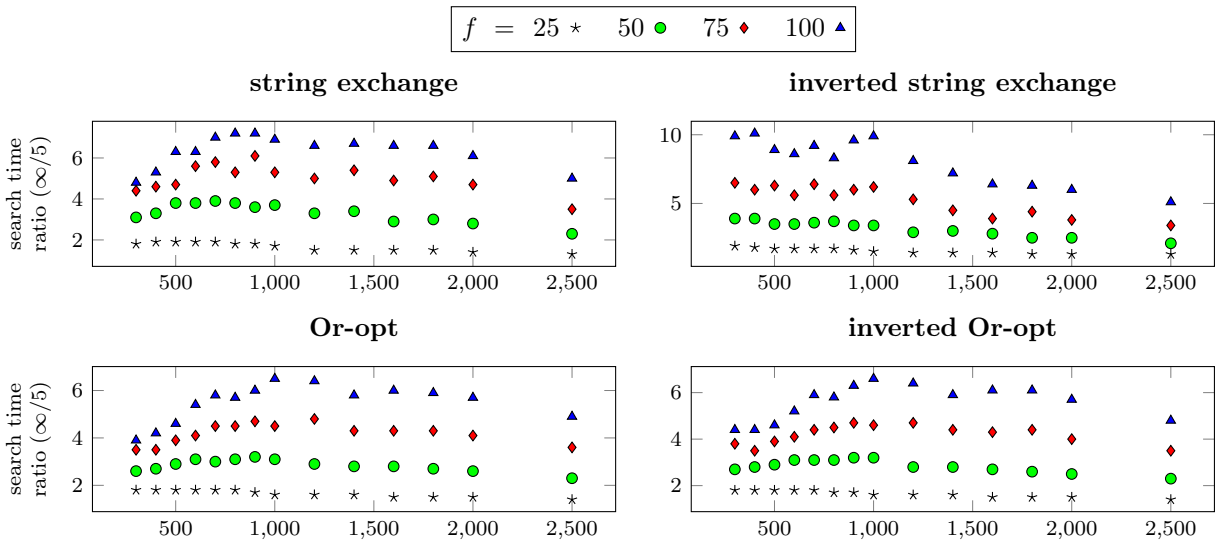


Figure 11: [MDVRP, large-scale instances] Dynamic-radius search computing time comparison of allowed length $L = \infty$ over $L = 5$ in string exchange and Or-opt neighborhood explorations for instances ranging from 300 to 2,500 in customer size.

The reader may expect now that we present a direct comparison of the implementations of 2-opt and 2-opt* using the correction terms of Section 4 and Or-opt and string exchange with unlimited string length $L = \infty$. Such a comparison would reveal that already longer neighborhood exploration times of Or-opt and string exchange (see Figure 9) must be compounded with the observed factors of Figure 11. However, by allowing arbitrary string lengths, we do not only recover all inter-depot 2-opt* and 2-opt moves, but we also enrich the local optima space: additional improving Or-opt and string exchange moves that do not represent 2-opt

or 2-opt* moves are found. Hence, such a direct comparison considering relative computation times is an oversimplification. We therefore omit further analyses.

Finally, we arrive at the conclusion that the simplest possibility is to rely on string exchange and Or-opt neighborhoods to produce inter-depot 2-opt and 2-opt* moves. However, even if this implementation shortcut is functional, it does not compete with a full-fledged inter-depot adaptation for the 2-opt* and 2-opt neighborhoods.

6.4. Comparison against Best Known Solutions from the Literature

Results are reported under the following general format. On the left side, we have the structural description of the instances obtained from the paper listed in the caption. Then follows the best known solutions (BKS) available in the literature $\bar{z}$ with a reference listing for the first occurrence and a bold *id* entry whenever optimality has been proven (Baldacci and Mingozzi 2009; Contardo and Martinelli 2014). Finally, we have our solutions $\tilde{z}$, running times $\mathbf{t}$ in seconds ([s]), and relative gaps computed in percent as $(\tilde{z} - \bar{z})/\bar{z} \times 100$.

The first set of results gathered in Table 5 are based on the instances of Cordeau et al. (1997). The average relative gap amounts to 0.46 % after some 250,000 local descents. Moreover, our heuristic takes an average of 6.5 minutes per instance, the longest time of 40 minutes being spent on pr10. In comparison, the state-of-the-art hybrid genetic algorithm which uses adaptive diversity control (HGSADC) of Vidal et al. (2012) establishes an average relative gap of virtually 0 % after an average running time of 42.40 minutes (10 runs of average 4.24 minutes).

