The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity

Oscar Tellez, Samuel Vercraene, Fabien Lehuédé, Olivier Péton, Thibaud Monteiro

The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity

Oscar Tellez[1], Samuel Vercraene[1], Fabien Lehuédé[2], Olivier Péton[2], Thibaud Monteiro[1]

[1] *Laboratoire DISP, INSA de Lyon, Bât Léonard de Vinci, 21 avenue Jean Capelle, 69621 Villeurbanne, France*
[2] *IMT Atlantique, 4 rue Alfred Kastler, F-44307 Nantes Cedex, France*
Laboratoire des Sciences du Numérique de Nantes (LS2N, UMR CNRS 6004)

Abstract

This paper introduces a fleet size and mix dial-a-ride problem with multiple passenger types and a heterogeneous fleet of reconfigurable vehicles. In this new variant of the dial-a-ride problem, en-route modifications of the vehicle's inner configuration are allowed. The main consequence is that the vehicle capacity is defined by a set of configurations and the choice of vehicle configuration is associated with binary decision variables.

The problem is modeled as a mixed-integer program derived from the model of the heterogeneous dial-a-ride problem. Vehicle reconfiguration is a lever to efficiently reduce transportation costs, but the number of passengers and vehicle fleet setting make this problem intractable for exact solution methods. A large neighborhood search metaheuristic combined with a set covering component with a reactive mechanism to automatically adjust its parameters is therefore proposed. The resulting framework is evaluated against benchmarks from the literature, used for similar routing problems. It is also applied to a real case, in the context of the transportation of disabled children from their home to medical centers in the city of Lyon, France.

Keywords: Dial-A-Ride Problem, Fleet Size and Mix Problem, reconfigurable vehicles, large neighborhood search, set-covering, feasibility check.

1. Introduction

Solving the Dial-A-Ride Problem (DARP) consists in designing vehicle routes in order to fulfil transportation requests scattered throughout a geographic area. The objective is to minimize costs while satisfying transportation requests. The DARP is a special case of the Pickup and Delivery Problem (PDP) in which constraints and objectives related to the quality of services offered to passengers are integrated (the reader may refer to Parragh et al. (2008) for a recent survey on PDP). The main applications concern door-to-door transportation of people, particularly elderly or disabled people.

In this paper, we propose a generalization of the heterogeneous PDP with configurable vehicle capacity introduced by Qu and Bard (2013). This problem includes both heterogeneous fleets of vehicles and heterogeneous users. We consider several types of users who do not occupy the same space in the vehicle (e.g. passengers using seats or wheelchairs). The vehicle capacity depends on the chosen *configuration*. A *configuration* is characterized by a multidimensional capacity vector indicating the maximum number of users of each type allowed in the vehicle. Figure 1 presents three configurations of a vehicle with two types of users (passengers using seats or wheelchairs). The first configuration has a capacity of 1 wheelchair and 7 seats, including 5 folding seats (the driver's seat is not available for passengers). In the second configuration, one folding seat has been lifted and replaced by a wheelchair space. This configuration has a capacity of 2 wheelchairs and 6 seats. Adding one more wheelchair requires two additional seats to be folded. This is represented by the third configuration, which has a capacity of 3 wheelchairs and 4 seats. Note that there is no linear correspondence between the number of seats and wheelchairs. Moreover, the total number of passengers can be either 7 or 8, depending on the chosen configuration.

Qu and Bard (2013) integrate the choice of the initial vehicle configurations as a decision variable in a PDP dedicated to the transport of people with reduced mobility. In such services, it is actually possible to switch en-route from one configuration to another to serve additional transportation requests. As a result, we propose a generalization to the problem presented by Qu and Bard (2013), by assuming the possibility of en-route reconfiguration. This extension is called the DARP with Re-Configurable vehicle capacity (DARP-RC). The present study considers an arbitrarily large fleet of heterogeneous vehicles that has to be set: the problem addressed here falls

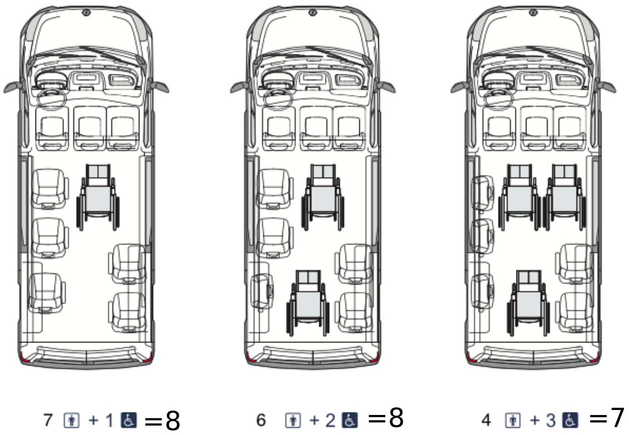

Figure 1: Vehicle with three configurations [http://handynamic.fr]

within the category of fleet size and mix problems. We therefore call it the Fleet Size and Mix DARP with reconfigurable vehicle capacity (FSM-DARP-RC).

This work is motivated by the daily transport of people with disabilities in the city of Lyon operated by the GIHP Company[1]. Every day, GIHP transports around 500 children from and to Medico-Social Institutions (MSI). GIHP works for 60 MSIs with about 180 adapted vehicles. From Mondays to Fridays, users are picked up at home in the morning and driven to their MSI. In the afternoon, they are transported back home. For MSIs, transportation is often considered the second-biggest expense after wages. Optimizing transport is therefore a priority. In this paper, without loss of generality, we present the case of morning trips. Every year, the company defines the fleet mix as well as the vehicles' itineraries. In addition, routing planners re-assess decisions daily, if required. All decisions are currently made without vehicle routing software. In order to design routes, route planners have to use simplifying assumptions, e.g. considering each MSI separately and ignoring the possibility of en-route vehicle reconfiguration. These assumptions lead to an over-constrained problem and consequently, to sub-optimal solutions. One aim of this paper is to solve the problem of having to stop at several MSIs on the same route. Using reconfigurable vehicles furthermore increases the possibility to pool routes between MSIs. Another aim of this paper is to

[1] www.gihp-sa.com

assess the impact of en-route reconfiguration.

The paper is organized as follows: Section 2 reviews related problems. Section 3 presents the problem settings and the mathematical formulation for the FSM-DARP-RC. Section 4 is dedicated to the solution method. In particular, we detail the Large Neighborhood Search (LNS) operators and the set covering component. The capacity check and the minimization of the number of reconfigurations are detailed in Section 5. We report extensive computational results and management insights in Section 6. Finally, Section 7 gives the conclusions and prospects.

2. Literature review

In this section, we first review some related work on DARP. We then present papers regarding the design and the routing of a fleet of heterogeneous vehicles with a special focus on DARP applications. Finally, we discuss how configuration and reconfiguration have been considered in the Vehicle Routing Problem (VRP) literature.

2.1. The dial-a-ride problem

The DARP is related to the optimization of a *multi-occupancy, door-to-door transport service for people* (Doerner and Salazar-Gonzàlez, 2014). Most commonly, this problem consists in designing the routes of a homogeneous fleet of vehicles to satisfy a set of transportation requests. The objective is to minimize the sum of route costs, satisfying constraints on vehicle capacity, time windows at origins and destinations of requests, and limits on the ride time of each passenger. Due to the combinatorial nature of the problem, exact methods (see for instance Ropke et al. (2007); Ropke and Cordeau (2009); Braekers et al. (2014)) are still limited in their ability to solve large size instances. Accordingly, several metaheuristics have been proposed to solve the DARP and its variants. In particular, Cordeau and Laporte (2003) developed a Tabu Search algorithm and proposed a set of instances that are still used to benchmark new algorithms. The most successful heuristics that have been applied to these instances in the last ten years are the Variable Neighborhood Search (VNS) of Parragh et al. (2010), the Hybrid LNS (HLNS) of Parragh and Schmid (2013), the Adaptive LNS (ALNS) of Masson et al. (2014), the deterministic annealing algorithm of Braekers et al. (2014), the evolutionary local search of Chassaing et al. (2016) and the ALNS of Gschwind and Drexl (2016).

One of the most challenging parts of DARP heuristics is the scheduling algorithm. It determines the feasibility of routes with respect to temporal constraints and, in some case, parts of the objective function such as the routes' duration. Several efficient algorithms have thus been designed to satisfy the most common DARP constraints with the best possible complexity (see e.g. Hunsaker and Savelsbergh (2002); Tang et al. (2010); Firat and Woeginger (2011); Gschwind, Timo (2015)). The latest contribution in this area is Gschwind and Drexl (2016), which proposes an incremental algorithm to evaluate, in constant time, the feasibility of request insertions in routes. The *minimization of route duration* is considered in Cordeau and Laporte (2003). This algorithm has been used and extended in many papers such as Parragh et al. (2010) and Braekers et al. (2014).

2.2. Routing a heterogeneous fleet of vehicles

In the FSM-DARP-RC, a heterogeneous fleet of vehicles has to be constituted to transport several types of users. In the VRP context, the routing of a heterogeneous fleet of vehicles was recently surveyed in Koç et al. (2016). Problems of this class fall into two main variants: (i) the Fleet Size and Mix VRP (FSM-VRP) introduced by Golden et al. (1984), which considers an unlimited fleet of various vehicle types and fixed costs for using vehicles. Solving the problem amounts to determining the fleet of vehicles and designing routes simultaneously; (ii) the Heterogeneous VRP (HVRP) introduced by Taillard (1999), which considers a given fleet of various types of vehicles. In such problems, vehicle types can induce differences both in constraints (e.g. capacity) or objectives (e.g. distance related or fixed related costs).

Regarding people transportation problems, the combined transportation of seated and wheelchair passengers by a heterogeneous fleet of vehicles has been studied in Toth and Vigo (1997). This problem falls into the class of Heterogeneous Dial-A-Ride-Problems (HDARP) introduced later by Parragh (2011). In the latter paper, the authors propose a branch-and-cut algorithm which optimally solves instances with up to 40 requests and a hybrid method which combines branch-and-price and VNS. The HDARP was also thoroughly investigated in Braekers et al. (2014). The authors propose a branch-and-cut algorithm and a deterministic annealing metaheuristic to solve a multi-depot version of this problem. To our knowledge, variants of the FSM-DARP have not yet been covered by the literature.

2.3. Configurable vehicle capacity

Finally, the FSM-DARP-RC extends the notion of vehicle configuration, which implies having multiple capacity options for some vehicles. To our knowledge, this notion was first introduced by Qu and Bard (2013) who define the heterogeneous PDP with configurable vehicle capacity. The problem could actually be called a DARP since it integrates considerations related to passenger transportation such as ride time minimization. These authors consider a limited and heterogeneous fleet of configurable vehicles that serve transportation requests coming from a heterogeneous set of passengers. In addition to routing decisions, one configuration must be chosen for each vehicle. Hence, this problem differs from the FSM-DARP-RC since a vehicle is assumed to keep the same configurations for its entire route. The authors propose a multi-start ALNS. The assignment of configurations for a given set of routes is done in a feasibility check either by a heuristic or by solving a general assignment problem when the heuristic fails. A branch-and-price algorithm for this problem was also proposed in Qu and Bard (2015) for instances with up to 30 requests.

Choosing a vehicle configuration can be compared to determining compartment sizes in the flexible multi-compartment VRP described in Derigs et al. (2011). In this paper, compartment sizes are determined together with a set of routes for vehicles with flexible compartments. The goal is to deliver various types of goods, given that some goods cannot be transported in the same compartment. Compartments can be *continuously flexible* as presented in Koch et al. (2016); Derigs et al. (2011), or *discretely flexible* if compartment options are defined beforehand as in Henke et al. (2015). The difference with our problem is twofold: first, in the FSM-DARP-RC, the total capacity changes from one configuration to another, in a non-linear way, whereas in multi-compartment problems, the sum of compartment sizes is assumed to remain constant. Second, we consider a PDP. Hence, a vehicle's load can increase and decrease along its route and en-route reconfiguration may be advantageous. In a VRP, a single configuration is needed since all goods are delivered at once. To our knowledge, the PDP with flexible compartment sizes has not been studied.

2.4. Contributions with respect to the literature

Regarding the literature, the main contributions of this paper are the following. First, we introduce the FSM-DARP-RC and propose a mathematical model for this problem. Second, we propose a matheuristic framework that combines a LNS with a reactive set covering component to solve this problem. This matheuristic is proven to be competitive on benchmark instances for the DARP and the heterogeneous PDP with configurable vehicle capacity. This matheuristic integrates innovative components related to the use of reconfigurable vehicles: (i) the *vehicle selection procedure*, which efficiently determines the minimum cost reconfigurable vehicle that is feasible for a given route, and (ii) the *reconfigurable vehicle capacity test*, which determines if a given vehicle can perform a given route together with its minimum number of reconfigurations. Finally, we present some managerial insights based on our experiments on the real instances provided by the GIHP. These insances will be made available to the community.

3. Problem settings and mathematical model

In this section, we start by providing an example of en-route reconfiguration. Then in Section 3.2 the FSM-DARP-RC problem is formally described, and is modeled in Section 3.3.

