Solving a Multi-objective Vehicle Routing Problem with Synchronization Constraints

Briseida Sarasola[1(✉)] and Karl F. Doerner[1,2]

[1] Institut für Business Decisions and Analytics, Oskar-Morgenstern-Platz 1, Wien, Austria
briseida.sarasola@univie.ac.at
[2] Data Science @ Uni Vienna, Vienna, Austria
karl.doerner@univie.ac.at

Abstract. In this paper, we solve a multi-objective vehicle routing problem with synchronization constraints at the delivery location. Our work is motivated by the delivery of parcels and consumer goods in urban areas, where customers may await deliveries from more than one service provider on the same day. In addition to minimizing travel costs, we also consider a second objective to address customer preferences for a compact schedule at the delivery location, so that all deliveries to a customer happen within a non-predefined time interval. To determine the Pareto fronts, three metaheuristic methods based on large neighborhood search are developed. The results on small instances are compared with an ϵ-constraint method using an exact solver. Results for large real-world instances are also presented.

Keywords: Vehicle routing problem · Synchronization · Multi-objective optimization

1 Introduction

The vehicle routing problem with synchronization constraints at the delivery location (VRPSCDL) is motivated by a current situation in urban transportation, where a single recipient often expects several orders from more than one service provider on the same day [14]. Service providers aim at minimizing the costs, while recipients are customers that wish to receive all orders approximately at the same time. Therefore the decision involves several stakeholders that need to find a compromise between transportation costs and compact schedules at the delivery location.

A real-life application of the VRPSCDL arises in housing and decoration logistics [16,20], where several furniture suppliers offer their products on a shared online platform. A customer may buy products from different suppliers, that are later shipped with different logistics companies. However, customers wish to receive all products within a small time interval. As a result, furniture suppliers

© Springer Nature Switzerland AG 2021
M. Mes et al. (Eds.): ICCL 2021, LNCS 13004, pp. 532–546, 2021.
https://doi.org/10.1007/978-3-030-87672-2_35

need to find a way to collaborate with each other to improve customer service. Other possible applications include supermarkets, construction sites, and in general, any business that expects deliveries from several suppliers.

The VRPSCDL has been solved in the literature by allowing a maximum amount of idle time at the delivery location, where idle time is defined as non service time between the first and the last delivery to a given location. Existing results show that it is possible to substantially reduce the idle time by more than 40% without sacrificing the travel cost (3% longer travel times) [14]. However, the problem was addressed by first determining a fix idle time/service time ratio and then solving the single-objective optimization problem, but it is in general not easy to find appropriate values for the ratios. We aim at supporting decision makers by providing several solutions, so that a suitable trade-off solution can be chosen. To tackle this problem we model the multi-objective problem as an extension of the single-objetive problem. We formulate two objectives to minimize travel cost and idle time at the delivery location.

Many of the initial approaches to tackle multi-objective routing problems in the literature use highly specialized population-based metaheuristics [5,9], such as evolutionary algorithms [6,13,17]. It has been recently shown that combining several single-objective methods [3,8,18] as well as using a single-objective algorithm within a solution framework that treats all other objectives as constraints [1,12] provide excellent results in solving multi-objective vehicle routing problems. In this work, we focus on solution techniques that use powerful single-objective neighborhood search methods to build the Pareto set.

The remainder of this paper is structured as follows. Section 2 introduces the problem. Our solution methods are described in Sect. 3. Results are presented in Sect. 4. Finally, Sect. 5 gives some conclusions and lines of future work.

2 Problem Statement

The multi-objective VRPSCDL (MO-VRPSCDL) is based on the VRPSCDL, which is defined using a multi-commodity flow formulation [14]. We introduce here the necessary notation to later define the multi-objective problem. In the single-objective formulation, the problem consists of minimizing the total travel time to serve n deliveries to m delivery locations from p depots. A solution to the VRPSCDL is a set of routes that serves all deliveries once. Each route starts at a departure depot and ends at the corresponding arrival depot, and only serves deliveries associated to that depot. A delivery location receives one or more deliveries, which must be fulfilled in a compact schedule. The notation for the sets, data, and variables of the VRPSCDL are summarized next.

Sets:

- D is the set of deliveries.
- P_1 is the set of departure depots.
- P_2 is the set of arrival depots.
- $N = P_1 \cup D \cup P_2$ is the total set of nodes.
- U is the set of delivery locations.

- D'_l is the set of all deliveries at location $l \in U$.
- D''_i is the set of all deliveries to be fulfilled by depot $i \in P_1$.
- V is the set of all vehicles.
- V_i is the set of all vehicles associated to depot $i \in P_1$.

Problem data:

- c_{ij} is the travel time from node i to j, $i, j \in N$.
- d_i is the service time at node $i \in N$.
- w_l is the maximum idle time at delivery location $l \in U$.
- Q is the vehicle capacity.
- T is the return time at the depot.

Variables:

- x_{ijk} is equal to 1 if vehicle k travels from node i to node j, 0 otherwise.
- s_{ik} is the time at which service starts at delivery i by vehicle k.
- $start_l$ is the time at which the first delivery at location l starts.
- $last_l$ is the time at which the last delivery at location l ends.
- z_{ij} is 1 if, given two deliveries i, j to the same location, i is scheduled before j, 0 otherwise.