Table 6 reads exactly in the same way except it is based on the MDVRPTW instances of Vidal et al. (2013). We compare with the results of Vidal et al. (2014a) and therefore likewise discard input of time windows and the depot fleet size limit m. The authors present various algorithms but two of them (HGSADC-noR and HGSADC+), each running for up to 5 hours per instance, produce almost all incumbent solutions. Using some 25,000 local descents, the quality of our solutions lies around 3 % above these but are obtained in 10 minutes rather than 10 hours. It is a good time to acknowledge that we obviously attain the limitation of this simple metaheuristic since additional iterations give less and less improved incumbent solutions.

While De Smet et al. (2006) have developed a software interface for `OptaPlanner` which deserves praise, we have no choice but to underscore that it clearly outputs subpar solutions. In fact, it makes us wonder whether the benchmark version is fully featured since a simple visual inspection is sometimes enough to discern possible solution improvements. The software has a hard coded time limit of 5 minutes but observing its behavior in the solution process suggests that the time limit could be halved without impacting solution quality. This does not bode well for the diversification mechanism that they employ.

The documentation is not explicitly clear about the variant used for the objective function but our understanding is that a hierarchy prioritizing minimal fleet size is used. We present results for our algorithm under the standard travel cost minimization objective as well as an adaptation for the hierarchical objective. In the case of the latter, it is well established that vehicle lower bounds computed based on expected route distributions are easier to reach as the instance grows in size. Since the fleet size is handled as a soft constraint, we slightly modified our algorithm to attempt to solve the problem with a vehicle limit that increases by one unit whenever the algorithm fails to achieve soft feasibility after its allocated runtime. With respect to smaller instances, since they are solved faster, the fact that the lower bound might be off by several units is not so concerning. Table 7 reports these results on 10 instances whose names `belgium-d-n-k` reflect the number of depots (d $= |D|$ if more than one), the number of vertices (n $= |N| + |D|$), and the depot fleet size limit (k $= m$). The objective priority is indicated by $\tilde{z}$ or $\tilde{\kappa}$. The number of local descents has been fixed to 500 to reproduce a similar running time. With average relative gaps of -13.54 % and -11.84 % respectively for routing costs and fleet-size objectives, it is clear that we obtain far better solutions. The cost disparity in fleet-size optimization yielding the same number of vehicles as in routing cost optimization can be explained by the δ-parameter which is set to the number of customers in the latter kind.

Instance						BKS		Dynamic-Radius Search		
	$\|N\|$	m	$\|D\|$	T	Q	$\bar{z}$	ref.	$\tilde{z}$	**t** [s]	% $\bar{z}$
p01	50	4	4	0	80	576.87	RLB96	576.87	71.0	0.00
p02	50	2	4	0	160	473.53	PR07	473.53	55.7	0.00
p03	75	3	5	0	140	641.19	PR07	641.19	94.9	0.00
p04	100	8	2	0	100	1,001.04	PR07	1,003.59	153.1	0.25
p05	100	5	2	0	200	750.03	VCGLR12	751.15	114.6	0.15
p06	100	6	3	0	100	876.50	RLB96	880.04	136.8	0.40
p07	100	4	4	0	100	881.97	PR07	891.33	189.2	1.06
p08	249	14	2	310	500	4,371.66	ELTG14	4,423.40	499.7	1.18
p09	249	12	3	310	500	3,858.66	VCGLR12	3,899.13	469.7	1.05
p10	249	8	4	310	500	3,629.60	ELTG14	3,668.89	440.2	1.08
p11	249	6	5	310	500	3,545.18	ELTG14	3,569.11	476.7	0.68
p12	80	5	2	0	60	1,318.95	RLB96	1,318.95	64.3	0.00
p13	80	5	2	200	60	1,318.95	RLB96	1,318.95	49.2	0.00
p14	80	5	2	180	60	1,360.12	CGL97	1,360.12	63.0	0.00
p15	160	5	4	0	60	2,505.42	PR07	2,505.42	181.0	0.00
p16	160	5	4	200	60	2,572.23	RLB96	2,572.23	139.2	0.00
p17	160	5	4	180	60	2,709.09	PR07	2,742.80	172.4	1.24
p18	240	5	6	0	60	3,702.85	PR07	3,702.85	351.6	0.00
p19	240	5	6	200	60	3,827.06	RLB96	3,827.06	258.1	0.00
p20	240	5	6	180	60	4,058.07	CGL97	4,091.78	308.4	0.83
p21	360	5	9	0	60	5,474.84	PR07	5,490.11	693.9	0.28
p22	360	5	9	200	60	5,702.16	PR07	5,702.16	487.2	0.00
p23	360	5	9	180	60	6,078.75	PR07	6,160.98	589.7	1.35
pr01	48	1	4	500	200	861.32	CGL97	861.32	82.1	0.00
pr02	96	2	4	480	195	1,307.34	PR07	1,307.34	369.7	0.00
pr03	144	3	4	460	190	1,803.80	VCGLR12	1,806.53	208.5	0.15
pr04	192	4	4	440	185	2,058.31	VCGLR12	2,073.65	422.8	0.75
pr05	240	5	4	420	180	2,331.20	VCGLR12	2,359.04	948.7	1.19
pr06	288	6	4	400	175	2,676.30	VCGLR12	2,703.31	708.5	1.01
pr07	72	1	6	500	200	1,089.56	PR07	1,089.56	191.0	0.00
pr08	144	2	6	475	190	1,664.85	PR07	1,667.24	764.1	0.14
pr09	216	3	6	450	180	2,133.20	VCGLR12	2,148.61	524.5	0.72
pr10	288	4	6	425	170	2,868.26	VCGLR12	2,912.24	2,486.7	1.53
Average									386.9	0.46