3.1. Example

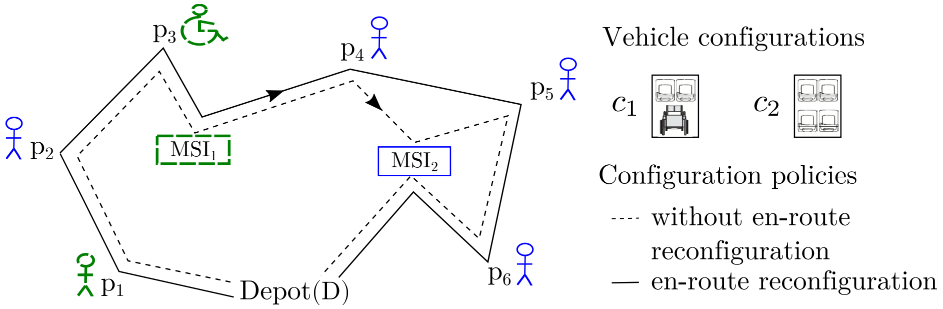

Figure 2: Comparison of routes with and without vehicle capacity reconfiguration

The example represented in Figure 2 illustrates the potential benefits of vehicle reconfiguration. We consider a single vehicle which starts and terminates its routes at depot D. This vehicle has two possible configurations, denoted as c_1 and c_2 respectively. Configuration c_1 consists of 2 seats and 1 wheelchair space. Configuration c_2 consists of 4 seats only. In this example, six users are transported to two MSIs. Users 1 and 3 are picked up at p_1 and p_3 respectively, and get off at MSI_1. Users 2, 4, 5 and 6 are picked up at p_2, p_4, p_5 and p_6 respectively, and get off at MSI_2. User 3 uses a wheelchair.

Without an en-route reconfiguration policy, the only solution is to leave the depot D with configuration c_1, which is the only one that includes a wheelchair space. The optimal solution for this policy is represented with dashed lines in Figure 2.

However, with en-route reconfiguration policy, the vehicle can be configured with configuration c_2 at the depot D and reconfigured with configuration c_1 after visiting MSI_1. This solution is represented with solid lines in Figure 2. This strategy reduces the length of the route by the value of the detour $\Delta_{p_4,MSI_2} + \Delta_{MSI_2,p_5} - \Delta_{p_4,p_5}$, where $\Delta_{i,j}$ represents the value of the shortest path from i to j.

3.2. Problem settings

The FSM-DARP-RC is modeled on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. The set $\mathcal{V}$ of nodes contains the set $\mathcal{O}^+$ of starting depots, the set $\mathcal{O}^-$ of arrival depots, the set $\mathcal{P}$ of pickup locations and the set $\mathcal{D}$ of delivery locations. We consider a large (potentially infinite) heterogeneous fleet of vehicles. We denote by $\mathcal{K}$ the set of vehicle types. Without loss of generality, we consider in this section that there is a single vehicle of each type, so that $\mathcal{K}$ also represents the set of vehicles. Each vehicle $k \in \mathcal{K}$ has a starting depot $o_k^+ \in \mathcal{O}^+$ and an ending depot $o_k^- \in \mathcal{O}^-$. In practice, o_k^+ and o_k^- often correspond to the same physical location. The set $\mathcal{A}$ contains three categories of arcs: Arcs (o_k^+, i) where $k \in \mathcal{K}, i \in \mathcal{P}$. Arcs (i, j) where $i, j \in \mathcal{P} \cup \mathcal{D}, i \neq j$. Arcs (i, o_k^-) where $k \in \mathcal{K}, i \in \mathcal{D}$. Every arc (i, j) represents the fastest path from nodes i to j. It is associated with a travel time t_{ij} and a distance Δ_{ij}.

The set of transport requests is denoted by $\mathcal{R}$. These requests concern passengers who may require various types of spaces in vehicles (e.g. seat, wheelchair). By extension, the type of space required by a passenger in a vehicle is called a user type and the set of user types is denoted by $\mathcal{U}$. Each request $r \in \mathcal{R}$ is characterized by a pickup node $p_r \in \mathcal{P}$, a delivery node $d_r \in \mathcal{D}$, a maximum ride time $\overline{T_r}$ and a quantity $q_{r,u}$ of users of each type to be transported from p_r to d_r. We define the following notations to represent the load variation at the pickup node and the delivery node of each request respectively: $\phi_{p_r,u} = q_{r,u}$ and $\phi_{d_r,u} = -q_{r,u}$.

Each node $i \in \mathcal{V}$ is associated with a service duration s_i and a time window $[a_i, b_i]$. At depots $o \in \mathcal{O}^+ \cup \mathcal{O}^-$, we assume that load variations are null for all user types, vehicles are empty and the service time s_o is null.

Each vehicle $k \in \mathcal{K}$ is associated with a fixed cost f^k, a cost per kilometer γ^k and a set $\mathcal{C}^k = \{1, \ldots, \bar{c^k}\}$ of possible configuration indexes. The set of configurations is defined by a set of capacity vectors $\mathcal{Q}^k = \{\mathbf{Q}^{k1}, \ldots, \mathbf{Q}^{k\bar{c}^k}\}$ with $\mathbf{Q}^{kc} = (Q_1^{kc}, \ldots, Q_{|\mathcal{U}|}^{kc})$, where Q_u^{kc} represents the maximal number of users of type $u \in U$ that can be carried by vehicle $k \in \mathcal{K}$ in configuration $c \in \mathcal{C}^k$.

Example 1. *Let us consider the vehicle $k \in \mathcal{K}$ represented in Figure 1. We have $\mathcal{C}_k = \{1, 2, 3\}$ and $\mathcal{Q}^k = \{\mathbf{Q}^{k1}, \mathbf{Q}^{k2}, \mathbf{Q}^{k3}\} = \{(4, 3), (6, 2), (7, 1)\}$.*

The sequence of nodes visited by a vehicle forms a *route*. A route is characterized by a vehicle $k \in \mathcal{K}$ (and its associated depots $o_k^+ \in \mathcal{O}^+$ and $o_k^- \in \mathcal{O}^-$) and a sequence of pickup and delivery nodes.

The maximal allowed route duration is denoted by $\overline{T}$. In this study, reconfigurable vehicles allow fast and easy reconfigurations. That is, for a given $k \in \mathcal{K}$, switching from a configuration $c \in \mathcal{C}^k$ to a configuration $c' \in \mathcal{C}^k$ can be done at no cost and within negligible time. However, to limit the inconvenience for the driver, related to this task, we introduce the parameter $\overline{R}$, which denotes the maximum number of reconfigurations allowed within a route. For each driving hour, there is a cost α related to the driver wages.

The FSM-DARP-RC consists in selecting a set of vehicles, designing their pickup and delivery routes, determining the time of service at each node and selecting the configuration used on each arc of the solution. The problem solutions must satisfy the vehicle capacity for all selected configurations, the maximum number of reconfigurations for each route, the time window constraint for each node and the maximum ride time constraint of each request.

3.3. Mathematical model

Let us introduce the following variables:

- x_{ij}^{kc} is a binary variable which is equal to 1 if vehicle $k \in \mathcal{K}$ uses arc $(i, j) \in \mathcal{A}$ with configuration $c \in \mathcal{C}_k$, and 0 otherwise,
- z_i^k is a binary variable which is equal to 1 if vehicle $k \in \mathcal{K}$ is reconfigured at node $i \in \mathcal{V}$, and 0 otherwise.
- $l_{i,u}^k$ is an integer variable representing the number of users of type $u \in \mathcal{U}$ in vehicle $k \in \mathcal{K}$ after visiting node $i \in \mathcal{V}$,
- w_i^k is a continuous variable representing the time of service of vehicle $k \in \mathcal{K}$ at node $i \in \mathcal{V}$.

The FSM-DARP-RC can then be formulated with the following mixed integer program:

$$\min \quad \sum_{k \in \mathcal{K}} f^k \sum_{i \in \mathcal{P}} \sum_{c \in \mathcal{C}^k} x_{o_k^+ i}^{kc} + \alpha \sum_{k \in \mathcal{K}} (w_{o_k^-}^k - w_{o_k^+}^k) + \sum_{k \in \mathcal{K}} \sum_{c \in \mathcal{C}^k} \sum_{(i,j) \in \mathcal{A}} \gamma^k \, \Delta_{ij} \, x_{ij}^{kc} \quad (1)$$

s.t.

$$\sum_{c\in\mathcal{C}^k}\sum_{(p_r,j)\in\mathcal{A}} x_{p_rj}^{kc} - \sum_{c\in\mathcal{C}^k}\sum_{(j,d_r)\in\mathcal{A}} x_{jd_r}^{kc} = 0 \qquad \forall r\in\mathcal{R}, k\in\mathcal{K} \tag{2}$$

$$\sum_{c\in\mathcal{C}^k}\sum_{k\in\mathcal{K}}\sum_{(j,p_r)\in\mathcal{A}} x_{jp_r}^{kc} = 1 \qquad \forall r\in\mathcal{R} \tag{3}$$

$$\sum_{c\in\mathcal{C}^k}\sum_{(j,i)\in\mathcal{A}} x_{ji}^{kc} - \sum_{c\in\mathcal{C}^k}\sum_{(i,j)\in\mathcal{A}} x_{ij}^{kc} = 0 \qquad \forall i\in\mathcal{P}\cup\mathcal{D}, k\in\mathcal{K} \tag{4}$$

$$\sum_{c\in\mathcal{C}^k}\sum_{i\in\mathcal{P}} x_{o_k^+i}^{kc} - \sum_{c\in\mathcal{C}^k}\sum_{i\in\mathcal{D}} x_{io_k^-}^{kc} = 0 \qquad \forall k\in\mathcal{K} \tag{5}$$

$$\sum_{c\in\mathcal{C}^k} x_{ij}^{kc} = 1 \Rightarrow \quad w_j^k \geq w_i^k + t_{ij} + s_i \qquad \forall (i,j)\in\mathcal{A}, k\in\mathcal{K} \tag{6}$$

$$a_i \leq w_i^k \leq b_i \qquad \forall i\in\mathcal{V}, k\in\mathcal{K} \tag{7}$$

$$w_{p_r}^k + s_{p_r} + t_{p_rd_r} \leq w_{d_r}^k \qquad \forall r\in\mathcal{R}, k\in\mathcal{K} \tag{8}$$

$$\sum_{c\in\mathcal{C}^k} x_{ij}^{kc} = 1 \Rightarrow l_{j,u}^k \geq l_{i,u}^k + \phi_{j,u} \qquad \forall (i,j)\in\mathcal{A}, u\in\mathcal{U}, k\in\mathcal{K} \tag{9}$$

$$l_{i,u}^k \leq \sum_{c\in\mathcal{C}^k}\sum_{(i,j)\in\mathcal{A}} Q_u^{kc} x_{ij}^{kc} \qquad \forall i\in\mathcal{P}, u\in\mathcal{U}, k\in\mathcal{K} \tag{10}$$

$$\sum_{c'\in\mathcal{C}^k/\{c\}}\sum_{(i,j)\in\mathcal{A}} x_{ji}^{c'k} + \sum_{(i,j)\in\mathcal{A}} x_{ij}^{ck} \leq 1 + z_i^k \qquad \forall i\in\mathcal{P}\cup\mathcal{D}, k\in\mathcal{K}, c\in\mathcal{C}^k \tag{11}$$

$$\sum_{i\in\mathcal{P}\cup\mathcal{D}} z_i^k \leq \overline{R} \qquad \forall k\in\mathcal{K} \tag{12}$$

$$w_{d_r}^k - w_{p_r}^k - s_{p_r} \leq \overline{T}_r \qquad \forall r\in\mathcal{R}, k\in\mathcal{K} \tag{13}$$

$$w_{o_k^-}^k - w_{o_k^+}^k \leq \overline{T} \qquad \forall k\in\mathcal{K} \tag{14}$$

$$x_{ij}^{kc} \in \{0,1\} \qquad \forall (i,j)\in\mathcal{A}, k\in\mathcal{K} \tag{15}$$

$$l_{i,u}^k \in \mathbb{Z}^+ \qquad \forall i\in\mathcal{V}, u\in\mathcal{U}, k\in\mathcal{K} \tag{16}$$

$$w_i^k \in \mathbb{R}^+ \qquad \forall i\in\mathcal{V}, k\in\mathcal{K} \tag{17}$$

$$z_i^k \in \{0,1\} \qquad \forall i\in\mathcal{P}\cup\mathcal{D}, k\in\mathcal{K}. \tag{18}$$

The objective function (1) to be minimized is the total transportation cost, including fixed costs associated with each selected vehicle, time-related costs proportional to the total route duration, and distance-related costs proportional to the total distance traveled. Constraints (2) and (3) ensure that all transportation requests are satisfied and carried by a single vehicle. Constraints (4) are flow conservation constraints at pickup and delivery nodes. Constraints (5) state that any vehicle $k \in \mathcal{K}$ leaving the node o_k^+ must end-up at the node o_k^-. Constraints (6) set arrival time variables. If arc $(i,j) \in \mathcal{A}$ is used by a vehicle k, then the arrival time at j is greater than the arrival time at i plus the service duration at i and the transportation time from i to j. These constraints can be linearized by using a value $M_{ij} = b_i + s_i + t_{ij}$. Constraints (6) can then be rewritten as

$$w_j^k \geq w_i^k + t_{ij} + s_i - M_{ij}(1 - \sum_{c \in \mathcal{C}^k} x_{ij}^{kc}) \qquad \forall (i,j) \in \mathcal{A}, k \in \mathcal{K}. \tag{19}$$

Constraints (7) set times windows for variables w_i^k. Constraints (8) state that the arrival at a delivery node cannot occur before the same vehicle has visited the associated pickup node and traveled between the pickup and the delivery node. Constraints (9) propagate the load on a vehicle along its route for each type of user. To linearize this constraint, we define the value $\bar{Q}_u^k$ as the maximum capacity of type $u \in \mathcal{U}$ among all configurations: $\bar{Q}_u^k = \max_{c \in \mathcal{C}^k}\{Q_u^{kc}\}$. The load propagation constraints (9) can then be expressed as follows:

$$l_{j,u}^k \geq l_{i,u}^k + \phi_{ju} - \bar{Q}_u^k(1 - \sum_{c \in \mathcal{C}^k} x_{ij}^{kc}) \qquad \forall (i,j) \in \mathcal{A}, u \in \mathcal{U}, k \in \mathcal{K}. \tag{20}$$

Constraints (10) are capacity constraints for each vehicle and each type of user. Note that, on delivery nodes, the number of users in a vehicle cannot increase. Thus, constraints (10) are defined at pickup nodes only. Constraints (11) and (12) concern the reconfiguration of vehicles. In constraints (11), the variable z_i^k is set at value 1 when the vehicle $k \in \mathcal{K}$ is reconfigured at node i. Constraints (12) limit the number of reconfigurations to be less than $\overline{R}$ for each vehicle. Constraints (13) limit the total ride time (including waiting and service times) of each user $i \in \mathcal{R}$ to be less than $\overline{T}_i$. Constraints (14) states that the duration of each route should be less than $\overline{T}$. Finally, the last constraints define the decision variables. Note that variables z_i^k do not need to be explicitly declared as binary variables and variables $l_{i,u}^k$ do not need to be declared as integer variables.