In previous work, the maximum idle time w_l at the delivery location has been defined as the maximum amount of non service time between the first and the last delivery at location l. The value of w_l depends on the total service time of all deliveries to location l and a parameter $\alpha \geq 0$, so that $w_l = \alpha \cdot \sum_{i \in D'_l} d_i$, where D'_l is the set of all deliveries to l. The value of α controls the percentage of allowed idle time at each delivery location. The maximum idle time is thus modeled as Constraint (1) in the single-objective problem.

$$last_l - start_l - \sum_{i \in D'_l} d_i \leq \alpha \cdot \sum_{i \in D'_l} d_i \qquad \forall l \in U \tag{1}$$

The model imposes that the service of two or more deliveries at a given location cannot overlap, so that a vehicle must wait if another vehicle is currently serving the customer at the delivery location. Moreover, a vehicle might also wait if it arrives to the location before service can start according to the synchronization constraints. Following previous work, we assume that vehicles leave each node as early as possible, so that they departure from the depot at time 0 and leave the delivery location right after serving the delivery.

The MO-VRPSCDL can be defined as a minimization problem of the form:

$$\min f(\boldsymbol{x}, \boldsymbol{\alpha}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{\alpha})) \tag{2}$$

s.t.

$$g_i(\boldsymbol{x}) \geq 0 \quad \forall i \tag{3}$$

$$h_j(\boldsymbol{x}) = 0 \quad \forall j \tag{4}$$

The first objective $f_1(\boldsymbol{x})$ is the total travel time of the VRPSCDL.

$$f_1(\boldsymbol{x}) = \sum_{i \in N} \sum_{j \in N} \sum_{k \in V} x_{ijk} \cdot c_{ij} \tag{5}$$

The second objective $f_2(\boldsymbol{\alpha})$ aims at minimizing the maximum α_l for every delivery location l in the problem.

$$f_2(\boldsymbol{\alpha}) = \max_{l \in D'_l} \left\{ \alpha_l \quad \middle| \quad \alpha_l = \frac{last_l - start_l}{\sum_{i \in D'_l} d_i} - 1 \right\} \tag{6}$$

The problem constraints of the MO-VRPSCDL are the same as in the VRPSCDL except Constraint (1), which is removed and treated as the second objective.

3 Solution Techniques

This section provides a description of our solution methods based on neighborhood search. All described methods provide an approximation of the Pareto front in the MO-VRPSCDL and use an archive to maintain the set of non-dominated solutions. A solution s_i is non-dominated if there are no other solutions s_j, $i \neq j$, with $f_1(s_j) < f_1(s_i)$ and $f_2(s_j) \leq f_2(s_i)$, or $f_1(s_j) \leq f_1(s_i)$ and $f_2(s_j) < f_2(s_i)$. Our local search algorithms try to improve one objective at each step, while the other objective values might deteriorate, so that each step follows a "pure local search" scheme [11].

We use the Solomon C1 heuristic to build initial solutions that are feasible with respect to some value of α. The algorithm selects depots in a random order and builds a set of routes considering all deliveries of the selected depot. To ensure that the solution is feasible, we use "self-imposed time windows", which provide an earliest and latest start time for all deliveries to the same location [14]. These time windows are not predefined, so all locations have an initial self-imposed time window equal to the work day duration $[0, T]$. When a delivery is inserted in the solution, its time window is updated. We denote this algorithm $Solomon\_C1(\alpha)$. If $\alpha = \infty$, self-imposed time windows are not used.

3.1 Multi-Directional Local Search

We solve the MO-VRPSCDL by integrating it in Multi-Directional Local Search (MDLS) [18]. This framework requires the definition of a local search method for each objective. Therefore, since the MO-VRPSCDL is a bi-objective problem, we need to define two local search methods. In our case, each execution of a local search is a single iteration of an Adaptive Large Neighborhood Search (ALNS). We denote this method MDLS-ALNS (see pseudocode in Algorithm 1).

For the first objective, we use $ALNS_1$, which is based on the existing ALNS for the VRPSCDL [14]. Its destroy and remove operators are described in detail

in previous work. Originally, it handles the maximum idle time constraint by setting self-imposed time windows on the delivery locations. However, MDLS considers the maximum idle time as a second objective, so self-imposed time windows are not used, i.e. an insertion is always feasible as long as other problem constraints are not violated (flow and timing constraints, vehicle capacity, return time at the depot, and deliveries served sequentially at the delivery location).

For the second objective, we define $ALNS_2$ as an ALNS with the following features. The destroy operators are the random removal and the worst removal operator. The first one selects ξ deliveries using a uniform distribution and removes them from the solution, whereas the second one removes ξ deliveries that correspond to delivery locations with high values of α. This latter uses a randomization factor to avoid always removing the same deliveries. The repair operators are the greedy insertion and the 2-regret insertion operator. Both of them consider the cost of inserting a delivery as the maximum value of α in the solution after inserting the delivery.

