Andelmin, Juho; Bartolini, Enrico

A multi-start local search heuristic for the green vehicle routing problem based on a multigraph reformulation

Published in:
Computers and Operations Research

DOI:
10.1016/j.cor.2019.04.018

Published: 01/09/2019

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Andelmin, J., & Bartolini, E. (2019). A multi-start local search heuristic for the green vehicle routing problem based on a multigraph reformulation. *Computers and Operations Research*, *109*(September 2019), 43-63. https://doi.org/10.1016/j.cor.2019.04.018

A multi-start local search heuristic for the green vehicle routing problem based on a multigraph reformulation

J. Andelmin[a,*], E. Bartolini[b]

Link to the published article: *https://doi.org/10.1016/j.cor.2019.04.018*

[a] *Department of Mathematics and Systems Analysis, Aalto University School of Science, P.O. Box 11100 FI-00076 Aalto, Finland*
[b] *Deutsche Post Chair - Optimization of Distribution Networks, School of Business and Economics, RWTH Aachen University, Kackertstr. 7 B, 52072 Aachen, Germany*

Abstract

We consider the Green Vehicle Routing Problem (G-VRP) which is an extension of the classical vehicle routing problem for alternative fuel vehicles. In the G-VRP, vehicles' driving autonomy and possible refueling stops en-route are explicitly modeled. We propose a multi-start local search algorithm that consists of three phases. The first two phases iteratively construct new solutions, improve them by local search, and store all vehicle routes forming these solutions in a route pool. Phase three optimally combines vehicle routes in the route pool by solving a set partitioning problem and improves the final solution by local search. The algorithm is based on a multigraph reformulation of the G-VRP in which nodes correspond to customers and a depot, and arcs correspond to possible sequences of refueling stops for vehicles traveling between two nodes. All local search operators used by our algorithm are tailored to exploit this reformulation and do not explicitly deal with refueling stations. We report computational results on benchmark instances with up to ~ 470 customers, showing that the algorithm is competitive with state-of-the-art heuristics.

Keywords: vehicle routing, alternative fuel vehicles, local search, multigraph

*Corresponding author
Email addresses: juho.andelmin@aalto.fi (J. Andelmin), bartolini@dpo.rwth-aachen.de (E. Bartolini)

1. Introduction

Vehicle Routing Problems (VRPs) are a class of optimization problems concerned with designing least-cost delivery routes for a fleet of vehicles to serve a set of customers. In the Capacitated Vehicle Routing Problem (CVRP) introduced by Dantzig and Ramser (1959), a set of customers with known demands is to be served by a fleet of vehicles with limited capacities, and the objective is to plan a set of minimum cost vehicle routes so that all customers are served. Several CVRP variants and generalizations of it have been studied since then (see, e.g., Golden et al., 2008; Laporte, 2009 for recent surveys). As gasoline- and diesel-powered vehicles have dominated the market, there has been little need to revise one major assumption underlying these models: each vehicle route can be represented as an ordered sequence of customer visits.

More recently, environmental concerns have driven governments to enact laws and regulations that require organizations to emphasize green logistic approaches in their operations, including the deployment of alternative fuel vehicles (AFVs). However, many AFVs have limited driving range and must often rely on a limited refueling infrastructure. Therefore, it may be necessary to include the exact locations of the refueling stations in the route planning. Furthermore, refueling delays may also have a considerable impact. This is a notorious issue, e.g., with electric vehicles due to their relatively short driving range and long recharging time, which together with a limited recharging infrastructure may cause range anxiety, the fear of not having enough energy to reach the desired destination (Franke et al., 2012). As a consequence, new routing models are emerging to provide reliable and adequate level of service when adopting AFVs.

There is a growing body of research addressing the routing of AFVs that consider intermediate refueling stops. Modeling vehicles' refueling strategies has been identified as one major challenge, and several routing models have been proposed with different trade-offs between the overall model complexity, fuel consumption representation, and vehicles' refueling policies. Special attention has been devoted to electric vehicles (EVs) (see, e.g., Conrad and Figliozzi, 2011, Felipe et al., 2014, Schneider et al., 2014, Hiermann et al., 2016, Montoya et al., 2017), and the impact of different recharging policies in this context is recently analyzed by Desaulniers et al. (2016) and Sweda et al. (2017). The study of Desaulniers et al. (2016) assesses the impact of different recharging policies (full vs partial recharges, and one vs multiple recharging stops) on the solution quality in presence of time windows constraints. The study highlights appreciable savings in terms of mileage and number of vehicles used when adopting the more flexible policies (partial recharges and multiple recharging stops). As the authors note, these savings are especially relevant in presence of relatively tight time windows.

In this paper, we focus on the Green Vehicle Routing Problem (G-VRP), introduced by Erdoğan and Miller-Hooks (2012), which is one of the first models tailored to AFV routing that includes refueling station visits. The G-VRP does not consider time windows, but imposes a maximum route duration to each vehicle. Although it assumes a full refuel policy and a constant refueling time, it captures the main features of AFV routing: explicit representation of vehicles' fuel levels and refueling stops. The G-VRP can thus be viewed as fundamental building block towards solving more complex and realistic problems. Moreover, it is motivated by AFV applications in which full refuel policies may be well justified, e.g., in presence of vehicles running on bio-diesel blends and EVs with replaceable batteries.

The G-VRP consists of a set of customers, a set of refueling stations, and a fleet of identical AFVs located at a central depot. The objective is to plan a set of vehicle routes, each starting and ending at the depot, so that each customer is visited exactly once and the total distance traveled by the vehicles is minimized. The vehicle fleet is assumed to be unlimited and no capacity constraints are considered, but the vehicles are subject to fuel constraints which limit their maximum driving range (autonomy). The vehicles can restore their driving autonomy by stopping at refueling stations en route (where they are assumed to fully refuel), and a constant refueling delay is incurred at each stop. Finally, each vehicle's trip is subject to a maximum duration constraint.

Therefore, the G-VRP has some similarities with the distance-constrained VRP (Laporte et al., 1984; Li et al., 1992).

1.1. Literature review

This section reviews VRP variants that are closest to the G-VRP in that they attempt to model the operational constraints arising with the adoption of AFVs as a consequence of refueling stops. It also provides an overview of heuristic methods that have been applied to solve the G-VRP.

The earliest studies of routing models that include refueling stations appear to be by Ichimori et al. (1981), who investigate the routing of a single vehicle with a limited driving autonomy through a network of regular nodes and refueling nodes at which the vehicle can refuel completely. The objective is to find the shortest path between two nodes along which the vehicle can traverse without running out of fuel. Ichimori et al. (1983) further study a problem in which a vehicle with a limited driving autonomy is initially located at a depot, visits a single customer after receiving a service call, and finally returns back to the depot, possibly stopping to refuel en route if necessary.

More recently, Conrad and Figliozzi (2011) introduce a variant of the VRP with time windows in which the vehicle fleet is composed of battery electric vehicles (BEVs). In their model, the BEVs are allowed to recharge their batteries at the customer locations, but dedicated recharging stations at which the vehicles can stop to refuel more than once are not considered.

Erdoğan and Miller-Hooks (2012) introduce the G-VRP and present two heuristic methods for solving it: a modified Clarke and Wright savings heuristic (Clarke and Wright, 1964) and a density-based clustering algorithm (Ester et al., 1996). The authors assess the heuristics by comparing their results to those obtained with the mixed-integer linear programming solver CPLEX (IBM ILOG CPLEX 11.2). The results indicate no significant differences between the proposed heuristics.

Schneider et al. (2014) introduce the electric VRP with time windows (E-VRPTW), which extends the G-VRP by incorporating customer time windows, customer demands, and vehicle capacities. The E-VRPTW also uses a new refueling policy in which recharging time depends on the battery level upon arrival at a station. The objective is to first minimize the number of vehicles and then the routing cost. The authors develop a hybrid metaheuristic combining variable neighborhood search (VNS) (Hansen and Mladenovic, 2003) and tabu search (TS) (Glover and Laguna, 1999). They apply their metaheuristic, called VNS/TS, also to the G-VRP and significantly improve the previous results.

Schneider et al. (2015) present the VRP with intermediate stops, which extends the E-VRPTW by incorporating both refueling stations and satellite facilities, as well as combinations of these two. In this model, vehicles can restore their driving autonomy at refueling stations and also replenish their supply at satellite facilities, thus allowing them to serve additional customers before returning to the depot. The authors develop a heuristic method called adaptive variable neighborhood search (AVNS), which combines ideas from VNS and adaptive large neighborhood search (Pisinger and Ropke, 2007). The AVNS is also applied to the G-VRP, improving the results obtained by VNS/TS with considerably smaller computation times. The E-VRPTW has also been extended to model a heterogeneous vehicle fleet (Hiermann et al., 2016) and a mixed fleet of conventional and electric vehicles (Goeke and Schneider, 2015).

Felipe et al. (2014) present the Green VRP with multiple technologies and partial recharges, which extends the G-VRP by incorporating customer demands, partial recharges, and different but constant recharging rates at different charging stations. The authors propose a heuristic method combining constructive local search heuristics within a simulated annealing framework (Suman and Kumar, 2006). Their heuristic yields competitive results with respect to VNS/TS, but is outperfomred by AVNS.

Montoya et al. (2016) study the G-VRP and propose a two-phase heuristic method called modified multi-space sampling heuristic, which is based on the multi-space sampling heuristic introduced by Mendoza and Villegas (2013). The first phase uses constructive heuristics to randomly generate several traveling salesman tours visiting only customer nodes. Each such tour is then optimally split into feasible vehicle routes while preserving the customer visiting order and adding possible refueling station visits to ensure feasibility (this is an adaptation of the optimal splitting procedure introduced by Prins (2004)). In the second phase, a solution to the G-VRP is computed by solving a set partitioning problem with all feasible routes obtained in the first phase, using the best solution cost obtained during the splitting procedure as an initial upper bound. The authors report competitive results with respect to AVNS.

Recently, Koç and Karaoglan (2016) present a new mathematical formulation for the G-VRP and a branch-and-cut algorithm. The authors also propose a simulated annealing heuristic and report results on small instances with up to 20 customers and 3-10 refueling stations.

The current best known solutions on benchmark G-VRP instances have been reported by Schneider et al. (2015) and Montoya et al. (2016).

1.2. Contributions of this paper

We propose a new matheuristic algorithm for solving the G-VRP. Our algorithm is based on a multigraph reformulation of the G-VRP in which refueling stations are not explicitly modeled. The node set in the multigraph consists of customers and a depot, and multiple arcs may exist between each pair of nodes. Each arc (i, j) in the multigraph represents a possible sequence of consecutive refueling stops for a vehicle traveling from node i to node j. We describe how to exploit this multigraph reformulation by tailoring classical local search operators to work directly on it, and by combining these operators to develop a multi-start local search heuristic algorithm. We demonstrate its effectiveness on benchmark instances with up to 500 customers. We report improved best known upper bounds for the largest benchmark instances, and we show that on instances with up to 100 customers our algorithm provides upper bounds that are, on average, within 0.27% from optimal.

Overall, our results suggest that high quality G-VRP solutions may be obtained by exploiting a multigraph reformulation, and by adapting classical local search operators to work directly on it, without dealing explicitly with refueling stations.

The heuristic we describe in this paper and the multigraph reformulation it is based upon were used by the exact method of Andelmin and Bartolini (2017). This article was quoted therein as a working paper and is the final version of that manuscript.

1.3. Organization of the paper

The rest of this paper is organized as follows. Section 2 describes the G-VRP. Section 3 describes the multigraph reformulation of the G-VRP and the construction of the multigraph. Section 4 details our multi-start local search heuristic and the local search operators used by it. Section 5 describes how the main parameters of the heuristic are determined. Section 6 reports a computational evaluation of our heuristic, and concluding remarks are given in Section 7.

2. Description of the G-VRP

The G-VRP is defined on a directed graph $G = (V, A)$, where the vertex set $V = N \cup F \cup \{0\}$ is a combination of a set $N = \{1, \ldots, n\}$ of n customers, a set $F = \{n + 1, \ldots, n + s\}$ of s refueling stations, and a vertex 0 representing the depot. Each customer $i \in N$ has a service time s_i, and each arc $(i, j) \in A$ is associated with a distance c_{ij} and a travel time t_{ij}.

The n customers are to be served by a (unlimited) homogeneous fleet of alternative fuel vehicles which are based at the depot. Each vehicle has a maximum driving time of T minutes and a limited amount of fuel which is expressed as a maximum fuel capacity C. All vehicles departing from the depot are assumed to be fully refueled having the maximum fuel amount C. It is worth noting that the G-VRP expresses the maximum driving range of a vehicle as its maximum fuel capacity C. However, since vehicle fuel consumption is assumed to be linearly dependent on the distance traveled, the maximum driving range can be equivalently expressed as a distance value. Letting K be the constant rate of fuel consumption per distance unit, the maximum distance Q that a vehicle can travel without refueling is thus defined as $Q = C/K$.

To avoid running out of fuel, vehicles can stop at any of the s refueling stations. After stopping at a refueling station, a vehicle is assumed to be fully refueled, and each stop is assumed to incur a refueling delay of δ minutes. It is also assumed that each vehicle incurs such a delay before leaving the depot for the first time (i.e., vehicles leave the depot fully refueled). This assumption is adopted by all the previous studies found in the literature.

Each vehicle is assumed to travel at a constant speed v. Thus, the travel time of each arc $(i, j) \in A$ can be defined as $t_{ij} = c_{ij}/v + s_i$ if $i \in N$ or $t_{ij} = c_{ij}/v + \delta$ if $i \in F \cup \{0\}$ (we include service times s_i and refueling delays δ in the arc travel times to simplify notation). Note that because of the maximum driving range Q and driving time T, any arc $(i, j) \in A$ with $c_{ij} > Q$ or $t_{ij} > T$ cannot be part of a feasible solution. We assume that those arcs are not present in G.

3. Multigraph reformulation of the G-VRP

In this section, we briefly describe a multigraph reformulation of the G-VRP, introduced by Andelmin and Bartolini (2017), which is used by our algorithm. For completeness, we also detail the procedure that we use to construct it.

3.1. Description of the multigraph

To define the multigraph, we denote by $\mathcal{N}_0$ its set of nodes where $\mathcal{N}_0 = N \cup \{0\}$. A *refuel path* from $i \in \mathcal{N}_0$ to $j \in \mathcal{N}_0$, $i \neq j$, is a simple path in G starting from i, visiting a subset $F' \subseteq F$ of refueling stations, and ending at j. For any pair of nodes $i, j \in \mathcal{N}_0$, $i \neq j$, we denote by $\mathscr{P}'_{ij}$ the index set of all refuel paths from i to j. For a refuel path P, its *total distance* $c(P)$ is equal to the sum of the distances of the arcs it traverses, and its *travel time* $t(P)$ is equal to the sum of the travel times of the arcs it traverses.

The multigraph is denoted by $\mathcal{G} = (\mathcal{N}_0, \mathcal{A})$, where $\mathcal{A}$ is the arc set containing arcs of the form (i, j, p), $\forall i, j \in \mathcal{N}_0$, $i \neq j$, $\forall p \in \mathscr{P}'_{ij}$. Note that the arcs of

$\mathcal{A}$ model either proper refueling paths that visit at least one station, or paths corresponding to a direct trip between two nodes $i, j \in \mathcal{N}_0$, $i \neq j$. We will call these latter arcs *direct arcs* and denote them as $(i, j, 0)$, or simply (i, j).

Thus, any arc $(i, j, p) \in \mathcal{A}$ either corresponds to a proper refuel path, or it is a direct arc indexed by $p = 0$.

Each arc $a = (i, j, p) \in \mathcal{A}$ has a distance $c(a) = c(i, j, p)$ and a travel time $t(a) = t(i, j, p)$ which are equal to the total distance and travel time, respectively, of the corresponding refuel path P_p, $p \in \mathscr{P}'_{ij}$ (we assume that $c(i, j, 0) = c_{ij}$ and $t(i, j, 0) = t_{ij}$). Moreover, for each arc $a = (i, j, p) \in \mathcal{A}$, we denote by $c_1(a) = c_1(i, j, p)$ and $c_2(a) = c_2(i, j, p)$ the distances of the first and the last arc, respectively, traversed by the refuel path P_p, $p \in \mathscr{P}'_{ij}$, $p \neq 0$.

