Discrete Applied Mathematics 346 (2024) 131–153

Contents lists available at ScienceDirect

Discrete Applied Mathematics

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

Inter-depot moves and dynamic-radius search for multi-depot vehicle routing problems

Jean Bertrand Gauthier *, Stefan Irnich

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

ARTICLE INFO

Article history:
Received 31 December 2022
Received in revised form 29 October 2023
Accepted 5 December 2023
Available online 22 December 2023

Keywords:
Routing
Local search
Sequential search
Dynamic-radius search
Inter-depot

ABSTRACT

Dynamic-radius search, formerly known as sequential 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 to identify 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 achieved compared to an also optimized implementation of lexicographic search, another effective neighborhood exploration technique using a feasibility-based pruning criterion. As one would expect, better local optima are found on average when allowing inter-depot moves in radius search (positive result). Against intuition, we do however not end up with a better ILS metaheuristic regarding best found solutions, i.e., better average results do not translate into better overall results. We can at least partly explain the latter negative result, which might be useful for other researchers and their attempt to algorithmically optimize their neighborhood exploration procedures.

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). Escobar et al. [8] note that it is rather straightforward to extend most classical neighborhoods to the multi-depot environment except 2-opt and 2-opt*. The description is however quite limited so we formalize the various inter-depot cases that arise in these two neighborhoods and sketch a few others that work naturally with multi-depot. This analysis also serves to equip dynamic-radius search with a modified pruning criterion so that best-improving moves which allow inter-depot edges are found with little additional computational effort. The idea behind this work revolves around the hypothesis that being systematically able to find moves with better gains leads to a local search that can find better local optima. Our analysis leads to positive and negative results. The positive result is that allowing inter-depot moves helps finding better local optima on average. This result unfortunately does not translate to the metastrategy where ultimately only the overall best local optimum is important. While one can compare this best

* Corresponding author.
E-mail addresses: jean-bertrand.gauthier@hec.ca (J.B. Gauthier), irnich@uni-mainz.de (S. Irnich).

https://doi.org/10.1016/j.dam.2023.12.004

gain idea to Dantzig's pivot rule in the primal simplex algorithm for which it is known that alternative less myopic rules perform better, it is not immediately clear to us that the behavior of a heuristic would be the same as that of an exact method.

Dynamic-radius search builds on several classical works on the symmetric *traveling salesman problem* (TSP). In the context of the TSP, Hoos and Stützle [13] use the expression *fixed-radius search* to collectively describe the idea of Steiglitz and Weiner [26] and numerous extensions such as Bentley [3], Martin et al. [21] and Johnson and McGeoch [17],Reinelt [23]. 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., $c_{ij} < c_{p_i,i}$ 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 known as the *gain criterion*. Lin and Kernighan [19] always choose edges according to the gain criterion in their famous Lin–Kernighan variable-depth neighborhood. Irnich et al. [16] have generalized this idea of using the gain criterion for the *(capacitated) vehicle routing problem* ((C)VRP) and to VRP-specific moves such as node/vertex relocation, Or-opt, string exchange, etc. Moreover, Irnich [15] 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, and others.

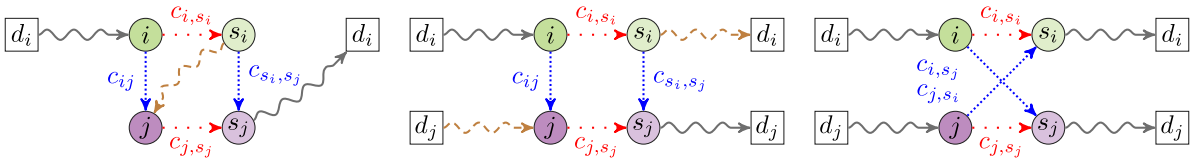
Irnich et al. [16] have introduced dynamic-radius search for VRPs under the name *sequential search*. This naming was inspired by the fact that moves are decomposed into partial moves and complete moves result from *sequentially* selecting the constituting partial moves. Note that cyclic independent move decomposition as coined in [16] 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 10, 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 $e \in E$, a routing cost c_e and a travel time t_e are given. 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 of 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 $\sum_{d \in D} |R \cap R_d| = |R| \leq \kappa$, and (2) all customers are serviced exactly once, i.e., $N = \bigcup_{r \in R} (V(r) \cap N)$ and $V(r_1) \cap V(r_2) \cap N = \varnothing$ 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 $R' = \mu(R)$. A move is improving if its marginal *gain* defined as $g(\mu, R) = c(R) - c(\mu(R))$ (or simply g when contextually clear) is positive, i.e., $g > 0$. Moreover, a solution R is a *local optimum* w.r.t. $\mathcal{N}$ if no improving move $\mu \in \mathcal{N}$ for R 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 a local optimum with respect to all $\mathcal{N} \in \Psi$. This solution $\tilde{R}$ depends on the starting solution R, the available neighborhoods 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 joint local optima $\tilde{R}$.

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.



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

Fig. 1. Intra-depot 2-opt and 2-opt* moves.

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 [2]. 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 $v \overleftsquigarrow w$ 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).

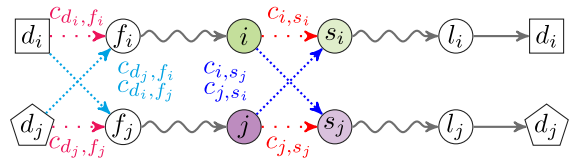
Developed with the TSP in mind, Croes [6] has devised an algorithm which performs so-called *inversions*. These are essentially 2-opt exchanges in the sense Lin [18] has generally coined λ-opt. The 2-opt neighborhood has been generalized to single-depot vehicle routing problems in a straightforward manner. Potvin and Rousseau [22] have introduced the 2-opt* neighborhood to tackle the *vehicle routing problem with time windows* [7]. The leading observation is that a 2-opt inter-tour move induces two segment inversions in the affected routes (see Fig. 1(b)). 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 Fig. 1(c)).

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 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. Fig. 1, which uses our multi-depot notation, therefore only captures intra-depot 2-opt and 2-opt* moves.

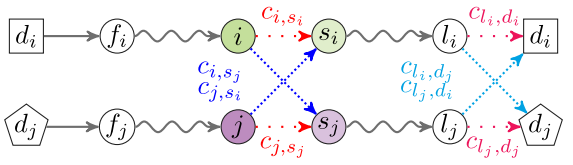
In the next two subsections, we show that the multi-depot case can nevertheless be correctly taken into account during the analysis of an *inter-depot* move. Observe that an inter-depot move must necessarily be inter-tour. In both neighborhoods, 2-opt and 2-opt*, we break down the possible cases that 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* moves

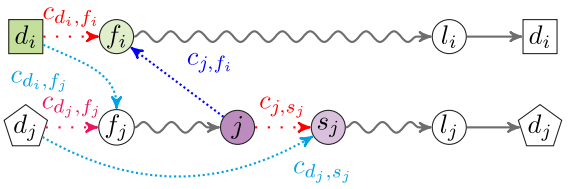
An inter-depot 2-opt* move happens when the two deleted arcs (i, s_i) and (j, s_j) belong to two different routes $r_i \neq r_j$ of two different depots $d_i \neq d_j$. As illustrated in Fig. 2, eight cases are sufficient to exhaustively cover repair options. Figs. 2(a) and 2(b) display the *standard case* where both deleted arcs (i, s_i) and (j, s_j) are either not the two first arcs or



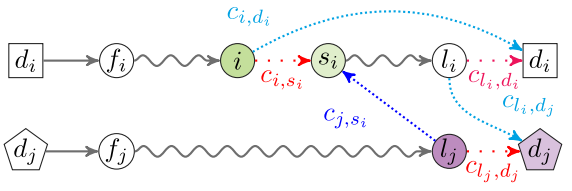
(a) Standard case: $i \neq d_i$, $j \neq d_j$.



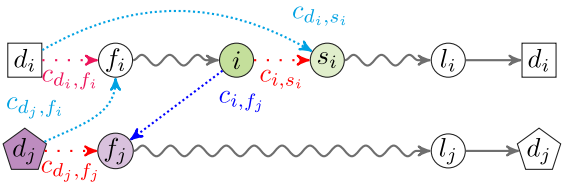
(b) Standard case: $s_i \neq d_i$, $s_j \neq d_j$



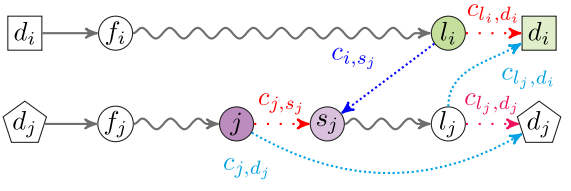
(c) Exception case: $i = d_i, j \neq d_j$, arc (i, s_j) is absent.



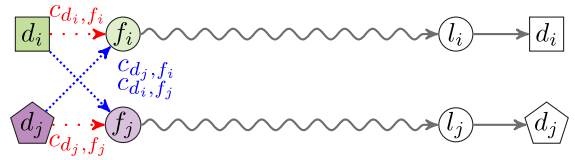
(d) Exception case: $s_i \neq d_i, s_j = d_j$, arc (i, s_j) is absent.