4. Solution method

In this section, we describe the solution method that we propose to solve the FSM-DARP-RC. It relies on a combination of two components: a Large Neighborhood Search (LNS) and the solution of a Set Covering Problem (SCP) using a layer with a Reactive adjustment of SCP parameters (denoted RSCP). The framework is denoted LNS-RSCP. This section is structured as follows: Section 4.1 presents the general framework of LNS-RSCP. LNS removal and repair operators are detailed in Section 4.2 and the RSCP is presented in Section 4.3.

4.1. Matheuristic framework (LNS-RSCP)

LNS was first proposed by Shaw (1998) in a constraint programming context and introduced under the name *ruin and recreate* in Schrimpf et al. (2000). In LNS, the current solution is improved by following an iterative process of destroying it (i.e. removing a part of it) and repairing it. This process is repeated until a stopping criterion is reached. In our case, the stopping criterion is either a maximum number of iterations or a maximum computational time.

The potential of LNS for solving a large variety of vehicle routing problems was revealed by Ropke and Pisinger who proposed an Adaptive version of LNS, known as ALNS, consisting of multiple search operators adaptively selected according to their past performance (see e.g. Ropke and Pisinger (2006) and Pisinger and Ropke (2007)). LNS was successfully applied to many variants of vehicle routing problems. In particular, it remains one of the best-known approaches to solve PDPs.

The RSCP component acts as a long-term memory. It is based on the idea that all iterations of the LNS, including the ones that do not yield good solutions, are likely to contain some good routes. Let $\Omega = \{1, \ldots, |\Omega|\}$ be a set of routes collected through LNS iterations, and let $\Pi(\omega) \in \mathbb{R}^+$ be the cost of route $\omega \in \Omega$. Route $\omega \in \Omega$ is said to *dominate* route $\omega' \in \Omega$ if both routes visit the same nodes and $\Pi(\omega) \leq \Pi(\omega')$. All non-dominated routes found by the LNS are stored in a pool of routes. The RSCP has the ability to put together good routes that were generated at distinct iterations of the LNS. It can be viewed as a route-based formulation of the FSM-DARP-RC built from the pool of routes. A SCP is solved with a MILP solver at regular intervals. The number of iterations between each SCP solution is adjusted in a reactive way, as described below. The solver is given a time limit to solve this problem.

The LNS-RSCP framework is described in Algorithm 1. The algorithm starts with an initial solution s, a set Σ^- of removal operators and a set of repair operators Σ^+. At every iteration, the proportion of requests Φ to be removed is determined randomly (following a uniform distribution) in the interval $[\Phi^-, \Phi^+]$ in line 6. A removal operator $\sigma^- \in \Sigma^-$ and a repair operator $\sigma^+ \in \Sigma^+$ are selected according to a discrete uniform distribution in line 7. The operator σ^- removes $\max\{1, \lfloor \Phi.|\mathcal{R}| \rfloor\}$ requests from the *current solution* s (line 8). These requests are placed in a set of unsatisfied requests called Request Bank ($\mathcal{B}$). From the request bank $\mathcal{B}$, they are reinserted into the partially destroyed solution using the repair operator σ^+ (line 9). If the solution cannot be completely repaired, its total cost is increased by the value *penalty* for each unsatisfied request. Once the solution s' is repaired, it can be either accepted or rejected as the next *current solution* using an acceptance criterion (line 10). We use the *record-to-record* acceptance criterion proposed by Dueck (1993): if $objective(s') \leq (1 + \chi).objective(s^*)$, then s' is accepted as the new *current solution* where χ is a small positive value. Similarly, the *best solution* s^* is updated every time a new *temporary solution* s' cheaper than s^* is found (lines 12-13). Finally, lines 14 to 21 describe the RSCP component of the algorithm. Every η iteration (line 15), SCP is solved with a MILP solver to find the best combination of routes generated during the previous iterations (line 14). A parameter t_{limit} is used to limit the time allocated to the solver. When the SCP cannot be solved to optimality within the time limit, the pool of routes Ω is cleared and reinitialized with the routes of the best-known solution s^* (line 18). The reactive layer of the RSCP readjusts parameter η every time the solver fails to optimally solve the SCP twice consecutively (line 19).

4.2. LNS operators

The sets of removal and repair operators are major components of the LNS method. Hence, many LNS operators have been developed. Pisinger and Ropke (2007) proposed a list of 7 destroy and 2 repair operators to solve a large variety of vehicle routing problems with a good performance. In practice, LNS operators should enable both intensification and diversification, but the exact list of implemented operators varies a lot from one implementation to another. Several papers (see e.g., Christiaens and Vanden Berghe (2016)) have shown that implementing only a few good operators is enough to get

Algorithm 1: The LNS-RSCP framework

```
Input: $s$: initial solution,
$\Sigma^{-}$: set of destroy operators,
$\Sigma^{+}$: set of repair operators,
$t_{limit}$: initial time limit for SCP,
$\eta$: initial number of iterations between two solutions of a SCP.
Output: Best solution found $s^{*}$.
Pool of routes: $\Omega \leftarrow \emptyset$
Request bank: $\mathcal{B} \leftarrow \emptyset$
$iter \leftarrow 0$
while Termination criterion not met do
                                                  /* LNS component */
    $s' \leftarrow s$
    Destroy quantity: Randomly select a proportion $\Phi$ of requests
      to remove
    Operator selection: Randomly select a destroy operator
      $\sigma^{-} \in \Sigma^{-}$ and a repair operator $\sigma^{+} \in \Sigma^{+}$
    Destroy: Apply $\sigma^{-}$ to remove $\max\{1, \lfloor \Phi . |\mathcal{R}| \rfloor\}$ requests from $s'$
      and place them into the request bank $\mathcal{B}$
    Repair: Apply $\sigma^{+}$ to reinsert requests from $\mathcal{B}$ into $s'$
    if Acceptance criterion is met then
        $s \leftarrow s'$
    if Cost of $s'$ is better than the cost of $s^{\star}$ then
        $s^{*} \leftarrow s'$
                                                  /* RSCP component */
    Update $\Omega$ with the non-dominated routes of $s'$
    if $iter$ modulo $\eta = 0$ (every $\eta$ iterations) then
        $s \leftarrow$ Solve SCP$(\Omega, s^{*}, t_{limit})$
        if the SCP is not solved to proven optimality then
            Reinitialize $\Omega$ with the routes of $s^{*}$
            Every two consecutive failures to optimally solve the SCP:
              update parameter $\eta$
        Remove duplicate requests in $s$
        $s^{*} \leftarrow s$
    $iter \leftarrow iter + 1$
return $s^{*}$
```

very good solutions. Additional operators may slightly improve the quality of the solutions. Following this principle, our implementation of LNS uses only a few operators in order to keep the framework as simple as possible, without sacrificing the solution quality.

Removal operators

These operators determine the set of requests to be removed from the current solution according to a given criterion. The following operators have been used:

- *Random removal:* This operator removes a proportion Φ of randomly selected requests from the current solution.
- *Historical node-pair removal:* This operator consists in removing a proportion Φ of requests from the current solution that were "better placed" in previous solutions. The LNS algorithm calculates a score for each arc $(i, j) \in \mathcal{A}$. The initial score of each arc is set to infinity. It is then updated at each iteration with the value of the best solution found so far that includes arc (i, j). The removal heuristic calculates the cost of a request $r \in \mathcal{R}$ from pickup node i to delivery node j by summing the score of the arcs that are incident to i and j in the current solution. It then removes requests according to a probabilistic rule used in Ropke and Pisinger (2006): Consider the list $\mathcal{R}$ of requests sorted by non-increasing costs. The operator iteratively removes the request at position $\xi^p \times |R|$, where $0 \leq \xi < 1$ is a random number and $p \in \mathbb{Z}$ is a deterministic parameter. This probabilistic choice gives higher probability to the requests with the highest cost to be removed.

We also implemented the following operators, that were not kept in the final version of the LNS: *worst removal*, which removes the request responsible for the longest detour in the solution; *distance-related removal* which removes the nearest nodes, based on a distance-related indicator; and *time-related removal*, which removes requests that are similar from a time point of view. For a full description of these operators, we refer to Pisinger and Ropke (2007).

Repair operators

Removed requests are stored in the request bank $\mathcal{B}$. Repair operators are intended to take them out of the request bank and to re-insert them into the current partial solution. . We implemented the two most common repair operators: *best insertion* and *k-regret insertion.*

- *Best insertion*: At each iteration, the best insertion position is calculated for each request $r \in \mathcal{B}$, in each route of the current solution. The request with the minimal insertion cost is then inserted at its best position. This process is repeated until the request bank is empty or no more feasible insertion exists.

- *K-regret*: This operator generalizes the best insertion with more looking-ahead information. At each iteration, the best insertion position of each request $r \in \mathcal{B}$ is calculated for each route of the current solution. Let Δf_r^j designate the insertion cost of a request $r \in \mathcal{R}$ in its j^{th} best route at its best position. Δf_r^1 denotes the insertion cost (min additional cost) of inserting request $r \in \mathcal{R}$ in its best route, Δf_r^2 is the insertion cost for the same request in its 2^{nd} best route, etc. The request $r^\star$ selected for insertion at its best position is: $r^\star = \arg\max_{r \in B} \left(\sum_{j=2}^{k} \Delta f_r^j - \Delta f_r^1 \right)$. The heuristic stops when the request bank is empty or when no more requests can be inserted. In this paper, we consider K-regret heuristics with values of K between 2 and 4 and regard them as 3 independent operators.

4.3. Set covering problem (SCP)

The use of heuristic approaches for generating routes to solve a set covering or set partitioning formulations of the VRP was first proposed by Foster and Ryan (1976): several routes, called *petals* are first generated and a set partitioning problem is then solved to build a VRP solution. This approach was then extended to multiple applications. In particular, it has been combined with local search methods in Rochat and Taillard (1995) and Subramanian et al. (2013) for the VRP, and has been combined with LNS-based heuristics in Parragh and Schmid (2013) and Gschwind and Drexl (2016) for the DARP. Note that, whereas Gschwind and Drexl (2016) solve a single SCP at the very end of their algorithm, the matheuristic of Parragh and Schmid (2013) uses reduced costs from the SCP solution to guide a variable neighborhood search that can possibly generate new routes. In the survey of Archetti and Speranza (2014) for VRP matheuristics, this approach is classified under the name of *column generation based heuristics.* The name is naturally chosen because of the similarity with set partitioning formulations in branch-and-price algorithms. More precisely, this method falls into the class of *restricted master heuristics* according to the same survey.

The proposed approach is based on the framework of Grangier et al. (2017) for a VRP with cross-docking. According to this framework, a solver is called every η iterations to solve the SCP on a pool Ω containing all non-dominated routes found by LNS in the previous iterations. When the number of routes in Ω makes the SCP intractable for the solver within a given time limit t_{limit}, Ω is cleared and reinitialized with the routes of the best solution found so far.

In the following sections, we detail the mathematical formulation of the SCP and the pool management process. We also develop a reactive version denoted RSCP where parameter η is automatically adjusted from one SCP solving to another.

Formulation of the SCP

Note that $\Omega = \{1, \dots, |\Omega|\}$ is the set of routes collected through LNS iterations, and $\pi_\omega \in \mathbb{R}^+$ denotes the cost of route $\omega \in \Omega$. Let $\rho_{r\omega}$ be a parameter equal to 1 if request $r \in \mathcal{R}$ is served by route $\omega \in \Omega$, and 0 otherwise.