Let us call *driving autonomy* the residual distance that a vehicle can travel without running out of fuel. Let q_i denote the driving autonomy after arriving at a node $i \in \mathcal{N}_0$. Notice that traversing an arc $(i, j, p) \in \mathcal{A}$, $p \neq 0$, in the multigraph corresponds to traversing the corresponding refuel path P_p, $p \in \mathscr{P}'_{ij}$. This is however possible only if the driving autonomy q_i upon arrival at i is at least $c_1(i, j, p)$. Moreover, the driving autonomy upon arrival at j after traversing an arc (i, j, p) is $Q - c_2(i, j, p)$. A *G-VRP route* can thus be modeled as a simple circuit $(0, a_0, i_1, a_1, \ldots, i_r, a_r, 0)$ in $\mathcal{G}$ traversing nodes $(0, i_1, \ldots, i_r)$ and arcs $a_0 = (0, i_1, p_0), \ldots, a_r = (i_r, 0, p_r)$ such that:

1. the sum of travel times of the arcs $a_0, ..., a_r$ does not exceed T
2. the driving autonomy q_{i_k} upon arrival at each node i_k, $k = 1, ..., r$, which is visited by the circuit is at least $c_1(i_k, i_{k+1}, p_k)$

Let $G_F = (F, A_F)$ be a subgraph of G induced by the set F of refueling stations with $A_F = \{(i, j) \in A : i, j \in F,\ c_{ij} \leq Q\}$. The multigraph construction is based on the following dominance rule.

Dominance 1. *Let $P = (i, u, \ldots, h, j)$ be a refuel path from $i \in N_0$ to $j \in N_0$ traversing the subpath $(u, \ldots, h)$ in G_F from $u \in F$ to $h \in F$ (possibly with $u = h$). P is said to be dominated if there exists another refuel path $P' =$*

$(i, u', \ldots, h', j)$ traversing a subpath $(u', \ldots, h')$ in G_F from $u' \in F$ to $h' \in F$ (possibly with $u' = h'$) such that

$$(i)\ c_{iu'} \leq c_{iu} \quad (ii)\ c_{h'j} \leq c_{hj} \quad (iii)\ c(P') \leq c(P) \quad (iv)\ t(P') \leq t(P)$$

and at least one of the inequalities is strict.

3.2. Construction of the multigraph

We represent a non-dominated refuel path as a simple path $P = (i, u, \ldots, h, j)$ which starts from $i \in \mathcal{N}_0$, traverses a subpath $P_{uh} = (u, \ldots, h)$ in G_F from $u \in F$ to $h \in F$, and finally ends at $j \in \mathcal{N}_0$. For any pair $u, h \in F$, let P^*_{uh} be the shortest $u - h$ path in G_F with respect to the arc distances $\{c_{ij}\}$, and let k^*_{uh} be its cardinality (i.e., the number of refueling station visits in P^*_{uh}). All the non-dominated refuel paths P from $i \in \mathcal{N}_0$ to $j \in \mathcal{N}_0$ must contain a $u - h$ path P_{uh} in G_F from some $u \in F$ to some $h \in F$ having cardinality less than or equal to k^*_{uh}. Thus, each non-dominated refuel path P from i to j is composed of

1. an arc $(i, u) \in A$, such that $u \in F$
2. a shortest $u - h$ path P^k_{uh} in G_F of cardinality k, for some $k \leq k^*_{uh}$
3. an arc $(h, j) \in A$, such that $h \in F$

The computation of non-dominated refuel paths is executed in the following three steps: (i) for every pair of nodes $u, h \in F$, compute the set $\mathcal{P}_{uh}$ of all shortest paths in G_F of cardinality $k = 1, 2, \ldots, k^*_{uh}$, (ii) for every pair of nodes $i, j \in \mathcal{N}_0$, $i \neq j$, compute the set $\mathcal{P}'_{ij}$ of all refuel paths from i to j satisfying 1 – 3, by using the paths $\mathcal{P}_{uh}$, $\forall u, h \in F$, (iii) extract the non-dominated paths from the sets $\mathcal{P}'_{ij}$ for all $i, j \in \mathcal{N}_0$, $i \neq j$. A step-by-step description of this procedure together with preprocessing steps that allow to reduce the size of the graph are provided in Andelmin and Bartolini (2017). Moreover, in this paper we also use a further reduction: we remove from A both arcs (i, j) and (j, i) if $i \in F_0$, $j \in N$, and $c_{ij} + c_{jk} > Q$, $\forall k \in V$, where $F_0 = F \cup \{0\}$.

In the following, $\mathscr{P}_{ij}$ denotes the final index set of all non-dominated paths from i to j obtained in this way $\forall i, j \in N_0$, $i \neq j$.

4. A Multi-start local search heuristic for the G-VRP

In this section, we describe our local search heuristic for solving the G-VRP. The core part of the heuristic builds on the idea of strategically sampling the solution space over a number of global iterations, a procedure more commonly known as the *multi-start method* (Martí et al., 2013). Each such global iteration typically consists of two phases: the first phase generates an initial solution and the second phase attempts to improve this solution by means of local search. Every global iteration thus produces a candidate solution that corresponds to a local optimum, and the best candidate solution over all global iterations is selected as the final output.

Our *Multi-Start Local Search* (MSLS) heuristic uses an approach similar to the multi-start method, but introduces some additional features. The vehicle routes of every phase one and phase two solution are added to a route pool $\mathcal{R}$, and a final component is included that solves a set partition problem over all the vehicle routes stored in $\mathcal{R}$ after all global iterations and improves the final solution by local search. Moreover, the algorithm directs the search to different parts of the solution space by dividing the set of all global iterations into four sequences. Each sequence uses different neighborhood structures in the construction phase (i.e., the first phase) of the corresponding global iteration sequence. Finally, in order to avoid generating identical solutions, the heuristic keeps track of each solution cost by using a cost table $\mathscr{H}$ which stores the cost of each solution found. If a candidate solution in phase one or two is found to have the same cost as one generated before, the heuristic iteratively modifies it (by also allowing its cost to become worse) until a solution with a unique cost is found. This is described in more detail in Section 4.3.1. Notice that if two different solutions have the same cost, they will be seen as the same solution. In this case, the vehicle routes in the solution that is constructed later will not be added to the route pool immediately but will enter the diversification phase instead. However, since the diversification phase rarely modifies several routes at a time, it is likely that even if several different solutions have the same cost, most of their routes will end up being added to the pool eventually.

Both phases of each global iteration use local search heuristics, hereafter called *operators*, to construct and improve G-VRP solutions. The local search operators used by the algorithm are divided into inter-route and intra-route operators. The inter-route operators operate on multiple routes simultaneously, while the intra-route operators operate on a single route at a time.

It is worth noting that although similar operators have been used by most of the previous heuristic methods developed for various alternative fuel VRPs, when adapted to work on our multigraph, they allow to simultaneously change both customer sequences and refueling stops of the routes they operate on. This allows to partially overcome one of the possible limitations of classical operators, namely, the lack of an integrated optimization approach with respect to routing and refueling decisions during local search.

The operators used by MSLS are described in Section 4.1, and Section 4.2 describes an efficient way to test the feasibility of the solutions in the neighborhoods explored by the operators. The overall structure of MSLS is detailed in Section 4.3.

4.1. Local search operators

All the operators used by MSLS are modified versions of the original ones which are tailored to work directly on the multigraph. Their pseudocode descriptions are given in Appendix B. When describing the operators, the depot is also considered to be a customer, except in the obvious case where a customer is relocated within or between routes.

The inter-route operators used by MSLS are the Clarke and Wright savings heuristic, the 2-OPT* and 3-OPT* heuristics, and the sequence relocate and cyclic exchange heuristics. MSLS also uses the intra-route 2-OPT and intra-route relocate heuristics which operate on a single route at a time, trying to improve its cost by removing and reconnecting arcs in $\mathcal{A}$ between customer pairs. Intra-route operators are used in the intensification phase, specifically, inside the function INTENSIFY, described in Algorithm 3 (see Section 4.3.1). Similar operators were also used by previous heuristics developed for the G-VRP or similar problems (Erdoğan and Miller-Hooks, 2012, Felipe et al., 2014, Schneider et al., 2014, 2015).

4.1.1. Clarke and Wright savings

We use the Clarke and Wright savings (Clarke and Wright, 1964) operator (hereafter abbreviated as CWS) to combine two vehicle routes into a single one. CWS computes all possible cost changes, or *savings*, resulting from merging two routes R_1 and R_2. This is done by reconnecting the last customer of R_1 to the first customer of R_2 so that a new feasible route is created. During each operator call, the best saving is computed by examining all pairs of routes, and for each such pair, by trying all possible arcs in $\mathcal{A}$ between the customers that are reconnected by the merge. The operator also considers the case where the route R_2 is reversed when computing the savings.

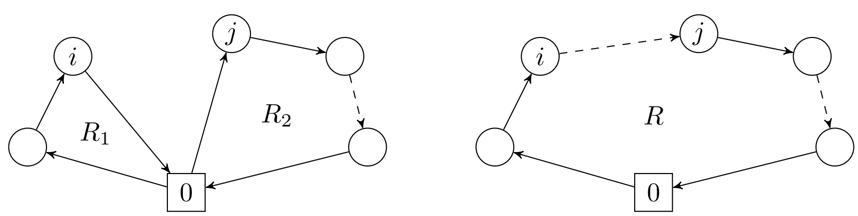

Figure 1: Example of the CWS operator merging two routes R_1 and R_2 into a new route R. Solid arrows represent direct arcs, and dashed arrows represent non-dominated refuel paths visiting one or more refueling stations.

We also use a modified version of the CWS operator (called CWS*) that operates in a similar fashion as the regular CSW, but it also checks if changing the arc between the depot and the first customer would be beneficial when merging two routes. An example of using the CWS* operator is presented in Figure 2.

We further modify CWS to use a regret-k heuristic (Ropke and Pisinger, 2006) for selecting a pair of routes to be merged. The regret-k heuristic computes

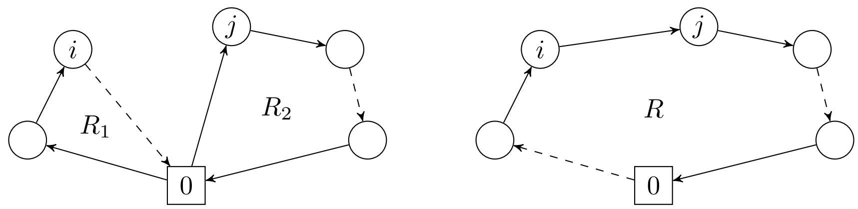

Figure 2: Example of using the CWS* operator. Solid arrows represent direct arcs, and dashed arrows represent non-dominated refuel paths visiting one or more refueling stations. Compared to Figure 1, the arc between i and j in R is now a direct arc, while the arc between the depot 0 and the first customer is now a non-dominated refuel path.

and stores, for each vehicle route R_i, the k best savings $s_1(R_i), \ldots, s_k(R_i)$ from merging R_i with every other route R_j. Then, for each route R_i, the following k-regret value $c_k(R_i)$ is computed:

$$c_k(R_i) = s_1(R_i) - \sum_{j=2}^{k} s_j(R_i).$$

Finally, the route R_i that obtains the greatest k-regret value $c_k(R_i)$ is merged with the route R_j that yields the best saving.

Different regret values (i.e., values of k) are used during the η first times that CWS or CWS* are used. The values of k and η are drawn randomly from a uniform distribution at the beginning of each global iteration (see Section 5).

4.1.2. 2-OPT and 3-OPT**

The 2-OPT* operator (Potvin and Rousseau, 1995) tries to reconnect customer sequence pairs of two vehicle routes in order to create two new routes with a smaller total cost. 2-OPT* first removes one arc from each route, thus creating four distinct customer sequences (two for each route), and then tries to reconnect the first customer sequence of the first route to the second customer sequence of the second route and vice versa. During each operator call, 2-OPT* computes cost savings for every pair or routes and selects the greatest one. The greatest cost saving for a given pair of routes is computed by examining all possible customer sequence pairs of the two routes, and for each such pair, by trying all possible combinations of arcs in $\mathcal{A}$ between the customers that can reconnect the customer sequences.

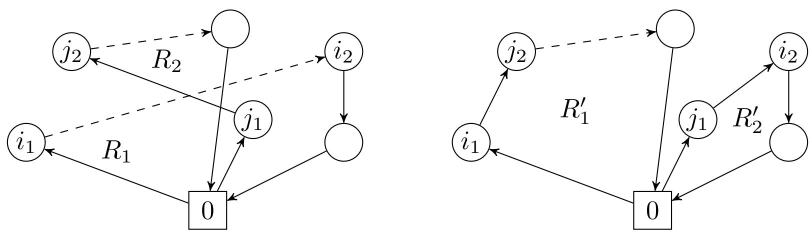

Figure 3: Example of the 2-OPT* operator removing one arc from route R_1 (between i_1 and i_2), one arc from route R_2 (between j_1 and j_2), and reconnecting the disconnected customer sequence pairs, thus creating two new routes R'_1 and R'_2. Solid arrows represent direct arcs, and dashed arrows represent non-dominated refuel paths visiting one or more refueling stations.

The 3-OPT* operator is otherwise similar to the 2-OPT*, but instead of operating on two routes, the 3-OPT* removes one arc from three distinct routes and then reconnects these routes optimally. The computational burden of 3-OPT* increases quickly when the number of routes becomes large, wherefore we use it only in the diversification phase (see function DIVERSIFY in Algorithm 4).

4.1.3. Sequence relocate and cyclic exchange

The sequence relocate operator selects a sequence of σ customers from one route and tries to insert it into another route between two successive customers. If some route has less than σ customers, the operator then tries to relocate the sequence containing all customers of that route between the two successive customers in the other route. During each call, the sequence relocate operator computes cost savings for every pair of routes by trying to relocate every possible customer sequence of length σ of the first route between every possible pair of successive customers in the second route. The greatest cost saving for a given relocation is computed by comparing all possible combinations of arcs in $\mathcal{A}$ between the corresponding customers when reconnecting the routes.

The cyclic exchange operator (Thompson and Orlin, 1989) selects a sequence of customers from each route within a set of two or more routes and exchanges these sequences such that each route obtains a new customer sequence from one

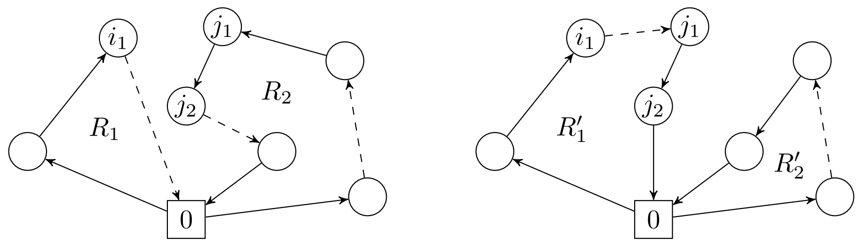

Figure 4: Example of the sequence relocate operator relocating customer sequence (j_1, j_2) of route R_2 between i_1 and 0 in route R_1, thus creating two new routes R_1' and R_2'. Solid arrows represent direct arcs, and dashed arrows represent non-dominated refuel paths visiting one or more refueling stations.

of the other routes. Each route is then connected to these new sequences, thus creating a set of new routes. The greatest cost saving for a given set of routes and sequence lengths for each route is computed by examining all possible customer sequence combinations with the given lengths, and for each such combination, by trying all possible customer sequence exchanges among the set of routes.

We limit the number of routes used by the the cyclic exchange operator to two, because the computational burden increases rapidly when operating on more than two routes simultaneously. When only two routes are considered, the operator is similar to a swap operator that selects a customer sequence from each route and swaps these customer sequences between the two routes.

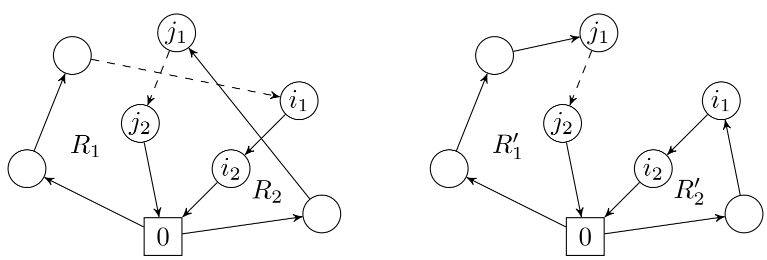

Figure 5: Example of the cyclic exchange operator exchanging customer sequences (i_1, i_2) of route R_1 and (j_1, j_2) of route R_2, thus creating two new routes R_1' and R_2'. Solid arrows represent direct arcs, and dashed arrows represent non-dominated refuel paths visiting one or more refueling stations.

4.1.4. Intra-route 2-OPT

The intra-route 2-OPT operator (Lin, 1965) is similar to the 2-OPT* operator except that 2-OPT operates on a single route instead of two. 2-OPT first removes two arcs from a given route and then reconnects the disconnected customer sequences optimally. The greatest cost saving for a given route R is computed by examining all possible pairs of arcs in R, and for each such pair, by removing these arcs from R and trying all possible combinations of arcs in $\mathcal{A}$ between the customers that can reconnect the route.