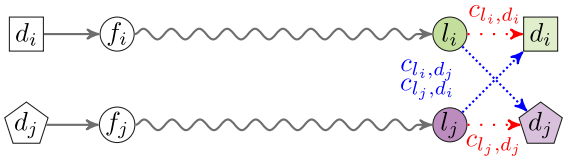
(e) Exception case: $i \neq d_i, j = d_j$, arc (j, s_i) is absent.



(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: $s_i = d_i, s_j = d_j$.

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

not the two last arcs in their respective routes r_i and r_j. This offers two possibilities of swapping the depots: swapping at the source (Fig. 2(a)) and at the sink (Fig. 2(b)). 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.

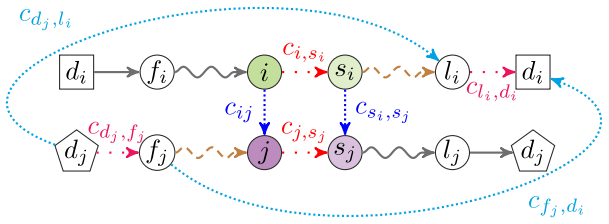
The next four *exception cases* depicted in Fig. 2(c)–2(f) happen if exactly one of the deleted arcs is the first (last) and the other one is not the first (not the last). In the exception cases, only three arcs instead of four are finally exchanged, one of the arcs inserted in the standard case is absent (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 Fig. 2(a)–2(f) cover all feasible inter-depot 2-opt* moves. There exist two more cases visualized in Figs. 2(g) and 2(h): (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 needs to delete to perform the source (sink) depot swap.

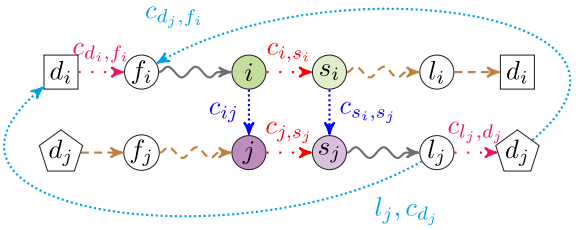
Ultimately, every 2-opt* inter-route move with different depots $d_i \neq d_j$ is evaluated with two repair operations (one source and one sink) based on the expressed conditions yielding up to two distinct inter-depot 2-opt* moves. Note that the conditions on i and j shown on the left-hand side of Fig.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

2.2. Inter-depot 2-opt moves

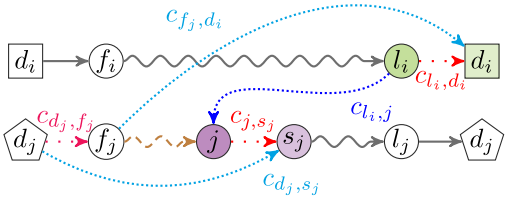
For the 2-opt neighborhood, we also raise standard and exception cases as presented in Fig. 3, but omit rejection cases for the sake of brevity. The repair operation must always swap sink and source depots of the two different routes. As a mnemonic device, depots finally always remain attached to their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



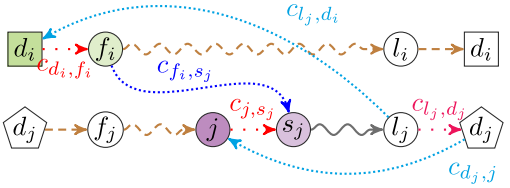
(a) Standard case: $s_i \neq d_i$, $j \neq d_j$.



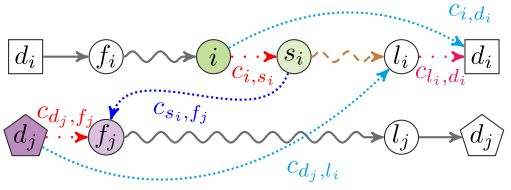
(b) Standard case: $i \neq d_i$, $s_j \neq d_j$.



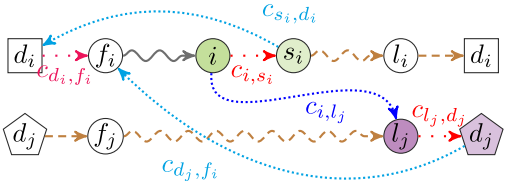
(c) Exception case: $s_i = d_i, j \neq d_j$, arc (s_i, s_j) is absent.



(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.

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

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 [29]. For the feasibility check, it is almost never a trivial task as per the inclusion of non-additive resource constraints. In accordance with Savelsbergh [24], Irnich [15], Vidal et al. [28], several resource constraints can nevertheless be tested in *constant time* 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 Fig. 1(a)) to explain the different search paradigms. A synopsis of the three search principles is shown in Fig. 4. All three Algorithms 1–3 work in a best-improvement manner, and they return the best gain γ found during exploration along with the vertices i, j that uniquely describe the corresponding 2-opt move. Lexicographic and radius are exact which means we also respectively get g^* and i^*, j^*. Details follow in Sections 3.1–3.3.

3.1. Lexicographic search

Lexicographic search as presented by Savelsbergh [24] consists of 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 in Fig. 4. 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 Fig. 1(a), 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 orientation of original path $(s_i \rightsquigarrow j)$ has been inverted. Since path P grows by one vertex in

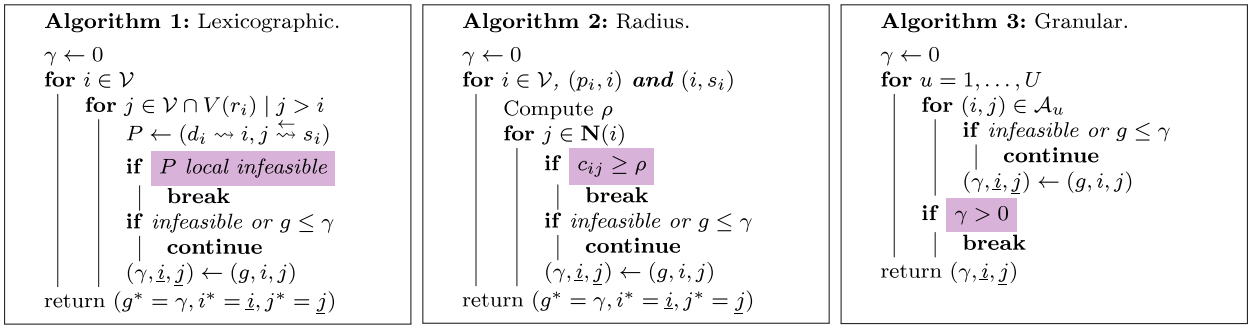


Fig. 4. Synopsis of the three neighborhood exploration methods for the intra-depot 2-opt neighborhood; highlighted lines show why and where early breaking occurs. It is assumed that both the feasibility and gain of a move are evaluated directly before the if-condition "*infeasible or* $g \le \gamma$" is reached.

every inner-loop iteration, the idea of Savelsbergh [24] is to prune the search based on the *local infeasibility* of P (see first if-condition in Algorithm 1 in Fig. 4), 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 in Fig. 4) must be performed. Note that in an inter-tour move, the inner loop would *skip* the remaining vertices $s_j, \dots, d_j$ 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 $\sum_{i \in V(P)} q_i \le Q$ (always fulfilled in the intra-tour case). Assuming that the triangle inequality holds for travel times, route duration constraints necessarily require $\sum_{e \in E(P)} t_e \le T$. For the MDVRP, all this can be tested in $\mathcal{O}(1)$ by summing up demand and travel times for the segment $(d_i \rightsquigarrow i)$ (outer loop over i) and the 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 $(d_i \rightsquigarrow i, j \overleftsquigarrow s_i, s_j \rightsquigarrow d_i) = (P, s_j \rightsquigarrow d_i)$. 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* [14] 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 be done with *a priori* computed bounded candidate lists. This idea can be combined with pruning using the gain criterion. Finally, Irnich et al. [16] 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 sorted according to arc costs are created once in initialization. For the special case of costs defined by Euclidean distances, one can imagine the search being conducted starting from the closest neighbor and spiraling outwards until some *break* condition is met. (This break condition remains correct in the non-Euclidean case.) Note that the 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 in Fig. 4, the first if-condition for breaking the inner loop is never fulfilled for a non-constraining radius such as $\rho = \infty$. In this case, 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 in Fig. 4, 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 [12], 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. For a more detailed discussion of move decomposition we refer to Funke et al. [9,10].

Definition 1. [16] A move μ is *cyclic-independent* if $\mu = p_{\pi(1)} \circ p_{\pi(2)} \circ \cdots \circ p_{\pi(k)}$ for all cyclic permutations of $\{1, 2, \ldots, k\}$ whereas it is *cost-independent* if $g(\mu) = \sum_{i=1}^{k} g(p_i)$, where $g(p_i)$ is a *partial gain* associated with p_i.

A neighborhood with cyclic-independent and cost-independent moves can be searched with the gain criterion in light of the following theorem:

Theorem 1. *[19] 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 $1 \le \ell \le k$.*

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 Figs. 1(a) and 1(b). 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 in Fig. 4 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 easily overlooked 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 is a nuisance, but the latter implies that this is an incomplete examination of the compound condition (1). One way to make the search exhaustive is to additionally test over neighbors $s_j \in \mathbf{N}(s_i)$ such that $c_{s_j,s_i} < \rho$. 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 in Fig. 4 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 $s_j \in \mathbf{N}(s_i)$. That is, we instantiate move-specific lists that reflect incoming/outgoing arcs in combination with inclusion/exclusion of depot origin/destination vertices.