The SCP aims at building a new solution by selecting a subset of independent routes in Ω. The SCP model uses binary variables y_ω, that are set at value 1 if route $\omega \in \Omega$ is part of the solution and 0 otherwise. The SCP can be defined by the following integer linear program.

$$
\begin{aligned}
&\min \sum_{\omega \in \Omega} \pi_\omega \; y_\omega && && (21)\\
&\text{s.t.} && && \\
&\sum_{\omega \in \Omega} \rho_{r\omega} y_\omega \geq 1 && \forall r \in \mathcal{R} && (22)\\
&y_t \in \{0, 1\} && \forall \omega \in \Omega. && (23)
\end{aligned}
$$

SCP solving

As introduced before, the SCP is solved by a MILP solver with a run time limited to t_{limit}. To improve the solver performance and guarantee that a feasible solution is returned, the solver is initialized with the current routes of s^* using a so-called *warm-start* function.

Given that constraints (22) of the SCP model allow request duplication, a request may appear in more than one route in the optimal solution of the SCP. In this case the request is left only in the route that yields the best cost (line 20, Algorithm 1).

Management of the pool of routes Ω

At each iteration of the LNS-SCP, the non-dominated routes in the current solution are added to the pool of routes Ω (line 14, Algorithm 1), possibly dominating some routes in the pool. Hence, the size of the pool generally increases after each iteration. Eventually, the pool becomes so large that the MILP solver cannot optimally solve the SCP within the time t_{limit}. When this happens, the pool Ω is cleared and reinitialized with the routes of s^*. This strategy used in Grangier et al. (2017) allows for the route pool to be maintained at a reasonable size. Of course, this strategy heavily depends on the solving time left to the MILP solver.

Reactive adjustment of SCP parameters (RSCP)

The combination of the run time t_{limit} and the frequency η determine the overall computational effort spent in solving the SCP. Thus, one issue in tuning the SCP component is to define a common policy to set the values of parameters t_{limit} and η in all instances.

In our numerical experiments we observe that, for a given value of η, the run time necessary to optimally solve the SCP is considerably longer on larger instances. Moreover, this time is longer at the beginning of the LNS execution than at the end. This has already been observed by Subramanian et al. (2013) in a VRP context. The authors developed a reactive strategy to limit the number of routes in the SCP. Their approach requires setting 5 parameters. Moreover, a threshold mechanism eliminates bad solutions even if they contain good routes. Thus, we propose a reactive layer RSCP that allows a simpler and automatic adjustment of parameter η, reducing its value geometrically such that $\eta \leftarrow \eta/\psi$ (Algorithm 1, line 19). This mechanism uses a single real parameter $\psi > 1$ and assures that every dominant route will be considered at least once in a SCP solving. Unlike the re-initialization of Ω which is done when the solver cannot solve the SCP optimally, the automatic adjustment of η is performed only when the solver cannot solve the SCP optimally twice consecutively.

5. Evaluation of the insertion of requests

In each iteration of the framework LNS-SCP (Algorithm 1), the repair operator checks the potential insertions of all unplanned requests in all routes at all positions. This represents a large number of insertion attempts. In practice, for every unplanned request, only the best insertion is performed. Repair operators therefore evaluate the feasibility and the performance of a very large number of unnecessary insertions, which has a major impact on the performance of the SCP-LNS algorithm. This section describes the core algorithms involved in the insertion of unplanned requests.

Let ω denote a feasible route with N nodes. Without loss of generality, let us denote by $1, ..., N$ the set of nodes visited by this route, where nodes 1 and N are the initial and the final depot of this route. Inserting request $r \in \mathcal{R}$ consists in inserting pickup node $p_r \in \mathcal{P}$ between nodes $i \in \omega$ and $i+1 \in \omega$ and inserting delivery node $d_r \in \mathcal{D}$ between nodes $j \in \omega$ and $j+1 \in \omega$. We call ω' the route resulting from this request insertion, as represented in Figure 3

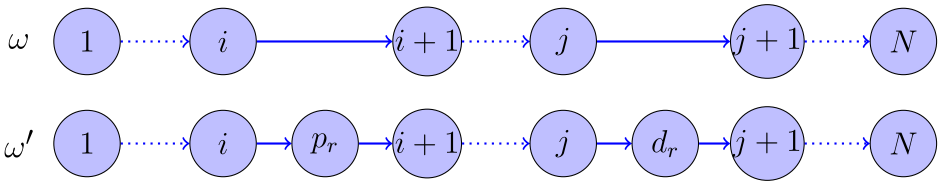

Figure 3: Insertion of request r into route ω

The feasibility evaluation of ω' has two parts: The *Capacity evaluation* checks that at least one vehicle type can serve all requests in ω' with at most $\bar{R}$ reconfigurations. The *Schedule evaluation* computes the minimal route duration and checks if there is a departure time for route ω' that complies with all users time windows and maximum ride times.

Let us denote by $time(\omega')$ the minimal duration of route ω', $\mathcal{K}(\omega')$ the set of vehicle types compatible with route ω', $k(\omega') \in \mathcal{K}(\omega')$ the cheapest vehicle type compatible with route ω', and $dist(\omega')$ the total length of route ω'. Note that $dist(\omega')$ can be computed in constant time with an incremental approach:

$$dist(\omega') = dist(\omega) + \begin{cases} \Delta_{i,p_r} + \Delta_{p_r,i+1} - \Delta_{i,i+1} + \Delta_{j,d_r} + \Delta_{d_r,j+1} - \Delta_{j,j+1} & \text{if } i < j, \\ \Delta_{i,p_r} + \Delta_{p_r,d_r} + \Delta_{d_r,i+1} - \Delta_{i,i+1} & \text{if } i = j. \end{cases}$$

Remember that α and γ^k represent the unitary costs related to the route duration and length respectively. Then, if route ω' is feasible regarding the *Capacity evaluation* and the *Schedule evaluation*, the minimal cost $\Pi(\omega')$ of route ω' is determined by

$$\begin{aligned} \Pi(\omega') &= \alpha\, time(\omega') + \min_{k \in \mathcal{K}(\omega')} \left(f^k + \gamma^k dist(\omega') \right), \\ &= \alpha\, time(\omega') + f^{k(\omega')} + \gamma^{k(\omega')} dist(\omega'). \end{aligned} \tag{24}$$

Note that in our application, all vehicles travel at the same speed. Hence, the time feasibility and the duration of a route can be evaluated without knowing the vehicle that travels that route.

Algorithm 2: Evaluation of a route ω'

```
   Input:  Routes $\omega$ and $\omega'$
   Output:  Total insertion cost (-1 if unfeasible)
1  $time(\omega') \leftarrow ScheduleEvaluation(\omega')$        /* Algorithm 3 */ ;
2  if $time(\omega') > -1$ then
3      $k(\omega') \leftarrow CapacityEvaluation(\omega, \omega')$      /* Algorithm 4 */ ;
4      if $k(\omega') > -1$ then
5          $\Pi(\omega') = \alpha\, time(\omega') + f^{k(\omega')} + \gamma^{k(\omega')} dist(\omega')$;
6          return $\Pi(\omega')$
7  return $-1$
```

Algorithm 2 shows how the *Schedule evaluation* and the *Capacity evaluation* algorithms are organized. Since these algorithms are mutually independent, they can be executed in any order. Although this order may impact computational times, there is no general rule to define which one should be placed first because the performance always depends on the considered data. *Schedule evaluation* (line 3) is performed first because our instances are mostly constrained by time windows and ride times. Accordingly, the *Capacity evaluation* (line 4) is run only on the resulting feasible routes. The algorithm returns the cost of an insertion, and the value -1 if the insertion is not possible.

Sections 5.1 and 5.2 detail the *Schedule evaluation* and the Capacity evaluation respectively.

5.1. Schedule evaluation

The scheduling of a route first consists in determining if the user time windows and maximum ride time are respected on the route. Second, it determines a minimal duration schedule. The *Schedule evaluation* is the most time-consuming operation in the algorithm due to the large number of evaluations performed each time a request insertion is evaluated.

The route scheduling for our problem is performed by Algorithm 3. It is based on Tang et al. (2010) as presented in Gschwind, Timo (2015). We add forward time slack calculations from Cordeau and Laporte (2003) in order to determine the minimum route duration. Note that some simple classical and necessary conditions are checked before running this algorithm. These are well summarized in Braekers et al. (2014).

5.2. Capacity evaluation

Let us consider a route $\omega = \{1, ..., N\}$. For each node $i \in \omega$, Algorithm 3 determines the beginning of service w_i in such a way that route duration is minimized. This algorithm has three phases. In the first phase, the earliest possible schedule is computed for every visited node (lines 5 to 9). At the same time, we calculate the forward time slack of node 1, denoted by F_1, and the total waiting time H on the route. The second phase minimizes the route duration by shifting the first node w_1 by F_1 units. Consequently, the beginning of service in all other nodes $i = 2, \ldots, N$ is updated (lines 11 to 13). Once the route duration is minimized, the maximum route duration constraint is verified in line 14.

The third phase checks ride-time constraints for every request (lines 16 to 27), starting from the end of the route. For any pickup node i, we denote as $r \in \mathcal{R}$ the request to which it belongs. Consequently, d_r is the corresponding delivery node and i is equal to p_r. The ride-time associated with this request is then $w_{d_r} - w_{p_r} + s_r$. If the value $(w_{d_r} - w_{p_r} + s_r) - \overline{T}_i$ is positive, then the max ride-time constraint associated with pickup p_r is violated. In this case, the beginning of service w_{p_r} is shifted by the value of this violation (line 21). This may lead to unfeasibility on time windows, which is checked in lines 22 to 25. Finally, since the beginning of service w_{p_r} may have been changed, the ride-time constraint for request r is checked again in line 26. If no unfeasibility is detected, the minimal route duration $w_N - w_1$ is returned (line 29).

The objective of this section is to complete the evaluation of a route by determining the cheapest vehicle type that satisfies capacity constraints as well

Algorithm 3: Schedule evaluation

```
Input:  Route $\omega = \{1, ..., N\}$.
Output: Minimal duration of route $\omega$ (-1 if unfeasible)
$w_1 \leftarrow a_1$                            /* Beginning of the service */
$H \leftarrow 0$                         /* Total waiting time on the route */
$F_1 \leftarrow b_1 - w_1$                   /* Forward time slack at node 1  */

/* Phase 1:  set up nodes at the earliest start          */
for  $i = 2, \ldots, N$ do
    $w_i \leftarrow \max\{a_i; w_{i-1} + s_{i-1} + t_{i-1,i}\}$
    if $w_i > b_i$ then return -1
    $H \leftarrow H + \max\{0; a_i - (w_{i-1} + t_{i-1,i} + s_{i-1})\}$
    $F_1 \leftarrow \min\{F_1; H + \max\{0; b_i - w_i\}\}$

/* Phase 2:  optimize route duration                     */
$w_1 \leftarrow w_1 + F_1$
for  $i = 2, \ldots, N$ do
    $w_i \leftarrow \max\{w_{i-1} + s_{i-1} + t_{i-1,i}; a_i\}$
/* Check route duration constraint                       */
if $(w_N - w_1) > \overline{T}$ then return -1

/* Phase 3:  check ride time constraints                 */
for  $i = N - 2, \ldots, 1$ do
    if  $i \in \mathcal{P}$ then
        $r \leftarrow$ request of pickup $i$            /* Implies $i = p_r$ */
        $\delta \leftarrow (w_{d_r} - w_{p_r} + s_r) - \overline{T}_r$
        if $(\delta > 0)$ then
            $w_{p_r} \leftarrow w_{p_r} + \delta$
            if $w_{p_r} > b_{p_r}$  then  return -1
            for $j = p_r + 1, \ldots, N$ do
                $w_j \leftarrow \max\{a_j; w_{j-1} + s_{j-1} + t_{j-1,k}\}$
                if $w_j > b_j$ then  return -1
            if $\overline{T}_r - (w_{d_r} - w_{p_r} + s_r) < 0$ then
                return -1

return $w_N - w_1$
```

as the constraint on the maximal number of reconfigurations. Section 5.2.1 presents the procedure that assigns a vehicle to a given route. Section 5.2.2 details the subroutine that checks if a given reconfigurable vehicle is feasible for a given route.

5.2.1. Vehicle type selection

We consider the resulting route ω' after the insertion of a request $r \in \mathcal{R}$ in route ω. A first observation is that the set of vehicle types that can perform route ω' is included in the set of vehicles that can perform route ω: $\mathcal{K}(\omega') \subset \mathcal{K}(\omega)$. Thus, the *Capacity evaluation* of ω' can be limited to the vehicle types in $\mathcal{K}(\omega)$.

A second observation is that it is sufficient to check the capacity only over a smaller subset of key nodes of the route. This leads to the following definitions:

Definition 1. *We call the load profile of a route the list of vectors representing its load at every node of this route. The load profile for route ω' is $\{\mathbf{l}_1, \dots, \mathbf{l}_N\}$, where $\mathbf{l}_i = \{l_{i,u}, i \in 1, \dots, N; u \in \mathcal{U}\}$, and $l_{i,u}$ represents the load of user type $u \in \mathcal{U}$ at node $i \in \omega'$.*

Load $l_{i,u}$ results from the accumulation of load variations of the route from node 1 to node i, i.e. $l_{i,u} = \sum_{j=1}^{i} \phi_{j,u}$.