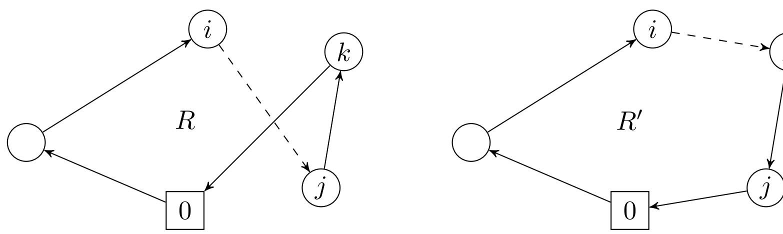

Figure 6: Example of the 2-OPT operator removing two arcs (between i and j, and between k and 0) and optimally reconnecting the disconnected customer sequences. Solid arrows represent direct arcs, and dashed arrows represent non-dominated refuel paths visiting one or more refueling stations.

4.1.5. Intra-route relocate

The intra-route relocate operator selects a customer from a given route and tries to relocate it between two successive customers in the same route. It first removes two arcs adjacent to a customer k which is to be relocated and one arc between two successive customers i and j. It then tries to reconnect the route so that it contains the customer sequence (i, k, j). The greatest cost saving for a given route is computed by examining all possible customer relocations and, for each such relocation, trying all possible combinations of arcs in $\mathcal{A}$ that can reconnect the route.

4.2. Feasibility tests

All the operators used by MSLS work by relocating or swapping customers or customer sequences (i.e., paths in $\mathcal{G}$) between the routes they operate on. This

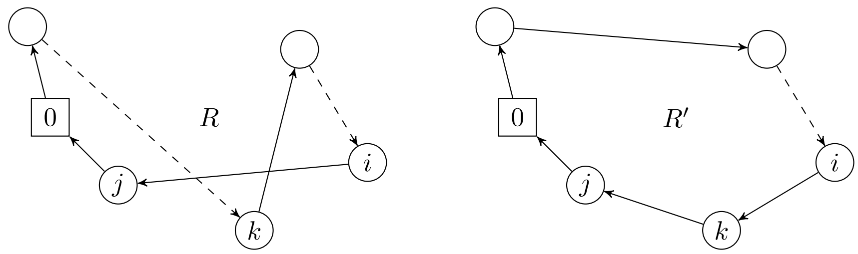

Figure 7: Example of the intra-route relocate operator relocating customer k between customers i and j. Solid arrows represent direct arcs, and dashed arrows represent non-dominated refuel paths visiting one or more refueling stations.

section describes an efficient way to evaluate feasibility of solutions resulting from these operations.

Let $R = (0, a_0, i_1, a_1, \dots, i_r, a_r, 0)$ be a G-VRP route, where i_0 and i_{r+1} both represent the depot 0. The route R starts from the depot i_0, visits customers $i_1, \dots, i_r$ while traversing arcs $a_0, ..., a_r$, and finally returns to the depot 0. In order to efficiently test the feasibility of solutions resulting from applying the operators described in Section 4.1, we define the following four labels for each vertex i_k visited by a G-VRP route R:

1. $C(i_k, R)$: The remaining fuel upon arriving at i_k with the convention that $C(0, R)$ denotes the fuel level at the end of the route).

2. $\overrightarrow{C}(i_k, R)$: The amount of fuel needed to reach the first refueling station, or the depot, starting from i_k.

3. $T(i_k, R)$: The total travel time upon arriving at i_k (with the convention that $T(0, R)$ denotes the total route time).

4. $\overrightarrow{T}(i_k, R)$: The time needed to reach the depot starting from i_k.

By using the labels defined above, it is easy to verify whether the modifications made on a solution by the MSLS operators preserve feasibility or not. We illustrate how this is done by using, as an example, a cyclic exchange relocation move (see Section 4.1.3), it being one of the most general cases.

Consider two feasible (customer-disjoint) routes $R = (0, a_0, i_1, a_1, \ldots, i_r, a_r, 0)$ and $R' = (0, a'_0, i'_1, a'_1, \ldots, i'_r, a'_r, 0)$. Suppose that a path $P = (i'_k, a'_k, ..., i'_h)$ is extracted from R' and inserted in R between two customers i_k and i_h. Suppose that the arcs $\bar{a}_k = (i_k, i'_k, p_k)$ and $\bar{a}_h = (i'_h, i_h, p_h)$ are used to connect P to R' (see Figure 8), thus creating a new route $\bar{R}$. To determine if $\bar{R}$ is feasible, it is sufficient to check that

A: The total duration of the route $\bar{R}$ (i.e., $T(0, \bar{R})$) does not exceed T.

B: The amount of fuel needed to reach the first refueling station (or the depot) starting from i_k (i.e., $\overrightarrow{C}(i_k, \bar{R})$) does not exceed $C(i_k, \bar{R})$.

C: The amount of fuel needed to reach the first refueling station (or the depot) starting from $i'_k \in \bar{R}$ (i.e., $\overrightarrow{C}(i'_k, \bar{R})$) does not exceed $C(i'_k, \bar{R})$.

D: The amount of fuel needed to reach the first refueling station (or the depot) starting from $i'_h \in \bar{R}$ (i.e., $\overrightarrow{C}(i'_h, \bar{R})$) does not exceed $C(i'_h, \bar{R})$.

E: The amount of fuel needed to reach the first refueling station (or the depot) starting from $i_h \in \bar{R}$ (i.e., $\overrightarrow{C}(i_h, \bar{R})$) does not exceed $C(i_h, \bar{R})$.

The above conditions can be checked in constant time without explicitly computing all the labels $C(i, \bar{R})$ and $T(i, \bar{R})$ of the new route $\bar{R}$ as follows.
Condition A can be verified in time $O(1)$ by computing

$$T(0, \bar{R}) = T(i_k, R) + \overrightarrow{T}(i'_k, R') - \overrightarrow{T}(i'_h, R') + \overrightarrow{T}(i_h, R) + t(\bar{a}_k) + t(\bar{a}_h)$$

and testing if $T(0, \bar{R}) - T \leq 0$.
For condition B, first notice that $C(i_k, \bar{R}) = C(i_k, R)$. Thus, we consider the following cases. If the arc $\bar{a}_k$ is not a direct arc, then $\overrightarrow{C}(i_k, \bar{R}) = Kc_1(\bar{a}_k)$ and condition B can be verified in time $O(1)$ by testing if $C(i_k, R) - Kc_1(\bar{a}_k) \geq 0$. Otherwise, $\bar{a}_k$ represents a direct arc and we can compute

$$\overrightarrow{C}(i_k, \bar{R}) = \begin{cases} Kc(\bar{a}_k) + \overrightarrow{C}(i'_k, R'), & \text{if } P \text{ contains a refueling station} \\ Kc(\bar{a}_k) + Kc(P) + \Gamma, & \text{otherwise} \end{cases}$$

where $c(P)$ is the total distance of the path P and $\Gamma = Kc(\bar{a}_h) + \overrightarrow{C}(i_h, R)$ if the arc $\bar{a}_h$ is a direct arc, or $\Gamma = Kc_1(\bar{a}_h)$ otherwise. Note that P contains a refueling station if and only if $Kc(P) \neq C(i'_k, R') - C(i'_h, R')$. Thus, $\overrightarrow{C}(i_k, \bar{R})$ can be obtained in time $O(1)$, and since $C(i_k, \bar{R}) = C(i_k, R)$, condition B can be verified in time $O(1)$ by testing if $C(i_k, \bar{R}) - \overrightarrow{C}(i_k, \bar{R}) \geq 0$.

Condition C is already verified by condition B if the arc $\bar{a}_k$ is a direct arc. Thus, we need to check condition C only if the arc $\bar{a}_k$ is not a direct arc. In this case, we have

$$\overrightarrow{C}(i'_k, \bar{R}) = \begin{cases} \overrightarrow{C}(i'_k, R'), & \text{if } P \text{ contains a refueling station} \\ Kc(P) + \Gamma, & \text{otherwise} \end{cases}$$

where $\Gamma = Kc(\bar{a}_h) + \overrightarrow{C}(i_h, R)$ if the arc $\bar{a}_h$ is a direct arc, or $\Gamma = Kc_1(\bar{a}_h)$, otherwise. Thus, $\overrightarrow{C}(i'_k, \bar{R})$ can be obtained in time $O(1)$, and since $C(i'_k, \bar{R}) = C - Kc_2(\bar{a}_k)$, condition C can be verified in time $O(1)$ by testing if $C(i'_k, \bar{R}) - \overrightarrow{C}(i'_k, \bar{R}) \geq 0$.

Condition D is already verified by condition C if the path P does not contain a refueling station. Thus, we need to verify condition D only if P contains a refueling station. In this case, we have $C(i'_h, \bar{R}) = C(i'_h, R')$. Moreover, we have $\overrightarrow{C}(i'_h, \bar{R}) = Kc(\bar{a}_h) + \overrightarrow{C}(i_h, R)$ if the arc $\bar{a}_h$ is a direct arc, or $\overrightarrow{C}(i'_h, \bar{R}) = Kc_1(\bar{a}_h)$ otherwise. Having computed $\overrightarrow{C}(i'_h, \bar{R})$, we can test if $C(i'_h, \bar{R}) - \overrightarrow{C}(i'_h, \bar{R}) \geq 0$ in time $O(1)$.

Condition E is already verified by condition D if the arc $\bar{a}_h$ is a direct arc. Thus, we need to verify condition E only if the arc $\bar{a}_h$ is not a direct arc. In this case, we have $C(i_h, \bar{R}) = C - Kc_2(\bar{a}_h)$, and since $\overrightarrow{C}(i_h, \bar{R}) = \overrightarrow{C}(i_h, R)$, we can test if $C(i_h, \bar{R}) - \overrightarrow{C}(i_h, \bar{R}) \geq 0$ in time $O(1)$.

Feasibility checks for routes that are modified by operators other than the cyclic exchange are derived in a similar way as above. However, when applying the intra-route 2-OPT operator to a route, the orientation of the middle path traversed by the new route must be reversed. Therefore, the corresponding feasibility check has to be adapted accordingly.

It is worth noting that the feasibility tests described in this section assume that those parts of the routes R and R' which occur before and after the relocated sequences (i.e., in the example above, before i'_k and after i'_h in R, and before i_k and after i_h in R'), as well as the relocated sequences themselves, are fixed. In other words, consistently with the definition of our local search operators (see Section 4.1), we do not attempt to make an infeasible move feasible by modifying either the fixed parts of R and R' or the relocated sequences.

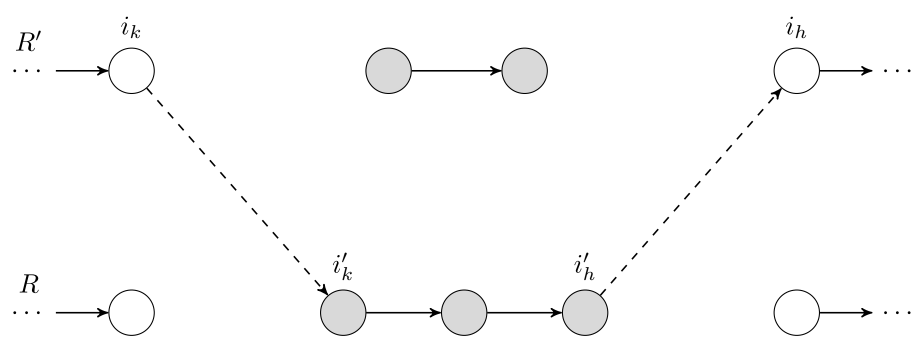

Figure 8: An example of the cyclic exchange operator relocating the customer sequence $P = (i'_k, \ldots, i'_h)$ of R between the customers i_k and i_h of R' to create a new route $\bar{R}$.

Note also that after each successful operator move, it is necessary to update the vertex labels of the routes involved in the move.

The vertex labels for each route are stored as doubly linked lists containing one node for each customer. After a successful operator move, we update the forward and backward time labels of each customer in the routes that are modified (by traversing the customer labels in their corresponding lists). The forward and backward fuel labels are also updated for those customers that are affected when traversing the lists, and the update process stops when the labels are guaranteed to not change (e.g., if we encounter a refueling station when traversing backwards from the customer i_k, we need not update any further backward fuel labels). The computational complexity of label updates is thus $\mathcal{O}(n)$ for every successful operator move, where n is the number of customers. Since the vertex labels need to be updated only after a successful operator move, label updates tend to have a negligible effect on the overall running time of the heuristic.

Finally, we note that the feasibility tests detailed in this section can be adapted to handle a partial refueling policy. Indeed, the labels $C(i_k, R)$ keep track of the maximum amount of fuel that a vehicle can have upon arriving at i_k, and the labels $\overrightarrow{C}(i_k, R)$ keep track of the least amount of fuel needed to reach the next refueling station or the depot. Thus, any partial refueling policy will be feasible as long as $C(i_k, R) \geq \overrightarrow{C}(i_k, R)$ for every customer $i_k \in R$. To compute possible time savings, we can introduce additional labels $T_{\min}(i_k, R)$ for every customer $i_k \in R$ to keep track of the minimum travel time upon arriving at i_k when using a partial refueling policy. If a route R using a full refueling policy has a travel time that is greater than the maximum T, it is then fast to check if R can be made feasible by using a partial refueling policy instead. A policy that uses the least amount of fuel (and time) is obtained by setting the refuel amount of each station visit to be the minimum required.

4.2.1. Feasibility and labeling comparison to similar approaches

It is worth noting that similar strategies to the one described in Section 4.2 have been proposed by several other authors to reduce the computational burden associated with the feasibility evaluation of local search moves applied to various generalizations of the G-VRP.

A major difference, however, is that all previous heuristics are based on formulations where refueling station visits are explicitly modeled as separate (duplicated) nodes. Differently from MSLS, these heuristics typically allow at most one refueling stop between two customers, thus simplifying the feasibility checks required by their local search moves.

An example is the Adaptive Large Neighbourhood Search (ALNS) metaheuristic developed by Hiermann et al. (2016) for the Electric Fleet Size and Mix Vehicle Routing Problem with Time Windows and Recharging Stations (E-FSMFTW). This algorithm uses similar labeling strategy to MSLS for handling fuel-level feasibility tests. It also uses similar local search operators as MSLS, including 2-OPT* and variants of the sequence relocate and cyclic exchange operators.

The operators used by MSLS are, however, modified to operate directly on the multigraph, and therefore require modified feasibility tests as well.

To appreciate the differences, consider the operation of concatenating two partial customer sequences $R_1 = (i_1, \ldots, i_r)$ and $R_2 = (j_1, \ldots, j_r)$ to form a new sequence $R = R_1 \oplus R_2 = (i_1, \ldots, i_r, j_1, \ldots, j_r)$. Although both ALNS and MSLS can assess the feasibility of this operation in constant time, ALNS disregards the possibility of inserting a charging station between the two customers i_r and j_1 to achieve feasibility. In contrast, MSLS can possibly make the concatenation feasible by connecting the two customer sequences through an arc of the multigraph that includes an arbitrary number of refueling stations.

Thus, even thought the ALNS and MSLS use a similar labeling strategy yielding the same time complexity for their feasibility checks, MSLS allows more options to repair infeasible routes without increasing the computational burden of testing their feasibility.

Similar observations can be made for other local search moves used by ALNS too. Overall, MSLS relies on a similar labeling strategy, but adapts it to work for local search operators that are tailored to the multigraph. This permits to explore wider neighborhoods (with regard to recharging station sequences) with no additional overhead in terms of complexity for the feasibility tests.

4.3. Overall description of the Heuristic

MSLS executes $L = 4l$ global iterations divided into four distinct *global iteration sequences* of length l (i.e., one global iteration sequence executes l global iterations). Each global iteration is divided into two distinct phases, both of which execute a number of local iterations by iteratively applying local search operators to modify a G-VRP solution. The two phases differ in the sets of operators they use and in their selection criteria. Phase one emphasizes the use of fast constructive operators, while phase two tries to improve each phase one solution by using a wider set of operators.

Phase one uses a different operator set $\mathcal{H}_i = \{H_1^i, \ldots, H_t^i\}$ in each of the four global iteration sequences $i = 1, ..., 4$ with the intention of exploring different regions of the solution space. Each set $\mathcal{H}_i$ is also associated with a probability set $P(\mathcal{H}_i) = \{Pr(H_1^i), \ldots, Pr(H_t^i)\}$ that controls the order and frequency with which the operators in the set $\mathcal{H}_i$ are applied. In phase two on the other hand, a same set of operators $\mathcal{O} = \{O_1, \ldots, O_k\}$ is used over all global iterations and no priority is given to any specific operator in this set. The operator sets $\mathcal{H}_i$, $i = 1, \ldots, 4$, and $\mathcal{O}$ are detailed in Section 4.3.1.

Figure 9 shows the structure of MSLS and visualizes its search strategy through the four global iteration sequences, followed by the set partitioning heuristic.