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.

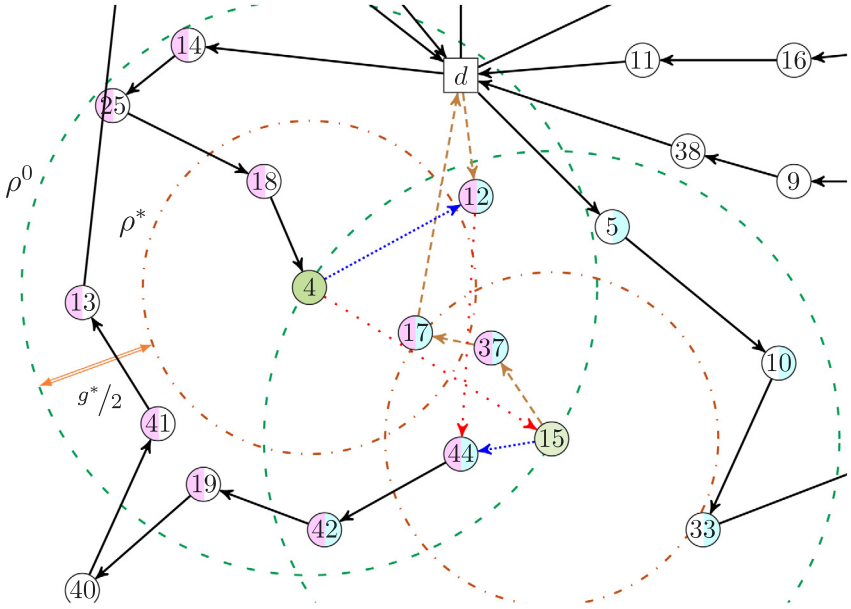


Fig. 5. Dynamic-radius search for **2-opt** where the first deleted arc is (4, 15), neighbors in $\mathbf{N}(4)$ and $\mathbf{N}(15)$ inspected with respect to threshold ρ^0 are highlighted. The selected move is an intra-depot (because $d_4 = d_{12} = d$) and inter-tour move. It deletes (12, 44) and inserts (4, 12) and (15, 44). At any time of the search, we have $\rho^* \le \rho \le \rho^0$ anywhere within the double arrow of width $g^*/2$ Some specific numbers are $c_{i,s_i} = c_{4,15} = 18.87$, $g^* = 17.28$, and $\rho^* = 18.87 - 17.28/2 = 10.23$ such that customer 40 is never tested whereas customer 13 is not if we already have identified a large enough gain.

3.2.3. Dynamic-radius search

As first discussed by Irnich et al. [16], 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 ([16, 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

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

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

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 the inner loop of Algorithm 2 in Fig. 4 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*.

Fig. 5 depicts part of an MDVRP solution in which we find an improving intra-depot 2-opt inter-tour move as seen in Fig. 1(b). 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 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 the 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 ρ 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 far fewer 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 $c_{i,s_i} = c_{j,s_j}$. Many 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 [8,25,27]. 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 $\mathcal{A}$.

Granular search can only guarantee that an improving move is found whenever one exists if the generator-arc list comprises all feasible arcs. 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 in Fig. 4.

The speed 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 Fig. 4 as well as 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 pre-selection of generator arcs, which can either be 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

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

2-opt*	Fig.	Gain g from edges												
Move type		Standard				Source				Sink				Exception
Intra-depot	1(c)	c_{i,s_i}	$-c_{j,s_i}$	$+c_{j,s_j}$	$-c_{i,s_j}$									
Inter-depot	2(a)	c_{i,s_i}	$-c_{j,s_i}$	$+c_{j,s_j}$	$-c_{i,s_j}$	$+c_{d_i,f_i}$	$-c_{d_j,f_i}$	$+c_{d_j,f_j}$	$-c_{d_i,f_j}$					
	2(b)	c_{i,s_i}	$-c_{j,s_i}$	$+c_{j,s_j}$	$-c_{i,s_j}$					$+c_{l_i,d_i}$	$-c_{l_i,d_j}$	$+c_{l_j,d_j}$	$-c_{l_j,d_i}$	
	2(c)	c_{i,s_i}	$-c_{j,s_i}$	$+c_{j,s_j}$				$+c_{d_j,f_j}$	$-c_{d_i,f_j}$					$-c_{d_j,s_j}$
	2(d)	c_{i,s_i}	$-c_{j,s_i}$	$+c_{j,s_j}$						$+c_{l_i,d_i}$	$-c_{l_i,d_j}$			$-c_{i,d_i}$
	2(e)	c_{i,s_i}		$+c_{j,s_j}$	$-c_{i,s_j}$	$+c_{d_i,f_i}$	$-c_{d_j,f_i}$							$-c_{d_i,s_i}$
	2(f)	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				Roles preserved				Roles interchanged				Exception
Intra-tour	1(a)	c_{i,s_i}	$-c_{ij}$	$+c_{j,s_j}$	$-c_{s_i,s_j}$									
Intra-depot	1(b)	c_{i,s_i}	$-c_{ij}$	$+c_{j,s_j}$	$-c_{s_i,s_j}$									
Inter-depot	3(a)	c_{i,s_i}	$-c_{ij}$	$+c_{j,s_j}$	$-c_{s_i,s_j}$	$+c_{l_i,d_i}$	$-c_{d_j,l_i}$	$+c_{d_j,f_j}$	$-c_{f_j,d_i}$					
	3(b)	c_{i,s_i}	$-c_{ij}$	$+c_{j,s_j}$	$-c_{s_i,s_j}$					$+c_{d_i,f_i}$	$-c_{d_j,f_i}$	$+c_{l_j,d_j}$	$-c_{l_j,d_i}$	
	3(c)	c_{i,s_i}	$-c_{ij}$	$+c_{j,s_j}$				$+c_{d_j,f_j}$	$-c_{f_j,d_i}$					$-c_{d_j,s_j}$
	3(d)	c_{i,s_i}		$+c_{j,s_j}$	$-c_{s_i,s_j}$							$+c_{l_j,d_j}$	$-c_{l_j,d_i}$	$-c_{d_j,j}$
	3(e)	c_{i,s_i}		$+c_{j,s_j}$	$-c_{s_i,s_j}$	$+c_{l_i,d_i}$	$-c_{d_j,l_i}$							$-c_{i,d_i}$
	3(f)	c_{i,s_i}	$-c_{ij}$	$+c_{j,s_j}$						$+c_{d_i,f_i}$	$-c_{d_j,f_i}$			$-c_{s_i,d_i}$

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

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 Figs. 1(a) and 1(b), 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 Fig. 2(a)–2(f) and 3(a)–3(f). 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) reflect the left-/right-side structure and refer, respectively, to move-defining, exchanged arcs of the *standard* cases 1(c), 2(a) and 2(b) of 2-opt* and *standard* cases 1(b), 3(a) and 3(b) of 2-opt, the arcs exchanged at the *source* (resp. roles preserved) and the *sink* (resp. roles interchanged) to repair the otherwise infeasible depot assignment, and the added arc of the *exception* cases that differs from the standard cases. It is obvious that the gain computations differ significantly from one another and especially compared to the standard ones.

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 Fig. 1(a)–1(c). 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 correction term. We now distinguish between the 2-opt* and the 2-opt neighborhoods.

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

Conditions on i and s_i	2-opt* (Fig. 2)		2-opt (Fig. 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

4.1. 2-Opt* moves

Recall that Figs. 1(c) and 2(a)–2(f) 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 1(c) is given by

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

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

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 2(e) and 2(f), 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 2(c) is symmetric in the affected routes to case 2(e) and likewise 2(d) to 2(f). 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 2(e) or 2(f), it is found by their symmetric counterpart. Our case-by-case analysis works as follows.

In the standard source case 2(a), 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 parts in parentheses. 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:

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

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 a nearest depot.

In the standard sink case 2(b), 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

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

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

In the exception case 2(c), 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 2(e) can be found anyway. The following threshold bounds from above the gain seen in case 2(c) thus fulfilling the latter requirement:

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

where $N(r)$ is the set of customers in route r. Moreover, 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 parentheses describe 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 parentheses describe some customer j for which short-cutting from d_j to s_j is favorable.

The exception case 2(d) slightly differs from the former case 2(c) 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 2(f). The resulting radius condition becomes

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

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 Eqs. (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 the 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):

$$\rho_{MD} = \rho + \max\left(\begin{cases} \tau_c & \text{if } i = d_i \\ \tau_a & \text{if } i \neq d_i \end{cases} , \begin{cases} 0 & \text{if } s_i = d_i \\ \max\{\ \tau_b\ ,\ \tau_d\ \} & \text{if } s_i \neq d_i \end{cases} \right). \tag{6}$$

In summary, 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 including inter-depot moves.