Table 5: [MDVRP, Cordeau et al. (1997)] Computational results of the multi-start iterated local search with $n_{\text{ILS}} = 250{,}000$ iterations.

Instance						BKS	Dynamic-Radius Search					
							$n_{\text{ILS}} = 2500$			$n_{\text{ILS}} = 25{,}000$		
	$\|N\|$	m	$\|D\|$	T	Q	$\bar{z}$	$\tilde{z}$	**t** [s]	% $\bar{z}$	$\tilde{z}$	**t** [s]	% $\bar{z}$
pr11a	360	10	4	450	200	4,994.67	5,207.44	12.7	4.26	5,129.05	122.4	2.69
pr12a	480	13	4	440	195	6,367.67	6,577.68	23.9	3.30	6,555.29	212.8	2.95
pr13a	600	16	4	430	190	7,645.29	8,011.74	31.5	4.79	7,917.86	343.3	3.57
pr14a	720	19	4	420	185	9,101.67	9,477.76	50.7	4.13	9,403.68	474.3	3.32
pr15a	840	22	4	410	180	10,598.70	10,961.81	76.1	3.43	10,920.67	740.4	3.04
pr16a	960	26	4	400	175	11,919.71	12,391.17	119.5	3.96	12,312.47	1,048.4	3.30
pr17a	360	7	6	460	200	4,761.70	4,877.06	15.5	2.42	4,856.80	153.4	2.00
pr18a	520	10	6	440	190	6,504.36	6,744.90	32.6	3.70	6,666.09	313.4	2.49
pr19a	700	13	6	420	180	8,639.44	8,995.29	77.3	4.12	8,971.01	649.9	3.84
pr20a	880	16	6	400	170	9,825.50	10,207.03	95.6	3.88	10,160.99	1,014.0	3.41
pr21a	420	4	12	475	200	4,582.62	4,709.52	23.1	2.77	4,677.77	228.7	2.08
pr22a	600	6	12	450	190	6,141.63	6,297.61	48.7	2.54	6,283.19	405.6	2.30
pr23a	780	8	12	425	180	8,014.10	8,273.65	81.8	3.24	8,218.96	764.2	2.56
pr24a	960	10	12	400	170	9,909.49	10,238.21	144.2	3.32	10,221.22	1,337.6	3.15
Average								59.5	3.56		557.7	2.91

Table 6: [MDVRP, Vidal et al. (2013)] Computational results of the multi-start iterated local search with $n_{\text{ILS}} = 2500$ and $n_{\text{ILS}} = 25{,}000$ iterations.