Definition 2. *The set of pickup nodes of a route that are immediately followed by a delivery node is called the kernel of this route. The restriction of the load profile $\{\mathbf{l}_1, \dots, \mathbf{l}_N\}$ to the nodes of its kernel is denoted by $\mathcal{L}$.*

Lemma 1. *A vehicle type $k \in \mathcal{K}$ can be assigned to a route ω' if, and only if*

- *each load $\mathbf{l} \in \mathcal{L}$ is compatible with at least one configuration of vehicle type k*
- *the number of reconfigurations required to carry every load $\mathbf{l} \in \mathcal{L}$ is less than or equal to the maximum number of allowed reconfigurations $\overline{R}$.*

Proof. If i is the delivery node of some request $r \in \mathcal{R}$. Then, $l_{i,u} \leq l_{i-1,u}$ for all user types $u \in \mathcal{U}$. Thus, if the capacity is satisfied at node $i-1$, it is also satisfied at node i.

If i is a pickup node followed by another pickup node, then $l_{i,u} \leq l_{i+1,u}$ and the capacity at i is satisfied if it is satisfied at $i+1$.

The second condition is straightforward. □

A corollary of Lemma 1 is that, in order to save computation time, the capacity evaluation of route ω' can be restricted to its kernel.

Figure 4 presents the successive loads of a vehicle on a route with two types of users (seats and wheelchairs). In Figure 4, pickup operations are illustrated by arcs pointing to the right or to the top and deliveries are represented by arcs pointing to the left or to the bottom. Although the route has 7 nodes, a sufficient *Capacity evaluation* of the route requires only a capacity check in the nodes of the kernel, this is nodes 3 and 5.

$$\mathcal{L} = \left\{ \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 3 \end{pmatrix} \right\}$$

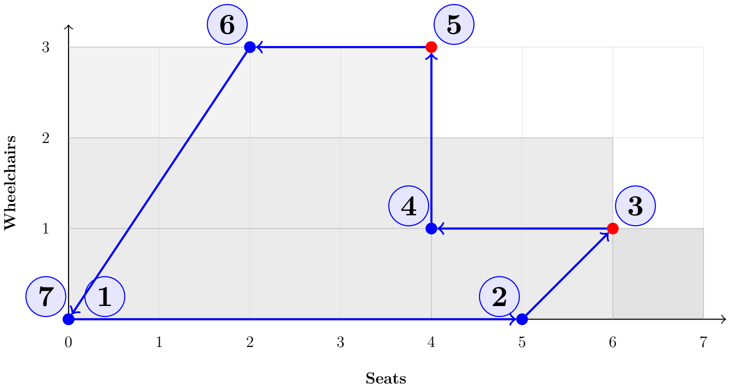

Figure 4: Graphical representation of a route load

Algorithm 4 describes the *Capacity evaluation* procedure. It first evaluates the fixed and distance-related costs for all vehicle types (lines 2–3). The route load profile is computed in lines 5–10. In line 10, a vertex are appended to the route kernel if it is a pickup, immediately followed by a delivery vertex. Checking if a vehicle type k is feasible for $\mathcal{L}$ is done on line 13. Considering that k can be reconfigured and that several types of users are considered, this evaluation is non-trivial and is detailed in Section 5.2.2.

Algorithm 4: Capacity evaluation

```
Input: Routes $\omega$ and $\omega'$, list $\mathcal{K}(\omega)$ of vehicle types compatible with $\omega$,
       distance $dist(\omega')$
Output: Cheapest vehicle type for the route $\omega'$ ($-1$ if unfeasible)

/* Estimate partial route costs                                    */
forall $k \in \mathcal{K}(\omega)$ do
    $\lambda^k = f^k + dist(\omega')v^k$

/* Compute the load profile for the kernel of the route           */
$\mathcal{L} = \emptyset$
$\mathbf{l}_1 = \mathbf{0}$
for $i \leftarrow 2, \dots, N$ do
    $\mathbf{l}_i = \mathbf{l}_{i-1} + (\phi_{j,u})_{u \in \mathcal{U}}$
    if $i$ belongs to the kernel of $\omega'$ then
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathbf{l}_i\}$

/* Find a feasible vehicle type                                    */
for $k \leftarrow \mathcal{K}(\omega)$ in non-decreasing order of costs $\lambda^k$ do
    if $k$ can carry $\mathcal{L}$ then                   /* Algorithm 5 */
        return $k$
return $-1$
```

Proposition 1. *The Capacity evaluation algorithm returns $k(\omega')$, the cheapest vehicle type for the route ω', or the value -1 if no vehicle can perform route ω'.*

Proof. From equation (24),

$$\Pi(\omega') = \alpha \, time(\omega') + \min_{k \in \mathcal{K}(\omega)} \lambda^k,$$

and

$$k(\omega') = \arg\min_{k \in \mathcal{K}(\omega)} \lambda^k.$$

Given that vehicles type $k \in \mathcal{K}(\omega)$ are tested one by one in non-decreasing order of values λ^k, as soon as ω' is feasible for a given $k \in \mathcal{K}(\omega')$, we have $k = k(\omega')$. □

5.2.2. Vehicle type feasibility

In this section, we focus on solving the question: given a vehicle type $k \in \mathcal{K}$ and a load profile $\mathcal{L}$, can this vehicle carry this load? (see line 13, Algorithm 4)

Recall that $\mathcal{Q}^k$ is the set of all configuration vectors for a vehicle type $k \in \mathcal{K}$, represented by the $|\mathcal{C}^k| \times |U|$ matrix:

$$\mathcal{Q}^k = \left\{ \begin{array}{c} \mathbf{Q}^{k1} \\ \dots \\ \mathbf{Q}^{k|\mathcal{C}^k|} \end{array} \right\} = \left\{ \begin{array}{ccc} Q_1^{k1} & \dots & Q_{|U|}^{k1} \\ \dots & \dots & \dots \\ Q_1^{k|\mathcal{C}^k|} & \dots & Q_{|U|}^{k|\mathcal{C}^k|} \end{array} \right\},$$

where $\mathbf{Q}^{kc}$ represents the c^{th} configuration ($c \in \mathcal{C}^k$) of vehicle k and Q_u^{kc} the capacity of vehicle type k in configuration c for user $u \in \mathcal{U}$.

Example 2. *Let us consider the vehicle type k described in Figure 1. Figure 5 represents $\mathbf{Q}^{k1}$, $\mathbf{Q}^{k2}$, $\mathbf{Q}^{k3}$.*

According to these notations, we can state that a route ω is feasible for a given vehicle type k if: (i) for each load $\mathbf{l}$ in the route kernel $\mathcal{L}$, there is a capacity vector in $\mathcal{Q}^k$ that is greater than $\mathbf{l}$; and (ii) if the minimum number of reconfigurations of k to operate $\mathcal{L}$ is less than $\overline{R}$.

The efficiency of these two checks is increased by computing *a priori* the list of feasible configurations for each possible load vector and for each vehicle

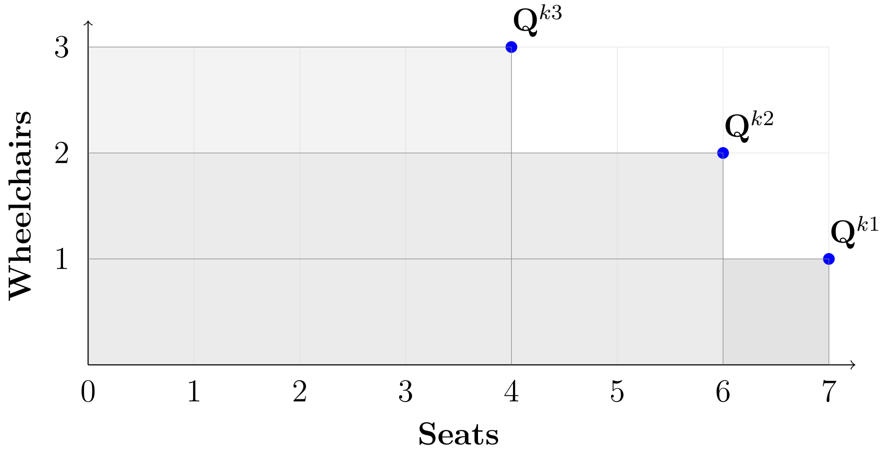

Figure 5: Capacity of the vehicle presented in Figure 1.

type. The envelope set of possible load vectors is defined by

$$\mathcal{T}^k = \{0, \ldots, \max_{c \in \mathcal{C}^k} Q_1^{kc}\} \times \cdots \times \{0, \ldots, \max_{c \in \mathcal{C}^k} Q_{|\mathcal{U}|}^{kc}\}.$$

Mathematically, we define $S^k(\mathbf{l})$ as the function that maps the set of feasible configurations of vehicle type k for a given load $\mathbf{l}$.

$$S^k : \mathcal{T}^k \rightarrow \mathbb{P}(\mathcal{C}^k),$$

where $\mathbb{P}(.)$ is the power-set function of set $\mathcal{C}^k$, i.e. the set of all subsets of $\mathcal{C}^k$, including the empty set and $\mathcal{C}^k$ itself.

The function $S^k(.)$ can be pre-processed and written as a matrix with $|\mathcal{U}|$ dimensions. Each vector of feasible configuration in this matrix is encoded using a *bitset*. This guarantees a limited size in memory and a $O(1)$ access to the list of configurations able to carry any element of $\mathcal{T}^k$.

Table 1 details the value of function $S^k(.)$ in the case described by Example 2 and Figure 5. For each possible load value, this table returns the list of configurations that are compatible with this load. For example, for a load of 4 seats and 2 wheelchairs, configurations 1 and 2 are feasible. For a load of 6 seats and 3 wheelchairs, there is no feasible configuration.

Algorithm 5 presents the procedure for evaluating the feasibility of vehicle k in route ω'. The algorithm initializes the number nr of necessary reconfigurations to 0 (line 1) and the set $ListConfig$ with all admissible

Seats

Wheelchairs	0	1	2	3	4	5	6	7
3	$\{1\}$	$\{1\}$	$\{1\}$	$\{1\}$	$\{1\}$	$\emptyset$	$\emptyset$	$\emptyset$
2	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$	$\{2\}$	$\{2\}$	$\emptyset$
1	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$	$\{2,3\}$	$\{3\}$
0	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$	$\{2,3\}$	$\{3\}$

Table 1: Matrix representation of S^k for the vehicle presented in Figure 1.

configurations (line 2). It then iteratively goes through the nodes of the entire route kernel $\mathcal{L}$. If the current node is not compatible with any vehicle configuration (line 4), vehicle k is unfeasible and the value -1 is returned. Otherwise, the list $ListConfig$ is updated with the set of feasible configurations common to this node and $ListConfig$ (line 6). If the load of the current node **l** requires a configuration not previously stored in $ListConfig$, a reconfiguration is needed. Thus, nr is increased by 1 and the $ListConfig$ is initialized with the set of feasible configurations of the current node (line 11). If nr exceeds the maximum value allowed, $\overline{R}$, vehicle k is unfeasible and the value -1 is returned. Finally, if every load in the nodes of the kernel are compatible with vehicle k the number of reconfigurations nr is returned.

Proposition 2. *Algorithm 5 finds the minimal number of reconfigurations.*

Proof. We define the graph $G = (V, A)$ in which nodes $v_1, \ldots, v_{|\mathcal{L}|}$ represent the ordered set of elements of the kernel. We define an arc between any pair of nodes which can be connected without reconfiguring the vehicle. So, if an arc (v_i, v_j) exists, then all arcs of the form $(v_{i'}, v_{j'})$, where $i \leq i' < j' \leq j$, also exist.

With all arcs weighted by 1, finding the minimum number of reconfigurations is equivalent to finding a shortest path between v_1 and $v_{|\mathcal{L}|}$.

Given a node $v_i \in V$, we define the *farthest* neighbor of v_i as the node v_j, such that the arc (v_i, v_j) exists and the arc (v_i, v_{j+1}) does not exist. Algorithm 5 starts from v_1 and iteratively looks for the farthest neighbor of the current node. Let $v_1, v_{opt(2)}, \ldots, v_{|\mathcal{L}|}$ be an optimal path from v_1 to $v_{|\mathcal{L}|}$. We shall prove by contradiction that if some node v_i is the farthest neighbor of v_1, then it belongs to an optimal path.

Assume that (v_1, v_i) is not in the shortest path from v_1 to $v_{|\mathcal{L}|}$. Because $(v_1, v_j) \notin A, \forall j > i$, it is not possible to go from l_1 to any successor of v_i with a weight of 1. So, $v_{opt(2)} < i$ and $v_{opt(3)} > i$. By construction, if arc

Algorithm 5: Feasibility of vehicle type k and number of necessary reconfigurations

```
Input: route $\omega'$, with the load profile $\mathcal{L}$ of its kernel
Input: vehicle $k$, with preprocessed values for $S^k(.)$ and $\mathcal{T}^k$
Output: number $nr$ of necessary reconfigurations
$nr \leftarrow 0$
$ListConfig \leftarrow \mathcal{C}^k$
forall $\mathbf{l} \in \mathcal{L}$ do
                                        /* capacity test */
    if $\mathbf{l} \notin \mathcal{T}^k$ or $S^k(\mathbf{l}) = \emptyset$ then
        return $-1$
                            /* number of reconfigurations test */
    $ListConfig \leftarrow ListConfig \cap S^k(\mathbf{l})$
    if $ListConfig = \emptyset$ then
        $nr \leftarrow nr + 1$
        if $nr > \overline{R}$ then
            return $-1$
        $ListConfig \leftarrow S^k(\mathbf{l})$
return $nr$
```

$(v_{opt(2)}, v_{opt(3)})$ exists and has weight 1, then arc $(v_i, v_{opt(3)})$ also exists and has the same weight. Thus, the cost of path $v_1 \to v_i \to v_{opt(3)}$ is 2. Hence, (v_1, v_i) is also an optimal solution.