In our pseudocode, $ALNS_1$ and $ALNS_2$ are called with two parameters, where the first one is the initial solution of the ALNS and the second one is the maximum allowed value of α. In particular, MDLS relies on $ALNS_2$ to find solutions that are good with respect to the second objective, so it does not impose a constraint on the maximum allowed idle time and the second parameter is $\alpha = \infty$.

The initial solution s_0 of MDLS is generated by solving the VRPSCDL with $\alpha = \infty$ (line 1) and it is used to initialize the archive A (line 2). Then, MDLS iteratively selects one random solution z from the archive (line 5) at iteration i and applies $ALNS_1$ and $ALNS_2$ to obtain two new solutions, $s_{i,1}$ and $s_{i,2}$ (lines 6–7), that are used to update the archive (line 8).

Algorithm 1. MDLS-ALNS

1: $s_0 \leftarrow Solomon\_C1(\infty)$
2: $A \leftarrow \{s_0\}$
3: **while** stopping condition not met **do**
4: $\quad i \leftarrow 1$
5: $\quad$ Select a random solution z from A
6: $\quad s_{i,1} \leftarrow ALNS_1(z, \infty)$
7: $\quad s_{i,2} \leftarrow ALNS_2(z, \infty)$
8: $\quad$ Update archive A with $s_{i,1}$ and $s_{i,2}$
9: **end while**

3.2 ϵ-Constraint Method

The ϵ-constraint method (ECM) for multi-objective problems consists of optimizing one single objective, while formulating the second objective as a constraint [4]. Although it has been mainly used in association with exact algorithms, the ECM and some of its variants has been shown to provide good results in combination with heuristics to approximate the Pareto front [1,7,12].

We embed $ALNS_1$ in the ECM framework and denote the resulting method ECM-ALNS (see Algorithm 2). The method proceeds as follows. We first solve the single-objective sub-problem π_0 with $\alpha_0 = 0$, so that no idle time is allowed at any delivery location (line 1). To obtain this first solution s_0, the solver is allowed to run until I_0 iterations without improvement are reached. Next, we solve the single-objective sub-problem π_∞ with $\alpha_\infty = \infty$ to get a minimum reference value for the first objective (line 2). The initial archive A contains thus s_0 and s_∞ (line 3). Then, the value of the maximum allowed α is set back to 0 (line 4) and iteratively increased by ϵ, so that the single-objective sub-problem π_i with $\alpha_i = \alpha_{i-1} + \epsilon$ is solved, $i > 0$ (lines 6-8). After I_n iterations without improvement, $ALNS_1$ stops and returns solution s_i, which is used to update the archive (line 9). Following previous work [12] and our own preliminary experiments, we allocate comparatively longer execution times to obtain the initial solution s_0 by setting $I_0 \gg I_n$.

The ECM can proceed with both increasing and decreasing constraint values. We choose to increase the values of α because the solutions of a sub-problem with α_i are also feasible solutions of sub-problems with α_j if $\alpha_j > \alpha_i$, but the opposite is in general not true. Preliminary experiments show that, instead of generating a new solution from scratch for each problem π_i, $i > 0$, better results can be obtained by using the solution s_{i-1} of π_{i-1} found in the previous iteration as the initial solution of the $ALNS_1$ to solve π_i. This can be achieved without repairing the solution or using penalties if the ECM operates for increasing values of α.

Algorithm 2. ECM-ALNS

1: $s_0 \leftarrow ALNS_1(Solomon\_C1(0), 0)$
2: $s_\infty \leftarrow ALNS_1(Solomon\_C1(\infty), \infty)$
3: $A \leftarrow \{s_0, s_\infty\}$
4: $\alpha_0 = 0$
5: **while** stopping condition not met **do**
6: $\quad i \leftarrow 1$
7: $\quad \alpha_i \leftarrow \alpha_{i-1} + \epsilon$
8: $\quad s_i \leftarrow ALNS_1(s_{i-1}, \alpha_i)$
9: $\quad$ Update archive A with s_i
10: **end while**

3.3 Heuristic Box Splitting

Heuristic Box Splitting (HBS) addresses some of the problems arising in the ECM [12]. Instead of iteratively solving a single-objective sub-problem with increasing (or decreasing) constraint values, HBS forms a rectangle determined by the minimum and maximum values of each objective. This rectangle is iteratively split in halves, so that a single-objective sub-problem with a constraint determined by the splitting value is solved.

We embed $ALNS_1$ in HBS (HBS-ALNS) as follows (see Algorithm 3). Similar to the ECM, the single-objective sub-problem π_0 with $\alpha_0 = 0$ is solved to obtain s_0 after I_0 iterations without improvement (line 1). Then, we solve the single-objective problem with $\alpha_\infty = \infty$ and obtain thus s_∞ with the same termination criterion as before (line 2). The initial archive contains solutions s_0 and s_∞ (line 3). Points $(f_1(s_\infty), f_2(s_\infty))$ and $(f_1(s_0), f_2(s_0))$ form a rectangle that determines the area where HBS searches for new solutions (lines 4–5). The initial rectangle is added to the rectangle set S (line 6) and HBS runs until no more rectangles are available as follows. It selects the rectangle $R(y^1, y^2)$ with the larger area (line 9). The rectangle is split in two halves, so that the line that halves the rectangle determines the value of the constraint α_i, where $i > 0$ is the current iteration. For example, in the first iteration after creating the initial rectangle, $\alpha_1 = 0.5 \cdot f_2(s_\infty)$, and in general, $\alpha_i = 0.5 \cdot (y_2^1 + y_2^2)$ (line 10). The solver runs using this constraint until I_n iterations without improvement are reached (line 12), and the found solution s_i is used to update the archive (lines 13–16). The solution also allows the algorithm to discard areas that are dominated by the found solutions and to create new rectangles that are added to S (line 17).

