Computers and Operations Research 129 (2021) 105196

Contents lists available at ScienceDirect

Computers and Operations Research

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

A hybrid algorithm for the multi-depot heterogeneous dial-a-ride problem

Igor Malheiros [a,*], Rodrigo Ramalho [b], Bruno Passeti [b], Teobaldo Bulhões [c], Anand Subramanian [d]

[a] Universidade Federal da Paraíba, Programa de Pós-Graduação em Informática, Centro de Informática, Rua dos Escoteiros s/n, Mangabeira 58055-000, João Pessoa, Brazil
[b] Universidade Federal da Paraíba, Centro de Informática, Rua dos Escoteiros s/n, Mangabeira 58055-000, João Pessoa, Brazil
[c] Universidade Federal da Paraíba, Departamento de Computação Científica, Centro de Informática, Rua dos Escoteiros s/n, Mangabeira 58055-000, João Pessoa, Brazil
[d] Universidade Federal da Paraíba, Departamento de Sistemas de Computação, Centro de Informática, Rua dos Escoteiros s/n, Mangabeira 58055-000, João Pessoa, Brazil

ARTICLE INFO

Article history:
Received 23 September 2020
Accepted 22 December 2020
Available online 30 December 2020

Keywords:
Routing
Dial-a-ride
Matheuristics
Iterated local search

ABSTRACT

Vehicle routing problems arise in many practical situations in the context of transportation logistics. Among them, we can highlight the problem of transporting customers from origin to destination locations, which is known as the dial-a-ride problem (DARP). This problem consists of designing least-cost routes to serve pickup-and-delivery requests, while meeting capacity, time window, maximum route duration, and maximum ride time constraints. This work proposes a hybrid algorithm to solve single and multi-depot DARP variants where both the demands and vehicle fleet are heterogeneous. The method combines the iterated local search metaheuristic with an exact procedure based on a set partitioning approach. In addition, several procedures were implemented to speedup the local search phase. Extensive computational experiments were conducted on existing and newly proposed benchmark instances in order to evaluate the impact of the different components of the algorithm, and to compare its performance against the best existing method. The results obtained suggest that the proposed algorithm is capable of producing highly competitive results regarding both solution quality and CPU time, and of improving several best known solutions.

1. Introduction

Many practical problems arising in transportation logistics, such as distribution of groceries, waste collection, pickup-and-delivery of goods or people, among others, can be modeled by a class of combinatorial optimization problems known as the *vehicle routing problem* (VRP). A typical VRP aims at determining least-cost routes that satisfy the demand of a set customers geographically distributed.

In this work, we are particularly interested in solving a VRP variant called the *multi-depot heterogeneous dial-a-ride problem* (MDHDARP) (Braekers et al., 2014), which is an extension of the classical *dial-a-ride* (DARP) problem (Cordeau and Laporte, 2003b), and the *heterogeneous* DARP (HDARP) (Parragh, 2011). The latter is also considered in this study. According to the classification scheme by Berbeglia et al. (2007), DARPs belong to the category of *one-to-one* pickup-and-delivery problems, in which customers must be collected from one location and delivered to another one.

DARP can be seen as a generalization of the classical *pickup-and-delivery problem with time windows* (PDPTW) (Dumas et al., 1991). Both problems share the same constraints, i.e., routes must respect capacity and time window constraints, but the former includes the so-called maximum ride time constraints, which makes the problem more challenging when compared to the latter. In the MDHDARP, vehicles are placed in multiple depots and both the user demands and vehicle fleet are heterogeneous. The problem is $\mathcal{NP}$-hard as it generalizes DARP and many other standard VRPs that are also $\mathcal{NP}$-hard.

Several real-life applications are formulated as a DARP variant (Ho et al., 2018). One of the most common examples concerns the transportation of elderly or disabled people from their residences to particular destinations (Madsen et al., 1995). Another example emerges in health-care systems, where patients must travel between medical facilities or from their residences to treatment centers. In those situations, patients may have a heterogeneous demand, that is, they might need a regular seat, stretcher, or even an wheelchair (Parragh, 2011), and in some cases the vehicles are located in multiple depots (Carnes et al., 2013; Detti et al., 2017).

* Corresponding author.
E-mail addresses: igormalheiros92@gmail.com (I. Malheiros), brunopasseti@eng.ci.ufpb.br (B. Passeti), tbulhoes@ci.ufpb.br (T. Bulhões), anand@ci.ufpb.br (A. Subramanian).

https://doi.org/10.1016/j.cor.2020.105196
0305-0548/

Other real-life DARP applications can be found in the works by Chevrier et al. (2012), Reinhardt et al. (2013), Muelas et al. (2013) and Markovićc et al. (2015).

The main contributions of this paper are as follows.

- We develop a hybrid algorithm that combines *iterated local search* (ILS) (Lourenço et al., 2019) and a *set partitioning* (SP) approach in a cooperative fashion. To our knowledge, this is the first time that a *matheuristic* based on ILS is employed to efficiently solve a DARP variant. The results obtained equaled or improved the best heuristic results for the problem. In addition, we also report very competitive results for HDARP.
- We present two inter-route neighborhood structures and a perturbation mechanism that are based on a subsequence of visits. Their effectiveness is empirically demonstrated by means of computational experiments.
- We introduce speedup mechanisms that not only avoid unnecessary move evaluations but also prevent the search to be repeated over a solution previously visited. Such mechanisms helped the proposed algorithm to run about 3 times faster, on average, than the best heuristic for the problem, when considering the existing benchmark dataset.
- We provide an extensive analysis on the impact of the main components of the algorithm, thus allowing one to properly assess their importance in the performance of the method.
- We propose a new set of instances for the MDHDARP involving up to 192 requests. The current dataset is limited to instances with up to 96 requests, and the existing bounds are already tight, thus motivating us to generate a novel benchmark containing test-problems of larger size.

The remainder of the paper is organized as follows. Section 2 briefly reviews the related literature. Section 3 formally defines the problem. Section 4 describes the proposed hybrid algorithm. Section 5 presents the results of the computational experiments. Section 6 contains the concluding remarks.

2. Related work

The first works to address dial-a-ride systems appeared in the 1970s (Wilson et al., 1971; Wilson and Weissberg, 1976), where the authors studied a dynamic version of a real-life problem. Early DARP variants were later approached in the 1980s and 1990s via *dynamic programming* (Psaraftis, 1980; Psaraftis, 1983; Desrosiers et al., 1986), *column generation* (CG) (Desrosiers et al., 1988), and heuristics (Jaw et al., 1986; Bodin and Sexton, 1986; Dumas et al., 1989; Ioachim et al., 1995; Toth and Vigo, 1996; Toth and Vigo, 1997).

The standard version of DARP was introduced by Cordeau and Laporte (2003b). The authors developed a *tabu search* (TS) heuristic, and a procedure to verify the maximum ride time constraints that remains widely used to this date. Exact algorithms based on *branch-and-cut* (BC) were proposed by Cordeau (2006, 2007), whereas a *branch-cut-and-price* algorithm was put forward by Gschwind and Irnich (2015). Parragh et al. (2010) implemented a *variable neighborhood search* (VNS) procedure for the problem, and they also considered a DARP version with a modified objective function. Hybrid *large neighborhood search* (LNS) algorithms were developed by Jain and Van Hentenryck (2011), Parragh and Schmid (2013), Gschwind and Drexl (2019). The first work combined LNS with *constraint programming*, the second with CG, while the third implemented an adaptive LNS combined with a *set covering* approach. Gschwind and Drexl (2019) also devised a procedure to check the feasibility of the maximum ride constraints in amortized $\mathcal{O}(1)$ time. Moreover, a dynamic version of DARP was tackled by Attanasio et al. (2004) via a parallel TS heuristic.

Motivated by the Austrian Red Cross, HDARP was introduced by Parragh (2011). The author proposed BC and VNS algorithms, the latter extends the one developed in Parragh et al. (2010). Masmoudi et al. (2017) later put forward a *genetic algorithm* combined with local search for both HDARP and DARP. MDHDARP was formally introduced by Braekers et al. (2014). The authors implemented BC and *deterministic annealing* (DA) algorithms that were capable of producing tight lower and upper bounds for the problem. They also addressed both HDARP and DARP. Masmoudi et al. (2016) devised three heuristic procedures for MDHDARP, in particular, *bees algorithm with deterministic annealing* (BA-DA), *bees algorithm with simulated annealing* (BA-SA) and ALNS. The *multi-depot dial-a-ride problem* (MDDARP) was one of the problems approached via a LNS algorithm by Molenbruch et al. (2017). Other DARP variants were recently suggested in Masmoudi et al. (2018), Masmoudi et al. (2020).

The reader is referred to Cordeau and Laporte (2003a), Molenbruch et al. (2017), Ho et al. (2018) for a comprehensive review on DARP variants.

3. Problem definition

Let n be the number of requests and v the total number of vehicles. Set $P = \{1, \ldots, n\}$ and $D = \{n+1, \ldots, 2n\}$ represent the pickup and delivery vertices, respectively. In addition, let $M = \{2n+1, \ldots, 2n+v\}$ be the set of depot vertices associated to each vehicle denoting the start and end of a route. However, more than one vehicle can have the same depot, meaning that M can contain vertices indicating the same depot. For each request, there is a pickup vertex $i \in P$ and an associated delivery vertex $i+n \in D$. Therefore, one can define set $V = P \cup D \cup M$, which contains the pickup and delivery vertices, as well as the depot vertices. Set A represents the possible arcs between the vertices of V, namely: arcs connecting a depot $m \in M$ to a pickup vertex $i \in P$; arcs connecting a pickup or delivery vertex to another pickup or delivery vertex $j \in P \cup D$ such that $i \neq j$ and $j+n \neq i$; arcs connecting a delivery vertex $j \in D$ to a depot $m \in M$; and arcs connecting a depot to itself to allow empty routes. Sets V and A define the directed graph $G = (V, A)$.

Moreover, let $K = \{1, \ldots, v\}$ be the set of vehicles and $R = \{0, \ldots, 3\}$ be the set of resources that can be used by a request. Resource 0 refers to staff seats, 1 to patient seats, 2 to stretchers, and 3 to wheelchair places. An accompanying person can consume resources 0, 1 or 2; the patient can consume resource 1 or 2; and travel requests using stretcher or wheelchair can only consume its respective resource.