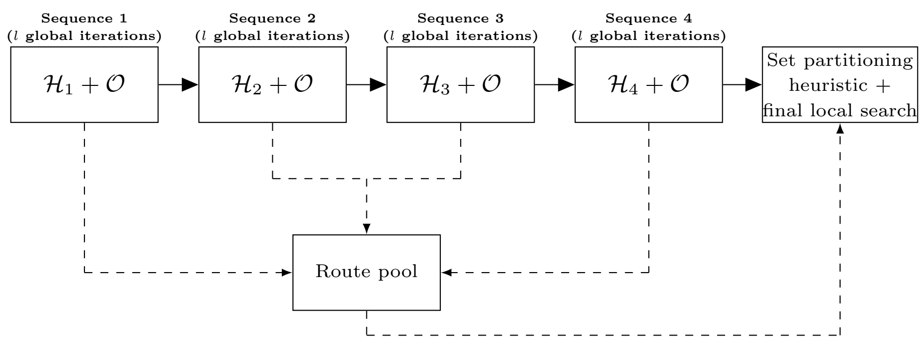

Figure 9: Structure of MSLS

The following section gives a detailed description of the two phases executed at each global iteration with the corresponding operator sets $\mathcal{H}_i$ and $\mathcal{O}$. A pseudocode description of MSLS is presented in Algorithm 1.

4.3.1. The two phases of a global iteration

Phase one of each global iteration starts by generating an initial solution S consisting of one vehicle route for each customer (i.e., n vehicle routes, each visiting a single customer). The function LocalSearch described in Algorithm 2 is then executed to improve S over a number of local iterations.

LocalSearch takes as input the current solution S, a set $\mathcal{H} = \{H_1, \ldots, H_t\}$ of operators, and a set $P(\mathcal{H}) = \{Pr(H_1), \ldots, Pr(H_t)\}$ of probabilities. Each operator $H \in \mathcal{H}$ is assigned a probability value $Pr(H)$, and the frequency with which the operators are applied depends on these probability values. Whenever

Algorithm 1 Multi-Start Local Search (MSLS)

Input: $L = 4l$: Number of global iterations. $\mathcal{R}$: Route pool for storing vehicle routes. $\mathcal{H} = \{\mathcal{H}_1, \ldots, \mathcal{H}_4\}$ and $P(\mathcal{H}) = \{P(\mathcal{H}_1), \ldots, P(\mathcal{H}_4)\}$, $i = 1, \ldots, 4$: Phase one operator and probability sets. $\mathcal{O} = \{O_1, \ldots, O_k\}$: Phase two operator set. $\mathscr{H}$: Table for storing solution costs. p: Max number of DIVERSIFY iterations.

1: **function** MSLS(L, $\mathcal{R}$, $\mathcal{H}_1, \ldots, \mathcal{H}_4, P(\mathcal{H}_1), \ldots, P(\mathcal{H}_4)$, $\mathcal{O}$, $\mathscr{H}$, p)
2:     **for** $i = 1, \ldots, 4$ **do**
3:         **for** $j = (i-1)l + 1, \ldots, il$ **do**
4:             **Execute phase one:**
5:                 Construct an initial solution S containing one route per customer
6:                 $S \longleftarrow$ LOCALSEARCH(S, $\mathcal{H}_i$, $P(\mathcal{H}_i)$)
7:                 $S' \longleftarrow$ INTENSIFY(S)
8:                 **if** $c(S') \in \mathscr{H}$ **then**
9:                     $S' \longleftarrow$DIVERSIFY(S', $\mathscr{H}$, p)
10:                 **end if**
11:                 ADDROUTES(S', $\mathcal{R}$, $\mathscr{H}$)
12:             **Execute phase two:**
13:                 $S \longleftarrow$ LOCALSEARCH(S, $\mathcal{O}$, unif$\{1, |\mathcal{O}|\}$)
14:                 $S' \longleftarrow$ INTENSIFY(S)
15:                 **if** $c(S') < c(S)$ **then**
16:                     **go to** row 13 and restart phase two using S'
17:                 **end if**
18:                 **if** $c(S') \in \mathscr{H}$ **then**
19:                     $S' \longleftarrow$DIVERSIFY(S', $\mathscr{H}$, p)
20:                 **end if**
21:                 ADDROUTES(S', $\mathcal{R}$, $\mathscr{H}$)
22:         **end for**
23:     **end for**
24:     Solve the problem (1) – (3) with the route set $\mathcal{R}$ to obtain a solution S^*
25:     $S^* \longleftarrow$ LOCALSEARCH(S^*, $\mathcal{O}$, unif$\{1, |\mathcal{O}|\}$)
26:     $S^* \longleftarrow$ INTENSIFY(S^*)
27:     **return** S^*
28: **end function**

an operator $H \in \mathcal{H}$ is applied but produces no improvement, it is removed from $\mathcal{H}$ (i.e., we set $\mathcal{H} \longleftarrow \mathcal{H} \setminus \{H\}$). However, every time some operator $H \in \mathcal{H}$ improves the current solution, all the previously removed operators are inserted back into $\mathcal{H}$. Applying an operator $H \in \mathcal{H}$ to a solution S is expressed as $H(S)$ in Algorithm 2.

When an operator $H \in \mathcal{H}$ is removed from $\mathcal{H}$, the probability value of H is divided among the remaining operators in $\mathcal{H}$ so that the ratio between their probability values remains unchanged. For example, suppose that $\mathcal{H}$ contains three operators H_1, H_2, and H_3 with probability values 0.10, 0.30, and 0.60, respectively, and suppose that we remove H_3 from $\mathcal{H}$. Then the probability values of H_1 and H_2 become $Pr(H_1) = 0.25$ and $Pr(H_2) = 0.75$, since the ratio between $Pr(H_1)$ and $Pr(H_2)$ before removing H_3 was $0.10 : 0.30 = 1 : 3$.

The phase one operators $H_1^i, \ldots, H_t^i$ included in the sets $\mathcal{H}_i$ and their selection probabilities $Pr(H_1^i), \ldots, Pr(H_t^i)$ for each of the $i = 1, \ldots, 4$ global iteration sequences are presented in Table 1, along with the number of routes r and the minimum and maximum number of customers used by these operators. Notice that in the operator set $\mathcal{H}_1$, the CWS operator is applied at each local iteration (its selection probability is 1.00), whereas at most one of the three remaining operators is applied depending on their probabilities. For the sets $\mathcal{H}_2$, $\mathcal{H}_3$, and $\mathcal{H}_4$, only one operator is instead applied per each local iteration.

Table 1: The phase one operator sets $\mathcal{H}_i$ and the corresponding probabilities $P(\mathcal{H}_i)$. The number of routes r and the minimum and maximum number of customers used by the operators (columns min and max, respectively) are also displayed.

Set $\mathcal{H}_1$		#cus		
Operator H	r	min	max	$Pr(H)$
CWS	2			1.00
2-OPT*	2			0.50
Sequence relocate	2	1	1	0.10
Cyclic exchange	2	1	1	0.10

Set $\mathcal{H}_2$		#cus		
Operator H	r	min	max	$Pr(H)$
CWS	2			0.25
2-OPT*	2			0.25
Sequence relocate	2	1	1	0.50

Set $\mathcal{H}_3$		#cus		
Operator H	r	min	max	$Pr(H)$
CWS	2			0.20
2-OPT*	2			0.20
Sequence relocate	2	1	1	0.10
Sequence relocate	2	2	2	0.10
Sequence relocate	2	1	3	0.10
Cyclic exchange	2	1	1	0.10
Cyclic exchange	2	1	2	0.10
Cyclic exchange	2	1	3	0.10

Set $\mathcal{H}_4$		#cus		
Operator H	r	min	max	$Pr(H)$
CWS	2			0.30
2-OPT*	2			0.15
Sequence relocate	2	1	3	0.10
Sequence relocate	2	3	6	0.10
Cyclic exchange	2	1	3	0.05
Cyclic exchange	2	1	6	0.20
Cyclic exchange	2	3	6	0.10

All the sets $\mathcal{H}_1, \ldots, \mathcal{H}_4$ include the constructive operators CWS and 2-OPT* because their combination forms a quick heuristic to construct initial solutions. Differences between the four sets arise from operators that exchange or relocate varying numbers of customers between two routes, and from the different selection probabilities. The rationale for selecting the operator sets and a computational evaluation of different probability values is presented in Section 5.

Algorithm 2 Local Search phase

Input: S: Initial solution. $\mathcal{H} = \{H_1, \ldots, H_t\}$: Set of operators used in local search.
$P(\mathcal{H}) = \{Pr(H_1), \ldots, Pr(H_t)\}$: Probability distribution of operators in $\mathcal{H}$.

function LocalSearch(S, $\mathcal{H}$, $P(\mathcal{H})$)
    $\mathcal{H}_0 \longleftarrow \mathcal{H}$
    while $\mathcal{H} \neq \emptyset$ **do**
        Randomly select an operator $H \in \mathcal{H}$ with probability $Pr(H)$
        $S' \longleftarrow H(S)$
        if $c(S') < c(S)$ **then**
            $S \longleftarrow S'$
            $\mathcal{H} \longleftarrow \mathcal{H}_0$
        else
            $\mathcal{H} \longleftarrow \mathcal{H} \setminus \{H\}$
        end if
    end while
    return S
end function

At termination of LocalSearch, the resulting solution S is passed to the function Intensify described in Algorithm 3. Intensify tries to obtain a new solution S' by modifying S, but in case its cost $c(S')$ is already included in $\mathscr{H}$, it uses the function Diversify described in Algorithm 4 to change S'. Diversify executes $p = 40$ iterations. It works by applying 2-OPT* and 3-OPT* operators to randomly selected routes, accepting also non-improving solutions. 2-OPT* and 3-OPT* were included in Diversify because they were found to provide diverse solutions significantly more often than the other operators. Finally, the function AddRoutes described in Algorithm 5 is applied to S'.

At this point, the solution S that was initially obtained by LocalSearch (before applying Intensify) is passed on to phase two. Notice that phase two operates on S instead of S' in order to reduce the probability of getting stuck in local optima.

Phase two proceeds in a similar fashion as phase one by first applying LocalSearch and then Intensify to the phase one solution S. The main difference with respect to phase one is the use of a wider set $\mathcal{O} = \{O_1, \ldots, O_k\}$ of

Algorithm 3 Intensification phase

```
Input: S: Initial solution.
function INTENSIFY(S)
    for each route R ∈ S do
        Iteratively apply intra-route relocate and 2-OPT operators defined in Section 4.1 each with probability 0.5 to R until no improvement is obtained
    end for
    return S
end function
```

Algorithm 4 Diversification phase

```
Input: S: Initial solution. ℋ: Cost table for storing solution costs. p: Number of iterations.
function DIVERSIFY(S, ℋ, p)
    for k = 1, . . . , p do
        Randomly select an operator H among 2-OPT* and 3-OPT* defined in Section 4.1.2.
        Apply H to randomly selected routes in S
        if c(S) ∉ ℋ then
            return S
        end if
    end for
    return S
end function
```

operators in LocalSearch. Moreover, the entire local search phase (Algorithm 2) is restarted whenever Intensify in row 14 improves the solution. Each operator $O \in \mathcal{O}$ is associated with a probability value $Pr(O) = 1/|\mathcal{O}|$, i.e., the operator selection probabilities follow a discrete uniform distribution unif$\{1, |\mathcal{O}|\}$, meaning that no operator is given priority with respect to the others. The set $\mathcal{O}$ of phase two operators used by our heuristic and their characteristics are presented in Table 2.

4.3.2. Set partitioning heuristic

The use of a set partitioning model within vehicle routing heuristics is not new (see, e.g., Rochat and Taillard, 1995, Groër et al., 2011, Subramanian et al., 2006), and it is known to yield appreciable improvements to the solution quality when properly combined with heuristic search. It also appears as a natural complement to the search strategy of our multi-start heuristics that is designed

Algorithm 5 Add vehicle routes to the route pool $\mathcal{R}$

Input: S: Initial solution. $\mathcal{R}$: Route pool. $\mathscr{H}$: Cost table to store solution costs.

```
function AddRoutes(S, R, H)
    if c(S) ∉ H then
        Add c(S) to H
        for each route R ∈ S do
            R ⟵ R ∪ {R}
        end for
    end if
end function
```

Table 2: The set $\mathcal{O} = \{O_1, \ldots, O_k\}$ of phase two operators used by our MSLS heuristic. The number of routes r and the minimum and maximum number of customers used by the operators are also displayed.

$\mathcal{O}$	Operator	r	min	max
O_1	2-OPT*	2		
O_2	Sequence relocate	2	1	1
O_3	Sequence relocate	2	2	2
O_4	Sequence relocate	2	3	3
O_5	Cyclic exchange	2	1	1
O_6	Cyclic exchange	2	1	3
O_7	Cyclic exchange	2	1	6
O_8	Cyclic exchange	2	3	6

to provide a large set of diverse solutions. In the context of the G-VRP, the set partitioning model is also used by Montoya et al. (2016).

The set partitioning heuristic is executed after all global iterations, and it solves the following set partitioning (SP) problem over all vehicle routes stored in the route pool $\mathcal{R}$. Let $\mathscr{R}$ be the index set of all vehicle routes in $\mathcal{R}$ after all global iterations, and let $\theta_{i\ell}$ be a binary coefficient that equals one if a route R_ℓ visits a customer $i \in N$, and zero otherwise. Moreover, for each route R_ℓ, $\ell \in \mathscr{R}$, let $c_\ell = c(R_\ell)$ be its cost, i.e., the sum of costs of the arcs it traverses. By defining a binary variables x_ℓ for each $\ell \in \mathscr{R}$ taking value one if and only if the route R_ℓ

is part of the solution, the SP problem can be modeled as follows

$$(SP) \quad z_{SP} = \min \sum_{\ell \in \mathscr{R}} c_\ell x_\ell \tag{1}$$

$$\text{s.t.} \sum_{\ell \in \mathscr{R}} \theta_{i\ell} x_\ell = 1, \quad \forall i \in N \tag{2}$$

$$x_\ell \in \{0, 1\}, \quad \forall \ell \in \mathscr{R}. \tag{3}$$

The SP problem (1) – (3) is solved by using the Mixed-Integer Linear Programming (MILP) solver CPLEX (IBM ILOG CPLEX 12.7.1). A time limit of $t_{SP} = 2000$ seconds is imposed on CPLEX, and the cost of the best phase one or phase two solution found is passed to CPLEX as an upper bound. A final LocalSearch followed by Intensify is applied to the solution obtained from the SP problem which tries to improve it one last time.

5. Parameter settings and implementation choices

This section describes how we determined the values of the main parameters used by MSLS to obtain the results reported in Section 6. The main choices concern the parameters η and k used by the regret-k heuristic, the number of iterations p used by the function Diversify, the time limit t_{SP} imposed on the set partitioning heuristic, and the sets of probabilities associated with the operator sets $\mathcal{H}_i$, $i = 1, \ldots, 4$.

The method we used to determine these parameter values is similar to the tuning strategy used by Ropke and Pisinger (2006). A fair parameter setting was produced by a trial-and-error phase while developing the heuristic. This produced the values $\eta \in \{1, \ldots, 15\}$ and $k \in \{1, 2, 3\}$ for the regret-k heuristics which seemed to generate a fair amount of variability in the phase 1 solutions, and $p = 40$ iterations for the function Diversify which was typically enough to transform a duplicate solution into a unique one. The time limit for the SP heuristic was set to $t_{SP} = 2000$ seconds. The idea was to set this limit large enough to find good solutions for larger instances without making it too big to become a bottleneck. This time limit was reached only in instances with 400 or more customers. More detailed results on the effect of this time limit on the overall computational time is given in Table A.10 of Appendix A.

Since the parameters having the largest impact on MSLS are the selection probabilities $Pr(H_1^i), \ldots, Pr(H_t^i)$ associated with the operator sets $\mathcal{H}_i$, $i = 1, \ldots, 4$, we used a more detailed tuning procedure to determine their values. We first focused on the probability values of the CWS and the 2-OPT* operators since they form the backbone of the construction phase. The probabilities of the remaining operators were set by considering six different combinations for each set $\mathcal{H}_i$, $i = 1, \ldots, 4$, and the probability combination that produced the best average solution was selected. The procedure was repeated for each operator set $\mathcal{H}_i$, $i = 1, \ldots, 4$, individually while keeping the probability combinations of the previous sets $\mathcal{H}_j$, $j < i$, fixed. Table 3 shows a computational evaluation with different probability settings for each of the operator sets $\mathcal{H}_i$, $i = 1, \ldots, 4$. This evaluation was executed using a set of 10 test instances (namely, the instances 111c_21s - 111c_28s of data set EMH and the instances AB101 - AB105 of data set AB that will be described in Section 6).

The following gives, for each set $\mathcal{H}_i$, $i = 1, \ldots, 4$, the rationale behind the probability values for CWS and 2-OPT*, and how the remaining operators are expected to affect the solutions.