Algorithm 3. HBS-ALNS

```
 1: s_0 ← ALNS_1(Solomon_C1(0), 0)
 2: s_∞ ← ALNS_1(Solomon_C1(∞), ∞)
 3: A ← {s_0, s_∞}
 4: z^1 ← (f_1(s_∞), f_2(s_∞))
 5: z^2 ← (f_1(s_0), f_2(s_0))
 6: S ← {R(z^1, z^2)}
 7: while stopping condition not met do
 8:     i ← 1
 9:     Select R(y^1, y^2) ∈ S with the largest area
10:     α_i ← 0.5 · (y_2^1 + y_2^2)
11:     Select z ∈ A with f_1(z) ≤ f_1(z_j) such that z_j ∈ A and f_2(z_j) ≤ α_i
12:     s_i ← ALNS_1(z, α_i)
13:     if s_i is dominated then
14:         y^2 ← (y_1^2, α_i)
15:     else
16:         Update archive A with s_i
17:         Update S according to HBS rules
18:     end if
19:     if S = ∅ then
20:         S ← {R(z^1, z^2)}
21:     end if
22: end while
```

Similar to the ECM, preliminary experiments show that better results can be obtained by using solutions found with α_i as the initial solution for solving problems with $\alpha_j > \alpha_i$ (line 11). In the ECM this step is straightforward, since α increases monotonically. However, HBS needs to select a solution $z \in A$ that

is feasible with respect to the current α_i. This is done by choosing the solution z with the best value of the first objective f_1 among those those solutions $z_j \in A$ that are feasible with respect to α_i (line 11).

The original HBS terminates when the rectangle set S is empty. We modify it to run until the maximum runtime is reached. If S is empty, the initial rectangle is added to S (lines 18–20).

4 Experiments and Results

This section presents the results obtained by our algorithms. We first compare the performance of our neighborhood search based methods with solutions obtained by an exact solver embedded in the ECM (see Sect. 4.1). Then, we evaluate our algorithms by solving large instances obtained from real-world data in Sect. 4.2. We use the set of instances for the VRPSCDL without instances 0, 1, 20, and 21, because those instances define a single depot and there is thus no synchronization involved.

Results report the normalized hypervolume (HV) as a percentage of the reference HV [21]. To calculate the reference HV for each instance, we build reference sets with all non-dominated solutions found in all our experiments. The nadir point is estimated to be 10% larger than the worst found values for each objective [10,12], so that extreme solutions also contribute to the HV. Although the HV is currently considered the most relevant performance indicator in multi-objective optimization [2], we also include some results concerning the overall non-dominated vector generation ($ONVG$) [19] and spacing (SP) [15] to obtain further information about the performance of our algorithms.

The exact solver is an implementation of the model of the VRPSCDL using CPLEX. We embed this solver in the ECM framework as described in Sect. 3.2, and denote this algorithm ECM-CPLEX. It runs for 86,400 s and dedicates a maximum of 3,600 s to solve each sub-problem. We use $\epsilon = 0.01$ in all experiments. Each call to CPLEX runs until it finds the optimum solution or the maximum runtime is reached. The exact solver is run only once for each instance.

We allow each combination of multi-objective framework and neighborhood search algorithm to run for a maximum time of 3,600 s. Following previous settings in the literature, each iteration of MDLS consists of one iteration of each ALNS. Same as described above, ECM-ALNS uses $\epsilon = 0.01$. ECM-ALNS and HBS-ALNS first obtain their reference points using $I_0 = 5,000$, and then solve each sub-problem with $I_n = 500$. For each instance and metaheuristic, we report the average and the best of 5 independent runs.

All algorithms were implemented in Java 1.7. The exact solver requires CPLEX 12.6.2. Each experiment runs on a Xeon core at 2.50 GHz with 64 GB shared RAM and deactivated hyperthreading.

4.1 Small Instances

First we compare our results on a small dataset with instances that contain $p \in [2, 3]$ depots, $n \in [10, 40]$ deliveries, and $m \in [6, 23]$ delivery locations.