□

6. Computational experiments

The matheuristic described in Section 4.1 was coded in C++ and run on a CPU Intel Xeon E5-1620 v3 @3.5Ghz. The SCP was solved with CPLEX 12.6 running on a single thread. The matheuristic was evaluated using real and benchmark data. The parameters shown in Table 2 provide the best average performance for the optimization problems solved in this paper.

After experimentation, we found that: K-regret operators with $K > 4$ did not improve the solution quality and two removal operators (historical removal and random removal) are sufficient when the SCP component is active. Note that values for parameters η, t_{limit} and ψ, can be automatically modified in the reactive version RSCP for different instance sizes (see Section 4.3).

$\chi = 5\%$	Record-to-record acceptance percentage.
$penalty = 10^4$	Penalty cost for incomplete solutions.
$\phi^- = 10\%$	Minimal proportion of removed request used by removal operators.
$\phi^+ = 40\%$	Maximal proportion of removed request used by removal operators.
$p = 6$	Roulette wheel parameter for the historical removal operator.
$\eta = 1\,000$	Launch frequency of the SCP.
$t_{limit} = 3$ sec	Imposed time limit for the SCP.
$\psi = 1.25$	RSCP coefficient to recompute the launch frequency of the SCP.

Table 2: Parameters (all defined in Section 4).

Regarding the sequencing of feasibility evaluation, performing the *Schedule evaluation* (Algorithm 3) before the *Capacity evaluation* (Algorithm 2) can reduce computation time up to 50%. This reduction is mostly explained by the fact our data set (derived from real data) is constrained more in time than in capacity. In other words, long pickup legs are seldom feasible because of time windows and maximum ride times. In addition, sorting vehicle types in non-increasing order of costs, before testing the vehicles capacity, can reduce computation time by up to 10% (Algorithm 4).

This section is structured as follows: First, we introduce the instances used to evaluate the algorithms. In Section 6.2 we present the experiments that determine the choice and calibration of the main components of the proposed matheuristic. The proposed algorithms are compared to state-of-the-art algorithms on benchmarks from Cordeau and Laporte (2003) and Qu and Bard (2013), which correspond to particular cases of our problem. Finally, we provide managerial insights, mainly regarding vehicle fleet aspects.

6.1. Description of instances

The real data comes from the GIHP Company[2]. An instance set of 14 instances is made from a bank of 576 requests decomposed into smaller subsets of three sizes. There are 8 small instances containing from 60 to 80 requests, 4 medium-size instances containing from 120 to 160 requests, and 2 large instances containing from 280 to 295 requests. We consider 2 different user types: users occupying seats and users with wheelchairs. Each type of user occupies dedicated spaces in vehicles. Travel times and distances are obtained from the Open Source Routing Machine[3] (OSRM) by Luxen and Vetter (2011). Common characteristics for the instances are: 1) maximum ride times are defined according to direct travel time (t_{p_r,d_r}) by the formula $RT = 15 \times \lceil (t_{p_r,d_r} + 15)/15 \rceil$; 2) time windows at medico-social institutions are 30 minutes wide while there is no time window at pickup locations; and 3) the service time for users using seats is 2 minutes at the pickup location and 1 minute at the delivery location. For users in a wheelchair, it is 5 minutes at pickup locations and 2 minutes at delivery locations.

The characteristics of vehicles considered in experiments are summarized in Table 4. There are four vehicle types, including one that is not configurable (V_0). Vehicles V_1 and V_2 have two possible configurations, while vehicles V_3 have three configurations. Costs for each vehicle type can be found in Table 4. Note that in terms of capacity, vehicle V_0 is a restricted case of vehicle V_3. It is nevertheless an interesting choice as its fixed cost is considerably lower (50€

[2] These instances will be made available on `www.vrp-rep.org` on acceptation of this paper.

[3] `http://project-osrm.org/`

against 63€). The fixed cost corresponds to the estimated vehicle ownership cost per day. The time-related cost corresponds to the driver's wages, and the distance-related cost corresponds to the fuel consumption and vehicle use.

Vehicle type	Fixed cost	Time-related cost	Distance-related cost
V_0	50 €	23.81 €/h	0.17 €/km
V_1	79 €	23.81 €/h	0.21 €/km
V_2	36 €	23.81 €/h	0.12 €/km
V_3	63 €	23.81 €/h	0.17 €/km

Table 3: Vehicle types – costs

	Configuration1		Configuration 2		Configuration 3	
Type	Seats	Wheelchairs	Seats	Wheelchairs	Seats	Wheelchairs
V_0	4	3	-	-	-	-
V_1	3	5	4	4	-	-
V_2	2	1	4	0	-	-
V_3	4	3	6	2	7	1

Table 4: Vehicle types – configurations

6.2. Evaluation of the metaheuristic components

Table 5 compares several LNS settings in order to evaluate the main components of the proposed matheuristic. These experiments are performed on the presented real instances with the four vehicles described in Table 4. From left to right, the first column (Inst) is the instance name composed of two numbers. The first number is the instance number and the second one indicates the number of requests in that instance. The second column (Time) is the computation time in minutes allowed to each solution method. For small instances, this time is 16 minutes. It is 40 minutes for medium-size instances and 100 minutes for large instances. Seven LNS variants are considered: The first variant is a classic implementation of LNS which implements operators k-regret, best insertion, random removal, historical node-pair removal, worse removal, time-related removal and distance related removal, as described in Pisinger and Ropke (2007). This variant is denoted by LNS(5) because it implements 5 destroy operators. The second metaheuristics ALNS(5) adds-up the adaptive layer of Ropke and Pisinger (2006) to the former LNS implementation; the parameters used can be found in Appendix 7. The third metaheuristic ALNS(5)–SCP integrates the SCP component described in Section 4.3. The fourth variant, denoted LNS(2)–SCP, integrates only 2 destroy operators (historical node-pair removal and random removal) and no adaptive layer. The fifth variant, denoted ALNS(5)–RSCP, is an extension of ALNS(5)–SCP integrating the reactive layer described in Section 4.3. The sixth variant, denoted LNS(5)–RSCP, does not include the adaptive layer. Finally, LNS(2)–RSCP extends the LNS(2)–SCP by including the reactive layer of Section 4.3.

The first observation is that the SCP component brings significant improvement in the solution quality. In addition, it can be observed that the LNS can be considerably simplified when the SCP component is used. Indeed, the matheuristic with two destroy operators performs as well as the versions with five destroy operators. Comparing LNS(2)–SCP with LNS(2)–RSCP, it can be seen that a second subsequent improvement is obtained when the SCP includes the reactive layer to adapt its parameters during the search.

According to this set of experiments, LNS(2)–RSCP configuration seems to outperform the other variants, either in performance or in simplicity.

		LNS(5)		ALNS(5)		ALNS(5)-SCP		LNS(2)-SCP		ALNS(5)-RSCP		LNS(5)-RSCP		LNS(2)-RSCP	
Inst	Time	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
I01-80	16	1,791.69	1,798.30	1,803.42	1,811.81	**1,757.22**	1,757.70	**1,757.22**	1,757.22	**1,757.22**	1,757.22	**1,757.22**	1,757.22	**1,757.22**	1,757.22
I02-60	16	**911.79**	913.64	911.88	926.49	925.09	926.38	**911.79**	913.21	915.02	922.48	**911.79**	914.45	915.34	922.63
I03-80	16	1,869.42	1,882.15	1,881.31	1,893.74	1,838.33	1,838.60	1,838.47	1,838.47	**1,811.86**	1,811.86	**1,811.86**	1,811.86	**1,811.86**	1,811.90
I04-70	16	1,600.54	1,603.65	1,599.53	1,602.58	**1,599.49**	1,599.52	**1,599.49**	1,599.50	**1,599.49**	1,599.51	**1,599.49**	1,599.49	**1,599.49**	1,599.49
I05-80	16	1,251.24	1,253.37	1,244.86	1,251.00	**1,242.48**	1,242.62	**1,242.48**	1,242.48	**1,242.48**	1,242.48	**1,242.48**	1,242.48	**1,242.48**	1,242.48
I06-80	16	1,451.57	1,452.69	1,460.48	1,464.95	**1,418.74**	1,421.09	**1,418.74**	1,422.19	**1,418.74**	1,421.31	**1,418.74**	1,420.88	**1,418.74**	1,418.75
I07-60	16	**1,537.63**	1,537.63	**1,537.63**	1,537.79	**1,537.63**	1,537.63	**1,537.63**	1,537.63	**1,537.63**	1,537.63	**1,537.63**	1,537.63	**1,537.63**	1,537.63
I08-65	16	**1,204.89**	1,205.68	**1,204.89**	1,218.95	**1,204.89**	1,205.11	**1,204.89**	1,205.11	**1,204.89**	1,205.54	**1,204.89**	1,204.89	**1,204.89**	1,205.11
I09-120	40	3,165.79	3,186.43	3,178.83	3,189.00	**3,128.00**	3,128.00	**3,128.00**	3,128.00	**3,128.00**	3,128.00	**3,128.00**	3,128.00	**3,128.00**	3,128.00
I10-135	40	2,196.36	2,208.65	2,169.89	2,219.19	2,059.99	2,069.46	**2,059.91**	2,070.72	2,071.42	2,087.16	2,081.24	2,088.84	**2,059.91**	2,079.79
I11-160	40	2,343.57	2,347.28	2,310.38	2,331.33	2,224.06	2,238.53	2,226.56	2,243.24	2,225.54	2,229.27	**2,222.14**	2,225.37	**2,222.14**	2,228.43
I12-160	40	2,851.89	2,864.54	2,880.07	2,911.25	2,615.00	2,627.80	2,620.93	2,638.95	2,623.44	2,628.19	**2,614.47**	2,631.98	2,623.24	2,634.22
I13-280	100	5,227.05	5,255.18	5,227.45	5,275.53	5,153.87	5,196.40	5,123.11	5,132.67	4,864.49	4,896.87	**4,843.39**	4,870.33	4,885.02	4,917.22
I14-295	100	5,475.69	5,497.11	5,476.19	5,533.58	5,352.61	5,411.52	5,387.16	5,431.81	**4,893.30**	4,927.77	4,921.74	4,948.96	4,904.47	4,931.81
	Avg	2,348.51	2,357.59	2,349.06	2,369.08	2,289.81	2,300.03	2,289.74	2,297.23	2,235.25	2,242.52	2,235.36	2,241.60	2,236.46	2,243.91
	Avg Gap	3.60%	3.94%	3.61%	4.46%	1.34%	1.64%	1.27%	1.52%	0.13%	0.38%	0.12%	0.31%	0.13%	0.39%
	NbBKS	3		2		7		9		9		12		10	

Average cost (Avg) is computed over 5 runs. Nb BKS refers to the number of best-known solutions (BKS). Time is expressed in minutes.
The Gap for every instance is computed as $(value - BKS)/BKS * 100$.
General parameters for all metaheuristics can be found in Table 2. Additional parameters for ALNS can be found in Table A.13.

Table 5: Performance comparison of LNS-based heuristics

Figure 6 shows the cost evolution considering the best run of four metaheuristics in instance I01-80. The other 3 variants are not shown because their performance is almost that of LNS(2)–RSP. Two separate groups of algorithms can be identified, corresponding to metaheuristics with and without an SCP component. The SCP increases not only solution quality, but also convergence speed.

Figure 7 shows the cost evolution on a large size instance I13-280. The variants without the reactive layer fail to improve the solution significantly. This is because the SCP parameters are not adapted to this instances and the SCP is never solved optimally. The variants with the reactive layer perform well since the number of iterations between two calls to the SCP is reduced. This approach proves efficient in large instance as shown in the figure.

6.3. Performance evaluation on benchmarks from the literature

To evaluate the performance of LNS(2)-RSCP, we apply it to two subproblems of the FSM-DARP-RC on benchmark instances. The first benchmark is made of the DARP instances of Cordeau and Laporte (2003) for which many elaborate methods were designed. In a second step, LNS(2)-RSCP is compared to the algorithm of Qu and Bard (2013) that was designed to solve a heterogeneous DARP with configurable vehicle capacity. This benchmark considers a limited fleet of vehicles that can be configured at the depot.