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 Fig. 3 together with their respective gains in Table 1b. Exception cases 3(c) and 3(f) can indeed be merged into a one-sided test because the arc costs are symmetric (same for 3(d) and 3(e)). The following correction terms cover the relevant cases 3(a), 3(b), 3(c), and 3(e):

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

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

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

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

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

$$\rho_{MD} = \rho + \max\left(\begin{cases} 0 & \text{if } i = d_i \\ \tau_b & \text{if } i \neq d_i \end{cases} , \begin{cases} \tau_c & \text{if } s_i = d_i \\ \max\{\ \tau_a\ ,\ \tau_e\ \} & \text{if } s_i \neq d_i \end{cases} \right), \tag{8}$$

where the same base term $\rho = c_{i,s_i} - \gamma/2$ (see (3)) only depends on the deleted arc (i, s_i). Furthermore, all the precomputed optima used in the partial terms τ_a, τ_b, τ_c, and τ_e would be decoupled if the deleted arc was changing. We point back to the two implementation choices discussed in Section 3.2.2 where using two distinct inner loops favors low-level efficiency.

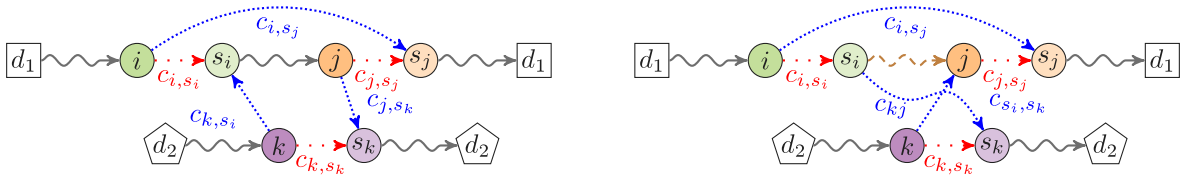
5. Iterated local search

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, 20] as it combines local search with a *perturbation* mechanism. Local optima are perturbed 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 artificially construed as a soft constraint by tolerating up to δm, $\delta \geq 1$, vehicles per depot in all components of the metaheuristic, i.e., construction, local search, and perturbation. We therefore speak of δ-fleet-size feasibility, where a solution R fulfills the instance requirements if $\delta = 1$. In addition, we introduce an intermediate *fleet-reduction operation* between local search and perturbation in an effort to recover fleet-size feasibility. This operation is allowed to fail in which case we do not obtain a local optimum.

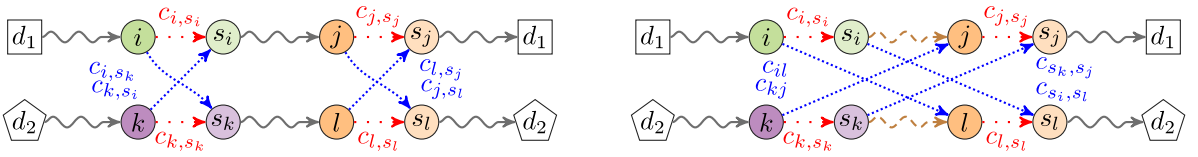
We then perform a perturbation operation to induce relatively significant routing changes in the current solution. The main complication is to maintain δ-fleet-size feasibility which we handle with a fail-safe savings heuristic. This explains why we had to design the ILS with a somewhat more involved 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.



(a) Ordered chain. (b) Inverted chain.

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



(a) Ordered chains. (b) Inverted chains.

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

5.1. Construction heuristic

Our construction procedure is based on the *savings* heuristic of Clarke and Wright [4]. 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 $\sigma_{ij}^d = -\lambda c_{ij} + c_{di} + c_{jd} + \zeta(c_{di} - c_{jd})$, where λ and ζ influence the comparison between the cost of arc (i, j) and of connecting i and j to d. We reject those combinations where (d, i, j, d) is an infeasible route. To further randomize the procedure, for each arc (i, j), we randomly take from these $|D|$ depot-specific savings values $(\sigma_{ij}^d)_{d \in D}$ 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 the parameter described in the introduction of Section 5.

The above procedure is repeated with $\delta = 1.25$ and new random parameters ζ and λ until the constructed solution R is δ-fleet feasible.

5.2. Local search

A reasonable local search-based ILS 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 [see, e.g.,1,10]. Figs. 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.

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, i.e., whenever an improving move is returned from a neighborhood exploration, it is performed and we continue the local search using the next neighborhood [11]. Moreover, we use a best-improvement strategy. Local search terminates when all neighborhoods are explored without success, so that the returned solution R is 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: Exactly $2\lfloor \delta m \rfloor$ copies are needed per depot $d \in D$ (two for each route, see Section 2).