For each arc $(i,j) \in A$, there is an associated time t_{ij} and cost c_{ij} that are assumed to satisfy the triangular inequality. For each vertex $i \in V$, there exists a service time d_i, a time window $[e_i, l_i]$ and a demand $q_i^r, r \in R$. For the pickup and delivery vertices we have that $q_{i+n}^r = -q_i^r, i \in P, r \in R$, whereas for the depot vertices we have that $q_{2n+k}^r = 0, k \in K, r \in R$, and that $d_{2n+k} = 0, k \in K$. Furthermore, a maximum ride time L_i is imposed for each $i \in P \cup M$. For each vehicle $k \in K$ and resource $r \in R$, there exists a capacity C^{rk}. The objective of MDHDARP is to minimize the sum of the travel costs while satisfying the following constraints: (i) each request i must be met by exactly one vehicle that must visit pickup vertex i before vising the corresponding delivery vertex $i+n$; (ii) each route must start and end at the same depot; (iii) the capacity of the vehicle $C^{rk}, r \in R, k \in K$, cannot be exceeded; (iv) each request i must have its pickup and delivery requests met within their respective time window $[e_i, l_i]$ and $[e_{i+n}, l_{i+n}]$; (v) the difference between the arrival time at a delivery vertex $i+n$ and the time to start serving the

corresponding pickup vertex i must be less than or equal to the maximum ride time L_i. This same constraint is used to limit the route duration, i.e, the difference between the arrival time at the depot and the starting time of the route must not exceed a maximum route duration $L_m, m \in M$.

4. Proposed hybrid algorithm

The proposed multi-start hybrid approach (ILS-SP) is presented in Algorithm 1. It first preprocesses the instance (line 1) by tightening the time windows and forbidding arcs that violate time windows or maximum ride time (Cordeau, 2006). In addition, one modifies the representation of the demands and capacities so that each resource can be processed and evaluated independently throughout the execution of the method (Parragh, 2011). ILS-SP performs I_R restarts (lines 4–13), where at each restart an initial solution is built (line 6) and then submitted to an ILS procedure (lines 7). Routes $\mathscr{R}_s$ from a local optimal solution s are stored in two structures, *pool* and $pool_{temp}$, which serve as input for the SP procedure. These structures were implemented using binary search trees because they allow one to efficiently identify and remove duplicates in $\mathcal{O}(\log \omega)$ operations, where ω is the number of solutions stored in the pool. The latter is reset at the beginning of each restart (line 5), while at the end the SP procedure is called (line 10) using such structure merged with a third one that stores the routes from the best solutions (line 11). After all restarts, and if the number of requests is smaller than 100, the SP procedure is called again (lines 15 and 16) but this time using the first structure (*pool*), i.e., the one that stores the routes from all local optima obtained during the execution of the algorithm, and the best solution is returned.

Algorithm 1: ILS-SP

1. Procedure ILS-SP(I_R, I_{ILS}, I_ρ)
2. `preProcessing()`
3. $pool \leftarrow pool_{best} \leftarrow s^* \leftarrow \varnothing; f^* \leftarrow \infty$
4. **for** $i \leftarrow 1$ to I_R **do**
5. $\quad pool_{temp} \leftarrow \varnothing$
6. $\quad s \leftarrow$ `generateInitialSolution()`
7. $\quad s \leftarrow$ ILS($I_{ILS}, I_\rho, pool, pool_{temp}, s$)
8. $\quad pool_{temp} \leftarrow pool_{temp} \cup pool_{best}$
9. $\quad modelSP \leftarrow$ `createModelSP(`$pool_{temp}$`)`
10. $\quad s \leftarrow$SP($modelSP, s, I_{ILS}, I_\rho, pool, pool_{temp}$)
11. $\quad pool_{best} \leftarrow pool_{best} \cup \mathscr{R}_s$
12. $\quad$ **if** $f(s) \leqslant f^*$ **do**
13. $\quad\quad s^* \leftarrow s; f^* \leftarrow f(s)$
14. **if** $n < 100$ **then**
15. $\quad modelSP \leftarrow$ `createModelSP(`$pool$`)`
16. $\quad s \leftarrow$ `SP(`$modelSP, s^*, I_{ILS}, I_\rho, pool, pool_{temp}$`)`
17. **if** $f(s^*) < f^*$ **then**
18. $\quad f^* \leftarrow f(s^*)$
19. **return** s^*

Starting from an initial solution, the ILS procedure described in Algorithm 2 iteratively applies local search (line 4) and perturbation mechanisms (line 10) with a view of finding promising local optimal solutions. After each local search, one stores the set of routes $\mathscr{R}_s$ from the solution found in the aforementioned structures (*pool* and $pool_{temp}$) (lines 5 and 6). The parameter I_{ILS} indicates the maximum number of consecutive ILS iterations without improvement on the best current solution.

Algorithm 2: ILS

1. Procedure ILS($I_{ILS}, I_\rho, pool, pool_{temp}, s$)
2. $s^* \leftarrow s; f^* \leftarrow f(s); iter \leftarrow 0$
3. **while** $iter \leqslant I_{ILS}$ **do**
4. $\quad s \leftarrow$ localSearch($s, iter, I_{ILS}$)
5. $\quad pool \leftarrow pool \cup \mathscr{R}_s$
6. $\quad pool_{temp} \leftarrow pool_{temp} \cup \mathscr{R}_s$
7. $\quad$ **if** $f(s) < f^*$ **then**
8. $\quad\quad s^* \leftarrow s; f^* \leftarrow f(s)$
9. $\quad\quad iter \leftarrow 0$
10. $\quad s \leftarrow$ `perturb(`s^*, I_ρ`)`
11. $\quad iter \leftarrow iter + 1$
12. **return** s^*

In what follows, we describe the main components of the proposed hybrid algorithm.

4.1. Set partitioning procedure

The vehicles with same attributes are grouped into a set of vehicle types $\mathscr{K}$, and U_k^m indicates the amount of each type of vehicle $k \in \mathscr{K}$ at each depot $m \in M$. Let $\mathscr{R}$ be the set of routes stored in a given pool ($pool, pool_{temp}$ or $pool_{best}$), and $\mathscr{R}_i \subseteq \mathscr{R}$ be the subset of routes containing request $i \in P$. We also define $\mathscr{R}_k^m \subseteq \mathscr{R}$ as the subset of routes in which a vehicle of type $k \in \mathscr{K}$ is at depot $m \in M$. The cost of each route $j \in \mathscr{R}$ is denoted by c_j. The binary variable y_j assumes value 1 in case route $j \in \mathscr{R}$ is selected, 0 otherwise. The SP model can be written as follows.

$$\min \sum_{j \in \mathscr{R}} c_j y_j \tag{1}$$

Subject to:

$$\sum_{j \in \mathscr{R}_i} y_j = 1 \quad i \in P \tag{2}$$

$$\sum_{j \in \mathscr{R}_k^m} y_j \leqslant U_k^m \quad k \in \mathscr{K}, \quad m \in M \tag{3}$$

$$y_j \in \{0, 1\} \quad j \in \mathscr{R}. \tag{4}$$

The objective function (1) minimizes the sum of the costs. Constraints (2) ensure that each request $i \in P$ is assigned to exactly one route $j \in \mathscr{R}_i$. Constraints (3) guarantee that the amount of vehicles of a certain type does not exceed the fleet limit. Constraints (4) define the domain of the variables.

The SP procedure was implemented based on the ideas presented in Subramanian et al. (2013), where the ILS algorithm is called every time an incumbent solution is found by the MIP solver. Note that in case ILS improves the incumbent solution, one does not provide it to the solver because it might possibly contain routes that are not part of the model. Instead, one provides the associated upper bound (UB) to be used as reference value when performing pruning decisions.

4.2. Constructive procedure

The initial solutions are generated by a parallel insertion-based procedure. Firstly, a candidate list (CL) is initialized with all requests sorted in non-decreasing order according to the earliest time of the pickup vertex, and an empty solution is initialized with all vehicles from the instance. Next, each empty route receives a seed request selected at random from CL. The remaining unse-

lected requests are iteratively included in the partial solution using the best feasible insertion strategy. The procedure ends when all requests have been inserted. In case it is not possible to perform any feasible insertion, the constructive algorithm restarts.

4.3. Local search

The local search employs three types of neighborhoods: inter-route ($\mathcal{N}_{inter}$), intra-route ($\mathcal{N}_{intra}$), and block-based inter-route ($\mathcal{N}_{blocks}$). The latter type was somewhat inspired by the *zero-split neighborhood* presented in Parragh et al. (2010), and it consists of moves involving a contiguous subsequence of vertices in which the vehicle load is zero both before immediately visiting the first vertex of the subsequence and immediately after visiting the last vertex of the subsequence. We denote this type of subsequence as *zero-sum block*. The difference with respect to the neighborhood implemented in Parragh et al. (2010) is that we do not modify the structure of the block during the move, while in their case they split the block.

All neighborhoods are explored exhaustively using the best improvement strategy, and only feasible moves are accepted.

Several search strategies involving the different types of neighborhoods mentioned above were tested, and the one presented in Algorithm 3 appeared to offer an interesting compromise between solution quality and CPU time. The local search algorithm relies partially on the *randomized variable neighborhood descent* (RVND) procedure (Subramanian et al., 2010), which is based on the traditional *variable neighborhood descent* (VND) by Hansen and Mladenović (2001).

The algorithms starts by updating the necessary *auxiliary data structures* (ADSs) (line 2) used during the search (see Section 4.3.2). Next, the RVND procedure is performed in a nested fashion (lines 3–11), but only considering two types of neighborhoods, namely: $\mathcal{N}_{block}$ and $\mathcal{N}_{inter}$. The neighborhoods of type $\mathcal{N}_{intra}$ are only explored in the last iteration of the ILS algorithm. The motivation for adopting the scheme described in Algorithm 3 was to decrease the number of times the search is performed on the $\mathcal{N}_{block}$ and $\mathcal{N}_{intra}$ neighborhoods. While the former type typically demand more operations, the latter sometimes lead to local optimal solutions that are difficult to escape from.

Algorithm 3: Local Search

1. Procedure localSearch($s, iter, I_{ILS}$)
2. Update ADSs
3. Initialize $\mathcal{N}_{block}$
4. **while** $\mathcal{N}_{block} \neq \varnothing$ **do**
5. Choose a neighborhood $\mathcal{N}^{(\eta)}_{block} \in \mathcal{N}_{block}$ at random
6. Remove $\mathcal{N}^{(\eta)}_{block}$ from $\mathcal{N}_{block}$
7. Find the best neighbor $s' \in \mathcal{N}^{(i)}_{block}(s)$
8. **if** $f(s') < f(s)$ **then**
9. $s \leftarrow s'$
10. Repopulate $\mathcal{N}_{block}$
11. RVND($s, \mathcal{N}_{inter}$, ADS)
12. **if** $iter = I_{ILS}$ **then**
13. $f \leftarrow f(s)$
14. RVND($s, \mathcal{N}_{intra}$, ADS)
15. **if** $f(s) < f$
16. **goto** line 3
17. **return** s

4.3.1. Neighborhood structures

The following neighborhood structures were implemented.