1. $\mathcal{H}_1$ executes CWS at every iteration to generate initial solutions quickly. To avoid having too much variability from 2-OPT*, we set an upper limit of 0.5 to its probability value. We tried two combinations with 2-OPT* having a smaller probability than 0.5, but they produced worse results. $\mathcal{H}_1$ occasionally relocates a single customer or exchanges a pair of customers between two routes to diversify solutions, i.e., the search is limited to small neighborhoods. Decreasing the probabilities of the relocate and exchange operators from 0.1 produced worse results.

2. For $\mathcal{H}_2$, we tried different probability values for CWS and 2-OPT* while keeping their ratio at 1. The probability value of sequence relocate opera-

Table 3: Computational evaluation with different probability settings $P_0, \ldots, P_5$ for each set $\mathcal{H}_1, \ldots, \mathcal{H}_4$. The best probability setting for each set is denoted by P_0. Rows Average and %Average report the average solution value and the percentage distance from the average solution of P_0, respectively. The number of routes r and the minimum and maximum number of customers used by the operators are also displayed.

Set $\mathcal{H}_1$		#cus		Probability settings					
Operator	r	min	max	P_0	P_1	P_2	P_3	P_4	P_5
CWS	2			**1.00**	1.00	1.00	1.00	1.00	1.00
2-OPT*	2			**0.50**	0.50	0.50	0.50	0.40	0.30
Sequence relocate	2	1	1	**0.10**	0.05	0.01	0.00	0.10	0.10
Cyclic exchange	2	1	1	**0.10**	0.05	0.01	0.00	0.10	0.10
Average				**3889.95**	3892.23	3900.44	3906.69	3890.80	3892.08
%Average				**0.00**	0.06	0.27	0.43	0.02	0.05

Set $\mathcal{H}_2$		#cus		Probability settings					
Operator	r	min	max	P_0	P_1	P_2	P_3	P_4	P_5
CWS	2			**0.25**	0.30	0.35	0.40	0.45	0.50
2-OPT*	2			**0.25**	0.30	0.35	0.40	0.45	0.50
Sequence relocate	2	1	1	**0.50**	0.40	0.30	0.20	0.10	0.00
Average				**3886.03**	3891.59	3888.57	3907.08	3906.37	3932.21
%Average				**0.00**	0.14	0.07	0.54	0.52	1.19

Set $\mathcal{H}_3$		#cus		Probability settings					
Operator	r	min	max	P_0	P_1	P_2	P_3	P_4	P_5
CWS	2			**0.20**	0.20	0.20	0.20	0.20	0.20
2-OPT*	2			**0.20**	0.20	0.20	0.20	0.20	0.20
Sequence relocate	2	1	1	**0.10**	0.15	0.15	0.10	0.24	0.28
Sequence relocate	2	2	2	**0.10**	0.10	0.10	0.15	0.03	0.01
Sequence relocate	2	1	3	**0.10**	0.05	0.05	0.05	0.03	0.01
Cyclic exchange	2	1	1	**0.10**	0.10	0.15	0.10	0.10	0.10
Cyclic exchange	2	1	2	**0.10**	0.10	0.10	0.15	0.10	0.10
Cyclic exchange	2	1	3	**0.10**	0.10	0.05	0.05	0.10	0.10
Average				**3893.47**	3894.53	3894.90	3897.16	3896.41	3897.58
%Average				**0.00**	0.03	0.04	0.09	0.08	0.11

Set $\mathcal{H}_4$		#cus		Probability settings					
Operator	r	min	max	P_0	P_1	P_2	P_3	P_4	P_5
CWS	2			**0.30**	0.30	0.30	0.30	0.30	0.30
2-OPT*	2			**0.15**	0.15	0.15	0.15	0.15	0.15
Sequence relocate	2	1	3	**0.10**	0.05	0.00	0.10	0.10	0.10
Sequence relocate	2	3	6	**0.10**	0.15	0.20	0.10	0.10	0.10
Cyclic exchange	2	1	3	**0.05**	0.05	0.05	0.05	0.05	0.00
Cyclic exchange	2	1	6	**0.20**	0.20	0.20	0.15	0.10	0.20
Cyclic exchange	2	3	6	**0.10**	0.10	0.10	0.15	0.20	0.15
Average				**3879.24**	3880.85	3882.19	3882.88	3879.73	3879.86
%Average				**0.00**	0.04	0.08	0.09	0.01	0.02

tor changes accordingly, and smaller probabilities seem to produce worse results. An upper bound of 0.5 was set to limit the variability of this operator on the generated solutions. The idea is to generate initial solutions quickly, but to allow more frequent customer relocations compared to $\mathcal{H}_1$.

3. For $\mathcal{H}_3$, we kept the probability values of both CWS and 2-OPT fixed at 0.2 to allow operators that swap or relocate customers have more effect on the generated solutions. $\mathcal{H}_3$ explores bigger neighborhoods than $\mathcal{H}_1$ and $\mathcal{H}_2$: up to 3 customers can be relocated or two customer sequences with up to 3 customers each can be exchanged between two routes at each local iteration. The best combination spreads the probabilities evenly among the different relocate and exchange operators, although all six probability combinations produced quite similar results.

4. For $\mathcal{H}_4$, we fixed the probability values of CWS and 2-OPT* to 0.3 and 0.15, respectively. The reason for CWS having a higher probability value than 2-OPT* is to decrease the average computation time. We also wanted to assign different probabilities to these operators than in the other operator sets to construct more diverse solutions. $\mathcal{H}_4$ explores even larger neighborhoods than $\mathcal{H}_3$: up to 6 customers can be relocated or two customer sequences with up to 6 customers each can be exchanged between two routes at each local iteration. Instead of dividing the remaining probability mass evenly, we wanted to prioritize operators that relocate or exchange up to 6 customers between two routes. However, the six tested combinations produced very similar results.

A final decision concerns the composition of the phase two operator set $\mathcal{O}$. The operators $\mathcal{O} = \{O_1, \ldots, O_k\}$ were selected as follows. We first included in $\mathcal{O}$ all inter-route operators working on the neighborhoods shown in Table 4. We then ran our MSLS heuristic on the test instanceof data set EMH (see Section 6) while collecting statistics about their success rates. The columns in Table 4 report the number of routes r affected by an operator move; the minimum (min) and maximum (max) number of customers used by an operator move; the total number of operator moves (#tot); the number of operator moves that successfully improved a solution (#succ); and the corresponding success rates (%succ). The success rates are computed as %succ = (#succ / #tot) $\times$ 100%. We finally removed from $\mathcal{O}$ all the operators with less than 1.5% success rate, namely, the CWS* and sequence relocate with 4, 5, and 6 customers.

Table 4: The initial set $\mathcal{O} = \{O_1, \ldots, O_k\}$ of operators used in phase two and their success rates over the test instances of data set EMH. The columns report the number of routes r affected by an operator move; the minimum (min) and maximum (max) number of customers used by the operators; the total number of operator moves (#tot); the number of operator moves that successfully improved a solution (#succ); and the corresponding success rates (%succ).

$\mathcal{O}$	Operator	r	min	max	#tot	#succ	%succ
O_1	CSW*	2			19909	226	1.14
O_2	2-OPT*	2			28327	16277	57.46
O_3	Sequence relocate	2	1	1	25553	8495	33.24
O_4	Sequence relocate	2	2	2	23571	1170	4.96
O_5	Sequence relocate	2	3	3	23715	421	1.78
O_6	Sequence relocate	2	4	4	23429	190	0.81
O_7	Sequence relocate	2	5	5	21984	120	0.55
O_8	Sequence relocate	2	6	6	22189	149	0.67
O_9	Cyclic exchange	2	1	1	30445	17882	58.74
O_{10}	Cyclic exchange	2	1	3	26798	4178	15.59
O_{11}	Cyclic exchange	2	1	6	26419	2896	10.96
O_{12}	Cyclic exchange	2	3	6	26802	3503	13.07

6. Computational experiments

We consider two sets of benchmark instances. The first set, called EMH, was proposed by Erdoğan and Miller-Hooks (2012) and comprises 52 test instances with 20 - 500 customers and 3 - 28 refueling stations. Most of these instances (40 out of 52) consist of 20 customers and 2 - 10 refueling stations and have been randomly constructed to represent different types of customer and refueling station configurations. The larger EMH instances are based on an actual case study which uses a medical textile supply company depot and a pool of customers in Virginia, Maryland and the District of Columbia. These instances contain 111 - 500 customers and 21 - 28 refueling stations and have been used to benchmark all the previous G-VRP algorithms. All these instances assume a maximum driving time of 11 hours and a maximum driving autonomy of 300 miles. Every customer has a service time of 30 minutes, and the refueling delay is assumed to be $\delta = 15$ minutes. The vehicles are assumed to travel at a constant speed of $v = 40$ miles per hour. The locations of the customers and refueling stations are provided as geographical coordinates (latitude and longitude). Thus, all distances c_{ij} have to be computed by using the Harvesine formula with an earth radius of 4,182.45 miles (this value has been used in all previous studies on the G-VRP). Moreover, in all previous G-VRP studies, customers that cannot be served by a route visiting at most one refueling station are considered *infeasible* and removed from the instance a-priori.

The second set of instances, which we denote by AB, was created by Andelmin and Bartolini (2017) by extracting a subset of customers from the larger EMH instances. The number of customers in the AB instances ranges between 50 and 100. These instances are divided into two subsets called AB1 and AB2. The AB1 instances have the same characteristics as the EMH instances, whereas the AB2 instances have the same customers and refueling stations as those in AB1, but the vehicles are assumed to travel at a higher speed of $v = 60$ miles per hour and have a maximum driving autonomy of 280 miles. Notice that the AB2 instances allow longer vehicle routes with respect to the EMH and AB1 instances due to the higher vehicle speed. All AB instances are available at the URL `http://www.vrp-rep.org/variants/item/g-vrp.html`.

All the results reported in this paper have been obtained on an Intel i5-3570K desktop clocked at 3.40 GHz with 8 GB RAM running Windows 10 Home x64 Edition. The heuristic was coded in C++.

In the remainder of this section, we report the results obtained by our algorithm on the three data sets AB1, AB2, and EMH. We first analyze the results obtained by MSLS on these data sets and then report a comparison with the other heuristics found in the literature that have been applied to the G-VRP.

6.1. *Computational results*

Our first set of experiments was aimed at assessing the quality of the best solutions found by MSLS. For these experiments, we ran MSLS 10 times on each instance using 240 global iterations per run, and we collected statistics relative to both the average and best results it found. Tables 5, 6 and 7 present the results obtained for the larger EMH instances with 111 – 500 customers and for the AB instances. The results obtained on the small EMH instances are instead reported in Table A.9 of Appendix A. Also, more detailed results of the algorithm on the large EMH instances in terms of the trade-off between the time spent and the resulting improvement in the solution quality at the different phases is given by Table A.10 in Appendix A.

In Tables 5 – 7, the columns "n", "s" and "$|\mathcal{A}|$" report for each instance the number of customers, the number of stations, and the number of arcs in $\mathcal{G}$, respectively, while the column "BKS" reports the Best Known Solution cost (hereafter abbreviated as BKS). Improved BKS values found by MSLS for the first time are marked in bold. The column "m^*" reports the number of vehicles in the best solution (BKS), and the column "LB^*" reports for each instance the best lower bound LB^* found by Andelmin and Bartolini (2017). The average gaps above the BKS values in percentages are reported under columns "%Best" and "%Avg." and are computed with respect to the best and average upper bound, respectively, obtained over 10 runs. As an example, the gap for an upper bound z is computed as $(z/\text{BKS} - 1)\times 100$. Finally, column "$m$" reports the number of vehicles in the solution of smallest cost found by MSLS, column "t" presents the average computing time over all the runs, and the column $\%opt$ reports the percentage gap between LB^* and the best upper bound over ten runs (computed as $(z/LB^* - 1) \times 100$ where z is the best upper bound found). Note that the BKS of instances 111c_21 – 111c_28 were proven to be optimal, whereas for most of the remaining EMH instances no lower bound is available. Therefore columns "LB^*" and $\%opt$ are not reported in Table 5.

Tables 5 – 7 show that MSLS consistently finds solutions of very high quality. Indeed, it is able to match or improve the best known solutions for all the

Table 5: Results on the large EMH instances. Times are in minutes.

Instance	n	s	$\|\mathcal{A}\|$	BKS	m^*	%Best	%Avg.	m	t
111c_21s	109	21	57462	**4770.47**	17	0.03	0.08	17	1.87
111c_22s	109	22	58480	**4767.21**	17	0.00	0.05	17	1.96
111c_24s	109	24	64588	**4767.14**	17	0.00	0.03	17	2.42
111c_26s	109	26	66814	**4767.14**	17	0.00	0.05	17	2.57
111c_28s	109	28	68878	**4765.52**	17	0.00	0.05	17	2.78
200c_21s	192	21	191884	**8766.04**	31	0.00	0.28	31	10.48
250c_21s	237	21	303962	**10379.98**	37	0.00	0.33	37	21.46
300c_21s	283	21	424602	**12202.49**	43	0.00	0.06	43	35.44
350c_21s	329	21	576896	**13908.96**	49	0.00	0.15	49	60.99
400c_21s	378	21	743346	**16398.13**	58	0.00	0.16	58	111.84
450c_21s	424	21	931852	**17938.85**	64	0.00	0.20	64	145.73
500c_21s	471	21	1128354	**20207.81**	71	0.00	0.18	71	198.97
Average				10303.31		0.003	0.136		49.71

Table 6: Results on the AB1 instances. Times are in seconds.

Instance	n	s	$\|\mathcal{A}\|$	BKS	m^*	LB^*	%Best	%Avg.	m	t	%*opt*
AB101	50	21	10590	**2566.62**	9	2566.62	0.00	0.00	9	10.98	0.00
AB102	50	21	12768	**2876.26**	10	2876.26	0.00	0.00	10	12.80	0.00
AB103	50	21	12604	**2804.07**	10	2804.07	0.00	0.00	10	15.82	0.00
AB104	47	25	7420	**2634.17**	9	2634.17	0.00	0.00	9	48.16	0.00
AB105	73	21	21002	**3939.96**	14	3939.96	0.00	0.00	14	31.50	0.00
AB106	74	21	24956	**3915.15**	13	3915.15	0.09	0.37	14	32.69	0.09
AB107	75	21	35694	**3732.97**	13	3732.97	0.00	0.04	13	43.44	0.00
AB108	75	21	31972	**3672.40**	13	3672.40	0.00	0.05	13	41.49	0.00
AB109	75	24	29358	**3722.17**	13	3722.17	0.00	0.01	13	43.27	0.00
AB110	75	24	29420	3612.95	13	3572.11	0.20	0.59	13	44.16	1.34
AB111	71	25	21462	**3996.96**	14	3996.96	0.00	0.06	14	142.90	0.00
AB112	100	21	52858	**5487.87**	18	5487.87	0.60	1.27	19	90.01	0.60
AB113	100	21	53902	**4804.62**	17	4804.62	0.04	0.30	17	93.22	0.04
AB114	100	21	53686	**5324.17**	18	5324.17	0.01	0.35	18	87.07	0.01
AB115	100	21	50764	**5035.35**	17	5035.35	0.00	0.27	17	84.08	0.00
AB116	100	21	58286	**4511.64**	16	4511.64	0.03	0.22	16	102.26	0.03
AB117	99	22	47174	**5370.28**	18	5370.28	0.12	0.18	18	80.83	0.12
AB118	100	22	48770	**5756.88**	19	5756.88	0.00	0.14	19	81.04	0.00
AB119	98	25	47884	**5599.96**	19	5599.96	0.00	0.00	19	95.21	0.00
AB120	96	25	47658	**5679.81**	19	5679.81	0.00	0.00	19	81.84	0.00
Average				4252.21			0.05	0.19		63.14	0.11

Table 7: Results on the AB2 instances. Times are in seconds.