Table 1 reports the HV obtained by the ECM-CPLEX as well as the average and best HV obtained by ECM-ALNS, HBS-ALNS, and MDLS-ALNS. Best results for each instance are highlighted in bold. Our results show that ECM-CPLEX is not always able to find the best-known Pareto front due to runtime restrictions. It obtains the best HV in 9 of 18 instances, but its results are weaker for the larger instances 8–9 and 16–19, each with 30 to 40 deliveries. In addition, the exact solver for the VRPSCDL only optimizes the travel cost, while its schedules are just determined to be feasible by CPLEX. For this reason, it often obtains worse Pareto fronts than the heuristic algorithms, which try to schedule deliveries in a compact manner. The best overall results are provided by HBS-ALNS, both regarding the average quality of the Pareto fronts as well as the best of 5 runs. In particular, it obtains the best average result in 11 of 18 instances with an average $HV = 98.5\%$. MDLS-ALNS provides competitive results and finds the best result for 8 of 18 instances. ECM-ALNS is outperformed by all algorithms.

Table 1. HV (%) obtained by each algorithm solving small instances.

Instance	ECM-	ECM-ALNS		HBS-ALNS		MDLS	
	CPLEX	Avg	Best	Avg	Best	Avg	Best
2	99.8	97.9	99.8	**99.9**	**99.9**	97.0	98.9
3	**100.0**	98.8	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
4	**98.8**	97.1	98.0	98.5	98.7	98.1	98.2
5	95.9	94.9	95.4	**96.0**	**96.0**	89.9	91.8
6	**100.0**	97.5	98.7	**100.0**	**100.0**	95.7	97.7
7	**100.0**	97.6	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
8	97.1	99.1	99.9	**100.0**	**100.0**	94.3	94.9
9	90.2	97.4	99.2	**100.0**	**100.0**	95.0	98.1
10	**100.0**	97.7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
11	97.7	95.2	96.8	**98.0**	**98.1**	95.0	95.2
12	**100.0**	94.4	99.5	96.3	**100.0**	**100.0**	**100.0**
13	**100.0**	97.9	99.1	99.6	99.9	**100.0**	**100.0**
14	**96.7**	93.1	95.7	96.0	**97.1**	95.8	96.3
15	**100.0**	79.9	86.7	97.3	98.5	**100.0**	**100.0**
16	97.8	96.7	97.9	**98.2**	98.3	98.0	**98.6**
17	95.8	95.6	96.6	97.5	**98.5**	**98.1**	98.4
18	91.1	94.7	96.4	**98.3**	**98.9**	95.3	96.0
19	83.6	95.2	96.6	97.6	98.9	**99.9**	**99.9**
Avg	96.9	95.6	97.6	**98.5**	**99.0**	97.3	98.0

4.2 Large Instances

Table 2 show the results obtained using the set of large instances with $p \in [2, 6]$ depots, $n \in [100, 300]$ deliveries, and $m \in [65, 100]$ delivery locations. For each algorithm and instance the average and the best HV (%) are reported. Our results show that HBS-ALNS outperforms the other solution methods in 15 of 18 instances regarding the average results, and it is able to find the best approximation of the Pareto front in 14 of 18 instances. In the remaining instances, ECM-ALNS finds the best results and always outperforms MDLS-ALNS.

Table 2. HV (%) obtained by each algorithm solving large instances.

Instance	ECM-ALNS		HBS-ALNS		MDLS-ALNS	
	Avg	Best	Avg	Best	Avg	Best
22	97.0	97.3	**98.4**	**98.5**	96.8	97.7
23	97.2	**98.0**	**97.9**	**98.0**	96.3	97.0
24	97.5	97.7	**98.2**	**98.4**	96.0	97.1
25	**95.9**	**96.6**	95.7	96.4	91.3	92.2
26	95.8	**96.5**	**96.3**	**96.5**	91.8	92.6
27	96.9	97.1	**97.8**	**98.2**	92.7	93.4
28	95.3	95.6	**96.3**	**96.9**	87.3	88.7
29	**95.1**	**95.6**	94.8	95.5	85.1	85.3
30	95.2	95.6	**96.6**	**97.0**	89.5	90.0
31	95.0	95.8	**96.4**	**96.6**	88.9	89.5
32	95.8	96.6	**96.8**	**97.1**	87.6	88.5
33	94.9	**96.3**	**96.0**	96.2	85.1	85.6
34	**95.9**	**96.9**	95.7	95.9	83.7	84.1
35	95.4	96.0	**96.4**	**96.9**	86.1	88.0
36	93.0	94.4	**95.6**	**95.9**	83.9	84.5
37	94.5	95.3	**95.1**	**95.9**	82.6	83.3
38	94.8	95.9	**95.6**	**96.0**	83.0	83.7
39	94.3	94.6	**95.6**	**96.3**	84.9	85.5
Avg	95.5	96.2	**96.4**	**96.8**	88.5	89.2