- ***Relocate-inter*** – $\mathcal{N}^{(1)}_{inter}$: The pickup and delivery vertices associated with a same request is moved from a route π_1 to a route π_2.
- ***Exchange-inter*** – $\mathcal{N}^{(2)}_{inter}$: Two requests, one from route π_1 and another from route π_2, are exchanged, i.e., the pickup vertices and delivery vertices are respectively swapped.
- ***2-opt**** – $\mathcal{N}^{(3)}_{inter}$: Two unloaded arcs are removed, one from route π_1 and another from route π_2. Next, two new routes are built by inserting another two arcs, one that combines the first part of route π_1 with the second part of route π_2, and vice versa.
- ***Exchange-vehicle*** – $\mathcal{N}^{(4)}_{inter}$: The vehicles associated with a pair of routes π_1 and π_2 are swapped.
- ***Relocate-intra*** – $\mathcal{N}^{(1)}_{intra}$: The positions of the pickup and delivery vertices of a same request are redefined in the same route.
- ***Exchange-intra*** – $\mathcal{N}^{(2)}_{intra}$: The position of two vertices of a same route are exchanged.
- ***Relocate-block*** – $\mathcal{N}^{(1)}_{block}$: A zero-sum block is moved from a route π_1 to route π_2.
- ***Exchange-block*** – $\mathcal{N}^{(2)}_{block}$: Two zero-sum blocks, one from route π_1 and another from route π_2 are exchanged. The pair of blocks involved in the move can be of different sizes.

We believe that this is the first time the latter two neighborhoods were employed for the MDHDARP. Furthermore, the size of the neighborhoods is presented in Table 1.

4.3.2. Move evaluation and feasibility checking

The move evaluation can be performed in $\mathcal{O}(1)$ operations by simply computing the difference associated with the cost of the removed and inserted arcs. Moreover, it is straightforward to observe that the pairing and precedence constraints can be directly checked in $\mathcal{O}(1)$ operations when performing any of the moves described in the previous section. To verify the capacity and time window constraints, the algorithm uses ADSs based on *subsequences*, which in turn are built and updated in $\mathcal{O}(n^2)$ time. They enable such constraints to be checked in $\mathcal{O}(1)$ operations. Time window constraints are checked as in Vidal et al. (2013), whereas capacity constraints are verified by means of an adaptation of the scheme presented in Bulhões et al. (2018), as described next.

Let $\sigma = (\sigma^0, \ldots, \sigma^{|\sigma|-1})$ be a subsequence, and σ^{ij} be a subsequence starting at the i-th position and ending at the j-th position, i.e., $\sigma^{ij} = (\sigma^i, \ldots, \sigma^j)$. For each possible subsequence σ of a route, the ADSs store and update the following values: $q^r_{sum}(\sigma)$ — cumulative sum of the pickup/delivery loads for each resource $r \in R$; and $q^r_{max}(\sigma)$ — maximum cumulative sum for each resource $r \in R$.

If a subsequence $\sigma\prime$ is only composed of a vertex i, then $q^r_{sum}(\sigma\prime) = q^r_i$ and $q^r_{max}(\sigma\prime) = max(0, q^r_i)$. Any subsequence σ, such that $|\sigma| > 1$, can be derived from two other subsequences, σ^1 and σ^2, by performing a concatenation operation, given by $\oplus$, as follows:

$$q^r_{sum}(\sigma^1 \oplus \sigma^2) = q^r_{sum}(\sigma^1) + q^r_{sum}(\sigma^2), \quad \forall r \in R \tag{5}$$

$$q^r_{max}(\sigma^1 \oplus \sigma^2) = max\{q^r_{max}(\sigma^1), q^r_{sum}(\sigma^1) + q^r_{max}(\sigma^2)\}, \quad \forall r \in R. \tag{6}$$

Table 1
Size of the neighborhoods.

$\mathcal{N}_{inter}$	Size	$\mathcal{N}_{intra}$	Size	$\mathcal{N}_{block}$	Size
Relocate-inter	$\mathcal{O}(n^3)$	Relocate-intra	$\mathcal{O}(n^3)$	Relocate-block	$\mathcal{O}(n^3)$
Exchange-inter	$\mathcal{O}(n^2)$	Exchange-intra	$\mathcal{O}(n^2)$	Exchange-block	$\mathcal{O}(n^4)$
2-opt*	$\mathcal{O}(n^2)$				
Exchange-vehicle	$\mathcal{O}(v^2)$				

A route denoted by subsequence $\sigma = (\sigma^0, \ldots, \sigma^{|\sigma|-1})$ is feasible with respect to the capacity constraints if $q^r_{max}(\sigma) \leqslant C^{rk}, \forall r \in R$, where k is the vehicle associated with the route. Remark that the vehicle load cannot be negative because we only deal with routes that satisfy the pairing and precedence constraints.

The maximum ride time constraints are checked by means of the eight-step procedure by Cordeau and Laporte (2003b), which is based on the concept of *forward time slack*. This procedure, whose complexity is $\mathcal{O}(n^2)$, is only called when the other constraints are satisfied and the move yields an improving cost. It is relevant to mention that we implemented the scheme proposed in Gschwind and Drexl (2019) to evaluate maximum ride time constraints of the neighborhood Relocate-inter in amortized $\mathcal{O}(1)$ operations, but it did not pay off when testing on the existing HDARP and MDHDARP benchmark instances, as well as on the newly proposed dataset. The preprocessing phase, whose complexity is $\mathcal{O}(n^3)$, ended up being time consuming to the point that one could not see the advantage of the constant time feasibility checking.

4.4. Perturbation mechanisms

Two perturbation mechanisms were implemented, and they are randomly chosen, with the same probability, at each call of the function `perturb` in Algorithm 2. In the first, denoted as ρ_1, a request is randomly selected from the solution and moved to an extra route. In the second, denoted as ρ_2, a request is also selected at random, but this time the algorithm moves the smallest zero-sum block containing that request to the extra route. A penalty is incurred on the total cost of the extra route, which allows time window and capacity constraints to be violated. The amount of consecutive perturbation moves performed is randomly determined by choosing a value from the interval $[1, I_\rho]$. Note that extra route tends to be naturally emptied during the local search.

4.5. Speedup mechanisms

With a view of improving the runtime performance of the algorithm, we implemented three speedup mechanisms. They all rely on structures that are part of the ADSs. Two of them store information related to solutions, individual routes or pair of routes that have been already explored. The third one stores the feasible search range with respect to time window constraints that a vertex can be inserted. Their detailed description is provided below.

- *Memoization Move Descriptor* (MMD) — This mechanism is an extension of the *Static Move Descriptor* proposed by Zachariadis and Kiranoudis (2010), and later improved by Beek et al. (2018). Given a route (in case of an intra-route search) or a pair of routes (in case of an inter-route search), once the best improving move of a particular neighborhood is determined, the structure stores the information regarding such move. If there is no improving move, the structure also stores that information. Therefore, before starting the search, the algorithm verifies whether the given route or pair or routes have been already explored. If so, MMD returns the information of the best improving move or that there is no such move. Otherwise, the search is performed normally, and at the end one stores the corresponding information regarding the route or pair of routes.
- *Memoization Solution Descriptor* (MSD) — The idea presented in the previous mechanism can be extended to store the best local optimal solutions found by ILS-SP after each restart. The structure also saves the corresponding restart in which the solution was achieved. In case such solution is found again, the search is interrupted and the algorithm restarts. This is to prevent ILS to unnecessarily perform the search again on local optima that have been proven difficult to escape from. Moreover, MSD also stores all local optimal solutions obtained after performing local search. Hence, if one of those solutions is found at any point of the search, the local search procedure is interrupted.
- *Feasible Search Range* (FSR) — The last mechanism takes advantage of the characteristics of the time windows. When two vertices have disjoint time windows, it can be easily verified that one of them must be visited before the other, if they are on the same route. This information can be stored during the preprocessing phase, which is useful for identifying the first and last positions of a route that a vertex can be inserted without violating the time windows constraints. Throughout the execution of the algorithm, FSR stores such positions for each vertex and each route that has been modified during the search. Fig. 1 shows an example illustrating the feasible search range of vertex 5 with respect to a route (1, 2, 3, 4). The depot vertex was omitted from the figure for simplicity.

MMD and MSD were implemented using binary search trees. For an improved performance, each element from MMD and MSD (routes, pair of routes or solution) must be represented by a key, which in turn must ensure that each element from its corresponding structure is unique. The binary search trees enable an element to be searched, checked and inserted in $\mathcal{O}(\log \tau)$ operations, where τ is the amount of elements (nodes) saved in the tree.

The structure that identifies the precedence between two vertices with disjoint time windows is filled in $\mathcal{O}(n^2)$ operations during the preprocessing phase. For each route modification during the search, one needs do update the FSR by determining the first and last possible insertion positions for all vertices of the instance in such route, and this is done in $\mathcal{O}(n^2)$ operations. During a move evaluation, the interval stored by FSR associated with the insertion of a given vertex in a certain route is retrieved in $\mathcal{O}(1)$ operations.

5. Computational experiments

The proposed algorithm was coded in C++, and all experiments were executed on a single thread of an Intel Intel(R) Core(TM) i5-9600KF CPU 3.70 GHz with 8 GB of RAM running Linux Ubuntu 18.04. CPLEX 12.7 was used as a MIP solver with a time limit of 60 s. The procedure was executed 10 times for each instance in all tests.

ILS-SP was tested on the set of HDARP and MDHDARP benchmark instances suggested by Parragh (2011) and Braekers et al. (2014), respectively. For each dataset, there are three groups: (i) *U*, in which only a single resource is considered and the fleet is homogeneous; *E*, where at most four resources are used and the fleet is homogeneous; *I*, in which at most four resources are considered and the fleet is heterogeneous. There is a total of 144 instances ranging from 2 to 8 vehicles, and 16 to 96 requests. The vertices of the graph are distributed in a plane $[-10, 10]^2$. For HDARP, the depot is placed at the center of the square, whereas for MDHDARP they are distributed in 4 points with the following coordinates: $[-5, -5], [5, 5], [-5, 5]$ and $[5, -5]$, where the first and fifth vehicles belong to the first depot, the second and sixth belong to the second depot, and so on. Moreover, we used the same procedure to generate a new benchmark dataset for the MDHDARP, considering only the characteristics of group *I*, containing 24 instances ranging from 9 to 16 vehicles, and 72 to 192 requests. The new instances are available at github.com/igormalheiros/Instances-MDHDARP.

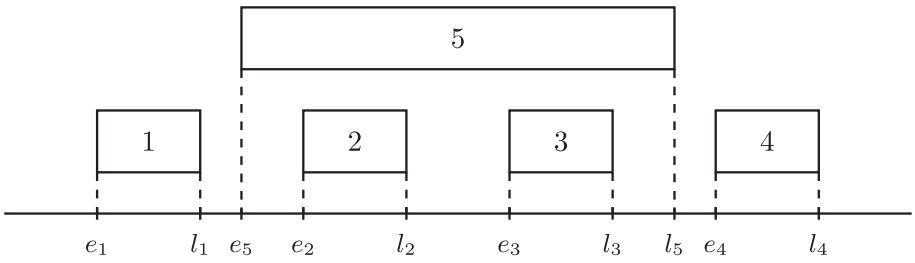

Fig. 1. Feasible search range for vertex 5 with respect to route (1, 2, 3, 4). Note that the time window of vertex 5 is disjoint from the one of vertices 1 and 4 because $e_5 > l_1$ and $l_5 < e_4$. However, the same is not true for the one of vertices 2 and 3, as there is an intersection between the time windows of vertices 2 and 5, as well as of those of 3 and 5, respectively. This way, vertex 5 must be inserted after vertex 1 and before vertex 4.