Instance	n	s	$\lvert\mathcal{A}\rvert$	BKS	m	LB^*	Best	Avg.	m^*	t	$\%opt$
AB201	50	21	19442	**1836.25**	6	1836.25	0.00	0.00	6	30.85	0.00
AB202	50	21	19978	**1966.82**	6	1966.82	0.00	0.02	6	58.10	0.00
AB203	50	21	19454	**1921.59**	6	1921.59	0.00	0.00	6	40.91	0.00
AB204	50	25	17874	**2001.70**	6	2001.70	0.00	0.00	6	130.92	0.00
AB205	75	21	42814	**2793.01**	9	2793.01	0.09	0.20	9	79.21	0.09
AB206	75	21	45478	**2891.48**	9	2891.48	0.00	0.00	9	79.23	0.00
AB207	75	21	54458	**2717.34**	8	2717.34	0.09	1.40	8	160.15	0.09
AB208	75	21	49572	**2552.18**	8	2552.18	0.00	0.17	8	110.63	0.00
AB209	75	24	51422	**2517.69**	8	2517.69	0.00	0.01	8	170.88	0.00
AB210	75	25	52968	**2479.97**	8	2479.97	0.00	0.02	8	158.25	0.00
AB211	75	24	47230	2970.56	9	2928.47	0.00	0.48	9	322.42	1.44
AB212	100	21	82248	**3341.43**	11	3341.43	0.70	0.71	11	230.68	0.70
AB213	100	21	90166	**3133.24**	10	3133.24	0.00	0.28	10	277.51	0.00
AB214	100	21	83186	3384.28	11	3364.16	0.03	0.50	11	210.35	0.63
AB215	100	21	83320	3480.52	11	3443.58	0.11	0.29	11	241.63	1.18
AB216	100	21	84618	3221.78	10	3200.47	0.55	1.22	10	259.79	1.22
AB217	100	22	87072	**3714.94**	11	3714.94	0.00	1.14	11	259.11	0.00
AB218	100	22	89430	**3658.17**	11	3658.17	0.14	0.29	11	256.52	0.14
AB219	100	25	103576	3790.71	11	3757.28	1.68	1.75	12	418.06	2.59
AB220	100	25	88330	**3737.88**	11	3737.88	0.35	0.51	11	281.61	0.35
Average				2905.64			0.19	0.45		188.84	0.42

large EMH instances but one within an average computing time of about 50 minutes. MSLS finds 8 new BKS (for instance 111c_22, and for instances 200c_21 – 500c_21) and matches 3 previously found BKS (for instances 111c_24, 111c_26, and 111c_28). When considering the best solution found over the 10 runs, the average gap with respect to the BKS is only 0.003%. When considering the average solution costs over the 10 runs, the solution quality appears convincing as well with an average gap of 0.136% above the BKS values.

For the AB1 instances, MSLS finds 13 optimal solutions and achieves an average gap of 0.05% with respect to the BKS and an average gap of 0.11% with respect to the best known lower bounds. For the AB2 instances, it finds 10 optimal solutions, its average gap with respect to the BKS is of 0.19%, and its average gap with respect to the best lower bounds is 0.42%. We also observe that the average solution quality appears rather stable, and the average solution costs are not too far from the best solutions found over the ten runs.

The main parameter of MSLS that allows to control the computing time is the number L of global iterations which is defined as $L = 4l$, where l is the

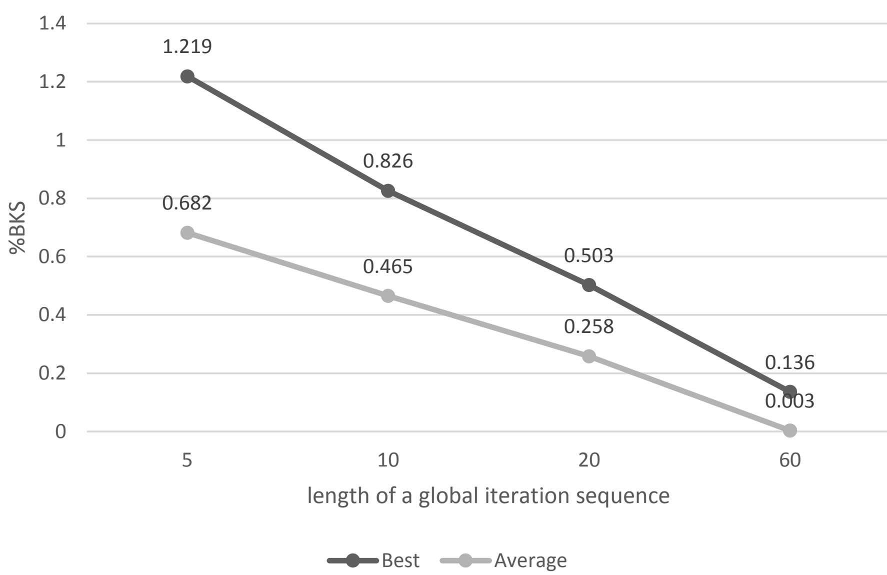

Figure 10: Average solution quality as a function of the length l of a global iteration sequence for the large EMH instances.

length of a global iteration sequence. To assess the trade-off between solution quality and the computing time, we have conducted a set of experiments with the larger EMH instances using four different values of l, i.e., $l = 5, 10, 20$ and 60, yielding $L = 20, 40, 80$ and 240 global iterations, respectively. Detailed results are reported in Table A.8 of Appendix A, while Figure 10 summarizes the impact of parameter l on the solution quality.

We see that the total computing time scales almost linearly with the value of l, thus allowing in effect to control the MSLS time by setting an upper bound on the maximum number of global iterations. Interestingly, Figure 10 shows that the solution quality scales almost linearly as well with the value of l. Overall, using the results obtained with $l = 5$ as a reference point, the average solution quality over the 10 runs of MSLS improves by 32.24%, 58.74%, and 88.84% when l is increased from 5 to 10, 20 and 60, respectively. Similarly, the best solution quality improves by 31.82%, 62.17%, and 99.56%, whereas the total computing time increases by approximately 1.9, 4.3 and 10.4 times. This analysis suggests that on the instances under consideration, the value of parameter l can be effectively used to control the trade off between solution quality and CPU time of MSLS, and the CPU time appears to be well utilized by the algorithm. It also suggests that the combination of intensification and diversification strategies adopted by MSLS are effective in guiding the algorithm across the solution space while avoiding it to get trapped in local minima.

6.2. Comparison with other heuristics

In this section, we benchmark our MSLS heuristic against other state-of-the-art heuristics found in the literature. Specifically, we consider four variants of MSLS that are obtained by setting l equal to 5, 10, 20, and 60. We refer to MSLS using $L = 4 \times l$ global iterations as MSLS-l. We compare MSLS-l on the large EMH instances to the Density-Based Clustering Algorithm/Modified Clarke and Write Savings (DBCA/MCWS) of Erdoğan and Miller-Hooks (2012), the 48A and Simulated Annealing (SA) heuristics of Felipe et al. (2014), the Variable Neighborhood Search / Tabu Search (VNS/TS) combination of Schneider et al. (2014), the Adaptive Variable Neighborhood Search (AVNS) by Schneider et al. (2015), and to the Multi-Space Sampling Heuristic (MSH) by Montoya et al. (2016). The number of test runs is 10 for all other heuristics (including our MSLS) except for the MCWS/DBCA for which the exact number of runs is not reported, and for the 48A and SA which have been ran only once.

The test runs of MCWS/DBCA were performed on a desktop with Pentium (4) CPU, 32-bit platform with 3.20 GHz processor and 2.00 GB of RAM. However, no computation times were presented. The algorithms 48A and SA were implemented in Fortran 95 and executed on an Intel Core i5, 2.8 GHz, 8 GB RAM, running Windows 7. Both algorithms VNS/TS and AVNS were implemented as single-thread code in Java, and both were evaluated on a desktop computer with an Intel Core i5 2.67 GHz processor with 4 GB RAM, running Windows 7 Professional. Finally, algorithm MHS was evaluated on a computing cluster with 2.33 GHz Intel Xeon E5410 processors with 16 GB of RAM running under Linux Rocks 6.1.1 (each replication was ran on a single processor).

Detailed results of this comparison are reported in Table A.8 of Appendix A,

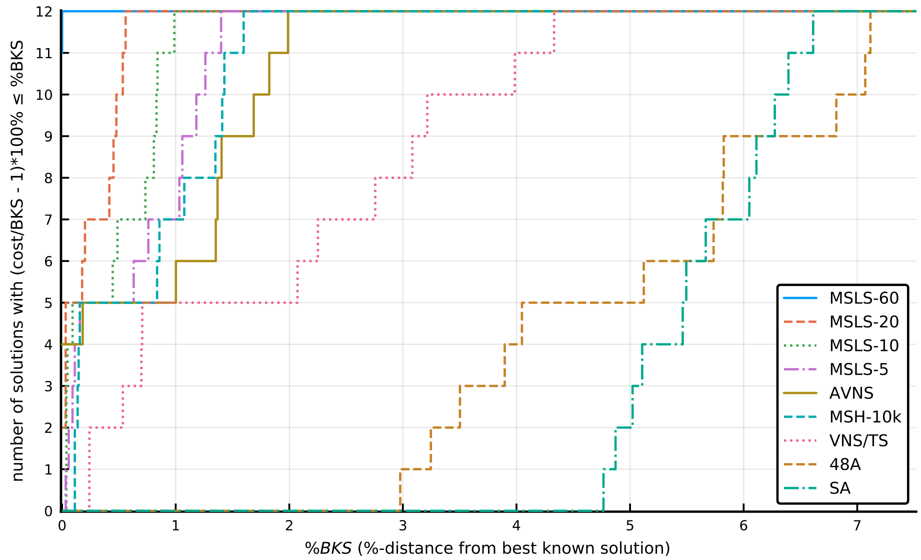

Figure 11: Performance profile comparing different heuristics (excluding MCWS/DBCA) with respect to the percentage distances from BKS costs of their best solution costs found for the large EMH instances.

while Figures 11 and 12 offer a visual comparison of the solution quality achieved by the different heuristics with respect to the best and average solution costs over 10 runs. For each value p, Figure 11 reports the number of instances for which the best solution costs found by each algorithm over the 10 runs is within p % from the BKS costs. Figure 12 reports the same information, but considers the % distances of the average solution costs obtained by each algorithm over the 10 runs from the BKS costs. Note that algorithms MCWS/DBCA are not considered in these figures because they are outperformed by all other algorithms (see Table A.8 in Appendix A) and no information is available regarding the number of runs they used and their average solution quality.

We observe that all versions of MSLS perform very well with regard to solution quality, and particularly MSLS-60, MSLS-20 and MSLS-10 demonstrate a clear advantage over the other heuristics. When looking at the upper half of Figure 11 (% distance from BKS with respect to at least 50% of the instances), a clear ranking emerges with the four MSLS variants in the first positions followed

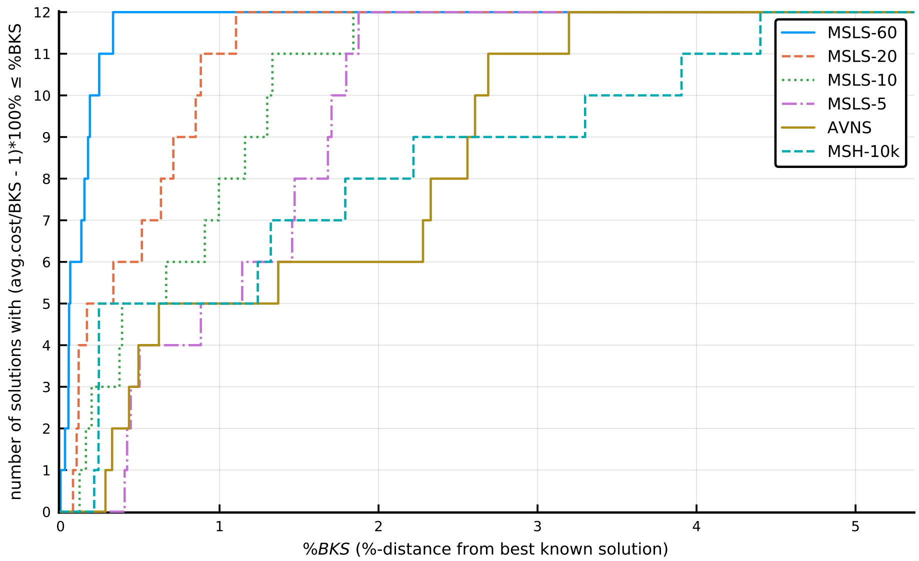

Figure 12: Performance profile comparing different heuristics with respect to the percentage distances from the BKS costs of their average solution costs obtained over 10 runs for the large EMH instances.

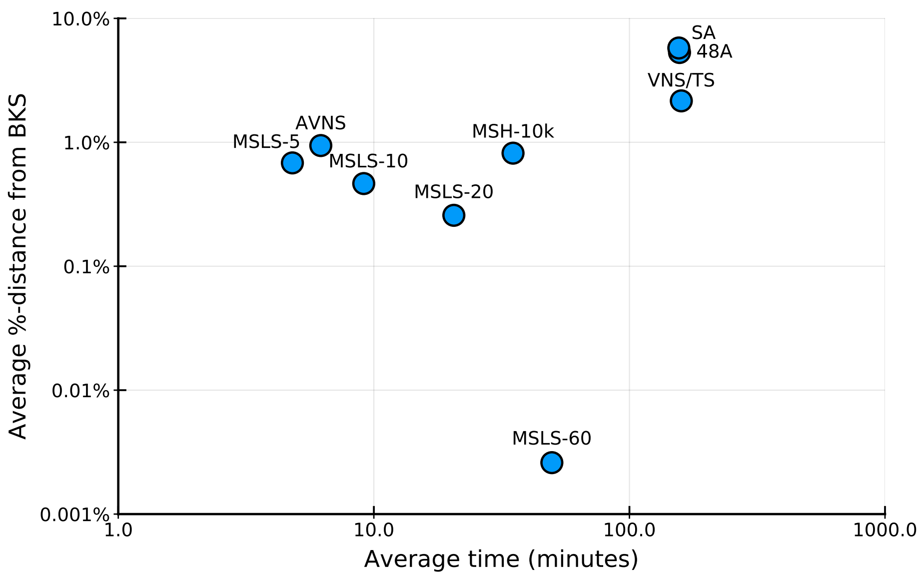

Figure 13: Comparison of average best upper bounds and cpu times of the different heuristics for the large EMH instances.

by MSH-10k and AVNS. Figure 12 shows that the situation is similar with even more pronounced differences when considering the average solution quality, although in this case MSH-10k falls behind AVNS when considering 9 or more instances.

A direct comparison of the computation times with the other heuristics is not straightforward as different programming languages and computers have been used for implementing and testing the heuristics. However, Table A.8 in Appendix A suggests that the computation time of our MSLS heuristic with $l = 60$ remains competitive with the other heuristics until the number of customers reaches 350, after which the solution time tends to slow down. This is because MSLS uses many operators whose running time depends on the number of routes, and this number increases rapidly with more customers for the large EMH instances. Nevertheless, by reducing the value of l, the computing time of MSLS decreases significantly, while the solution quality remains competitive. This is illustrated in Figure 13 which shows the trade-off between solution quality (average best solution costs over 10 runs) and cpu time realized by each algorithm.

We see that when $l = 5$, the average CPU time of MSLS drops to about 5 minutes while, on average, the distance from BKS over the 10 runs is still competitive with respect to the other heuristics. Finally, we note that algorithm AVNS appears to scale better than the other heuristics in terms of computation time. Compared to MSLS, it is indeed faster on most of the larger instances, although its solution quality becomes worse.

Finally, a comparison of all the algorithms on the small EMH instance can be found in Table A.9 of Appendix A. On these instances, our MSLS heuristic obtains 39 out of 40 BKS, and the average solution cost over the 10 runs is the same as the best solution cost for most of the instances. We used MSLS-60 for these instances and the computation times were between 0.1 and 1.0 seconds.

7. Conclusions

We have developed a Multi-Start Local Search (MSLS) heuristic for the Green Vehicle Routing Problem (G-VRP) which iteratively constructs new solutions, stores the vehicle routes forming these solutions in a pool, and finally optimally combines these routes by solving a set partitioning problem. We tested our MSLS heuristic on a set of 52 benchmark instances from the literature and found it competitive with recent heuristics. MSLS found 8 new best known solutions and matched another 43. Moreover, although MSLS contains some randomized components, the average solution costs it found over ten runs on each instance were not far from the best found solutions, suggesting that the heuristic is "robust" in the sense that it produces solutions whose costs do not vary much between different test runs. We also applied our MSLS heuristic to a new set of 40 instances with up to 100 customers for which MSLS found 23 optimal solutions and provided upper bounds on average within 0.27% far from optimal.

A distinguishing feature of our heuristic is that it is based on the multigraph reformulation of the G-VRP which does not require to explicitly model the refueling stops, and which excludes solutions containing sub-optimal refuel paths. This multigraph forms a core part of our MSLS heuristic because all the local search operators used by the MSLS are tailored to exploit it. As it is shown in Andelmin and Bartolini (2017) (see Appendix A3 and tables therein), the construction of the multigraph $\mathcal{G}$ and subsequent arc reductions on it exclude a significant number of sub-optimal customer-station sequences. We believe this, coupled with the fact that $\mathcal{G}$ allows local search heuristics to operate simultaneously on customer sequencing and refueling decisions, provides a rationale for the good performance of MSLS and motivates the investigation of multigraph reformulations in the context of AFV routing heuristics.

We finally note that the multigraph reformulation used by MSLS can be adapted to other VRP variants that include a limited fuel autonomy and refueling stations where this autonomy can be restored (either fully or partially), such as the electric VRP with time windows. A description of how the multigraph can be constructed in order to model EV routing problems with partial recharges is detailed in the Master's thesis of the first author (see Chapter 4 of Andelmin, 2014). We refer the interested reader to that document for a detailed description.

Acknowledgements

We wish to thank four anonymous referees for helpful comments that improved the presentation of our paper.

References

Andelmin, J., 2014. Optimal routing of electric vehicles. Department of Mathematics and Systems Analysis, Aalto University School of Science.