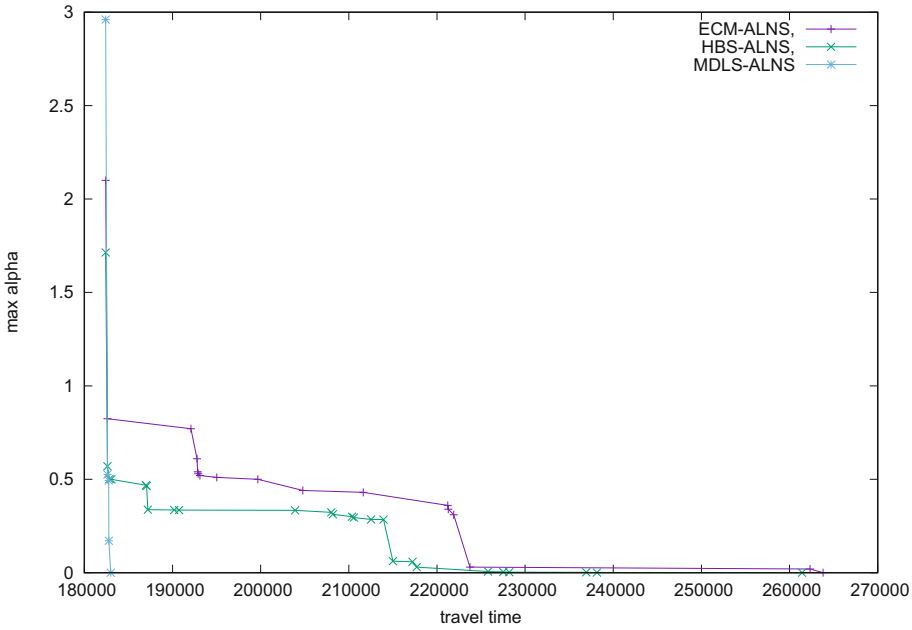
Table 3 shows average and best results of the $ONVG$ indicator obtained by each algorithm on each instance. This measure is the number of solutions in the Pareto front approximation and larger values are considered to be better. Although it poses some problems, as bad approximations with many non-dominated points are preferred over better approximations with only a few points, it can be used together with the HV to provide additional insights about the algorithms. Our results show that ECM-ALNS consistently finds mores solutions than the other algorithms, both in the average and in the best case. It also

finds more solutions for larger instances (80–100) than for medium-sized ones (40–80), which seems a priori logical because the solution space is likely larger. HBS-ALNS finds more solutions than MDLS-ALNS, but it should be noted that the number of solutions for both algorithms does not depend on the instance size. The difference between ECM-ALNS and HBS-ALNS can be explained by how they explore the solution space. While ECM-ALNS increases α by very small values in each iteration and is therefore able to find many similar (but different) solutions, HBS-ALNS tries to explore different regions of the solution space and therefore finds less (but more diverse) solutions.

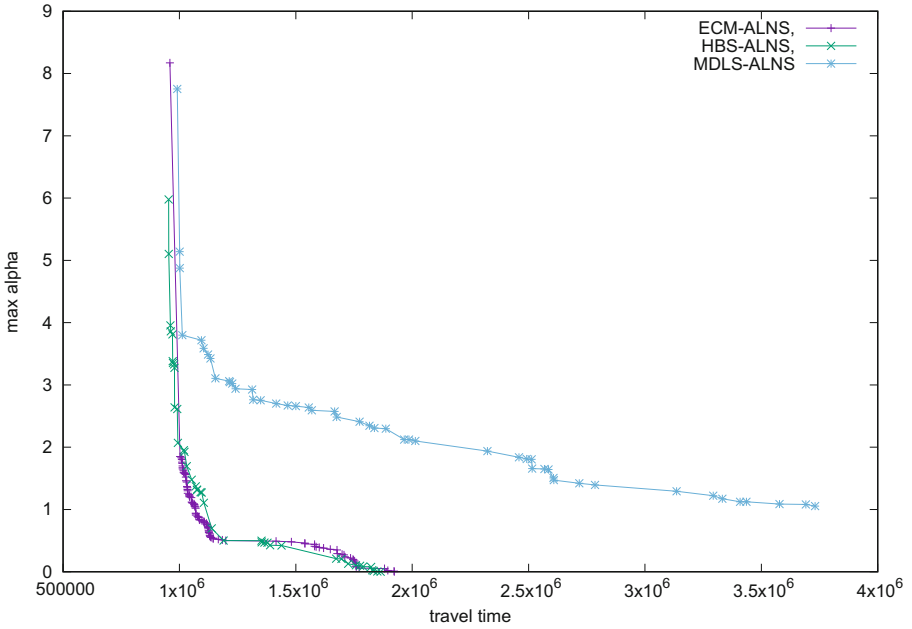
Table 3. $ONVG$ obtained by each algorithm solving large instances.

Instance	ECM-ALNS		HBS-ALNS		MDLS-ALNS	
	Avg	Best	Avg	Best	Avg	Best
22	**45.2**	**57.0**	42.0	48.0	28.4	42.0
23	**42.4**	**51.0**	32.6	41.0	23.2	28.0
24	**57.6**	**71.0**	47.2	59.0	32.2	45.0
25	**69.8**	**80.0**	52.6	73.0	39.6	47.0
26	**63.0**	**75.0**	53.2	61.0	35.4	40.0
27	**65.6**	**78.0**	49.0	56.0	42.2	49.0
28	**76.9**	**86.0**	45.6	54.0	36.4	40.0
29	**85.4**	**96.0**	47.8	64.0	38.6	44.0
30	**79.1**	**92.0**	39.8	56.0	37.6	43.0
31	**78.7**	**90.0**	39.0	54.0	39.6	45.0
32	**81.3**	**88.0**	42.8	49.0	42.2	50.0
33	**91.1**	**100.0**	47.6	60.0	36.2	39.0
34	**84.3**	**94.0**	38.8	46.0	31.8	41.0
35	**89.8**	**98.0**	56.4	70.0	30.4	42.0
36	**86.1**	**93.0**	40.0	51.0	35.8	39.0
37	**96.1**	**112.0**	41.0	55.0	32.6	34.0
38	**90.5**	**100.0**	38.6	54.0	26.4	34.0
39	**95.2**	**106.0**	45.8	54.0	40.6	50.0
Avg	76.6	87.1	44.4	55.8	35.0	41.8