Instance		BKS		Dynamic-Radius Search							
				Distance objective				Fleet-size objective			
	Q	$\bar{z}$	$\bar{\kappa}$	$\tilde{z}$	κ	**t** [s]	% $\bar{z}$	z	$\tilde{\kappa}$	**t** [s]	% $\bar{z}$
belgium-d2-n50-k10	125	15.47	8	15.46	9	0.1	−0.06	16.17	8	0.2	4.52
belgium-d3-n100-k10	250	17.43	8	17.33	8	0.3	−0.57	17.40	8	0.3	−0.57
belgium-d5-n500-k20	250	54.70	15	38.43	16	3.1	−29.74	39.92	15	6.9	−27.02
belgium-d8-n1000-k20	500	68.04	16	50.56	17	13.6	−25.69	51.12	15	37.7	-24.87
belgium-d10-n2750-k55	500	119.77	40	90.70	43	65.2	−24.27	94.85	40	304.5	−20.81
belgium-n50-k10	125	21.16	8	21.02	8	0.1	−0.66	20.81	8	0.1	−0.66
belgium-n100-k10	250	22.98	8	23.11	8	0.3	0.57	23.11	8	0.3	0.57
belgium-n500-k20	250	57.49	14	47.75	15	3.1	−16.94	49.56	14	9.3	−13.79
belgium-n1000-k20	500	77.59	14	59.20	14	9.9	−23.70	60.55	14	22.2	−23.70
belgium-n2750-k55	500	150.22	39	128.67	41	100.1	−14.35	132.03	39	274.0	−12.11
Average						19.6	−13.54			65.6	−11.84

Table 7: [MDVRP, De Smet et al. (2006)] Computational results of the multi-start iterated local search with $n_{\text{ILS}} = 500$ iterations.

7. Conclusion and Outlook

In this paper, we have revisited radius search, an effective neighborhood exploration technique, which distinguishes itself from other techniques such as lexicographic search by the way the neighborhood is explored: A lexicographic search prunes a exploration branch whenever a local infeasibility is observed. Dynamic-radius search is closer in spirit to the optimization paradigm, since the pruning is based on coefficients of the objective function, that is, the threshold bound is a function of the best gain found at any given time.

We have extended previous works on radius search to the multi-depot vehicle routing problem including capacity and tour-duration constraints. The focus of our research is on the two fundamental neighborhoods 2-opt and 2-opt* and their exploration which not includes standard intra-depot moves but also inter-depot moves. Case-dependent correction terms to be added to the otherwise incorrect standard search radii have been derived. Dynamic-radius search equipped with this modified pruning criterion allows identifying a best-improving move, either inner-depot or inter-depot, with little additional computational effort.

In comparison to lexicographic search, speedups of factors of 100 and more are observed for 2-opt, 2-opt*, Or-opt, swap, and string-exchange neighborhoods. Furthermore, we have confirmed with statistical tests that allowing depot swapping strongly favors heuristic solution quality, especially for multi-depot configurations where depots are not located close to each other.

While there certainly are some fancier metaheuristics out there, we believe our basic iterated-local search implementation is legitimate enough. Summarizing the results on three benchmark sets from the literature, we can state that we have created a single implementation with very little parametrization which successfully competes with state-of-the-art metaheuristics.

We can think of the following research paths. First, the way the threshold is constructed is particularly interesting because it relies on cost upper bounds rather than the actual cost. Tackling alternative vehicle routing problem variants where the objective function is not exactly a sum of arc costs such as time-dependent travel costs then becomes possible. Second, for asymmetric problems, the redundancy in the exploration does not mean we get to test asymmetric arc costs for free. Indeed, the worst-case factor is eight rather than four which fortunately is still prone to significant empirical reduction. Finally, we venture that machine learning may help answer the question we raised at the end of Section 3.2.3 concerning the prediction of the smallest but sufficiently large radius ensuring that a move with maximum gain is identified.

Acknowledgement

This research was funded by the Deutsche Forschungsgemeinschaft (DFG) under grant no. IR 122/7-1.

References

Emile H. L. Aarts and Jan Karel Lenstra, editors. *Local Search in Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley & Sons, New York, NY, USA, 1997.

Roberto Baldacci and Aristide Mingozzi. A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming*, 120(2):347–380, 2009. doi:10.1007/s10107-008-0218-9.

Mandell Bellmore and Saman Hong. Transformation of multisalesmen problem to the standard traveling salesman problem. *Journal of the Association for Computing Machinery*, 21:500–504, 1974. doi:10.1145/321832.321847.

Jon Jouis Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992. doi:10.1287/ijoc.4.4.387.