Andelmin, J., Bartolini, E., 2017. An exact algorithm for the green vehicle routing problem. Transportation Science 51 (4), 1288–1303.

Clarke, G., Wright, J. W., 1964. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12 (4), 568–581.

Conrad, R. G., Figliozzi, M. A., 2011. The recharging vehicle routing problem. In: Proceedings Industrial Engineering Research Conference, Reno, NV.

Dantzig, G. B., Ramser, J. H., 1959. The truck dispatching problem. Management Science 6 (1), 80–91.

Desaulniers, G., Errico, F., Irnich, S., Schneider, M., 2016. Exact algorithms for electric vehicle-routing problems with time windows. Operations Research 64 (6), 1388–1405.

Erdoğan, S., Miller-Hooks, E., 2012. A green vehicle routing problem. Transportation Research Part E: Logistics and Transportation Review 48 (1), 100–114.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the International Conference on Knowledge, Discovery and Data Mining (KDD). Vol. 96. pp. 226–231.

Felipe, Á., Ortuño, M. T., Righini, G., Tirado, G., 2014. A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges. Transportation Research Part E: Logistics and Transportation Review 71, 111–128.

Franke, T., Neumann, I., Bühler, F., Cocron, P., Krems, J. F., 2012. Experiencing range in an electric vehicle: understanding psychological barriers. Applied Psychology 61 (3), 368–391.

Glover, F., Laguna, M., 1999. Tabu Search. Springer.

Goeke, D., Schneider, M., 2015. Routing a mixed fleet of electric and conventional vehicles. European Journal of Operational Research 245 (1), 81–99.

Golden, B. L., Raghavan, S., Wasil, E. A., 2008. The Vehicle Routing Problem: Latest Advances and New Challenges. Vol. 43. Springer.

Groër, C., Golden, B., Wasil, E., 2011. A parallel algorithm for the vehicle routing problem. INFORMS Journal on Computing 23 (2), 315–330.

Hansen, P., Mladenovic, N., 2003. Variable neighbourhood search. Handbook of Metaheuristics, Dordrecht, Kluwer Academic Publishers.

Hiermann, G., Puchinger, J., Ropke, S., Hartl, R. F., 2016. The electric fleet size and mix vehicle routing problem with time windows and recharging stations. European Journal of Operational Research 252 (3), 995–1018.

Ichimori, T., Ishii, H., Nishida, T., 1981. Routing a vehicle with the limitation of fuel. J. OPER. RES. SOC. JAPAN. 24 (3), 277–281.

Ichimori, T., Ishii, H., Nishida, T., 1983. Two routing problems with the limitation of fuel. Discrete Applied Mathematics 6 (1), 85–89.

Koç, Ç., Karaoglan, I., 2016. The green vehicle routing problem: A heuristic based exact solution approach. Applied Soft Computing 39, 154–164.

Laporte, G., 2009. Fifty years of vehicle routing. Transportation Science 43 (4), 408–416.

Laporte, G., Desrochers, M., Nobert, Y., 1984. Two exact algorithms for the distance-constrained vehicle routing problem. Networks 14 (1), 161–172.

Li, C.-L., Simchi-Levi, D., Desrochers, M., 1992. On the distance constrained vehicle routing problem. Operations Research 40 (4), 790–799.

Lin, S., 1965. Computer solutions of the traveling salesman problem. Bell System Technical Journal, The 44 (10), 2245–2269.

Martí, R., Resende, M. G., Ribeiro, C. C., 2013. Multi-start methods for combinatorial optimization. European Journal of Operational Research 226 (1), 1–8.

Mendoza, J. E., Villegas, J. G., 2013. A multi-space sampling heuristic for the vehicle routing problem with stochastic demands. Optimization Letters 7 (7), 1503–1516.

Montoya, A., Guéret, C., Mendoza, J. E., Villegas, J. G., 2016. A multi-space sampling heuristic for the green vehicle routing problem. Transportation Research Part C: Emerging Technologies 70, 113–128.

Montoya, A., Guéret, C., Mendoza, J. E., Villegas, J. G., 2017. The electric vehicle routing problem with nonlinear charging function. Transportation Research Part B: Methodological 103, 87–110.

Pisinger, D., Ropke, S., 2007. A general heuristic for vehicle routing problems. Computers & Operations Research 34 (8), 2403–2435.

Potvin, J.-Y., Rousseau, J.-M., 1995. An exchange heuristic for routeing problems with time windows. Journal of the Operational Research Society 46 (12), 1433–1446.

Prins, C., 2004. A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research 31 (12), 1985–2002.

Rochat, Y., Taillard, É. D., 1995. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1 (1), 147–167.

Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 40 (4), 455–472.

Schneider, M., Stenger, A., Goeke, D., 2014. The electric vehicle-routing problem with time windows and recharging stations. Transportation Science 48 (5), 500–520.

Schneider, M., Stenger, A., Hof, J., 2015. An adaptive VNS algorithm for vehicle routing problems with intermediate stops. OR Spectrum 37 (2), 353–387.

Subramanian, A., Uchoa, E., Ochi, L. S., 2006. A hybrid algorithm for a class of vehicle routing problems. Computers & Operations Research 40, 2519–2531.

Suman, B., Kumar, P., 2006. A survey of simulated annealing as a tool for single and multiobjective optimization. Journal of the Operational Research Society 57 (10), 1143–1160.

Sweda, T. M., Dolinskaya, I. S., Klabjan, D., 2017. Adaptive routing and recharging policies for electric vehicles. Transportation Science 51 (4), 1326–1348.

Thompson, P. M., Orlin, J. B., 1989. The theory of cyclic transfers. Working paper, OR 200-89, Massachusetts Institute of Technology, Operations Research Center.

Appendix A. Detailed computational results

This section provides a detailed comparison of the four variants of our MSLS heuristic, using $L = 20, 40, 80$ and 240 global iterations, with the other heuristics in the literature (see Section 6.2). We also provide here some additional details on the behavior of the algorithm when applied to the large EMH instances.

Table A.8 compares all the heuristics on the large EMH instances whereas Table A.9 presents the result on the 20-customer EMH instances. The columns in these tables report similar information as in Tables 5 – 7 of Section 6.1. Columns "n", "s" and "$|\mathcal{A}|$" report the number of customers, the number of stations, and the number of arcs in $\mathcal{G}$, respectively, while the column "BKS" reports the Best Known Solution. Columns "Best" and "Avg." report the best upper bound and average upper bound (when available) found by each heuristic during the runs performed. Finally, column "t" reports the average computing time over all the runs. The table also presents the average gaps above the BKS values in percentages for each heuristic. The average gaps are obtained by first computing the percentage gaps for each instance individually and then taking the average. As an example, the gap for an individual instance for some heuristic with a solution cost z is computed as $(z/\text{BKS} - 1) \times 100\%$.

To allow a more detailed analysis of the algorithm's behavior we report in Table A.10 more detailed statistics about the different phases of the algorithm. In columns "t_{P1}", "t_{P2}", and "t_{SP}" we report for each instance the total time spent by the MSLS on phase one, phase two, and on solving the SP problem described in Section 4.3.2, respectively. The column "Best$_{P2}$" reports for each instance the best solution found at termination of the 4 global iteration sequences (i.e., before applying the set partitioning heuristic), while columns "%$P2$" and "%$P2$" report the % distance from the BKS of the best solution cost found at the end of the 4 global iteration sequences, and after the set partitioning heuristic, respectively.

Table A.8: Results on large EMH instances. Time is measured in minutes.

					MCWS/DBCA		48A			SA			VNS/TS			AVNS		
Instance	n	s	$\|\mathcal{A}\|$	BKS	m	Best	m	Best	t	m	Best	t	m	Best	t	Best	Avg.	t
111c_21s	109	21	57462	**4770.47**	20	5626.64	18	4960.60	21.74	18	5062.06	12.35	17	4797.15	21.76	**4770.47**	4791.53	1.78
111c_22s	109	22	58480	**4767.21**	20	5610.57	18	4914.20	21.23	18	5029.17	12.30	17	4802.16	23.56	4776.81	4797.31	1.94
111c_24s	109	24	64588	**4767.14**	20	5412.48	18	4952.90	21.27	18	5010.59	12.13	17	4786.96	21.90	**4767.14**	4790.84	2.16
111c_26s	109	26	66814	**4767.14**	20	5408.38	18	4934.11	21.25	18	5092.06	12.07	17	4778.62	25.12	**4767.14**	4782.60	2.04
111c_28s	109	28	68878	**4765.52**	20	5331.93	18	4971.93	21.29	18	5038.84	11.99	17	4799.15	24.17	**4765.52**	4781.26	1.73
200c_21s	192	21	191884	**8766.04**	35	10413.59	32	9276.63	76.41	32	9206.28	73.84	35	8963.46	76.65	8886.00	8970.14	3.61
250c_21s	237	21	303962	**10379.98**	41	11886.61	39	11007.98	120.67	38	10885.71	108.10	39	10800.18	120.90	10487.15	10531.20	3.67
300c_21s	283	21	424602	**12202.49**	49	14229.92	46	12869.17	184.27	45	12827.35	302.04	46	12594.77	182.23	12374.49	12514.78	4.94
350c_21s	329	21	576896	**13908.96**	57	16460.30	54	14954.83	227.30	52	14828.63	332.38	51	14323.02	232.03	14103.66	14271.56	7.11
400c_21s	378	21	743346	**16398.13**	67	19099.04	61	17351.92	302.03	60	17327.27	384.68	61	16850.21	305.12	16697.21	16839.23	12.70
450c_21s	424	21	931852	**17938.85**	75	21854.17	68	19215.38	514.68	67	19085.91	456.26	68	18521.23	525.52	18310.60	18512.47	13.19
500c_21s	471	21	1128354	**20207.81**	84	24517.08	76	21636.59	352.16	75	21475.71	154.45	76	21170.90	356.01	20609.67	20874.50	19.51
Average				10303.31		12154.23		10920.52	157.03		10905.80	156.05		10598.98	159.58	10442.99	10538.12	6.20
Avg. Gap above BKS (%)						16.867		5.769			5.305			2.160		0.941	1.697	
Computer					P4 3.2 GHz		Core I5 2.8 GHz			Core I5 2.8 GHz			Core I5 2.67 GHz			Core I5 2.67GHz		
Runs					Not reported		1			1			10			10		

	MSH(10k)				MSLS-5 ($L = 4 \times 5$)				MSLS-10 ($L = 4 \times 10$)				MSLS-20 ($L = 4 \times 20$)				MSLS-60 ($L = 4 \times 60$)			
Instance	m	Best	Avg.	t	m	Best	Avg.	t	m	Best	Avg.	t	m	Best	Avg.	t	m	Best	Avg.	t
111c_21s	17	4777.91	4781.85	4.94	17	4775.81	4790.44	0.18	17	4775.81	4780.56	0.33	17	**4770.47**	4775.86	0.63	17	4771.97	4774.20	1.87
111c_22s	17	4774.65	4778.80	4.69	17	4771.59	4787.74	0.18	17	4770.01	4775.12	0.34	17	**4767.21**	4772.66	0.65	17	**4767.21**	4769.77	1.96
111c_24s	17	4773.67	4778.62	5.64	17	4768.94	4790.50	0.22	17	4769.40	4785.58	0.43	17	4768.65	4772.84	0.82	17	**4767.14**	4768.48	2.42
111c_26s	17	4773.67	4778.62	5.23	17	4775.55	4791.52	0.23	17	4769.40	4786.36	0.46	17	4768.65	4771.97	0.87	17	**4767.14**	4769.50	2.57
111c_28s	17	4772.46	4777.03	5.54	17	4770.86	4808.75	0.26	17	4767.32	4771.75	0.48	17	4767.03	4773.76	0.93	17	**4765.52**	4767.97	2.78
200c_21s	31	8839.62	8879.98	19.96	31	8836.86	8895.10	1.06	31	8812.87	8853.33	1.93	31	8807.93	8843.39	3.67	31	**8766.04**	8790.80	10.48
250c_21s	37	10482.52	10518.32	21.58	37	10520.07	10610.38	2.24	37	10468.93	10574.56	3.98	37	10445.39	10498.57	7.54	37	**10379.98**	10414.45	21.46
300c_21s	44	12367.60	12421.75	47.53	43	12287.64	12344.05	3.69	43	12257.71	12289.04	6.43	43	12224.63	12247.72	12.20	43	**12202.49**	12209.94	35.44
350c_21s	50	14073.34	14226.03	63.01	50	14084.36	14158.02	6.03	50	14014.54	14099.40	10.77	50	13976.72	14031.65	22.44	49	**13908.96**	13929.89	60.99
400c_21s	59	16660.20	17119.89	71.70	58	16574.54	16673.88	10.93	58	16534.45	16601.55	21.87	58	16471.18	16507.07	47.23	58	**16398.13**	16424.29	111.84
450c_21s	65	18241.48	18902.03	80.75	64	18190.71	18269.19	14.58	64	18118.59	18176.05	26.31	64	18034.79	18091.34	66.57	64	**17938.85**	17973.93	145.73
500c_21s	73	20496.50	20997.04	89.95	72	20421.72	20552.23	18.01	72	20376.79	20430.86	36.26	71	20256.21	20332.82	83.19	71	**20207.81**	20245.13	198.97
Average		10419.47	10580.00	35.04		10398.22	10455.98	4.80		10369.65	10410.35	9.13		10338.24	10368.31	20.56		**10303.44**	**10319.86**	49.71
		0.817	1.799			0.682	1.219			0.465	0.826			0.258	0.503			**0.003**	**0.136**	
Computer	XEON E5410 2.33 GHz				Core I5 3.40 GHz															
Runs	10				10															

Table A.9: Results on the 20 customer EMH instances. Time is measured in minutes.