The performance of MDLS is hindered by the poor performance of $ALNS_2$ when the instance size increases. Although it obtains reasonably good HV values for middle-sized instances such as 22–24 (100 deliveries), its results quickly degradate for larger instances. Figure 1 shows the median Pareto front obtained by each method based on neighborhood search for two instances. The median Pareto front is the front that corresponds to the median HV of 5 independent runs. Figures 1a and 1b show the results obtained for instances 19 and 39 (40 and



(a) Instance 19 with 3 depots, 40 deliveries, and 17 locations.



(b) Instance 39 with 6 depots, 300 deliveries, and 100 locations.

Fig. 1. Median Pareto front obtained by each solution method based on neighborhood search on (a) a small instance and (b) a large instance.

300 deliveries, respectively). On the smaller instance, MDLS-ALNS is fast and outperforms the other two solution methods, while it fails at finding good solutions in the larger instance. Figure 1b also shows that EPS-ALNS finds many good solutions, but it does not explore the front for high values of α, while HBS-ALNS finds reasonably good spread solutions along the complete Pareto front.

We quantify this latter effect by calculating the average SP for each algorithm and instance (see Fig. 2, only ECM-ALNS and HBS-ALNS are shown for better readability). The SP indicator measures how uniformly the solutions of a Pareto front approximation are spread (lower values are considered to be better). Our results show that ECM-ALNS finds good spread Pareto fronts for smaller instances (22–26), while the fronts become more sparse for larger instances (27–39). The performance of HBS-ALNS on this indicator, however, does not depend so strongly on the instance size, i.e. similar SP values are obtained for most instances, and they are on average better than those of ECM-ALNS, specially for larger instances.

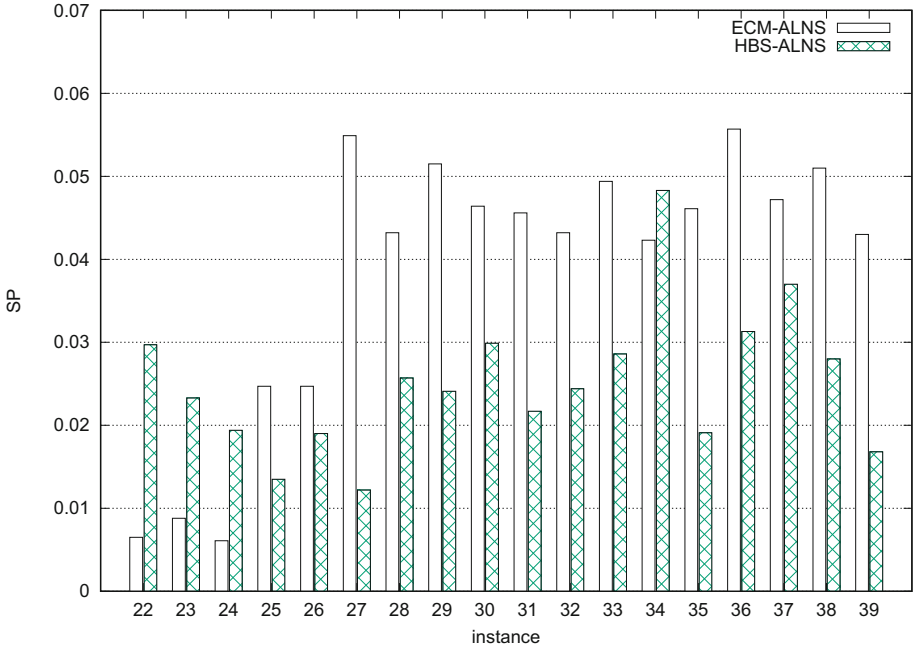


Fig. 2. Average SP obtained by ECM-ALNS and HBS-ALNS.

5 Conclusions

In this paper, we defined and solved a multi-objective vehicle routing problem with synchronization constraints at the delivery location. We proposed

three methods based on single-objective neighborhood search that are embedded within more general optimization frameworks for multi-objective optimization. In particular, we used ECM, HBS, and MDLS together with two ALNS algorithms. Experiments showed that HBS-ALNS provide the best Pareto fronts regarding the HV, also outperforming an exact solver on small instances. Future research should consider other quality indicators for multi-objective optimization problems, additional analysis on how fast the methods are at finding new solutions, tune the ALNS for the second objective in MDLS, and implement some restart criteria to better explore the Pareto front in the ECM.