Geoff Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964. doi:10.1287/opre.12.4.568.

Claudio Contardo and Rafael Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129–146, 5 2014. doi:10.1016/j.disopt.2014.03.001.

Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997. doi:10.1002/(SICI)1097-0037(199709)30:2<105::AID-NET5>3.0.CO;2-G.

G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

Geoffrey De Smet et al. *OptaPlanner User Guide 7.17.0*. Red Hat and the community, 2006. URL `https://www.optaplanner.org`. OptaPlanner is an open source constraint satisfaction solver in Java.

Guy Desaulniers, Oli B. G. Madsen, and Stefan Røpke. The vehicle routing problem with time windows. In Paolo Toth and Daniele Vigo, editors, *Vehicle Routing*, chapter 5, pages 119–159. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2014. doi:10.1137/1.9781611973594.ch5.

John Willmer Escobar, Rodrigo Linfati, Paolo Toth, and Maria G. Baldoquin. A hybrid granular tabu search algorithm for the multi-depot vehicle routing problem. *Journal of Heuristics*, 20(5):483–509, 2014. doi:10.1007/s10732-014-9247-0.

Birger Funke, Tore Grünert, and Stefan Irnich. A note on single alternating cycle neighborhoods for the TSP. *Journal of Heuristics*, 11(2):135–146, 2005. doi:10.1007/s10732-005-0713-6.

Keld Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. doi:10.1016/s0377-2217(99)00284-2.

Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. The Morgan Kaufmann Series in Artifical Intelligence. Elsevier, San Francisco, CA, USA, 2005.

Stefan Irnich. Resource extension functions: properties, inversion, and generalization to segments. *OR Spectrum*, 30(1):113–148, 2008a. doi:10.1007/s00291-007-0083-6.

Stefan Irnich. A unified modeling and solution framework for vehicle routing and local search-based metaheuristics. *INFORMS Journal on Computing*, 20(2):270–287, 2008b. doi:10.1287/ijoc.1070.0239.

Stefan Irnich, Birger Funke, and Tore Grünert. Sequential search and its application to vehicle-routing problems. *Computers & Operations Research*, 33(8):2405–2429, 2006. doi:10.1016/j.cor.2005.02.020.

David Stifler Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study in local optimization. In *Local Search in Combinatorial Optimization*, chapter 8, pages 215–310. 1997.

Shen Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269, 1965. doi:10.1002/j.1538-7305.1965.tb04146.x.

Shen Lin and Brian Wilson Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, USA, 2002. doi:10.1007/0-306-48056-5_11.

Olivier Martin, Steve W. Otto, and Edward William Felten. Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 1(4):219–224, 1992. doi:10.1016/0167-6377(92)90028-2.

Jean-Yves Potvin and Jean-Marc Rousseau. An exchange heuristic for routeing problems with time windows. *The Journal of the Operational Research Society*, 46(12):1433–1446, 1995. doi:10.2307/2584063.

Gerhard Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer, Berlin Heidelberg, Germany, 1994. doi:10.1007/3-540-48661-5.

Martin W. P. Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47(1):75–85, 1990. doi:10.1016/0377-2217(90)90091-O.

Michael Schneider, Fabian Schwahn, and Daniele Vigo. Designing granular solution methods for routing problems with time windows. *European Journal of Operational Research*, 263(2):493–509, 2017. doi:10.1016/j.ejor.2017.04.059.

Kenneth Steiglitz and Peter Weiner. Some improved algorithms for computer solution of the traveling salesman problem. In *Proceedings of the Sixth Allerton Conference on Circuit and System Theory*, pages 814–821, Urbana, IL, USA, 1968.

Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003. doi:10.1287/ijoc.15.4.333.24890.

Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012. doi:10.1287/opre.1120.1048.

Thibaut Vidal, Gabriel Crainic Teodor, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489, 2013. doi:10.1016/j.cor.2012.07.018.

Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. Implicit depot assignments and rotations in vehicle routing heuristics. *European Journal of Operational Research*, 237(1):15–28, 2014a. doi:10.1016/j.ejor.2013.12.044.

Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673, 2014b. doi:10.1016/j.ejor.2013.09.045.

Thomas Visser and Remy Spliet. Efficient move evaluations for time-dependent vehicle routing problems. Technical Report EI2017-23, Erasmus University Rotterdam, 2017. URL `http://hdl.handle.net/1765/100852`.