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
```

5.3. Perturbation and fleet-reduction

The perturbation consists of a multi-phase re-clustering on a δ-fleet-size feasible solution R. 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 1-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 likely 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. This modified solution is then perturbed irrespective of its feasibility status, but we do save it as a local optimum if the fleet-reduction operation is successful.

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.

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, the algorithm moves 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 with the previous dynamic-radius search implementation follows in Section 6.2. We then take a look in Section 6.3 at an alternative way to recover inter-depot 2-opt and 2-opt* moves that foregoes both the challenging implementation and the correction term. The impact of the correction term is analyzed in Section 6.4 by evaluating its contribution under various usage scenarios.

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) $\times$ 4 (load factor) $\times$ 14 (size)) from Irnich et al. [16], where the load factor f is the quotient of the capacity Q and the average demand q_i of the customers $i \in N$. As a result, we see on average approximately f customers per route in good solutions.

We then look at how dynamic-radius search behaves with respect to multi-depots and the presence or not of the correction terms using the proposed ILS on commonly used MDVRP instances from the literature:

- 33 MDVRP instances (p01–pr10) from Cordeau et al. [5];

The CVRP instances from the previous work [16] are available at https://logistik.bwl.uni-mainz.de/research/benchmarks/.

6.2. Improved implementation

Dynamic-radius search has been re-implemented with a greater focus on low-level efficiency but we also gave due attention to lexicographic search. We believe that technological progress since 2006 cannot single-handedly explain the improvement that have been incorporated. We support this claim by replicating the previous results [16, Figures 7 and 9] with our improved implementation. Figs. 8 and 9 therefore refer to the same 560 CVRP instances but we distinguish ordered/inverted variants of Or-opt and string exchange neighborhood while also increasing the allowed string lengths from 3 to 5. These figures compare lexicographic and radius search by analyzing two indicators: the acceleration factor describing the average ratio of computing times needed with lexicographic compared to dynamic-radius search (ratios are dimensionless) and 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 the *load factor* f. Therefore, the results presented in Figs. 8 and 9 are exactly grouped by both number $|N|$ of customers and load factor f. In the former figure, a horizontal dashed line positioned at 1 on the ordinate would separate observations where dynamic-radius search is comparatively slower (which never happens here). As we just elicited, one would expect lexicographic search to take over when it is used on shorter routes.

Table 3 compares the result numbers from [16] to those of this paper. Specifically, we compare the largest numbers for both the acceleration factor and the search time for every comparable neighborhood regardless of the instance size. Furthermore, since the exact numbers from the previous study are not available, we rounded the pixel ratio of the y-axis to reasonable values. All else being equal, a larger number for the acceleration factor is better, whereas a smaller one is preferred for the search times (in milliseconds). Let us explain why there might appear to be contradictory results at first glance. For instance, comparing relative performance to lexicographic search with the 2-opt neighborhood (old factor 32 vs. new factor 18.8) and the string exchange neighborhood (800 vs. 427.9), it appears that the benefit of dynamic-radius search has slightly decreased. This is due to significant improvements in the lexicographic search implementation and the increased maximum string length from $L = 3$ to 5 favoring the lexicographic paradigm.

In any case, we achieve a reduction in absolute computation time by roughly a factor of 10 or more for every neighborhood including those with larger maximal string lengths L. As a case in point, the average search times of the swap neighborhood (62 vs. 4.7) and string exchange neighborhood (310 vs. 7.4) give a ratio of 13.2 and 41.9 respectively. These ratios account for both technological and algorithmic changes. For perspective, Passmark (https://cpubenchmark.net/compare) rates the Intel Pentium 4@2.40 GHz single thread performance at 360, whereas that of our processor is 2301, which gives a factor of 6.4. This means that any speedup larger than 6.4 is likely to incorporate algorithmic progress. We point back to the aforementioned transition to an explicit double inner-loop design, which allows finer treatment of depot arcs, see Section 3.2.2. 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. The particularly large ratio of the string exchange neighborhood can be explained by the fact that we have incorporated feasibility pruning inside the string examinations of dynamic-radius search. It is also worth to underscore that this significant improvement is reflected in the trend lines of Fig. 8, which no longer show any significant negative slope when increasing the number of customers. This observation is in fact generally true for every neighborhood under test except Or-opt (ordered and inverted) with $L = \infty$.

As a last note, we rectify a mistake in the pseudo-code [16, 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 **2** in the last denominator.

Finally, Fig. 10 shows that this behavior remains consistent on the benchmark MDVRP instances of Cordeau et al. [5]. In particular, we see that despite being considerably smaller than 2500, the largest instance (#23 with 360 customers and 9 depots) benefits comparatively more from dynamic-radius search. We also use the same plot symbol as for load factor $f = 25$, since it is usually does not surpass 10 in these instances.

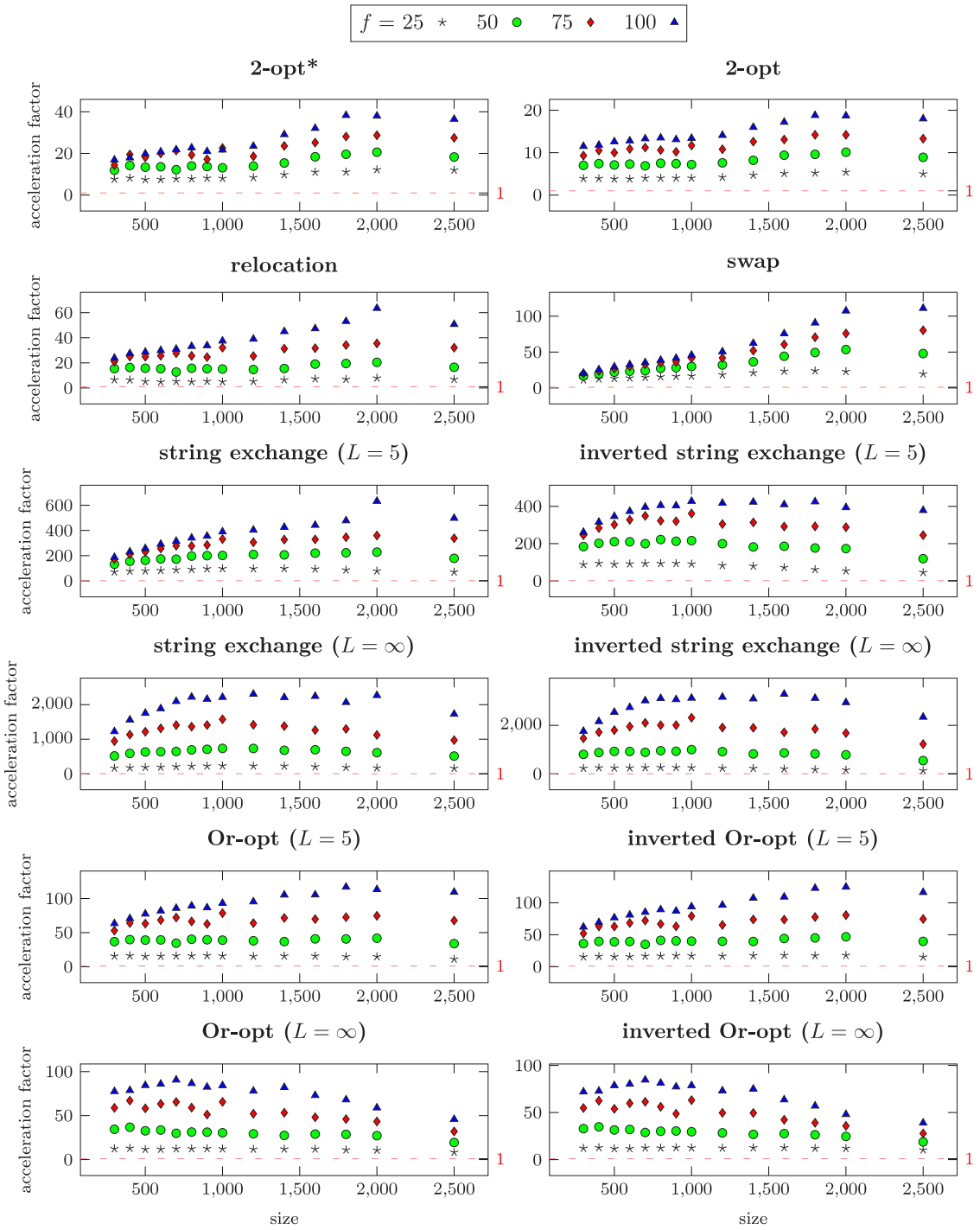


Fig. 8. *Average acceleration factor* of dynamic-radius search over lexicographic search for various neighborhood operators and instances ranging from 300 to 2500 in customer size.
Source: CVRP, instances from Irnich et al. [16].

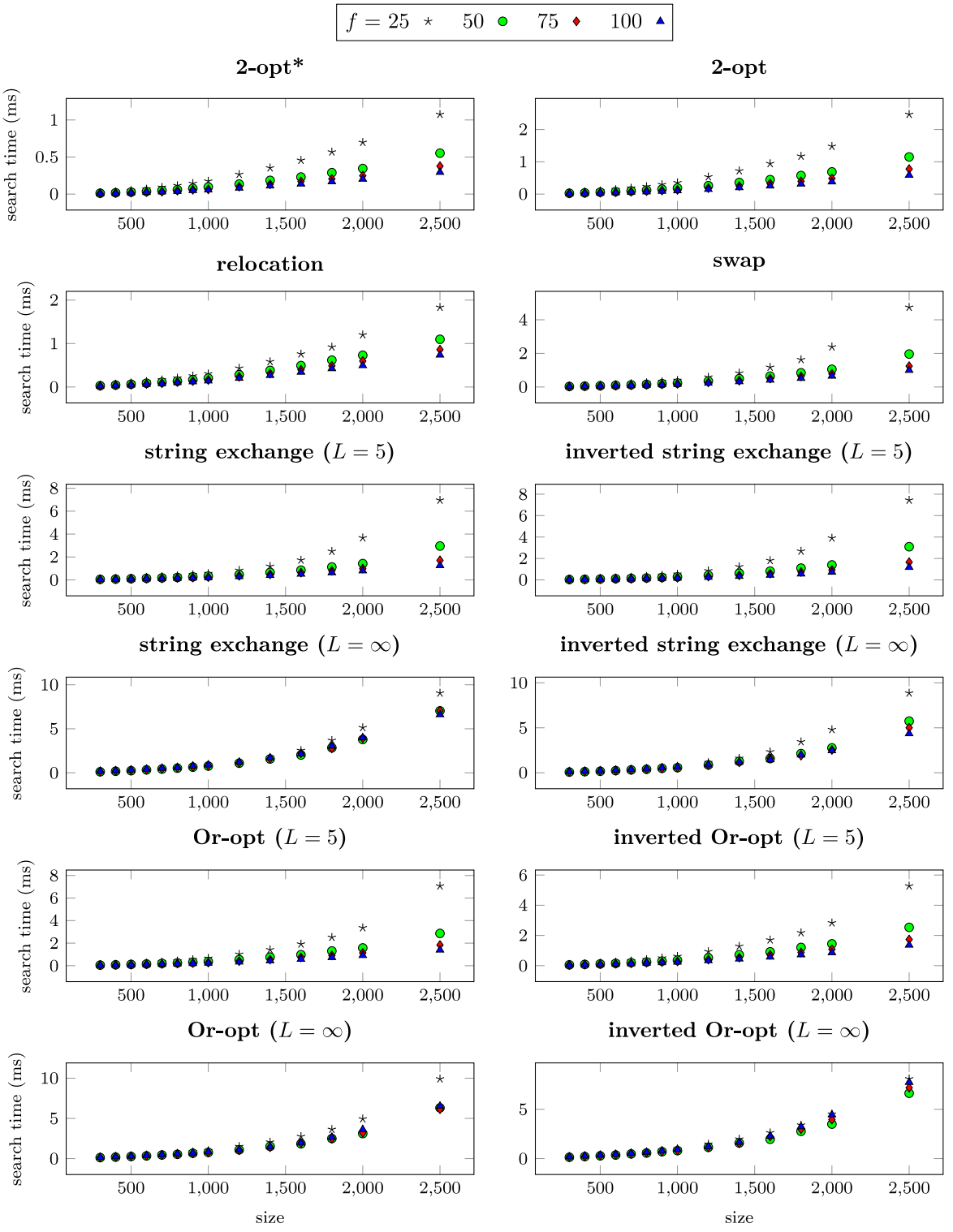


Fig. 9. *Average neighborhood exploration time* (in milliseconds) of dynamic-radius search for various neighborhood operators and instances ranging from 300 to 2500 in customer size.
Source: CVRP, instances from Irnich et al. [16].