We selected the 72 most challenging instances, among the current benchmark, for performing parameter tuning experiments, as well as to evaluate the effect of different components of the algorithm. In some tests we used all instances, and we specify when it is the case. The results are reported in the following.

5.1. Impact of the block-based neighborhoods

We conducted experiments without the block-based neighborhoods, just with one of them, and with both. For each of the four combinations, the algorithm was executed without the SP procedure and with $I_R = 1$ and $I_{ILS} = 0$, meaning that only a single call to the local search procedure is performed, and no restarts or perturbation moves are done. Table 2 reports, for each combination, the average gap with respect to the lower bound (LB) obtained by the exact method presented in Braekers et al. (2014), and the average CPU time in seconds. These two measures are also used in the other experiments.

The results obtained suggest that the neighborhood block-exchange ($\mathcal{N}^{(2)}_{block}$) seems to have a more significant impact when compared to block-relocate ($\mathcal{N}^{(1)}_{block}$). This was somewhat expected because $\mathcal{N}^{(2)}_{block}$ has a larger search space. However, when both neighborhoods were put together, they yielded the best average gap without an increase on the CPU time, which indicates that $\mathcal{N}^{(1)}_{block}$ is also useful.

5.2. Impact of the perturbation mechanisms

To evaluate the impact of the perturbation mechanisms, we experimented with 18 different versions, and the average results are presented in Fig. 2. Note that we were interested not only in measuring the effect of each perturbation alone and both combined, but also to determine the value of parameter I_p, for which 6 different values were tried. The SP procedure was not considered during this testing, and we set $I_R = 1$ and $I_{ILS} = 2 \times n$.

The graph shows that the points (settings) 5, 7, 9, 10, 11, 12 and 18 are dominated by the other settings. Disregarding the dominated points, the smaller average gaps were found by points 6, 16 and 17, with an associated CPU time of around 0.8 seconds. Points 1, 13 and 2 achieved smaller values for the CPU time, but with an average gap of more than 1%. We decided to adopt the non-dominated setting 15, i.e., $\rho_1 + \rho_2$ and $I_\rho = 4$, because it is appears to offer an interesting compromise between solution quality and CPU time.

Table 2
Impact of the block-based neighborhoods.

Setting	Avg. Gap (%)	CPU (s)
Without $\mathcal{N}_{block}$	7,46%	0,026
$\mathcal{N}^{(1)}_{block}$	7,32%	0,031
$\mathcal{N}^{(2)}_{block}$	6,61%	0,033
$\mathcal{N}^{(1)}_{block}$ + $\mathcal{N}^{(2)}_{block}$	**6,48%**	0,033

5.3. Parameter tuning

Concerning the main parameters of the algorithm, that is, I_R and I_{ILS}, 9 different settings were tested and the results obtained are depicted in Fig. 3. The values considered for I_R were 15, 20 and 25, wheres the values considered for I_{ILS} were $max(2 \times n, 100)$, $max(2 \times n, 150)$ and $max(2 \times n, 200)$.

By observing the graph, we can verify that points 2, 4, 7 and 8 were dominated by the other settings. Such dominated points are associated with settings with $I_{ILS} = max(2 \times n, 200)$. Points 1, 3 and 5 appear to require less CPU time among the non-dominated settings, but wih average gaps larger than 0.22%. The non-dominated points with most competitive average gaps were 6 and 9, where the first seem to be faster. Therefore, we decided to adopt setting 6, that is, $I_R = 20$ and $I_{ILS} = max(2 \times n, 200)$, as it presented a good balance between solution quality and CPU time.

5.4. Local search runtime analysis

Fig. 4 shows the average percentage time of each local search component. In this experiment, we consider the parameter values previously defined and executed the algorithm on all instances.

The graph shows that Relocate-inter and Exchange-inter account, on average, for more than half of the runtime spent during local search. It is also interesting to observe that the time required to update the ADSs is higher than other components, more precisely, Relocate-block, Exchange-vehicle and the intra-route neighborhoods. Furthermore, 2-opt* and Exchange-block together consume, on average, around 27% of the total time.

The analysis show that although Exchange-block is the neighborhood with the largest search space ($\mathcal{O}(n^4)$), it is not the most time consuming in practice. This is related to the way the local search procedure was designed, which resulted in many more calls to the $\mathcal{N}_{inter}$ operators than the $\mathcal{N}_{block}$ ones.

5.5. Impact of the speedup mechanisms

To measure the impact of the speedup mechanisms, we conducted experiments on all instances. The idea is to observe the effect of each mechanism, and all their possible combinations, as the size of the instance increases. In this case, the algorithm was executed using the parameter values determined in the previous sections but again without the SP procedure. Figs. 5 and 6 show the graphs of the average CPU times obtained on the existing and newly proposed instances, respectively, when adopting each of the possible 8 settings, including the version without any speedup mechanism.

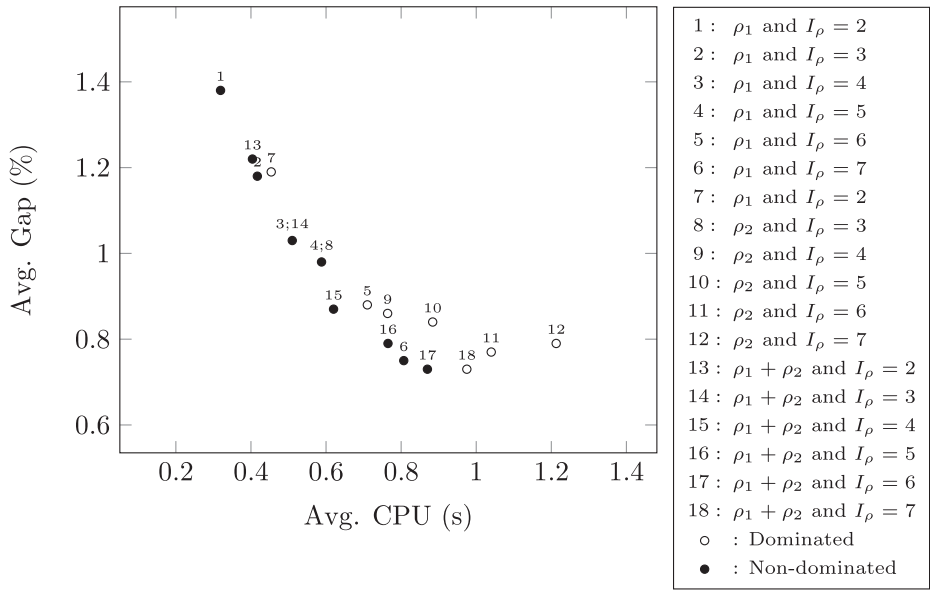

Fig. 2. Impact of the perturbation mechanisms.

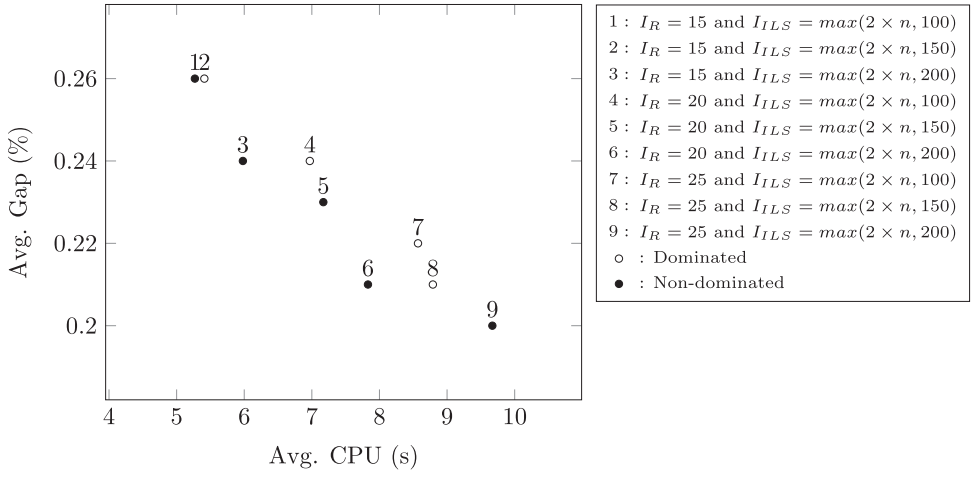

Fig. 3. Parameter tuning: I_R and I_{ILS}.

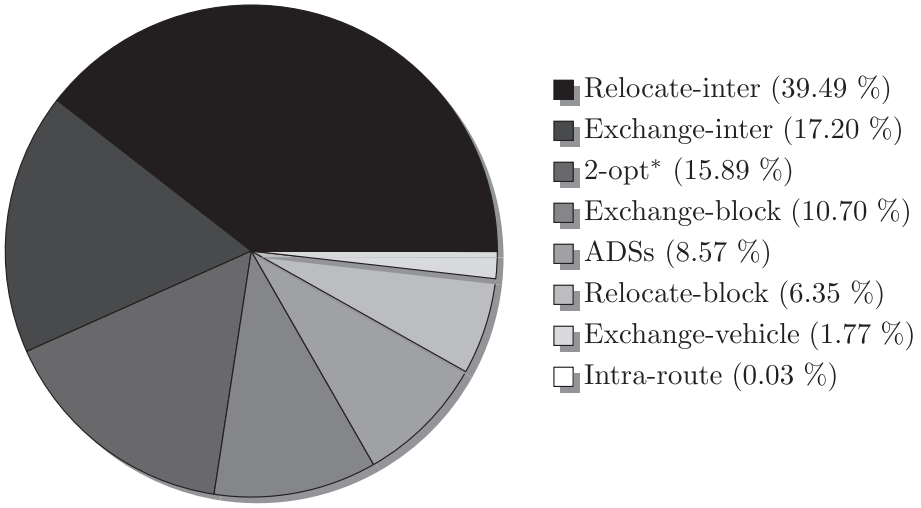

Fig. 4. Percentage of time of each local search component.

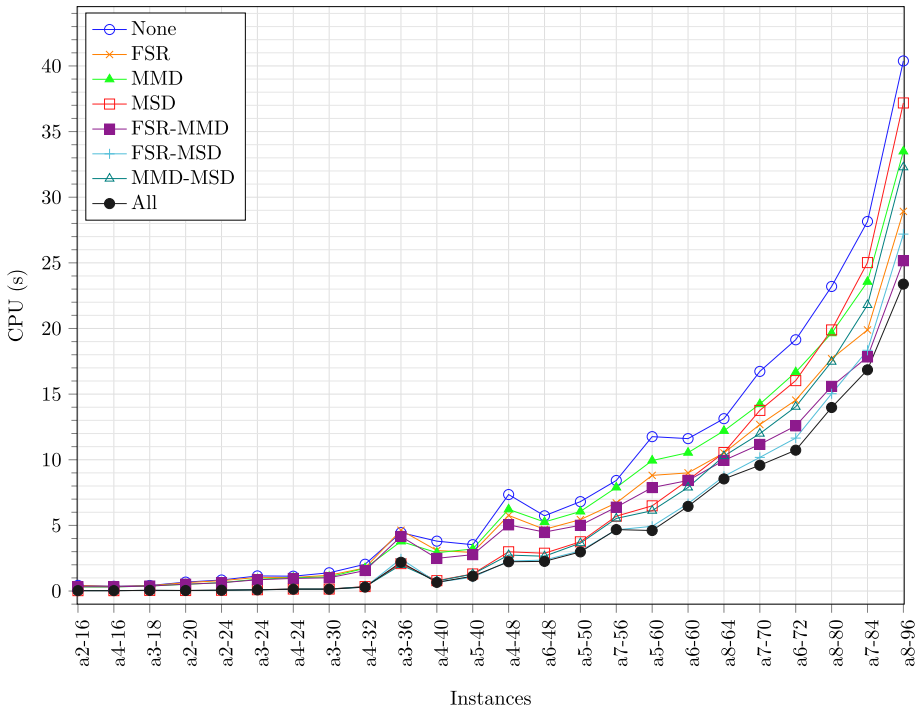

Fig. 5. Impact of the speedup mechanisms on the existing benchmark instances.