					MCWS/DBCA		48A			VNS/TS			AVNS			MHS (10k)				MSLS-60 ($L = 4 \times 60$)			
Instance	n	s	$\|\mathcal{A}\|$	BKS	m	Best	m	Best	t	m	Best	t	Best	Avg.	t	m	Best	Avg.	t	m	Best	Avg.	t
20c3sU1	20	3	558	1797.49	6	1797.51	6	1805.41	0.03	6	1797.49	0.69	1797.49	1797.49	0.16	6	1797.49	1797.49	0.08	6	1797.50	1797.50	0.005
20c3sU2	20	3	570	1574.77	6	1613.53	6	1574.78	0.02	6	1574.77	0.64	1574.78	1574.78	0.15	6	1574.78	1574.78	0.07	6	1574.78	1574.78	0.005
20c3sU3	20	3	594	1704.48	7	1964.57	6	1704.48	0.02	6	1704.48	0.64	1704.48	1704.48	0.13	6	1704.48	1704.48	0.07	6	1704.48	1704.48	0.005
20c3sU4	20	3	588	1482.00	6	1487.15	5	1482.00	0.03	5	1482.00	0.65	1482.00	1482.00	0.17	5	1482.00	1482.00	0.07	5	1482.00	1482.00	0.005
20c3sU5	20	3	558	1689.37	5	1752.73	6	1689.37	0.04	6	1689.37	0.67	1689.37	1689.37	0.18	6	1689.37	1689.37	0.07	6	1689.37	1689.37	0.005
20c3sU6	20	3	586	1618.65	6	1668.16	6	1618.65	0.03	6	1618.65	0.67	1618.65	1618.65	0.15	6	1618.65	1618.65	0.07	6	1618.65	1618.65	0.005
20c3sU7	20	3	528	1713.66	6	1730.45	6	1713.67	0.03	6	1713.66	0.64	1713.66	1713.66	0.19	6	1713.66	1713.87	0.07	6	1713.67	1713.67	0.004
20c3sU8	20	3	544	1706.50	6	1718.67	6	1722.78	0.03	6	1706.50	0.67	1706.50	1706.50	0.16	6	1706.50	1706.50	0.07	6	1706.50	1706.50	0.004
20c3sU9	20	3	488	1708.81	6	1714.43	6	1708.82	0.04	6	1708.81	0.66	1708.82	1708.82	0.19	6	1708.82	1709.65	0.07	6	1708.82	1708.82	0.004
20c3sU10	20	3	624	1181.31	5	1309.52	4	1181.31	0.02	4	1181.31	0.64	1181.31	1181.31	0.23	4	1181.31	1181.31	0.07	4	1181.31	1181.31	0.005
20c3sC1	20	3	772	1173.57	5	1300.62	4	1178.97	0.03	4	1173.57	0.62	1173.57	1173.57	0.38	4	1173.57	1173.57	0.07	4	1173.57	1177.49	0.006
20c3sC2	19	3	538	1539.97	5	1553.53	5	1539.97	0.02	5	1539.97	0.58	1539.97	1539.97	0.21	5	1539.97	1539.97	0.08	5	1539.97	1539.97	0.005
20c3sC3	12	3	278	880.20	4	1083.12	3	880.20	0.01	3	880.20	0.25	880.20	880.20	0.15	3	880.20	880.20	0.04	3	880.20	880.20	0.004
20c3sC4	18	3	608	1059.35	5	1091.78	4	1059.35	0.02	4	1059.35	0.53	1059.35	1077.71	0.23	4	1059.35	1059.94	0.06	4	1059.35	1059.35	0.005
20c3sC5	19	3	362	2156.01	7	2190.68	7	2156.01	0.02	7	2156.01	0.60	2156.01	2156.01	0.14	7	2156.01	2156.04	0.10	7	2156.01	2156.01	0.004
20c3sC6	17	3	276	2758.17	9	2883.71	8	2758.17	0.02	8	2758.17	0.71	2758.17	2758.17	0.14	8	2758.17	2758.17	0.08	8	2758.17	2758.17	0.003
20c3sC7	6	3	38	1393.99	5	1701.40	4	1393.99	0.00	4	1393.99	0.18	1393.99	1393.99	0.04	4	1393.99	1393.99	0.06	4	1393.99	1393.99	0.002
20c3sC8	18	3	232	3139.72	10	3319.74	9	3139.72	0.02	9	3139.72	0.62	3139.72	3139.72	0.08	9	3139.72	3139.72	0.12	9	3139.72	3139.72	0.003
20c3sC9	19	3	480	1799.94	6	1811.05	6	1799.94	0.02	6	1799.94	0.60	1799.94	1799.94	0.16	6	1799.94	1799.94	0.10	6	1799.94	1799.94	0.004
20c3sC10	15	3	222	2583.42	8	2644.11	8	2583.42	0.02	8	2583.42	0.45	2583.42	2600.39	0.09	8	2583.42	2583.42	0.07	8	2640.00	2640.00	0.003
S1_2i6s	20	6	896	1578.12	6	1614.15	6	1578.12	0.03	6	1578.12	0.71	1578.12	1578.12	0.16	6	1578.12	1578.12	0.07	6	1578.12	1578.12	0.007
S1_4i6s	20	6	972	1397.27	5	1541.46	5	1413.97	0.03	5	1397.27	0.75	1397.27	1397.27	0.16	5	1397.27	1397.27	0.07	5	1397.27	1397.27	0.008
S1_6i6s	20	6	744	1560.49	6	1616.20	6	1571.30	0.03	5	1560.49	0.73	1560.49	1560.49	0.20	5	1560.49	1560.49	0.07	5	1560.49	1560.49	0.005
S1_8i6s	20	6	822	1692.32	6	1882.54	6	1692.33	0.03	6	1692.32	0.74	1692.32	1692.32	0.17	6	1692.32	1692.32	0.07	6	1692.32	1692.32	0.007
S1_10i6s	20	6	1186	1173.48	5	1309.52	4	1173.48	0.03	4	1173.48	0.71	1173.48	1173.48	0.24	4	1173.48	1173.48	0.07	4	1173.48	1173.48	0.009
S2_2i6s	20	6	848	1633.10	6	1645.80	6	1645.80	0.03	6	1633.10	0.75	1633.10	1633.10	0.19	6	1633.10	1633.10	0.09	6	1633.10	1633.10	0.008
S2_4i6s	19	6	920	1505.07	6	1505.07	6	1505.07	0.02	5	1532.96	0.88	1505.07	1505.07	0.14	6	1505.07	1505.07	0.09	6	1505.07	1505.07	0.007
S2_6i6s	20	6	560	2431.33	10	3115.10	8	2660.49	0.04	7	2431.33	0.78	2431.33	2431.33	0.13	7	2431.33	2431.33	0.07	7	2431.33	2431.33	0.007
S2_8i6s	16	6	292	2158.35	9	2722.55	7	2175.66	0.02	7	2158.35	0.57	2158.35	2158.35	0.09	7	2158.35	2158.35	0.06	7	2158.35	2158.35	0.004
S2_10i6s	16	6	466	1585.46	6	1995.62	5	1585.46	0.02	6	1958.46	0.61	1585.46	1585.46	0.15	5	1585.46	1585.46	0.06	5	1585.46	1585.46	0.005
S1_4i2s	20	2	518	1582.20	6	1582.20	6	1598.91	0.03	6	1582.21	0.63	1582.21	1582.21	0.13	6	1582.21	1582.21	0.07	6	1582.21	1582.21	0.004
S1_4i4s	20	4	708	1460.09	6	1580.52	5	1483.19	0.03	5	1460.09	0.68	1460.09	1460.09	0.16	5	1460.09	1460.09	0.07	5	1460.09	1460.09	0.006
S1_4i6s	20	6	972	1397.27	5	1541.46	5	1413.97	0.03	5	1397.27	0.75	1397.27	1397.27	0.16	5	1397.27	1397.27	0.07	5	1397.27	1397.27	0.008
S1_4i8s	20	8	1320	1397.27	6	1561.29	6	1397.27	0.03	6	1397.27	0.82	1397.27	1397.27	0.17	5	1397.27	1397.27	0.07	5	1397.27	1397.27	0.010
S1_4i10s	20	10	1494	1396.02	5	1529.73	5	1396.02	0.03	5	1396.02	0.85	1396.02	1396.02	0.23	5	1396.02	1396.02	0.07	5	1396.02	1396.02	0.012
S2_4i2s	18	2	548	1059.35	5	1117.32	4	1059.35	0.02	4	1059.35	0.51	1059.35	1069.42	0.23	4	1059.35	1059.94	0.06	4	1059.35	1059.35	0.004
S2_4i4s	19	4	838	1446.08	6	1522.72	5	1446.08	0.02	5	1446.08	0.60	1446.08	1449.17	0.21	5	1446.08	1446.08	0.09	5	1446.08	1446.08	0.006
S2_4i6s	20	6	924	1434.14	6	1730.47	5	1434.14	0.02	5	1434.14	0.69	1434.14	1445.35	0.20	5	1434.14	1435.95	0.08	5	1434.14	1434.14	0.007
S2_4i8s	20	8	1256	1434.14	6	1786.21	5	1434.14	0.02	5	1434.14	0.75	1434.14	1434.14	0.20	5	1434.14	1435.95	0.08	5	1434.14	1434.14	0.010
S2_4i10s	20	10	1528	1434.13	6	1729.51	5	1434.13	0.02	5	1434.13	0.78	1434.13	1455.31	0.24	5	1434.13	1435.94	0.09	5	1434.13	1434.13	0.014
Average				1635.43		1774.15		1644.75	0.03		1645.45	0.65	1635.43	1637.45	0.17		1635.43	1635.62	0.07		1636.84	1636.94	0.006
Gap above BKS (%)						8.72		0.46			0.63		0.00	0.15			0.00	0.01			0.05	0.06	
NBKS						2		29			38		40				40				39		
Computer					P4 3.2 GHz		Core I5 2.8 GHz			Core I5 2.67 GHz			Core I5 2.67 GHz			XEON E5410 2.33 GHz				Core I5 3.40 GHz			
Runs					N.A.		1			10			10			10				10			

Table A.10: Detailed results of MSLS-60 ($L = 4 \times 60$) on the large EMH instances. Time is measured in minutes.

Instance	t_{P1}	t_{P2}	t_{SP}	Best$_{P2}$	$\%P1$	$\%P2$
111c_21s	1.25	0.60	0.01	4778.36	0.17	0.03
111c_22s	1.30	0.62	0.01	4770.76	0.07	0.00
111c_24s	1.61	0.78	0.01	4775.11	0.17	0.00
111c_26s	1.68	0.85	0.01	4772.00	0.10	0.00
111c_28s	1.80	0.93	0.01	4772.41	0.14	0.00
200c_21s	8.02	2.32	0.04	8918.41	1.74	0.00
250c_21s	16.29	4.73	0.20	10557.43	1.71	0.00
300c_21s	27.92	6.79	0.32	12344.03	1.16	0.00
350c_21s	44.53	10.24	4.97	14084.01	1.26	0.00
400c_21s	67.62	15.77	26.70	16658.52	1.58	0.00
450c_21s	97.61	23.36	22.40	18231.13	1.63	0.00
500c_21s	134.02	29.39	31.72	20467.61	1.40	0.00
Average	33.64	8.03	7.20	10427.48	0.927	0.003

Appendix B. Operator pseudocodes

In the pseudocode descriptions, we will use the notation $[c, A_{out}, A_{in}]$ where c denotes the cost of the best saving found by the operator, A_{out} denotes the set of arcs removed by the best operator move, and A_{in} denotes the set of arcs added by the best operator move to reconstruct the routes.

Algorithm 6 Clarke and Wright Savings (CWS)

Input: S: Initial feasible solution. $c = 0$: cost of the best saving. $A_{out} = \emptyset$: arcs removed by the best operator move. $A_{in} = \emptyset$: arcs added by the best operator move to reconstruct the routes.

Output: S': Improved solution.

$[c, A_{out}, A_{in}] \longleftarrow [0, \emptyset, \emptyset]$
for each route $R_1 \in S$ **do**
  for each route $R_2 \in S \setminus \{R_1\}$ **do**
    Let i be the last customer of R_1 and j the first customer of R_2
    Let $(i, 0, p_{i0})$ be the last arc of R_1, and $(0, j, p_{0j})$ be the first arc of R_2
    for each $p' \in \mathscr{P}_{ij}$ **do**
      $\hat{c} \longleftarrow c(i, 0, p_{i0}) + c(0, j, p_{0j}) - c(i, j, p')$
      if $\hat{c} > c$ **then**
        $R \longleftarrow R_1 \setminus (i, 0, p_{i0}) \cup R_2 \setminus (j, 0, p_{j0}) \cup (i, j, p')$
        if R is feasible **then**
          $c \longleftarrow \hat{c}$
          $A_{out} \longleftarrow \{(i, 0, p_{i0}), (j, 0, p_{j0})\}$
          $A_{in} \longleftarrow \{(i, j, p')\}$
        end if
      end if
    end for
  end for
end for
if $[c, A_{out}, A_{in}] \neq [0, \emptyset, \emptyset]$ **then**
  Remove the arcs in A_{out} and add the arcs in A_{in} to obtain a new solution S'
end if

Algorithm 7 2-OPT*

Input: S: Initial feasible solution. $c = 0$: cost of the best saving. $A_{out} = \emptyset$: arcs removed by the best operator move. $A_{in} = \emptyset$: arcs added by the best operator move to reconstruct the routes.

Output: S': Improved solution.

$[c, A_{out}, A_{in}] \longleftarrow [0, \emptyset, \emptyset]$
for each route $R_1 \in S$ **do**
  for each route $R_2 \in S \setminus \{R_1\}$ **do**
    for all arcs $(i_1, i_2, p_{i_1 i_2}) \in R_1$ and $(j_1, j_2, p_{j_1 j_2}) \in R_2$ **do**
      for all $p_1 \in \mathscr{P}_{i_1 j_2}$ and $p_2 \in \mathscr{P}_{j_1 i_2}$ **do**
        $\hat{c} \longleftarrow c(i_1, i_2, p_{i_1 i_2}) + c(j_1, j_2, p_{j_1 j_2}) - c(i_1, j_2, p_1) - c(j_1, i_2, p_2)$
        if $\hat{c} > c$ **then**
          $R'_1 \longleftarrow R_1 \setminus (i_1, i_2, p_{i_1 i_2}) \cup (i_1, j_2, p_1)$
          $R'_2 \longleftarrow R_2 \setminus (j_1, j_2, p_{j_1 j_2}) \cup (j_1, i_2, p_2)$
          if R'_1 and R'_2 are feasible **then**
            $c \longleftarrow \hat{c}$
            $A_{out} \longleftarrow \{(i_1, i_2, p_{i_1 i_2}), (j_1, j_2, p_{j_1 j_2})\}$
            $A_{in} \longleftarrow \{(i_1, j_2, p_1), (j_1, i_2, p_2)\}$
          end if
        end if
      end for
    end for
  end for
end for
if $[c, A_{out}, A_{in}] \neq [0, \emptyset, \emptyset]$ **then**
  Remove the arcs in A_{out} and add the arcs in A_{in} to obtain a new solution S'
end if

Algorithm 8 Sequence relocate

Input: S: Initial feasible solution. σ: customer sequence length. $c = 0$: cost of the best saving. $A_{out} = \emptyset$: arcs removed by the best operator move. $A_{in} = \emptyset$: arcs added by the best operator move to reconstruct the routes.

Output: S': Improved solution.

$[c, A_{out}, A_{in}] \longleftarrow [0, \emptyset, \emptyset]$
for each route $R_1 \in S$ **do**
  for each route $R_2 \in S \setminus \{R_1\}$ **do**
    for all pairs of customer sequences $(i, i_1, \dots, i_\sigma, j) \in R_1$ and $(j_1, j_2) \in R_2$ **do**
      for all triplets $p_1 \in \mathscr{P}_{ij}$, $p_2 \in \mathscr{P}_{j_1 i_1}$ and $p_3 \in \mathscr{P}_{i_\sigma j_2}$ **do**
        $\hat{c} \longleftarrow c(i, i_1, p_{ii_1}) + c(i_\sigma, j, p_{i_\sigma j}) + c(j_1, j_2, p_{j_1 j_2}) - c(i, j, p_1) - c(j_1, i_1, p_2) - c(i_\sigma, j_2, p_3)$
        if $\hat{c} > c$ **then**
          $R_1' \longleftarrow R_1 \setminus (i, i_1, p_{ii_1}) \setminus (i_\sigma, j, p_{i_\sigma j}) \cup (i, j, p_1)$
          $R_2' \longleftarrow R_2 \setminus (j_1, j_2, p_{j_1 j_2}) \cup (j_1, i_1, p_2) \cup (i_\sigma, j_2, p_3)$
          if R_1' and R_2' are feasible **then**
            $c \longleftarrow \hat{c}$
            $A_{out} \longleftarrow \{(i, i_1, p_{ii_1}), (i_\sigma, j, p_{i_\sigma j}), (j_1, j_2, p_{j_1 j_2})\}$
            $A_{in} \longleftarrow \{(i, j, p_1), (j_1, i_1, p_2), (i_\sigma, j_2, p_3)\}$
          end if
        end if
      end for
    end for
  end for
end for
if $[c, A_{out}, A_{in}] \neq [0, \emptyset, \emptyset]$ **then**
  Remove the arcs in A_{out} and add the arcs in A_{in} to obtain a new solution S'
end if

Algorithm 9 Cyclic Exchange (2 routes)

Input: S: Initial feasible solution. σ: customer sequence length. w: customer sequence length. $c = 0$: cost of the best saving. $A_{out} = \emptyset$: arcs removed by the best operator move. $A_{in} = \emptyset$: arcs added by the best operator move to reconstruct the routes.

Output: S': Improved solution.

$[c, A_{out}, A_{in}] \longleftarrow [0, \emptyset, \emptyset]$
for each route $R_1 \in S$ **do**
  for each route $R_2 \in S \setminus \{R_1\}$ **do**
    for all customer sequences $(i, i_1, \ldots, i_\sigma, j) \in R_1$ and $(k, j_1, \ldots, j_w, l) \in R_2$ **do**
      for all path combinations $p_1 \in \mathscr{P}_{ij_1}$, $p_2 \in \mathscr{P}_{j_w j}$, $p_3 \in \mathscr{P}_{ki_1}$, $p_4 \in \mathscr{P}_{i_\sigma l}$ **do**
        $\hat{c} \longleftarrow c(i, i_1, p_{ii_1}) + c(i_\sigma, j, p_{i_\sigma j}) + c(k, j_1, p_{kj_1}) + c(j_w, l, p_{j_w l}) - c(i, j_1, p_1) - c(j_w, j, p_2) - c(k, i_1, p_3) - c(i_\sigma, l, p_4)$
        if $\hat{c} > c$ **then**
          $R'_1 \longleftarrow R_1 \setminus (i, i_1, p_{ii_1}) \setminus (i_\sigma, j, p_{i_\sigma j}) \cup (i, j_1, p_1) \cup (j_w, j, p_2)$
          $R'_2 \longleftarrow R_2 \setminus (k, j_1, p_{kj_1}) \setminus (j_w, l, p_{j_w l}) \cup (k, i_1, p_3) \cup (i_\sigma, l, p_4)$
          if R'_1 and R'_2 are feasible **then**
            $c \longleftarrow \hat{c}$
            $A_{out} \longleftarrow \{(i, i_1, p_{ii_1}), (i_\sigma, j, p_{i_\sigma j}), (k, j_1, p_{kj_1}), (j_w, l, p_{j_w l})\}$
            $A_{in} \longleftarrow \{(i, j_1, p_1), (j_w, j, p_2), (k, i_1, p_3), (i_\sigma, l, p_4)\}$
          end if
        end if
      end for
    end for
  end for
end for
if $[c, A_{out}, A_{in}] \neq [0, \emptyset, \emptyset]$ **then**
  Remove the arcs in A_{out} and add the arcs in A_{in} to obtain a new solution S'
end if