Table 3
Comparison of computational results with the previous implementation used in [16].

		2-opt*	2-opt	Relocation	Swap	Inverted string exchange	Inverted Or-opt
Acceleration factor	2006	95	32	64	97	800	128
	This paper	38.3	18.8	63.5	111.0	427.9	124.7
	Ratio	2.5	1.7	1.0	0.9	1.9	1.0
Search time (ms)	2006	11	24	16	62	310	54
	This paper	1.1	2.5	1.8	4.7	7.4	5.3
	Ratio	10.0	9.6	8.9	13.2	41.9	10.2

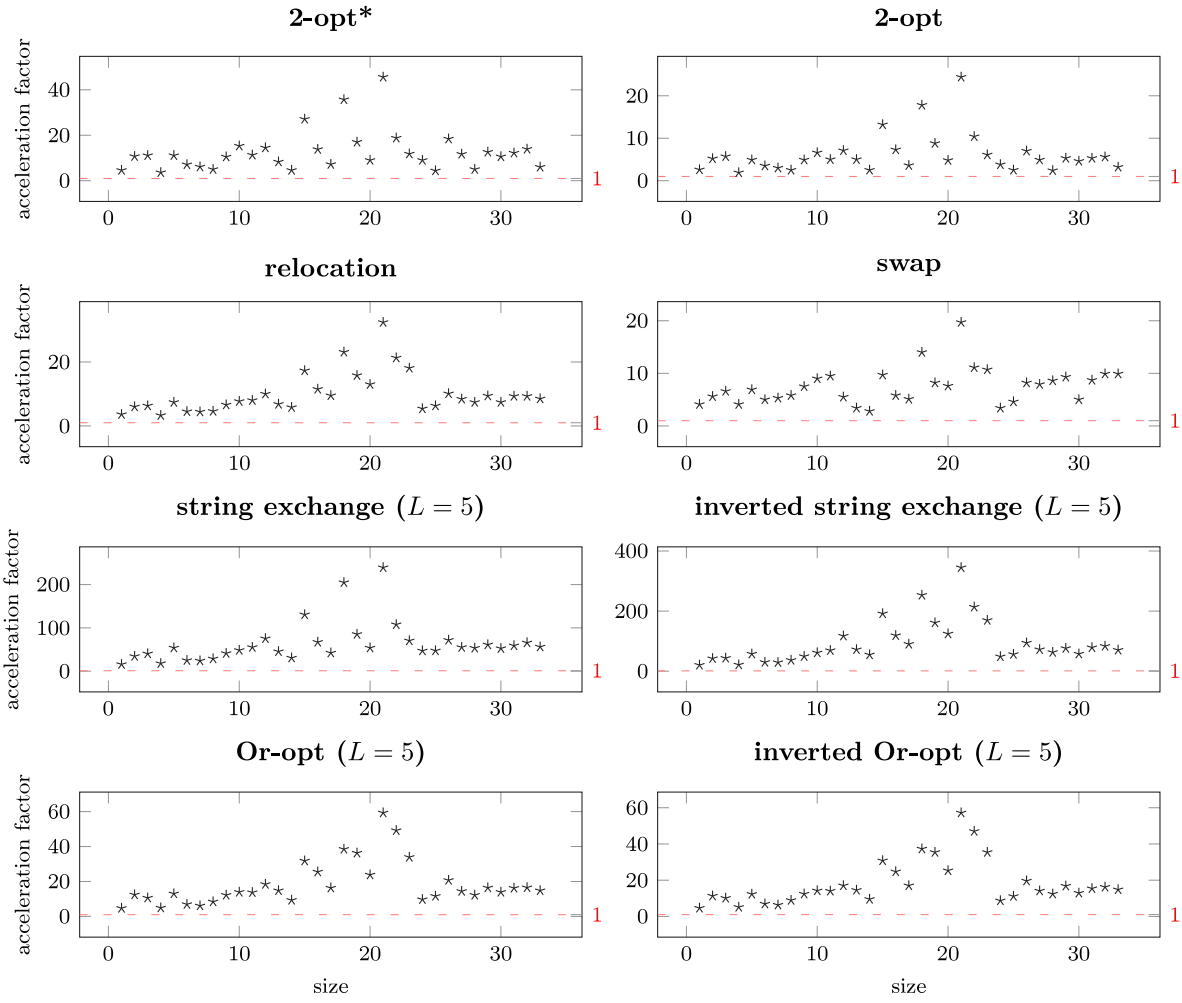


Fig. 10. *Average acceleration factor* of dynamic-radius search over lexicographic search for various neighborhood operators and instances ranging from 48 to 360 in customer size.
Source: MDVRP, instances from Cordeau et al. [5].

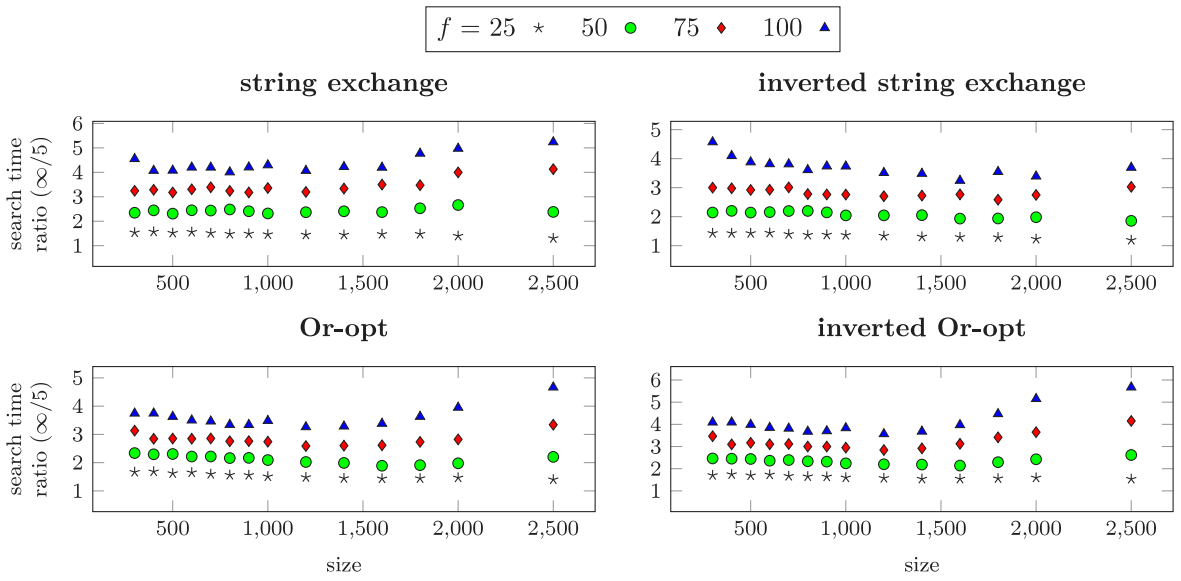


Fig. 11. Dynamic-radius search computing time comparison of allowed length $L = \infty$ over $L = 5$ in string exchange and Or-opt neighborhood explorations for instances ranging from 300 to 2500 in customer size.
Source: CVRP, instances from Irnich et al. [16].

6.3. Inter-depot moves via string exchange and Or-opt

In this section, we consider alternative implementation choices for inter-depot 2-opt* and 2-opt moves. More specifically, 2-opt* moves (Fig. 2) can be reproduced by ordered variants of the string exchange (standard cases 2(a), 2(b)) and Or-opt (exception cases 2(c), 2(d), 2(e), 2(f)) moves. In particular, one can see that the string $f_i \rightsquigarrow i$ in Fig. 2(e) (2-opt*) can have any length and corresponds to the string $s_i \rightsquigarrow j$ in Fig. 6(a) (ordered Or-opt). With respect to 2-opt moves (Fig. 3), inverted variants of the string exchange and Or-opt respectively reproduce standard and exception cases.

Implementing string exchange and Or-opt moves seems to be simpler than implementing inter-depot 2-opt* and 2-opt moves. This raises the question of whether the implementation burden is worth it if the former neighborhoods are already available. Moreover, for exact dynamic-radius search, the implementation includes the computation of the correction terms. 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 in the candidate solution. The results of the comparison between the maximum string length $L = \infty$ and $L = 5$ are shown in Fig. 11. We ultimately observe a factor of at least 1.5 which reaches 4 on instances with load factor $f = 100$.

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 versus 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 Fig. 9) must be compounded with the observed ratios of Fig. 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. Inter-depot moves and the correction term

In this section, we investigate the central hypotheses that we already discussed in Section 1 as a main motivation for our work:

Hyp1: Allowing inter-depot moves helps finding better *local optima* on average.
Hyp2: Allowing inter-depot moves helps finding better *best solutions* on average.

To this end, we computationally compare five implementations of ILS that differ in the way in which the 2-opt and 2-opt* neighborhoods are explored, i.e., the only neighborhoods that implement inter-depot moves. More precisely, we examine how the following five options *With*, *Without*, *Last Resort*, *Random*, and *Forbid* perform against each other by comparing the respective relative gap measures:

(i): Option *With* (W) uses the correction term (6) and thus guarantees that a move with provably maximum gain is identified
(ii): Option *Without* (WO) uses the simple radii of the intra-depot cases, i.e., (3) and (4) It loses the guarantee but still requires the whole machinery of the depot repair operation
(iii): Option *Last Resort* (LR) uses option *Without* until no improving move is found at which point it switches to option *With*
(iv): Option *Forbid* (F) literally forbids inter-depot moves from happening thus eliminating the need for the repair implementation
(v): Option *Random* (R) uniformly picks among those four options at every local search iteration.

More specifically, all algorithmic settings are kept fixed except for the option studied. In particular, we use a best-improvement strategy for local search plus a perturbation mechanism which means that we are warm-starting at almost every iteration. It is clear that, for a fixed number n_{ILS} of iterations, the five options differ in the computation time consumed by the respective ILS. Formally, we have two vectors of result data $[z^I_{no}]$ and $[t^I_{no}]$, where $n \in \{1, \dots, n_{\text{ILS}}\}$ denotes an iteration, o an option, and I an instance. The vectors respectively give the local optima we have found at each iteration and the total time it takes since the beginning. In order to be able to summarize the benchmark results, Fig. 12 illustrates how we normalize the computation log over $n_{\text{ILS}} = 50{,}000$ iterations for each option on a specific instance I. On the left, we have iterations on the abscissa and objective values on the ordinate. Normalizing the ordinate is fairly intuitive since we simply compute relative gaps with respect to the instance's best known solution $\bar{z}^I$:

$$\gamma^I(z^I_{no}) = 100(z^I_{no} - \bar{z}^I)/\bar{z}^I.$$

Normalizing the abscissa in the range $[0, 100]$ allows to compare where we stand for each option with respect to a computation time budget. Since every option impacts the search time of neighborhood explorations, the cumulative time difference across the options can become significant. Each circle marker indicates the last iteration index for which the computation time of the fastest option, say τ^I, does not exceed that of the corresponding option, i.e., $\arg\max_{n\in\{1,\dots,n_{\text{ILS}}\}} t^I_{no} \le \tau^I$. In Fig. 12(a), the ILS with option WO is the fastest whereas option R is comparatively out of time around iteration 40,000. In Fig. 12(b), one can see the time normalization as pulling rightwards the curve of every option by its circle marker until it reaches 100%, thus more or less keeping the shape of its iteration-based plot but also neglecting all points after from further comparison. By construction, the curve of option WO is the reference with respect to time and thus does not get modified. Formally, we define, for each option o and instance I, the function $h^I_o : [0, 100] \mapsto \mathbb{R}_+$ giving the relative gaps in percent of the average local optima up to a percent time t, given by

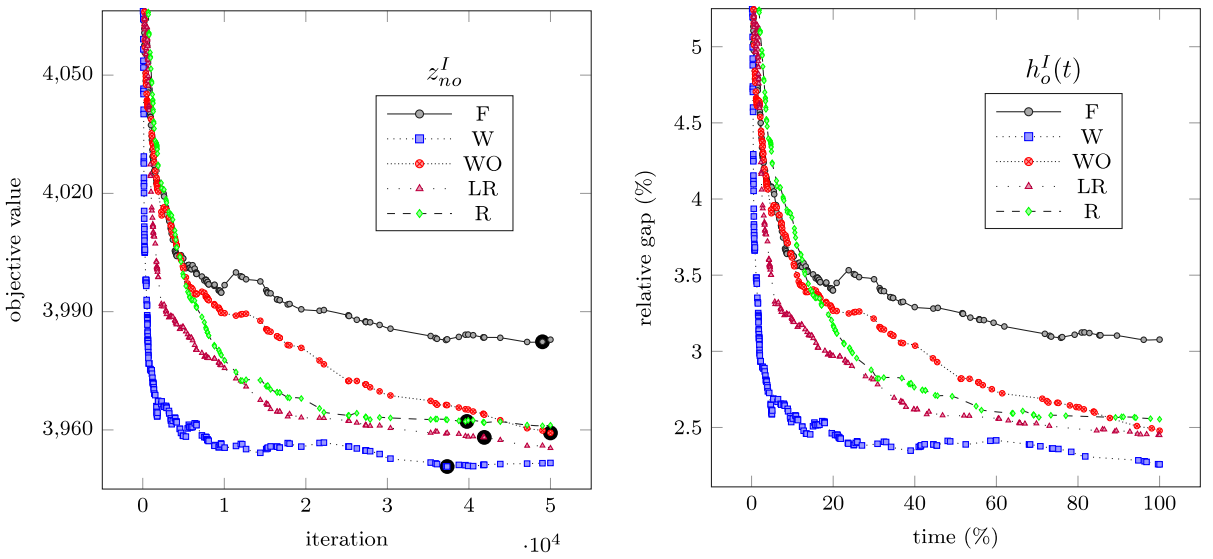
$$h^I_o(t) = \frac{1}{\nu^I_{to}} \sum_{n=1}^{\nu^I_{to}} \gamma^I(z^I_{no}) \tag{9}$$

where $\nu^I_{to} = \arg\max_{n\in\{1,\dots,n_{\text{ILS}}\}} \left\{ t^I_{no} \le t\,\% \cdot \tau^I \right\}$ denotes the largest iteration index that is completed within $t\,\%$ of the normalized time.