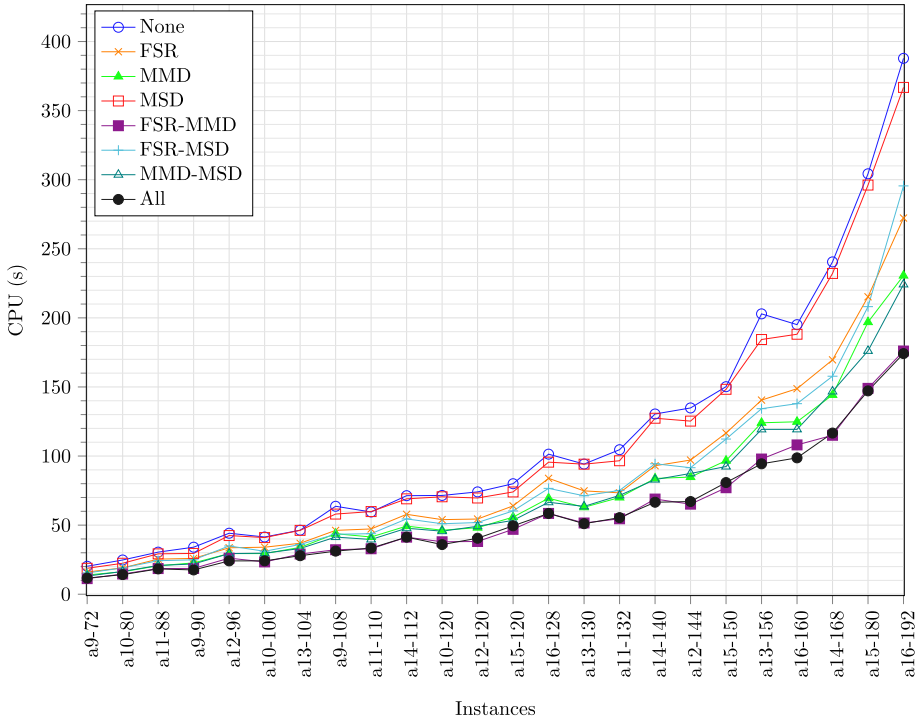

Fig. 6. Impact of the speedup mechanisms on the newly proposed instances.

Regarding the existing instances with up to 96 requests, the results achieved visibly shows the benefit of incorporating the speedup mechanisms. When analyzing each mechanism individually, MSD produced better gains on the instances of smaller size and such gains tend to decrease as the instance size increases. This happens because the same local optimal solutions are likely to be obtained for smaller instances, as opposed to larger instances where one hardly finds the same local optima. On the other hand, MMD and FSR attained proportional gains as the instance size increases, and the latter appeared to be the most effective. Overall, the best setting is the one where all mechanisms were considered. For example, when considering the instances with 96 requests, i.e. the larger ones, it can be verified that the version with all three mechanisms was approximately 1.73 times faster than the one with no mechanism.

A similar conclusion can be derived when analyzing the CPU times obtained on the new benchmark dataset ranging from 72 to 192 requests. As expected, the gains due to MSD on larger instances are relatively marginal, but the other mechanisms clearly play a critical role in the speedup of the method, especially when put together, running about 2.23 times faster than the version in which no speedup mechanism was considered, on the largest instance. Moreover, one can observe that the larger the size of the instance, the larger the speedup gains.

5.6. Impact of the SP procedure

We examined the impact of the SP procedure over the instances used in Sections 5.1, 5.2 and 5.3. In this testing, we compare three versions of the algorithm: one without SP and with the parameter values defined in the previous sections; another with the double of restarts, that is $I_R = 40$, and also without SP; and a third version with the same parameter values of the first one but with the SP procedure. The motivation for experimenting with the second version is that the CPU times are more comparable with the version that includes SP. Table 3 reports the average results obtained, including the number of best solutions found (#BKS) and the number of improved solutions (#Imp).

Overall, we can verify that the version with SP finds better solutions, and that the version without SP and with $I_R = 40$ is completely dominated by the former. The results suggest that the SP procedure is clearly beneficial for the method, and that increasing the number of restarts of the version without SP improves the average gap, but it is still not as effective as incorporating the SP procedure. Note that several BKSs were only found after its inclusion.

5.7. Comparison with the literature

This section presents the results found by ILS-SP on the HDARP and MDHDARP benchmark instances, which are compared with those obtained by Braekers et al. (2014). Because we had access to the source code of the DA metaheuristic implemented in their work, we were also capable of providing the average CPU times found when running the code 10 times for each instance in our machine, for a fair comparison. It is worth mentioning that Braekers et al. (2014) ran their experiments on a laptop equipped with Intel Core 2.6 GHz with 4 GB of RAM. Moreover, we also report improved LBs that were attained by means of long runs of the exact algorithm proposed in Braekers et al. (2014). When then observed that several UBs reported in Masmoudi et al. (2016) and Masmoudi et al. (2017) for HDARP and MDHDARP, respectively, were smaller than such LBs, possibly due to precision issues. For this reason, it was thought advisable not to compare our results with those provided in their works.

5.7.1. HDARP

Table 4 presents the results found on the HDARP instances. For each instance, we report the best known LB, the BKS reported in Braekers et al. (2014), the values of the best and average solutions found by DA and ILS-SP, as well as the average CPU time in seconds and the average gap between the value of the average solution and the LB. The average CPU time achieved by running the DA algorithm in our machine is provided in column CPU$*$ (s). Improved solutions are highlighted in bold. We remark that Braekers et al. (2014) only reported results for the instances with up to 4 vehicles, and the results attained for the remaining instances were later made available by the authors.

The results obtained show that ILS-SP and DA found similar results in terms of solution quality, but the former visibly outperformed the latter with regards to CPU time. It can be observed that there is considerable runtime difference between the methods, especially for the instances with up to 60 requests, where the proposed algorithm spent up to 10 s, and DA spent up to 40 s. In addition, ILS-SP managed to attain all BKSs, including all known optima, and to improve the result of one instance.

5.7.2. MDHDARP

Table 5 presents the results found on the existing MDHDARP instances. The information reported have the same meaning as those provided in Table 4. The runtime behavior of the results are similar to the one for the HDARP, but this time the gains on the solution quality are more noticeable for the MDHDARP, where the result of 6 instances were improved. There were significant improvements on the larger instances of set *I*, in which there are multiple resources associated with the demands and the vehicle fleet is heterogeneous. Also, ILS-SP was capable of finding all known optima, and also several BKSs that DA could not.

5.7.3. Results for the newly proposed instances

Table 6 illustrates the results obtained on the newly proposed benchmark dataset for the MDHDARP. For a fair comparison, we report the results achieved by DA, and those attained by our algorithm when imposing the average CPU time of the former as time limit for the latter (ILS-SP*). We also provide the results found by ILS-SP using the regular stopping criterion. Column BKS indicates the best results found among all experiments involving our algorithm, whereas the values highlighted in bold indicate when the BKS has been found by a given method.

The results suggest that both ILS-SP* and ILS-SP outperformed DA in terms of solution quality. Note that ILS-SP* dominates DA

Table 3
Impact of the SP procedure.

	ILS (I_R = 20)				ILS (I_R = 40)				ILS-SP (I_R = 20)			
	#BKS	#Imp	Gap (%)	CPU (s)	#BKS	#Imp	Gap (%)	CPU (s)	#BKS	#Imp	Gap (%)	CPU (s)
HDARP												
U	12	0	0.066	7.77	12	0	0.030	14.86	12	0	0.004	12.13
E	11	0	0.031	7.42	11	0	0.024	14.07	12	0	0.005	11.96
I	11	1	0.128	7.67	12	0	0.085	14.74	11	1	0.056	11.54
Total HDARP	34	1	0.075	7.62	35	0	0.046	14.56	35	1	0.022	11.87
MDHDARP												
U	8	0	0.295	8.34	9	0	0.254	16.28	11	1	0.167	14.14
E	9	1	0.158	8.10	10	1	0.139	15.71	11	1	0.061	14.68
I	7	3	0.602	7.70	8	4	0.554	15.36	8	4	0.455	12.07
Total MDHDARP	24	4	0.352	8.05	27	5	0.316	15.78	30	6	0.228	13.63
Total/ Mean	58	5	0.213	7.83	62	5	0.181	15.17	65	7	0.125	12.75

Table 4
Results obtained on the HDARP instances.