6.3.1. Performance evaluation on the DARP

The Cordeau and Laporte (2003) instances are still a reference point to evaluate the efficiency of algorithms on the DARP. Table 6 compares LNS(2)–RSCP after 50,000 Iterations to the latest heuristics that have been designed specifically to solve this problem. The best known solutions (BKS column) are taken from Gschwind and Drexl (2016) and have been found either by the cited meta-heuristics, or during the parameter tuning of Gschwind and Drexl (2016). Looking at the average gap (Avg Gap) we note that our matheuristic competes with those three dedicated methods. LNS(2)–RSCP has the second-best average gap of 0.72% with 8 solutions among the best known (Nb. BKS), just after the ALNS of Gschwind and Drexl (2016) with 0.5% Avg Gap and 9 BKS. An interesting remark is that the reactive layer of the SCP components was almost never activated since the initial values of $\eta = 1000$ and $t_{limit} = 3$ secs is sufficient for these instance sizes. The other

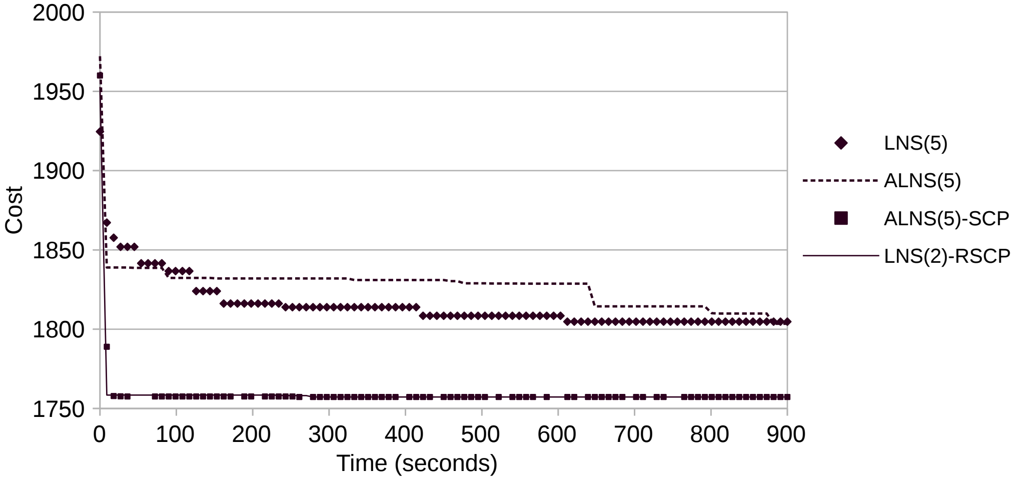

Figure 6: Impact of SCP component (Instance I01-80)

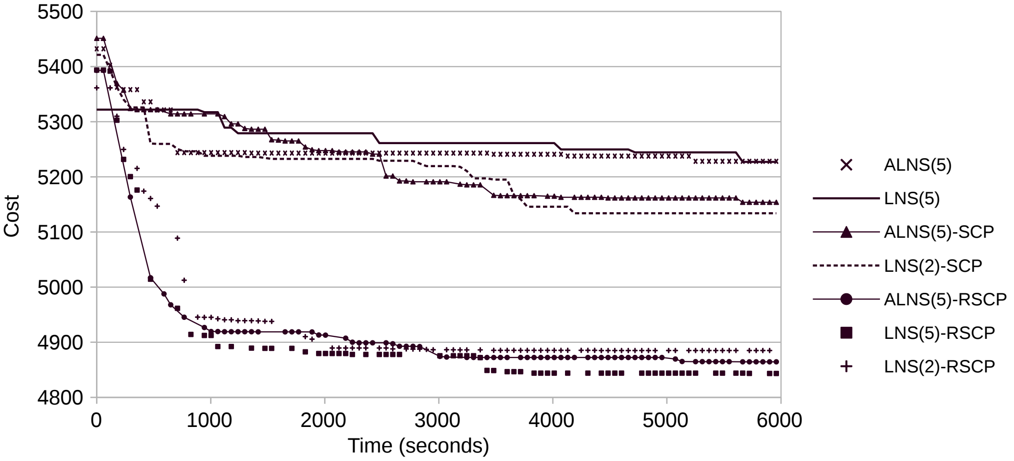

Figure 7: Reactive SCP (Instance I13-280)

two methods are the threshold acceptance (TA) based local search of Braekers et al. (2014) which has the highest average gap (1.16%) but also by far the lowest computation time, and the evolutionary local search (ELS) proposed by Chassaing et al. (2016).

Note that we do not scale computational times here as CPU information is not available for all methods and our goal is not to compete with these three methods. Nevertheless, these experiments show that the performance of LNS(2)–RSCP without any modification from the calibration on the FSM-DARP-RC instances, remains competitive on simpler benchmark problems.

		TA			ELS			ALNS[1]			LNS(2)-RSCP[2]		
		Braekers et al. (2014)			Chassaing et al. (2016)			Gschwind and Drexl (2016)					
Inst	BKS	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
pr01	190.02	**190.02**	190.02	16.6	**190.02**	190.02	15.0	**190.02**	190.02	15.1	**190.02**	190.02	24.0
pr02	301.34	**301.34**	301.34	42.0	**301.34**	301.34	75.0	**301.34**	301.34	41.1	**301.34**	301.34	145.9
pr03	532.00	532.10	533.54	48.8	532.43	533.86	138.0	**532.00**	532.01	62.1	**532.00**	532.00	256.6
pr04	570.25	577.16	580.52	74.6	570.54	574.47	442.2	**570.25**	570.61	138.7	571.25	572.23	493.9
pr05	**625.64**	629.80	632.06	89.2	630.82	637.59	724.2	628.59	630.99	267.4	631.38	633.20	735.7
pr06	**783.78**	797.78	800.68	107.0	792.80	796.10	1,315.2	789.36	790.44	397.5	789.51	790.23	1,216.6
pr07	291.71	292.23	292.23	22.6	**291.71**	292.96	28.2	**291.71**	291.71	19.7	**291.71**	291.71	55.3
pr08	**487.84**	490.94	491.00	48.6	491.60	493.16	160.8	489.83	491.13	74.1	489.33	492.49	329.4
pr09	**653.94**	662.64	666.65	72.2	672.86	681.35	675.0	659.69	660.13	151.7	659.10	663.00	668.8
pr10	**845.47**	853.98	860.83	114.4	857.36	860.68	1,279.8	853.07	857.18	389.2	856.95	860.91	1,236.6
pr11	164.46	**164.46**	164.46	23.8	**164.46**	164.46	16.8	**164.46**	164.46	17.7	**164.46**	164.46	43.6
pr12	295.66	295.69	296.06	51.4	**295.66**	295.72	82.2	**295.66**	296.22	48.1	295.67	296.83	227.6
pr13	484.83	488.61	490.03	76.2	489.00	490.70	222.0	**484.83**	484.83	101.3	488.30	488.70	478.4
pr14	**529.33**	534.99	540.99	117.0	531.08	531.98	612.0	531.19	531.92	204.2	**529.33**	532.13	966.2
pr15	**573.56**	581.46	584.33	155.2	578.44	580.23	1,195.8	576.70	578.17	491.6	577.00	579.47	1,584.0
pr16	**725.22**	743.56	747.19	180.6	731.25	736.59	1,939.2	731.50	736.08	692.6	732.60	737.12	2,331.9
pr17	248.21	249.33	249.33	34.0	**248.21**	248.21	34.8	**248.21**	248.21	22.8	**248.21**	248.21	104.8
pr18	458.73	461.77	462.38	81.0	461.21	462.40	259.2	461.48	461.67	103.0	**458.73**	461.42	520.8
pr19	**592.23**	598.23	600.63	146.4	595.39	597.53	745.8	593.83	596.77	361.4	594.22	597.65	1,225.3
pr20	**783.81**	795.08	801.89	162.8	796.60	803.99	1,887.0	787.14	789.83	591.6	789.49	799.38	1,895.8
Avg Gap		0.81%	1.16%	83.2	0.64%	1.04%	592.4	0.32%	0.50%	209.5	0.38%	0.72%	727.1
Nb BKS			3			6			9			8	

Average cost (Avg) is computed over 5 runs. (Time) is the average computation time in seconds.
Nb BKS refers to the number of best knows solutions. The Gap for an instance is computed as $(value - BKS)/BKS \times 100$

[1] Running 75 000 iterations including the Balas-Simonetti operator $Op_k = 3$ and solve a SCP at the very end of the solution method.

[2] Running 50 000 iterations

Table 6: Results on the DARP instances of Cordeau and Laporte (2003)

6.3.2. Performance evaluation on the heterogeneous DARP with configurable vehicle capacity

Another relevant benchmark was proposed by Qu and Bard (2013) for the heterogeneous dial-a-ride with configurable vehicle capacity (HDARP-C). The three main differences between this problem and the FSM-DARP-RC are: (i) a limited fleet of heterogeneous vehicles with configurable capacity; (ii) the vehicle configuration is decided at the depot, not en-route; and (iii) the presence of users with walkers in some instances, which means that two seats or one seat and half a wheelchair space are occupied.

These differences were integrated into our algorithm to run Qu and Bard (2013) instances. The SCP was adapted in a simple manner to consider a limited fleet of vehicles as follows: first, every route was explicitly assigned the vehicle type for which it was generated, and second, a constraint limiting the number of vehicles for each vehicle type was added to the SCP model.

In this benchmark, two instance sets of 100 requests are proposed. A first set “A” is characterized by integrating various proportions of appointment requests (between 5% to 25%, distinguished in the first two digits of the name). The second instance set “B” instead differentiates the proportion of users with walkers, between 20% to 60% of clients. This proportion is distinguished in the first two digits in the name and the number of appointments is always constant at 10%.

According to Qu and Bard (2013), in the following tables we denote by C_t, the cost related to travel time. It is computed as the total traveled distance divided by *speed*, times the time cost $c_{time} = \$0.377/min$. C_p denotes the total passenger ride time cost with the unitary cost $\tau_c = \$0.0001$ per minute. The vehicle ownership cost is denoted as $C_v = \$0.01/van$. Finally, the total cost is denoted by $C_{total} = C_t + C_p + C_v$. For the subsequent experiments, LNS(2)–RSCP uses the same parameter tuning proposed in Table 2.

Table 7 compares the performance of our algorithm with respect to the multi-start ALNS (MSALNS) heuristic of Qu and Bard (2013) for the instance set A. From the Avg row we observe an average total cost improvement of 1.02% over 3 runs and all instances. Looking at the best solution found over those three runs, a 1.34% improvement is obtained on average over 3 runs with respect to the MSALNS best solutions. 6 new best solutions were found out of the 10 instances of the benchmark. These results demonstrate the good performance and stability of LNS(2)-RSCP from an experimental point of view.

A second set of experiments is performed for instance set “B” with two scenarios. The first scenario in Table 8 presents the results obtained when walkers are taken into account. The main difference between set “B” and set “A” is the presence of very few time windows. The ten instances are built up upon two instances of set “A”, then varying the proportion of walkers while all other characteristics remain constant. Looking at the total cost of

	MSALNS				LNS(2)-RSCP					
Inst	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}[1]	Gap (*)	Gap Avg
A051	43.00	0.20	0.04	43.24	42.29	0.22	0.09	**42.60**	-1.47%	-1.71%
A151	49.82	0.23	0.06	**50.11**	50.18	0.27	0.06	50.51	0.80%	0.18%
A101	43.99	0.22	0.05	**44.26**	44.75	0.25	0.08	45.07	1.84%	1.17%
A251	62.42	0.29	0.06	62.77	55.18	0.28	0.08	**55.54**	-11.52%	-10.95%
A201	53.57	0.23	0.06	53.86	52.81	0.27	0.06	**53.14**	-1.33%	-2.28%
A052	36.86	0.20	0.05	**37.11**	42.25	0.23	0.09	42.58	14.73%	13.99%
A102	45.19	0.19	0.05	**45.43**	45.34	0.30	0.07	45.71	0.61%	2.92%
A152	49.64	0.21	0.05	49.90	44.37	0.29	0.09	**44.75**	-10.32%	-8.17%
A202	49.79	0.24	0.06	50.09	48.56	0.30	0.09	**48.94**	-2.29%	-0.76%
A252	55.21	0.28	0.06	55.55	52.73	0.28	0.09	**53.10**	-4.42%	-4.62%
Avg	48.95	0.23	0.05	49.23	47.85	0.27	0.08	48.19	-1.34%	-1.02%

1. C_{total} is the best total cost found over 3 runs. The stopping criteria is 1 hour.
Gaps were computed with respect to C_{total} from Qu and Bard (2013).

Table 7: Benchmark Qu and Bard (2013) instance A scenario (iii) without groups

the solutions returned by LNS(2)-RSCP, we find that it even more clearly outperforms the MSALNS of Qu and Bard (2013) than on instances A. A second observation is that LNS(2)-RSCP is barely affected by the proportion of walkers. This result is confirmed by the second scenario in Table 9 where walkers requirements are ignored. Without accounting for walkers there is no difference among all instances ranging from B201 to B601 instances. The same applies to BX02 instances. In Table 8 we present our aggregated results for five runs of each instance type BX01 and BX02. Looking at the average gap, we find that LNS(2)-RSCP improves the solution by 13.32% when compared to the MSALNS.

	MSALNS				LNS(2)-RSCP					
Inst	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}[1]	Gap	Avg Gap
B201	43.99	0.22	0.05	44.25	37.01	0.37	0.06	**37.44**	-15.39%	-15.09%
B301	44.84	0.25	0.05	45.15	37.19	0.30	0.06	**37.54**	-16.85%	-16.07%
B401	47.65	0.21	0.05	47.91	38.82	0.26	0.05	**39.13**	-18.33%	-18.28%
B501	47.87	0.20	0.05	48.12	38.07	0.30	0.05	**38.43**	-20.15%	-19.69%
B601	45.83	0.22	0.05	46.10	40.10	0.20	0.05	**40.36**	-12.46%	-12.12%
B202	45.19	0.19	0.05	45.43	36.34	0.31	0.05	**36.70**	-19.21%	-18.66%
B302	44.42	0.19	0.05	44.66	37.73	0.21	0.05	**37.98**	-14.95%	-14.83%
B402	47.90	0.19	0.05	48.14	36.63	0.24	0.05	**36.92**	-23.30%	-23.10%
B502	46.50	0.18	0.05	46.73	36.93	0.24	0.05	**37.21**	-20.36%	-17.73%
B602	45.72	0.20	0.05	45.97	37.73	0.30	0.05	**38.08**	-17.16%	-16.75%
Avg	45.99	0.21	0.05	46.25	37.66	0.27	0.05	37.98	-17.82%	-17.23%

1. C_{total} is the best total cost found over 3 runs. The stopping criteria is 1 hour.
Gaps were computed with respect to C_{total} from Qu and Bard (2013).