From Fig. 12(b) we see that the function h^I_o is generally not monotone but it does have a decreasing trend, which can be attributed to the design of the metaheuristic, e.g., the acceptance criterion and the perturbation that reuses solution parts. If randomized initial solutions were used at every iteration, we would by the law of large numbers expect rather horizontal lines in Fig. 12 after sufficiently many iterations.

Overall average quality of local optima and hypothesis Hyp1. The normalization allows us to aggregate the functions h^I_o over several MDVRP instances for a given option o. Recall that hypothesis Hyp1 refers to the *average* quality of local minima. Formally, let $\mathcal{I}$ be a finite set of instances $I \in \mathcal{I}$. The aggregated function is

$$h_o(t) = \frac{1}{|\mathcal{I}|} \sum_{I\in\mathcal{I}} h^I_o(t), \tag{10}$$



(a) Iteration and average objective value.

(b) Axes normalized; functions $h^I_o(t)$.

Fig. 12. Normalizing iterations and average objective values of five options for a specific MDVRP instance *I*.
Source: MDVRP, instances from Cordeau et al. [5].

and the upper part of Fig. 13 shows the result for the benchmark instances described in Section 6.1. We can see that overall averages are worst in ILS with options *Forbid* and *Random*, while they are significantly better with options *With*, *Without*, and *Last Resort*. Moreover, differences between the latter three ILS are minor. The point is that options *With*, *Without*, and *Last Resort* are those that systematically allow inter-depot moves. This confirms our leading hypothesis Hyp1, i.e., the ability to identify better gains by allowing inter-depot moves pays off with regard to the average quality of local minima.

Overall average quality of best solutions and hypothesis Hyp2. To analyze the average quality of best solutions found, we introduce functions f^I_o and f_o in a similar manner as (9) and (10):

$$f^I_o(t) = \min_{n\in\{1,\dots,\nu^I_{to}\}} \gamma(z^I_{no}) \qquad \text{and} \qquad f_o(t) = \frac{1}{|\mathcal{I}|}\sum_{I\in\mathcal{I}} f^I_o(t). \tag{11}$$

The functions f^I_o and herewith also f_o for the incumbent relative gap are monotonically non-increasing for all options o by definition.

We can see in the lower part of Fig. 13 that the best local optima plots are tightly bunched together fairly early on. For the sake of better visibility, we added markers to the plots in Figs. 12 and 13, even though functions h^I_o and f^I_o are defined for continuous values (see (9) and (11)). These coordinates are established by filtering out every iteration for which none of the incumbents across any of the options is improved within 0.01% which explains why the plots get sparser towards the right. This means that the changes in average relative gap values do not appear where the minimum functions f^I_o are plateaus. This void is instead filled by interpolation simply for easily keeping track of each curve. Deriving coordinates for the aggregated functions follows the same treatment but we additionally match time percentages within some small tolerance across all instances and options simultaneously to derive an average relative gap coordinate for the whole benchmark.

There are several results that were completely unexpected for us:

- The difference between the ILS equipped with the five different options is very small It means that although the average quality of local optima is improved with 2-opt and 2-opt* moves that allow a depot swap, average quality does not translate into better overall quality Hypothesis Hyp2 is not confirmed but rather proven false.
- The ILS with option *With* (W) is performing worst, while the ILS with option *Without* (WO) is performing best Even though options W and WO allow inter-depot 2-opt and 2-opt* moves, it is hard to make an option recommendation when compared to option *Forbidden* (F) which is somewhere in between The relationship shown with f_o is really inconsistent with the relationship shown with h_o.

Retrospectively, we can offer the following possible explanation for the unexpected outcome: For the overall best solution, we only need to find an excellent local optimum one time. It seems to be more important to inspect more local optima (even if not ensured that they are true optima) than proving that the solutions cannot be improved with the given neighborhoods. The explanation goes partly in line with the computation times that we can see in Fig. 12(a). The ranking

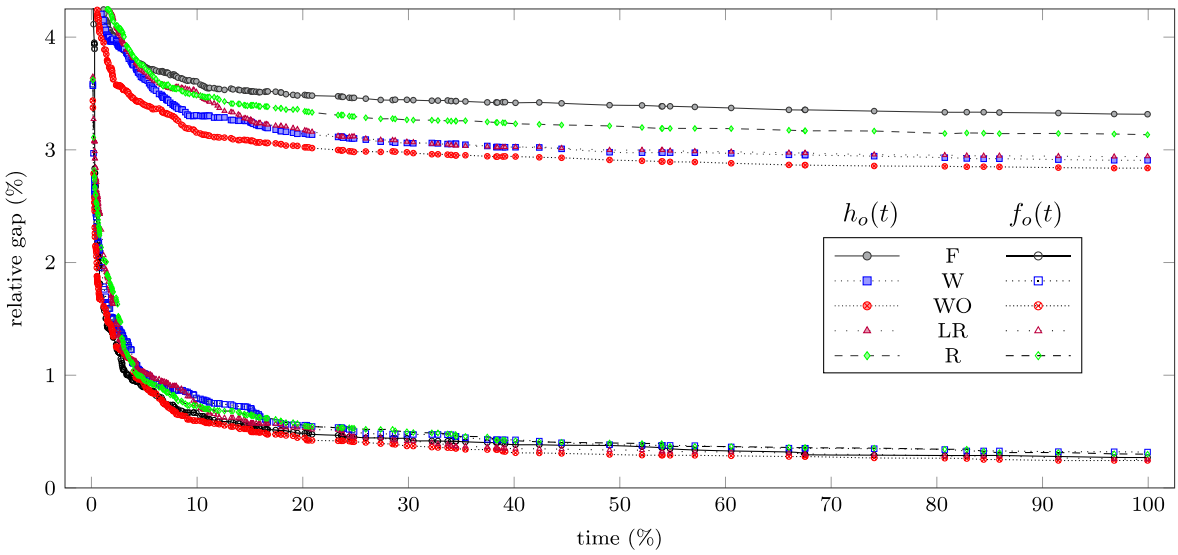


Fig. 13. Average and minimum relative gaps evolution over normalized axes averaged across benchmark instances for five options. *Source:* MDVRP, instances from Cordeau et al. [5].

by decreasing speed of the options is WO, F, LR, R, and W, while the ranking by increasing best local optima WO, F and LR, R and W (the latter pairs are performing equally well). This outcome is very consistent for values of t between 20 and 100.

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 a neighborhood is explored: A lexicographic search prunes an 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*, more specifically, their exploration which 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 to identify a provably best-improving move, either intra-depot or inter-depot, with little additional computational effort. Such inter-depot moves can also be found without the repair machinery but we found that it is a computationally expensive neighborhood exploration that results from the need to increase the allowed string length in string exchange and Or-opt neighborhoods.

Our improved implementation is considerably faster than the original from Irnich et al. [16] on the capacitated vehicle routing problem for both lexicographic and dynamic-radius search. While the relative competitiveness to lexicographic search is sometimes consequently reduced, we still observe acceleration factors in the order of hundreds or even thousands for some neighborhoods. Furthermore, these factors are generally scaling well with larger instances. Moreover, the adaptation to the multi-depot vehicle routing problem benefits in the same orders of magnitude.

While there certainly are some fancier metaheuristics out there, we believe our basic iterated local search implementation is legitimate enough with respect to behavior one would observe in state-of-the-art metaheuristics. In particular, we conjectured and have later shown that identifying moves with better gains lead to better local optima on average. We also conjectured that better average local optima translate to better incumbent solutions in local search-based metaheuristics like the iterated local search that we implemented. This behavior unfortunately does not hold true, proving our second hypothesis as false. Even if false, this is an important insight that has not been reported so explicitly in the literature.