			DA					ILS-SP			
Instance	BKS	LB	Best	Avg.	Gap (%)	CPU (s)	CPU* (s)	Best	Avg.	Gap (%)	CPU (s)
U											
a2-16	294.25	294.25	294.25	294.25	0.00	8.60	7.80	294.25	294.25	0.00	0.44
a2-20	344.83	344.83	344.83	344.83	0.00	20.20	19.80	344.83	344.83	0.00	0.62
a2-24	431.12	431.12	431.12	431.12	0.00	17.40	17.65	431.12	431.12	0.00	0.74
a3-18	300.48	300.48	300.48	300.48	0.00	9.40	7.30	300.48	300.48	0.00	0.39
a3-24	344.83	344.83	344.83	344.83	0.00	16.60	14.40	344.83	344.83	0.00	0.92
a3-30	494.85	494.85	494.85	494.85	0.00	18.80	18.10	494.85	494.85	0.00	1.18
a3-36	583.19	583.19	583.19	583.19	0.00	28.40	28.10	583.19	583.19	0.00	1.74
a4-16	282.68	282.68	282.68	282.68	0.00	9.80	6.90	282.68	282.68	0.00	0.40
a4-24	375.02	375.02	375.02	375.02	0.00	13.00	10.50	375.02	375.02	0.00	0.97
a4-32	485.50	485.50	485.50	485.50	0.00	25.60	21.70	485.50	485.50	0.00	1.53
a4-40	557.69	557.69	557.69	557.69	0.00	26.40	22.30	557.69	557.69	0.00	2.19
a4-48	668.82	668.82	668.82	668.82	0.00	35.40	33.50	668.82	668.82	0.00	4.63
a5-40	498.41	498.41	498.41	498.41	0.00	21.80	19.20	498.41	498.41	0.00	2.45
a5-50	686.62	686.62	686.62	686.62	0.00	30.60	29.60	686.62	686.62	0.00	5.63
a5-60	808.42	808.42	808.42	808.42	0.00	43.80	44.60	808.42	808.42	0.00	7.61
a6-48	604.12	604.12	604.12	604.12	0.00	30.60	27.00	604.12	604.12	0.00	4.07
a6-60	819.25	819.25	819.25	819.25	0.00	37.00	36.30	819.25	819.25	0.00	9.64
a6-72	916.05	916.05	916.05	916.05	0.00	53.60	54.10	916.05	916.24	0.02	17.32
a7-56	724.04	724.04	724.04	724.04	0.00	32.40	30.50	724.04	724.04	0.00	6.62
a7-70	889.12	889.12	889.12	889.30	0.02	43.60	42.90	889.12	889.12	0.00	13.51
a7–84	1033.37	1033.37	1033.37	1033.38	0.00	58.20	58.00	1033.37	1033.66	0.03	23.68
a8-64	747.46	747.46	747.46	747.46	0.00	37.80	36.20	747.46	747.46	0.00	13.40
a8-80	945.73	945.73	945.73	945.73	0.00	57.60	57.40	945.73	945.73	0.00	22.70
a8-96	1229.66	1229.66	1229.66	1232.03	0.19	66.40	65.60	1229.66	1229.66	0.00	35.67
Avg.	627.73	627.73	627.73	627.84	0.009	30.96	29.56	627.73	627.75	0.002	7.42
E											
a2-16	331.16	331.16	331.16	331.16	0.00	9.40	7.80	331.16	331.16	0.00	0.44
a2-20	347.03	347.03	347.03	347.03	0.00	19.60	19.20	347.03	347.03	0.00	0.63
a2-24	450.25	450.25	450.25	450.25	0.00	15.80	15.30	450.25	450.25	0.00	0.76
a3-18	300.63	300.63	300.63	300.63	0.00	9.60	7.20	300.63	300.63	0.00	0.44
a3-24	344.91	344.91	344.91	344.91	0.00	14.60	12.20	344.91	344.91	0.00	0.91
a3-30	500.58	500.58	500.58	500.58	0.00	17.00	16.00	500.58	500.58	0.00	1.15
a3-36	583.19	583.19	583.19	583.19	0.00	23.60	24.20	583.19	583.19	0.00	1.53
a4-16	285.99	285.99	285.99	285.99	0.00	8.20	5.90	285.99	285.99	0.00	0.41
a4-24	383.84	383.84	383.84	383.84	0.00	12.20	9.90	383.84	383.84	0.00	0.94
a4-32	500.24	500.24	500.24	500.24	0.00	22.80	20.70	500.24	500.24	0.00	1.47
a4-40	580.42	580.42	580.42	580.42	0.00	24.20	22.30	580.42	580.42	0.00	2.23
a4-48	670.52	670.52	670.52	670.52	0.00	33.60	32.70	670.52	670.52	0.00	4.41
a5-40	500.06	500.06	500.06	500.06	0.00	21.80	18.70	500.06	500.06	0.00	2.46
a5-50	693.77	693.77	693.77	693.77	0.00	28.00	26.30	693.77	693.77	0.00	4.52
a5-60	828.90	828.90	828.90	829.28	0.05	40.40	38.90	828.90	828.90	0.00	7.03
a6-48	614.36	614.36	614.36	614.36	0.00	29.40	25.80	614.36	614.36	0.00	4.13
a6-60	847.58	847.58	847.58	848.48	0.11	41.60	38.60	847.58	847.58	0.00	11.27
a6-72	949.17	949.17	949.17	949.17	0.00	55.40	54.60	949.17	949.35	0.02	16.47
a7-56	740.63	740.63	740.63	740.63	0.00	32.80	29.70	740.63	740.63	0.00	5.99
a7-70	946.32	946.32	946.32	946.32	0.00	46.60	41.80	946.32	946.32	0.00	13.50
a7–84	1092.90	1092.90	1092.90	1092.93	0.00	57.20	56.20	1092.90	1093.07	0.02	26.23
a8-64	762.81	762.81	762.81	762.81	0.00	39.00	34.90	762.81	762.81	0.00	10.64
a8-80	982.71	982.71	982.71	982.71	0.00	57.40	57.30	982.71	982.71	0.00	17.12
a8-96	1265.36[1]	1265.36	1265.54	1266.78	0.11	63.00	63.80	1265.36	1265.74	0.03	38.03
Avg.	645.98	645.97	645.98	646.09	0.011	30.13	28.33	645.97	646.00	0.003	7.20
I											
a2-16	294.25	294.25	294.25	294.25	0.00	7.20	6.60	294.25	294.25	0.00	0.37
a2-20	355.74	355.74	355.74	355.74	0.00	17.40	17.50	355.74	355.74	0.00	0.47
a2-24	431.12	431.12	431.12	431.12	0.00	12.60	12.90	431.12	431.12	0.00	0.44
a3-18	302.17	302.17	302.17	302.17	0.00	8.40	6.60	302.17	302.17	0.00	0.35
a3-24	344.83	344.83	344.83	344.83	0.00	13.40	12.00	344.83	344.83	0.00	0.71
a3-30	494.85	494.85	494.85	494.85	0.00	14.80	14.00	494.85	494.85	0.00	1.03
a3-36	618.15	618.15	618.15	618.59	0.07	22.60	23.40	618.15	618.15	0.00	3.80
a4-16	299.05	299.05	299.05	299.05	0.00	7.20	5.30	299.05	299.05	0.00	0.27
a4-24	375.02	375.02	375.02	375.02	0.00	12.00	10.30	375.02	375.02	0.00	0.93
a4-32	486.93	486.93	486.93	487.50	0.12	21.00	19.60	486.93	486.93	0.00	1.42
a4-40	557.69	557.69	557.69	557.69	0.00	23.80	22.50	557.69	557.69	0.00	2.06
a4-48	670.72	670.72	670.72	670.72	0.00	30.00	29.40	670.72	670.72	0.00	4.87
a5-40	507.18	507.18	507.18	507.18	0.00	19.20	17.70	507.18	507.18	0.00	2.46
a5-50	690.99	690.99	690.99	690.99	0.00	27.80	27.10	690.99	690.99	0.00	5.37
a5-60	816.15	816.15	816.15	816.81	0.08	39.00	39.50	816.15	816.15	0.00	6.84
a6-48	604.12	604.12	604.12	604.12	0.00	29.00	26.90	604.12	604.12	0.00	3.87
a6-60	829.23	829.23	829.23	829.26	0.00	34.20	35.00	829.23	829.23	0.00	9.99
a6-72	938.46	938.46[2]	938.46	939.32	0.09	48.60	48.30	938.46	938.46	0.00	14.94
a7-56	727.20	727.20	727.2	727.20	0.00	29.20	28.60	727.20	727.20	0.00	6.94
a7-70	925.39	923.19[2]	925.39	927.60	0.48	39.20	38.60	925.39	925.64	0.27	13.53

Table 4 (*continued*)

			DA					ILS-SP			
Instance	BKS	LB	Best	Avg.	Gap (%)	CPU (s)	CPU* (s)	Best	Avg.	Gap (%)	CPU (s)
a7–84	1037.04	1037.04[2]	1037.04	1037.88	0.08	54.00	54.70	1037.04	1037.04	0.00	23.01
a8-64	748.04	748.04	748.04	748.04	0.00	37.20	34.10	748.04	748.04	0.00	13.50
a8-80	969.87	968.84[2]	969.87	970.83	0.21	50.20	47.90	**968.84**	969.60	0.08	20.75
a8-96	1237.89	1233.86[2]	1237.89	1237.96	0.33	61.80	60.00	1237.89	1237.97	0.33	32.30
Avg.	635.92	635.62	635.92	636.20	0.061	27.49	26.60	635.88	635.92	0.028	7.09
Total Avg.	636.54	636.44	636.54	636.71	0.03	29.53	28.17	636.53	636.56	0.01	7.24

1 Result presented in Masmoudi et al. (2018).
2 New LBs provided by the BC algorithm proposed in Braekers et al. (2014).

Table 5
Results obtained on the MDHDARP instances.