References

1. Anderluh, A., Nolz, P.C., Hemmelmayr, V.C., Crainic, T.G.: Multi-objective optimization of a two-echelon vehicle routing problem with vehicle synchronization and 'grey zone' customers arising in urban logistics. Eur. J. Oper. Res. **289**(3), 940–958 (2021). https://doi.org/10.1016/j.ejor.2019.07.049
2. Audet, C., Bigeon, J., Cartier, D., Digabel, S.L., Salomon, L.: Performance indicators in multiobjective optimization. Eur. J. Oper. Res. **292**(2), 397–422 (2021). https://doi.org/10.1016/j.ejor.2020.11.016
3. Eskandarpour, M., Ouelhadj, D., Hatami, S., Juan, A.A., Khosravi, B.: Enhanced multi-directional local search for the bi-objective heterogeneous vehicle routing problem with multiple driving ranges. Eur. J. Oper. Res. **277**(2), 479–491 (2019). https://doi.org/10.1016/j.ejor.2019.02.048
4. Haimes, Y.Y., Lasdon, L.S., Wismer, D.A.: On a bicriterion formation of the problems of integrated system identification and system optimization. IEEE Trans. Syst. Man Cybern., 296–297 (1971). https://doi.org/10.1109/TSMC.1971.4308298
5. Jozefowiez, N., Semet, F., Talbi, E.: Multi-objective vehicle routing problems. Eur. J. Oper. Res. **189**(2), 293–309 (2008). https://doi.org/10.1016/j.ejor.2007.05.055
6. Lacomme, P., Prins, C., Sevaux, M.: A genetic algorithm for a bi-objective capacitated arc routing problem. Comput. Oper. Res. **33**(12), 3473–3493 (2006). https://doi.org/10.1016/j.cor.2005.02.017. Part Special Issue: Recent Algorithmic Advances for Arc Routing Problems
7. Laumanns, M., Thiele, L., Zitzler, E.: An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. Eur. J. Oper. Res. **169**(3), 932–942 (2006). https://doi.org/10.1016/j.ejor.2004.08.029
8. Lian, K., Milburn, A.B., Rardin, R.L.: An improved multi-directional local search algorithm for the multi-objective consistent vehicle routing problem. IIE Trans. **48**(10), 975–992 (2016). https://doi.org/10.1080/0740817X.2016.1167288
9. Liu, Q., Li, X., Liu, H., Guo, Z.: Multi-objective metaheuristics for discrete optimization problems: a review of the state-of-the-art. Appl. Soft Comput. **93**, 106382 (2020). https://doi.org/10.1016/j.asoc.2020.106382
10. Maltese, J., Ombuki-Berman, B.M., Engelbrecht, A.P.: A scalability study of many-objective optimization algorithms. IEEE Trans. Evol. Comput **22**(1), 79–96 (2018). https://doi.org/10.1109/TEVC.2016.2639360
11. Martí, R., Campos, V., Resende, M.G., Duarte, A.: Multiobjective GRASP with path relinking. Eur. J. Oper. Res. **240**(1), 54–71 (2015). https://doi.org/10.1016/j.ejor.2014.06.042

12. Matl, P., Hartl, R.F., Vidal, T.: Leveraging single-objective heuristics to solve bi-objective problems: Heuristic box splitting and its application to vehicle routing. Networks **73**(4), 382–400 (2019). https://doi.org/10.1002/net.21876
13. Ombuki, B.M., Ross, B., Hanshar, F.: Multi-objective genetic algorithms for vehicle routing problem with time windows. Appl. Intell. **24**(1), 17–30 (2006). https://doi.org/10.1007/s10489-006-6926-z
14. Sarasola, B., Doerner, K.F.: Adaptive large neighborhood search for the vehicle routing problem with synchronization constraints at the delivery location. Networks **75**(1), 64–85 (2020). https://doi.org/10.1002/net.21905
15. Schott, J.R.: Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master's thesis, Department of Aeronautics and Astronautics, MIT (1995)
16. Shao, S., Xu, G., Li, M., Huang, G.Q.: Synchronizing e-commerce city logistics with sliding time windows. Transp. Res. Part E Logist. Transp. Rev. **123**, 17–28 (2019). https://doi.org/10.1016/j.tre.2019.01.007
17. Tan, K.C., Cheong, C.Y., Goh, C.K.: Solving multiobjective vehicle routing problem with stochastic demand via evolutionary computation. Eur. J. Oper. Res. **177**(2), 813–839 (2007). https://doi.org/10.1016/j.ejor.2005.12.029
18. Tricoire, F.: Multi-directional local search. Comput. Oper. Res. **39**(12), 3089–3101 (2012). https://doi.org/10.1016/j.cor.2012.03.010
19. Veldhuizen, D.A.V.: Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations. Ph.D. thesis, Graduate School of Engineering, Air Force Institute of Technology (1999)
20. Xu, S.X., Shao, S., Qu, T., Chen, J., Huang, G.Q.: Auction-based city logistics synchronization. IISE Trans. **50**(9), 837–851 (2018). https://doi.org/10.1080/24725854.2018.1450541
21. Zitzler, E.: Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. Ph.D. thesis, ETH Zurich, Switzerland (1999)