Clearly, the overarching metastrategy does strongly influence the empirical behavior we illustrate in our analysis. This means that the metaheuristic's components like the initial solution construction, the acceptance criterion, and the depth of local search including the choice of neighborhoods all impact the results. However, there is absolutely no reason to believe that switching to another search paradigm like lexicographic would yield any difference. (The metaheuristic would just become slower and therefore lose competitiveness.)

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

in which 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.

Data availability

No data was used for the research described in the article.

Acknowledgments

This research was funded by the Deutsche Forschungsgemeinschaft (DFG), Germany under grant nos. IR 122/7-1 and IR 122/7-2 of project 315139873.

References

[1] Emile H.L. Aarts, Jan Karel Lenstra (Eds.), Local Search in Combinatorial Optimization, John Wiley & Sons, New York, 1997.
[2] Mandell Bellmore, Saman Hong, Transformation of multisalesmen problem to the standard traveling salesman problem, J. Assoc. Comput. Mach. 21 (3) (1974) 500–504, http://dx.doi.org/10.1145/321832.321847.
[3] Jon Louis Bentley, Fast algorithms for geometric traveling salesman problems, ORSA J. Comput. 4 (4) (1992) 387–411, http://dx.doi.org/10.1287/ijoc.4.4.387.
[4] Geoff Clarke, J.W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, Oper. Res. 12 (4) (1964) 568–581, http://dx.doi.org/10.1287/opre.12.4.568.
[5] Jean-François Cordeau, Michel Gendreau, Gilbert Laporte, A tabu search heuristic for periodic and multi-depot vehicle routing problems, Networks 30 (2) (1997) 105–119, http://dx.doi.org/10.1002/(SICI)1097-0037(199709)30:2<105::AID-NET5>3.0.CO;2-G.
[6] G.A. Croes, A method for solving traveling-salesman problems, Oper. Res. 6 (6) (1958) 791–812, http://dx.doi.org/10.1287/opre.6.6.791.
[7] Guy Desaulniers, Oli B.G. Madsen, Stefan Røpke, The vehicle routing problem with time windows, in: Paolo Toth, Daniele Vigo (Eds.), Vehicle Routing, SIAM, Philadelphia, 2014, pp. 119–159, http://dx.doi.org/10.1137/1.9781611973594.ch5, Chapter 5.
[8] John Willmer Escobar, Rodrigo Linfati, Paolo Toth, Maria G. Baldoquin, A hybrid granular tabu search algorithm for the multi-depot vehicle routing problem, J. Heuristics 20 (5) (2014) 483–509, http://dx.doi.org/10.1007/s10732-014-9247-0.
[9] Birger Funke, Tore Grünert, Stefan Irnich, Local search for vehicle routing and scheduling problems: Review and conceptual integration, J. Heuristics 11 (4) (2005) 267–306, http://dx.doi.org/10.1007/s10732-005-1997-2.
[10] Birger Funke, Tore Grünert, Stefan Irnich, A note on single alternating cycle neighborhoods for the TSP, J. Heuristics 11 (2) (2005) 135–146, http://dx.doi.org/10.1007/s10732-005-0713-6.
[11] Pierre Hansen, Nenad Mladenović, Raca Todosijević, Saïd Hanafi, Variable neighborhood search: Basics and variants, EURO J. Comput. Optim. 5 (3) (2017) 423–454, http://dx.doi.org/10.1007/s13675-016-0075-x.
[12] Keld Helsgaun, An effective implementation of the Lin-Kernighan traveling salesman heuristic, European J. Oper. Res. 126 (1) (2000) 106–130, http://dx.doi.org/10.1016/s0377-2217(99)00284-2.
[13] Holger H. Hoos, Thomas Stützle, Stochastic Local Search: Foundations and Applications, in: The Morgan Kaufmann Series in Artifical Intelligence, Elsevier, San Francisco, 2005, http://dx.doi.org/10.1016/B978-1-55860-872-6.X5016-1.
[14] Stefan Irnich, Resource extension functions: Properties, inversion, and generalization to segments, OR Spectrum 30 (1) (2008) 113–148, http://dx.doi.org/10.1007/s00291-007-0083-6.
[15] Stefan Irnich, A unified modeling and solution framework for vehicle routing and local search-based metaheuristics, INFORMS J. Comput. 20 (2) (2008) 270–287, http://dx.doi.org/10.1287/ijoc.1070.0239.
[16] Stefan Irnich, Birger Funke, Tore Grünert, Sequential search and its application to vehicle-routing problems, Comput. Oper. Res. 33 (8) (2006) 2405–2429, http://dx.doi.org/10.1016/j.cor.2005.02.020.
[17] David S. Johnson, Lyle A. McGeoch, The traveling salesman problem: a case study, in: Emile H.L. Aarts, Jan Karel Lenstra (Eds.), Local Search in Combinatorial Optimization, John Wiley & Sons, New York, 1997, pp. 215–310, http://dx.doi.org/10.1515/9780691187563-011, Chapter 8.
[18] Shen Lin, Computer solutions of the traveling salesman problem, Bell Syst. Tech. J. 44 (10) (1965) 2245–2269, http://dx.doi.org/10.1002/j.1538-7305.1965.tb04146.x.
[19] Shen Lin, Brian Wilson Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Oper. Res. 21 (2) (1973) 498–516, http://dx.doi.org/10.1287/opre.21.2.498.
[20] Helena R. Lourenço, Olivier C. Martin, Thomas Stützle, Iterated local search, in: Fred Glover, Gary A. Kochenberger (Eds.), Handbook of Metaheuristics, Springer, Boston, 2002, pp. 321–353, http://dx.doi.org/10.1007/0-306-48056-5_11.
[21] Olivier Martin, Steve W. Otto, Edward William Felten, Large-step Markov chains for the TSP incorporating local search heuristics, Oper. Res. Lett. 1 (4) (1992) 219–224, http://dx.doi.org/10.1016/0167-6377(92)90028-2.
[22] Jean-Yves Potvin, Jean-Marc Rousseau, An exchange heuristic for routeing problems with time windows, J. Oper. Res. Soc. 46 (12) (1995) 1433–1446, http://dx.doi.org/10.2307/2584063.
[23] Gerhard Reinelt, The Traveling Salesman: Computational Solutions for TSP Applications, in: Lecture Notes in Computer Science, vol. 840, Springer, Berlin Heidelberg, 1994, http://dx.doi.org/10.1007/3-540-48661-5.
[24] Martin W.P. Savelsbergh, An efficient implementation of local search algorithms for constrained routing problems, European J. Oper. Res. 47 (1) (1990) 75–85, http://dx.doi.org/10.1016/0377-2217(90)90091-O.
[25] Michael Schneider, Fabian Schwahn, Daniele Vigo, Designing granular solution methods for routing problems with time windows, European J. Oper. Res. 263 (2) (2017) 493–509, http://dx.doi.org/10.1016/j.ejor.2017.04.059.
[26] Kenneth Steiglitz, 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, Urbana, IL, USA, 1968, pp. 814–821.
[27] Paolo Toth, Daniele Vigo, The granular tabu search and its application to the vehicle-routing problem, INFORMS J. Comput. 15 (4) (2003) 333–346, http://dx.doi.org/10.1287/ijoc.15.4.333.24890.
[28] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Christian Prins, A unified solution framework for multi-attribute vehicle routing problems, European J. Oper. Res. 234 (3) (2014) 658–673, http://dx.doi.org/10.1016/j.ejor.2013.09.045.
[29] Thomas R. Visser, Remy Spliet, Efficient move evaluations for time-dependent vehicle routing problems, Transp. Sci. 54 (4) (2020) 1091–1112, http://dx.doi.org/10.1287/trsc.2019.0938.