			DA					ILS-SP			
Instance	BKS	LB	Best	Avg.	Gap (%)	CPU (s)	CPU* (s)	Best	Avg.	Gap (%)	CPU (s)
U											
a2-16	284.18	284.18	284.18	284.18	0.00	5.40	4.63	284.18	284.18	0.00	0.37
a2-20	343.43	343.43	343.43	343.43	0.00	16.00	14.77	343.43	343.43	0.00	0.58
a2-24	427.17	427.17	427.17	427.17	0.00	15.80	15.71	427.17	427.17	0.00	0.77
a3-18	289.67	289.67	289.67	289.67	0.00	8.00	5.95	289.67	289.67	0.00	0.39
a3-24	348.30	348.30	348.30	348.30	0.00	14.60	12.36	348.30	348.30	0.00	0.86
a3-30	469.16	469.16	469.16	469.16	0.00	14.40	13.08	469.16	469.16	0.00	0.99
a3-36	592.42	592.42	592.42	592.42	0.00	24.60	23.87	592.42	592.42	0.00	1.79
a4-16	262.44	262.44	262.44	262.44	0.00	7.20	5.17	262.44	262.44	0.00	0.28
a4-24	355.72	355.72	355.72	355.72	0.00	10.80	8.58	355.72	355.72	0.00	0.69
a4-32	461.65	461.65	461.65	461.65	0.00	17.80	15.58	461.65	461.65	0.00	1.55
a4-40	540.34	540.34	540.34	540.34	0.00	20.00	18.21	540.34	540.34	0.00	2.51
a4-48	631.75	631.75	631.75	632.31	0.09	25.80	24.96	631.75	631.75	0.00	4.90
a5-40	482.19	482.19	482.19	482.19	0.00	19.80	17.62	482.19	482.19	0.00	3.01
a5-50	664.54	664.54	664.54	665.17	0.09	28.60	26.61	664.54	664.54	0.00	6.70
a5-60	789.87	789.87	789.87	789.87	0.00	37.80	35.87	789.87	789.90	0.00	10.23
a6-48	586.08	586.08	586.08	586.08	0.00	25.20	22.31	586.08	586.08	0.00	5.12
a6-60	776.63	776.63	776.63	776.65	0.00	32.20	30.67	776.63	776.68	0.01	10.99
a6-72	883.78	883.78	883.78	883.78	0.00	49.20	48.22	883.78	883.78	0.00	18.91
a7-56	680.08	680.08	680.08	682.14	0.30	28.40	25.18	680.08	680.08	0.00	8.34
a7-70	854.22	854.22	855.76	857.67	0.40	40.40	38.46	854.22	855.04	0.10	15.45
a7–84	1007.33	1007.33	1007.33	1009.92	0.26	51.80	51.73	1007.33	1007.91	0.06	28.10
a8-64	713.11	713.11	713.11	713.11	0.00	35.60	32.94	713.11	713.11	0.00	13.14
a8-80	889.29[1]	885.45	890.69	892.79	0.83	47.60	46.09	**889.10**	889.67	0.48	26.14
a8-96	1185.45[1]	1170.91	1187.26	1189.74	1.61	61.00	60.19	1185.45	1186.87	1.36	42.99
Avg.	604.95	604.18	605.15	605.66	0.149	26.58	24.95	604.94	605.09	0.083	8.53
E											
a2-16	327.67	327.67	327.67	327.67	0.00	6.40	5.22	327.67	327.67	0.00	0.43
a2-20	345.59	345.59	345.59	345.59	0.00	17.80	14.01	345.59	345.59	0.00	0.55
a2-24	445.88	445.88	445.88	445.88	0.00	15.20	14.74	445.88	445.88	0.00	0.72
a3-18	289.67	289.67	289.67	289.67	0.00	7.80	5.95	289.67	289.67	0.00	0.39
a3-24	348.61	348.61	348.61	348.61	0.00	12.40	10.49	348.61	348.61	0.00	0.86
a3-30	471.43	471.43	471.43	471.43	0.00	12.80	11.80	471.43	471.43	0.00	1.05
a3-36	593.84	593.84	593.84	593.84	0.00	20.80	20.73	593.84	593.84	0.00	1.71
a4-16	262.44	262.44	262.44	262.44	0.00	6.00	4.30	262.44	262.44	0.00	0.26
a4-24	364.54	364.54	365.54	365.54	0.27	10.40	8.42	364.54	364.54	0.00	0.77
a4-32	476.59	476.59	476.59	476.59	0.00	16.20	14.29	476.59	476.59	0.00	1.48
a4-40	562.86	562.86	562.86	562.86	0.00	19.40	17.62	562.86	562.86	0.00	2.25
a4-48	633.49	633.49	633.49	633.49	0.00	25.00	23.97	633.49	633.49	0.00	4.24
a5-40	483.84	483.84	483.84	483.84	0.00	19.40	17.37	483.84	483.84	0.00	2.93
a5-50	674.19	674.19	674.19	674.71	0.08	26.80	24.93	674.19	674.19	0.00	5.98
a5-60	813.96	813.96	813.96	814.00	0.00	34.60	33.91	813.96	813.96	0.00	8.32
a6-48	599.76	599.76	599.76	599.91	0.03	24.20	21.62	599.76	599.76	0.00	5.38
a6-60	802.49	802.49	802.49	803.59	0.14	34.40	32.33	802.49	802.49	0.00	10.44
a6-72	915.03	915.03	915.03	915.53	0.05	50.00	48.81	915.03	915.26	0.03	18.61
a7-56	703.62	703.62	703.62	703.62	0.00	28.20	25.49	703.62	703.62	0.00	8.20
a7-70	910.91	910.91	911.06	913.51	0.29	42.00	39.02	910.91	910.91	0.00	16.15
a7–84	1059.12	1059.12	1060.21	1062.97	0.36	51.80	51.27	1059.12	1061.46	0.22	44.21
a8-64	731.11	731.11	731.11	733.01	0.26	44.00	32.45	731.11	731.11	0.00	14.49
a8-80	925.72	925.72	927.89	930.95	0.56	71.20	45.19	925.72	925.98	0.03	22.66
a8-96	1222.43	1215.38	1222.43	1225.02	0.79	94.40	57.96	**1219.40**	1220.96	0.46	40.60
Avg.	623.53	623.24	623.72	624.34	0.118	28.80	24.25	623.41	623.59	0.031	8.86
I											
a2-16	284.18	284.18	284.18	284.18	0.00	7.60	3.96	284.18	284.18	0.00	0.30
a2-20	358.88	358.88	358.88	358.88	0.00	19.60	11.43	358.88	358.88	0.00	0.48
a2-24	439.29	439.29	439.29	439.29	0.00	13.80	11.98	439.29	439.29	0.00	0.45
a3-18	292.41	292.41	292.41	292.41	0.00	7.80	5.09	292.41	292.41	0.00	0.27

(*continued on next page*)

Table 5 (*continued*)

			DA					ILS-SP			
Instance	BKS	LB	Best	Avg.	Gap (%)	CPU (s)	CPU* (s)	Best	Avg.	Gap (%)	CPU (s)
a3-24	348.54	348.54	348.54	348.54	0.00	13.20	9.73	348.54	348.54	0.00	0.73
a3-30	486.04	486.04	486.04	486.04	0.00	14.80	12.09	486.04	486.04	0.00	0.89
a3-36	626.96	626.96	626.96	627.73	0.12	23.60	20.81	626.96	626.96	0.00	10.58
a4-16	285.40	285.40	285.40	285.40	0.00	6.00	3.88	285.40	285.40	0.00	0.33
a4-24	357.51	357.51	357.51	357.51	0.00	12.00	8.42	357.51	357.51	0.00	0.70
a4-32	471.54	471.54	471.54	471.54	0.00	16.60	13.26	471.54	471.54	0.00	1.32
a4-40	542.56	542.56	542.56	542.56	0.00	21.40	17.04	542.56	542.56	0.00	2.13
a4-48	637.58	637.58	637.58	637.58	0.00	26.80	23.25	637.58	637.58	0.00	4.48
a5-40	496.36	496.36	496.36	496.36	0.00	18.80	14.73	496.36	496.36	0.00	3.13
a5-50	669.30	669.30	669.30	669.67	0.06	27.40	22.79	669.30	669.30	0.00	4.83
a5-60	800.10	800.10	800.10	802.42	0.29	37.60	32.65	800.10	800.10	0.00	8.47
a6-48	586.08	586.08	586.08	586.08	0.00	27.20	21.60	586.08	586.08	0.00	4.95
a6-60	789.40	789.40	789.40	789.80	0.05	32.60	27.37	789.40	789.41	0.00	8.38
a6-72	910.24	910.24	910.24	910.24	0.00	47.60	41.81	910.24	910.63	0.04	12.42
a7-56	688.51	688.51	688.51	688.51	0.00	27.80	22.63	688.51	688.51	0.00	8.38
a7-70	887.53	878.91[2]	887.53	889.75	1.23	39.40	32.81	**885.89**	885.89	0.79	12.90
a7–84	1014.12	1012.24[2]	1014.12	1016.67	0.44	56.40	48.64	**1012.24**	1013.40	0.11	28.55
a8-64	713.11	713.11	713.11	715.56	0.34	38.20	31.47	713.11	713.11	0.00	11.43
a8-80	925.56	904.73	925.56	926.63	2.42	48.00	40.02	**923.87**	925.10	2.25	19.59
a8-96	1196.35	1169.75	1196.79	1204.49	2.97	63.80	55.35	**1195.85**	1196.22	2.26	31.66
Avg.	616.90	614.57	617.00	617.83	0.330	27.00	22.20	616.74	616.87	0.228	7.39
Total Avg.	615.13	614.00	615.29	615.94	0.20	27.46	23.80	615.03	615.18	0.11	8.26

1 Results presented in Molenbruch et al., 2017.
2 New LBs provided by the BC algorithm proposed in Braekers et al., 2014.

Table 6
Results obtained on the MDHDARP new instances.

		DA				ILS-SP*			ILS-SP			
Instance	BKS	Best	Avg.	Gap (%)	CPU (s)	Best	Avg	Gap	Best	Avg.	Gap (%)	CPU (s)
a9-72	828.81	829.92	833.87	0.61	41.40	**828.81**	829.65	0.10	**828.81**	829.43	0.07	17.34
a9-90	1050.44	1053.58	1056.73	0.60	53.80	**1050.44**	1052.85	0.23	1052.24	1053.73	0.31	30.79
a9-108	1260.75	1261.51	1267.71	0.55	71.50	1262.65	1265.15	0.35	1261.45	1265.49	0.38	42.84
a10-80	949.10	**949.10**	952.75	0.38	46.60	**949.10**	949.33	0.02	**949.10**	949.65	0.06	20.85
a10-100	1147.39	1152.56	1160.96	1.18	62.30	**1147.39**	1151.95	0.40	1147.80	1153.13	0.50	67.96
a10-120	1379.26	1383.68	1391.16	0.86	79.50	**1379.26**	1391.91	0.92	1384.52	1390.39	0.81	47.89
a11-88	906.48	907.35	914.78	0.92	51.30	**906.48**	908.58	0.23	**906.48**	908.25	0.20	32.62
a11-110	1256.99	1263.65	1271.04	1.12	65.20	1257.43	1264.59	0.60	**1256.99**	1262.75	0.46	42.16
a11-132	1355.56	1363.57	1372.00	1.21	83.40	1361.59	1365.17	0.71	**1355.56**	1362.43	0.51	65.95
a12-96	1070.59	1080.93	1085.94	1.43	55.00	**1070.59**	1071.29	0.07	1070.83	1071.83	0.12	41.55
a12-120	1351.38	1357.36	1360.99	0.71	69.00	**1351.38**	1359.06	0.57	1354.44	1358.41	0.52	50.47
a12-144	1592.23	1601.03	1613.55	1.34	85.30	1592.89	1601.01	0.55	**1592.23**	1600.12	0.50	82.46
a13-104	1147.70	1153.10	1158.70	0.96	51.20	**1147.70**	1154.41	0.58	1150.70	1155.16	0.65	34.46
a13-130	1348.83	1359.82	1365.34	1.22	74.40	1352.57	1359.72	0.81	**1348.83**	1357.80	0.67	65.94
a13-156	1670.19	1674.25	1685.09	0.89	90.70	1673.23	1679.94	0.58	1672.38	1679.47	0.56	120.52
a14-112	1186.92	1190.54	1198.72	0.99	60.50	1186.98	1190.23	0.28	1190.00	1193.88	0.59	53.94
a14-140	1492.18	1503.11	1512.18	1.34	84.60	**1492.18**	1506.67	0.97	1498.43	1502.94	0.72	85.60
a14-168	1680.76	1697.54	1707.08	1.57	97.10	1689.05	1699.98	1.14	**1680.76**	1696.71	0.95	147.15
a15-120	1251.55	1252.62	1262.22	0.85	61.70	1254.92	1257.06	0.44	1252.55	1257.29	0.46	58.83
a15-150	1606.26	1620.61	1624.37	1.12	77.40	1606.82	1613.79	0.46	**1606.26**	1611.68	0.34	100.66
a15-180	1878.70	1890.33	1904.83	1.39	100.50	1878.81	1888.36	0.51	1881.01	1885.60	0.37	185.12
a16-128	1359.23	1367.16	1376.39	1.26	67.60	1363.21	1372.39	0.97	**1359.23**	1368.07	0.65	74.21
a16-160	1652.98	1676.22	1686.25	2.01	81.90	1666.38	1673.23	1.23	1658.69	1667.95	0.91	123.06
a16-192	2024.88	2055.58	2064.86	1.97	107.30	2034.51	2046.42	1.06	2028.69	2038.29	0.66	217.29
Avg.	1352.05	1360.21	1367.81	1.10	71.63	1354.35	1360.53	0.57	1353.67	1359.19	0.50	75.40

not only when observing the best solution achieved, but also when comparing the average values. Nonetheless, DA is still capable of obtaining high quality solutions, as ratified by its average gap of 1.10% with respect to the BKSs. Furthermore, ILS-SP obtained a slightly better average gap than ILS-SP* (0.50% against 0.57%) at the expense of a larger average CPU time (75.40 s against 71.63 s).

6. Concluding remarks

In this work, we developed a matheuristic algorithm for solving two DARP variants, namely: HDARP and MDHDARP. The proposed approach combines ILS with a SP procedure, where the former generates promising routes that are optimality combined by the latter. Both of them also share a cooperative aspect during the search by helping each other's performance. For example, when the solver finds an incumbent solution, ILS is called by means of a callback function with a view of improving such solution. If so, the associated UB is then used as a better cutoff value for the solver. In addition, we implemented two neighborhood structures that either move or swap subsequences of contiguous visits denoted as blocks. Moreover, we showed how to considerably improve the local search performance by presenting three speedup mechanisms.

Extensive computational experiments were carried out not only to measure the impact of the different components of the algorithm, but also to compare its performance with the best known heuristic, in particular, the DA-based approach by Braekers et al. (2014). The results obtained showed that ILS-SP can obtain high quality solutions in a very limited amount of time,

clearly outperforming DA, on average. We provide 7 new improved solutions, 1 one for HDARP benchmark dataset, and 6 for the MDHDARP dataset. Furthermore, all known optima were found by our algorithm. We also conducted tests on newly proposed instances containing up to 192 requests, which are clearly more challenging than those from the existing benchmark dataset, composed of instances with up to 96 requests, and whose current associated bounds are already of high quality. The results achieved show the superiority of our matheuristic when compared to DA, which in turn is still capable of achieving promising solutions.

Future work may include the extension of the proposed method to address other multi-attribute DARPs, such as the *mixed fleet heterogeneous dial-a-ride problem* (Masmoudi et al., 2020) and the *dial-a-ride problem with electric vehicles* (Masmoudi et al., 2018), as well as similar emerging variants such as the *share-a-ride problem* (Li et al., 2014), in which one combines the pickup-and-delivery services of passengers and parcels.

Declaration of Competing Interest

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

Acknowledgements

We would like to warmly thank Dr. Kris Braekers for kindly providing both the source code of the DA metaheuristic and the updated LBs. This research was partially supported by the Brazilian research agencies CNPq, grant 307843/2018, and CAPES, Finance code – 001.

References

Attanasio, A., Cordeau, J.-F., Ghiani, G., Laporte, G., 2004. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. Parallel Comput. 30 (3), 377–387.
Beek, O., Raa, B., Dullaert, W., Vigo, D., 2018. An efficient implementation of a static move descriptor-based local search heuristic. Comput. Oper. Res. 94, 1–10.
Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., Laporte, G., 2007. Static pickup and delivery problems: a classification scheme and survey. Top 15 (1), 1–31.
Bodin, L.D., Sexton, T.R., 1986. The multi-vehicle subscriber dial-a-ride problem. TIMS Stud. Manage. Sci. 22.
Braekers, K., Caris, A., Janssens, G.K., 2014. Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. Transp. Res. B Methodol. 67, 166–186.
Bulhões, T., Subramanian, A., Erdoğan, G., Laporte, G., 2018. The static bike relocation problem with multiple vehicles and visits. Eur. J. Oper. Res. 264 (2), 508–523.
Carnes, T.A., Henderson, S.G., Shmoys, D.B., Ahghari, M., MacDonald, R.D., 2013. Mathematical programming guides air-ambulance routing at ornge. INFORMS J. Appl. Anal. 43 (3), 232–239.
Chevrier, R., Liefooghe, A., Jourdan, L., Dhaenens, C., 2012. Solving a dial-a-ride problem with a hybrid evolutionary multi-objective approach: application to demand responsive transport. Appl. Soft Comput. 12 (4), 1247–1258.
Cordeau, J.-F., 2006. A branch-and-cut algorithm for the dial-a-ride problem. Oper. Res. 54 (3), 573–586.
Cordeau, J.-F., Laporte, G., 2003a. The dial-a-ride problem (DARP): variants, modeling issues and algorithms. Quart. J. Belgian, French Ital. Oper. Res. Soc. 1 (2), 89–101.
Cordeau, J.-F., Laporte, G., 2003b. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transp. Res. B Methodol. 37 (6), 579–594.
Desrosiers, J., Dumas, Y., Soumis, F., 1986. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. Am. J. Math. Manage. Sci. 6 (3–4), 301–325.
Desrosiers, J., Dumas, Y., Soumis, F., 1988. The multiple vehicle dial-a-ride problem. In: Daduna, J.R., Wren, A. (Eds.), Computer-Aided Transit Scheduling. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 15–27.
Detti, P., Papalini, F., de Lara, G.Z.M., 2017. A multi-depot dial-a-ride problem with heterogeneous vehicles and compatibility constraints in healthcare. Omega 70, 1–14.
Dumas, Y., Desrosiers, J., Soumis, F., 1989. Large scale multi-vehicle dial-a-ride problems. Les Cahiers du GERAD, G-89-30.
Dumas, Y., Desrosiers, J., Soumis, F., 1991. The pickup and delivery problem with time windows. Eur. J. Oper. Res. 54 (1), 7–22.
Gschwind, T., Drexl, M., 2019. Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. Transp. Sci. 53 (2), 480–491.
Gschwind, T., Irnich, S., 2015. Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. Transp. Sci. 49 (2), 335–354.
Hansen, P., Mladenović, N., 2001. Variable neighborhood search: Principles and applications. Eur. J. Oper. Res. 130 (3), 449–467.
Ho, S.C., Szeto, W., Kuo, Y.-H., Leung, J.M., Petering, M., Tou, T.W., 2018. A survey of dial-a-ride problems: literature review and recent developments. Transp. Res. B Methodol. 111, 395–421.
Ioachim, I., Desrosiers, J., Dumas, Y., Solomon, M.M., Villeneuve, D., 1995. A request clustering algorithm for door-to-door handicapped transportation. Transp. Sci. 29 (1), 63–78.
Jain, S., Van Hentenryck, P., 2011. Large neighborhood search for dial-a-ride problems. In: International Conference on Principles and Practice of Constraint Programming. Springer, pp. 400–413.
Jaw, J.-J., Odoni, A.R., Psaraftis, H.N., Wilson, N.H., 1986. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. Transp. Res. B Methodol. 20 (3), 243–257.
Li, B., Krushinsky, D., Reijers, H.A., Van Woensel, T., 2014. The share-a-ride problem: people and parcels sharing taxis. Eur. J. Oper. Res. 238 (1), 31–40.
Lourenço, H.R., Martin, O.C., Stützle, T., 2019. Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.-Y. (Eds.), Handbook of Metaheuristics. Springer International Publishing, Cham, pp. 129–168.
Madsen, O.B., Ravn, H.F., Rygaard, J.M., 1995. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. Ann. Oper. Res. 60 (1), 193–208.
Markovićc, N., Nair, R., Schonfeld, P., Miller-Hooks, E., Mohebbi, M., 2015. Optimizing dial-a-ride services in maryland: benefits of computerized routing and scheduling. Transp. Res. C Emerg. Technol. 55, 156–165.
Masmoudi, M.A., Braekers, K., Masmoudi, M., Dammak, A., 2017. A hybrid genetic algorithm for the heterogeneous dial-a-ride problem. Comput. Oper. Res. 81, 1–13.
Masmoudi, M.A., Hosny, M., Braekers, K., Dammak, A., 2016. Three effective metaheuristics to solve the multi-depot multi-trip heterogeneous dial-a-ride problem. Transp. Res. E Logist. Transp. Rev. 96, 60–80.
Masmoudi, M.A., Hosny, M., Demir, E., Genikomsakis, K.N., Cheikhrouhou, N., 2018. The dial-a-ride problem with electric vehicles and battery swapping stations. Transp. Res. E Logist. Transp. Rev. 118, 392–420.
Masmoudi, M.A., Hosny, M., Demir, E., Pesch, E., 2020. Hybrid adaptive large neighborhood search algorithm for the mixed fleet heterogeneous dial-a-ride problem. J. Heurist. 26 (1), 83–118.
Molenbruch, Y., Braekers, K., Caris, A., 2017. Benefits of horizontal cooperation in dial-a-ride services. Transp. Res. E Logist. Transp. Rev. 107, 97–119.
Molenbruch, Y., Braekers, K., Caris, A., 2017. Typology and literature review for dial-a-ride problems. Ann. Oper. Res. 259 (1–2), 295–325.
Muelas, S., LaTorre, A., Peña, J.-M., 2013. A variable neighborhood search algorithm for the optimization of a dial-a-ride problem in a large city. Expert Syst. Appl. 40 (14), 5516–5531.
Parragh, S.N., 2011. Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. Transp. Res. C Emerg. Technol. 19 (5), 912–930.
Parragh, S.N., Doerner, K.F., Hartl, R.F., 2010. Variable neighborhood search for the dial-a-ride problem. Comput. Oper. Res. 37 (6), 1129–1138.
Parragh, S.N., Schmid, V., 2013. Hybrid column generation and large neighborhood search for the dial-a-ride problem. Comput. Oper. Res. 40 (1), 490–497.
Psaraftis, H.N., 1980. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. Transp. Sci. 14 (2), 130–154.
Psaraftis, H.N., 1983. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. Transp. Sci. 17 (3), 351–357.
Reinhardt, L.B., Clausen, T., Pisinger, D., 2013. Synchronized dial-a-ride transportation of disabled passengers at airports. Eur. J. Oper. Res. 225 (1), 106–117.
Ropke, S., Cordeau, J.-F., Laporte, G., 2007. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks Int. J. 49 (4), 258–272.
Subramanian, A., Drummond, L.M. d. A., Bentes, C., Ochi, L.S., Farias, R., 2010. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. Comput. Oper. Res. 37 (11), 1899–1911.
Subramanian, A., Uchoa, E., Ochi, L.S., 2013. A hybrid algorithm for a class of vehicle routing problems. Comput. Oper. Res. 40 (10), 2519–2531.
Toth, P., Vigo, D., 1996. Fast Local Search Algorithms for the Handicapped Persons Transportation Problem. Springer US, Boston, MA, pp. 677–690.
Toth, P., Vigo, D., 1997. Heuristic algorithms for the handicapped persons transportation problem. Transp. Sci. 31 (1), 60–71.
Vidal, T., Crainic, T.G., Gendreau, M., Prins, C., 2013. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Comput. Oper. Res. 40 (1), 475–489.
Wilson, N., Weissberg, H., 1976. Advanced dial-a-ride algorithms research project final report. Tech. Rep. R, 20–76.
Wilson, N.H., Sussman, J.M., Wong, H.-K., Higonnet, T., 1971. Scheduling algorithms for a dial-a-ride system. Massachusetts Institute of Technology. Urban Systems Laboratory.
Zachariadis, E.E., Kiranoudis, C.T., 2010. A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. Comput. Oper. Res. 37 (12), 2089–2105.