Table 8: Benchmark Qu and Bard (2013) instance “B” scenario (iii) without groups

	MSALNS				LNS(2)-RSCP					
Inst	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}[1]	Gap	Avg Gap
B201	43.44	0.22	0.05	43.72	36.34	0.35	0.05	**36.74**	-15.97%	-15.59%
B301	43.90	0.27	0.05	44.21	36.47	0.35	0.05	**36.86**	-16.62%	
B401	43.99	0.29	0.05	44.32	37.20	0.25	0.04	**37.50**	-15.39%	
B501	42.94	0.22	0.05	43.21	36.43	0.32	0.05	**36.81**	-14.82%	
B601	43.22	0.21	0.05	43.48	36.47	0.37	0.05	**36.90**	-15.14%	
B202	42.13	0.27	0.05	42.45	36.87	0.28	0.05	**37.19**	-12.38%	-9.82%
B302	42.13	0.27	0.05	42.45	36.87	0.28	0.05	**37.19**	-12.38%	
B402	42.13	0.27	0.05	42.45	37.99	0.25	0.04	**38.28**	-9.82%	
B502	40.38	0.23	0.04	40.65	37.69	0.26	0.04	**37.99**	-6.54%	
B602	39.70	0.23	0.05	39.98	36.49	0.25	0.05	**36.79**	-7.99%	
Avg	42.40	0.25	0.05	42.69	36.88	0.30	0.05	37.22	-12.71%	-12.71%

[1] C_{total} after 1 hour of computation time.

Table 9: Benchmark Qu and Bard (2013) instances "B" ignoring walker requirements.

6.4. Managerial insights

Having shown the efficiency of our matheuristic, we performed simulations in order to provide some general insights regarding, in particular, the relevance of en-route reconfiguration of vehicles. First, we measured the impact of en-route reconfiguration for two variants of the DARP. We then varied the fixed cost of reconfigurable vehicles to determine under which conditions it is worth buying the vehicles.

6.4.1. Vehicle fleet insights

In this section we analyze the gain of enabling en-route reconfigurations. Two variants of the DARP are compared with the real instance set, as shown in Table 10. Both variants employ the vehicle fleet of Table 4. The first variant, denoted as FSM-DARP-C, consist in allowing vehicles to be configured only once: at the depot, before starting their respective routes. The second variant, denoted FSM-DARP-RC, allows vehicles to be reconfigured en-route. Strictly speaking FSM-DARP-RC is a relaxed problem of FSM-DARP-C, which explains why gains in the last column are always greater than 0. Cost reductions can go up to 2.45% of the total cost, depending on the instance.

Regarding the structure of solutions, let us consider the best solution (Best) found among the 5 runs for each instance. The next columns (Routes, Rec, RR) refer to some characteristics of these Best solutions. Passing from FSM-DARP-C to FSM-DARP-RC, we observe that the number of routes (Routes) is similar for most instances. However, in some instances, like I14-280, cost savings are related to a reduction in the number of routes (from 37

		FSM-DARP-C			FSM-DARP-RC					
Inst	Time	Avg	Best	Routes	Avg	Best	Routes	Rec	RR	Savings
I01-80	16	1,774.74	1,774.74	18	1,757.22	1,757.22	18	1	1	0.99%
I02-60	16	924.68	923.51	8	916.22	912.28	8	1	3	1.22%
I03-80	16	1,838.57	1,838.47	15	1,838.41	1,838.33	15	1	1	0.01%
I04-70	16	1,602.48	1,602.48	18	1,599.49	1,599.49	18	1	1	0.19%
I05-80	16	1,242.55	1,242.48	15	1,242.48	1,242.48	15	0	0	0.00%
I06-80	16	1,419.81	1,418.74	12	1,418.75	1,418.74	12	0	0	0.00%
I07-60	16	1,546.61	1,546.61	11	1,537.63	1,537.63	11	2	3	0.58%
I08-65	16	1,223.65	1,223.58	10	1,205.11	1,204.89	10	1	2	1.53%
I09-120	40	3,167.81	3,166.29	23	3,129.75	3,129.15	23	2	5	1.17%
I10-135	40	2,094.77	2,068.71	17	2,083.10	2,061.52	17	1	3	0.35%
I11-160	40	2,283.03	2,271.57	21	2,249.11	2,233.19	21	3	7	1.69%
I12-160	40	2,659.34	2,653.40	24	2,653.78	2,629.19	23	2	3	0.91%
I13-280	100	5,027.30	5,008.69	37	4,901.06	4,885.77	36	4	13	2.45%
I14-295	100	5,017.01	4,982.85	45	5,026.49	4,981.14	44	2	8	0.03%

Table 10: Savings due to en-route configuration

to 36).

The maximum number of performed reconfigurations among the solution routes (Rec) shows that en-route reconfiguration is usually performed once or twice inside a route. The number of routes performing en-route reconfiguration (RR) indicates how many configurable vehicles are actually reconfiguring en-route in the whole solution. For example, in instance I02-60, we find that 3 out of 8 routes actually reconfigure en-route, this is 37.5% of the vehicle fleet. In average this proportion is 10.28% for small instances, 21.43% for medium-size and 26% for large instances.

6.4.2. Fixed cost analysis

So far, we have seen that reconfigurations can allow cost reductions; however, the resulting gain is correlated to the cost of buying reconfigurable vehicles rather than standard ones. Figure 8 pictures how, for the instance in I13-280, the percentage of reconfigurable vehicles decreases when their cost increases compared to the cost of standard vehicles. In this figure, the point with coordinates (2%;12%) means that re-configurable vehicles are 2% more expensive than standard vehicles, and in the cheapest solution found, 12% of the vehicles are reconfigurable. In the same way, when reconfigurable vehicles are 20% more expensive than standard vehicles, they are completely excluded from the solution.

The set of vehicle types considered in these experiments can be found in Table 11 and their respective costs in Table 12.

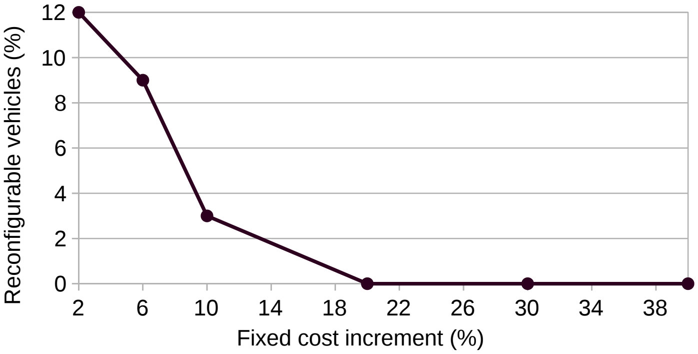

Figure 8: Fixed cost impact of reconfigurable vehicles (Instance I13-280)

	Configuration 1		Configuration 2		Configuration 3	
Vehicle type	Seats	Wheelchairs	Seats	Wheelchairs	Seats	Wheelchairs
V_4	7	1	-	-	-	-
V_5	6	2	-	-	-	-
V_6	4	3	-	-	-	-
V_7	4	3	6	2	7	1

Table 11: Vehicle types used in the fixed cost analysis

Vehicle type	Fixed cost	Time-related cost	Distance-related cost
V_4	50€	23.81 €/h	0.17 €/km
V_5	50€	23.81 €/h	0.17 €/km
V_6	50€	23.81 €/h	0.17 €/km
V_7	(+2%) 51€ (+6%) 53€ (+10%) 55€ (+20%) 60€ (+30%) 65€ (+40%) 70€	23.81 €/h	0.17 €/km

Table 12: Costs of vehicle types used in the fixed cost analysis

7. Conclusion

In this paper, we investigated a new variant of the dial-a-ride problem characterized by en-route reconfiguration of vehicle capacity. This feature was studied in the context of door-to-door transportation of children with disabilities, considering heterogeneous users and vehicles. We aimed to determine the size and composition of the fleet as a strategic decision. A matheuristic based on Large Neighbourhood Search and a Reactive Set Covering Problem (LNS–RSCP)was proposed to solve this problem for real instances of up to 295 user requests. Results on the real data set showed significant performance compared with six other LNS-based metaheuristics variants. Experiments show that the SCP component increases not only solution quality but also convergence speed. LNS–RSCP was also tested on literature instances that achieved competitive results for the classic DARP and outstanding results in the heterogeneous DARP with configurable vehicle capacity. Although en-route reconfiguration is not a usual practice in companies, companies often own configurable vehicles, as in the case study. Yet route designers do not plan routes considering this extra degree of flexibility. In this study we show that companies can easily save up to 2.5% in the total cost just by allowing vehicles to use en-route reconfiguration. Finally, we show that the utilization of reconfigurable vehicles is strongly dependent on the vehicle ownership cost (fixed cost). For the evaluated instance, supposing that all operations are the same on each day, we found that reconfigurable vehicles are advantageous for companies when their cost are no more than 20% of the cost of standard non-reconfigurable vehicles.

Appendix A. Parameters used for ALNS-based metaheuristics

Reinitialize incumbent solution with the best solution every 2000 iters.
Score for new best solutions σ_1= 33
Score for new improving solutions σ_2 = 20
Score for new accepted solutions σ_3 = 15
Reaction factor r = 0.1
Recompute operator weights = 100 iters
Minimum weight operators = 0.1
Maximum weight operators = 5

Table A.13: ALNS Parameters as in Masson et al. (2014)

References

Archetti, C., Speranza, M. G., 2014. A survey on matheuristics for routing problems. EURO Journal on Computational Optimization 2 (4), 223–246.

Braekers, K., Caris, A., Janssens, G. K., 2014. Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. Transportation Research Part B: Methodological 67, 166–186.

Chassaing, M., Duhamel, C., Lacomme, P., 2016. An ELS-based approach with dynamic probabilities management in local search for the Dial-A-Ride Problem. Engineering Applications of Artificial Intelligence 48, 119–133.

Christiaens, J., Vanden Berghe, G., 2016. A fresh ruin & recreate implementation for the capacitated vehicle routing problem. Technical report, KU Leuven, Department of Computer Science.

Cordeau, J.-F., Laporte, G., 2003. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B: Methodological 37 (6), 579–594.

Derigs, U., Gottlieb, J., Kalkoff, J., Piesche, M., Rothlauf, F., Vogel, U., 2011. Vehicle routing with compartments: applications, modelling and heuristics. OR spectrum 33 (4), 885–914.

Doerner, K. F., Salazar-Gonzàlez, J.-J., 2014. Pickup-and-delivery problems for people transportation. In: Vigo, D., Toth, P. (Eds.), Vehicle Routing: Problems, Methods, and Applications. Vol. 18. SIAM, Ch. 7, pp. 193–212.

Dueck, G., 1993. New optimization heuristics. Journal of Computational Physics 104 (1), 86 – 92.

Firat, M., Woeginger, G. J., 2011. Analysis of the dial-a-ride problem of Hunsaker and Savelsbergh. Operations Research Letters 39 (1), 32–35.

Foster, B. A., Ryan, D. M., 1976. An integer programming approach to the vehicle scheduling problem. Journal of the Operational Research Society 27 (2), 367–384.

Golden, B., Assad, A., Levy, L., Gheysens, F., 1984. The fleet size and mix vehicle routing problem. Computers & Operations Research 11 (1), 49–66.

Grangier, P., Gendreau, M., Lehuédé, F., Rousseau, L.-M., 2017. A matheuristic based on large neighborhood search for the vehicle routing problem with cross-docking. Computers & Operations Research 84, 116–126.

Gschwind, T., Drexl, M., 2016. Adaptive Large Neighborhood Search with a Constant-Time Feasibility Test for the Dial-a-Ride Problem. LM-2016-08, Gutenberg School of Management and Economics, Johannes Gutenberg University.

Gschwind, Timo, 2015. Route Feasibility Testing and Forward Time Slack for the Synchronized Pickup and Delivery Problem. Tech. Rep. 1503, Gutenberg School of Management and Economics, Johannes Gutenberg University.

Henke, T., Speranza, M. G., Wäscher, G., 2015. The multi-compartment vehicle routing problem with flexible compartment sizes. European Journal of Operational Research 246 (3), 730–743.

Hunsaker, B., Savelsbergh, M., 2002. Efficient feasibility testing for dial-a-ride problems. Operations Research Letters 30 (3), 169–173.

Koç, c., Bektas, T., Jabali, O., Laporte, G., 2016. Thirty years of heterogeneous vehicle routing. European Journal of Operational Research 249 (1), 1–21.

Koch, H., Henke, T., Wäscher, G., 2016. A genetic algorithm for the multi-compartment vehicle routing problem with flexible compartment sizes. Tech. rep., Otto-von-Guericke-Universität Magdeburg, Faculty of Economics and Management.

Luxen, D., Vetter, C., 2011. Real-time routing with openstreetmap data. In: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. GIS'11. ACM, New York, NY, USA, pp. 513–516.

Masson, R., Lehuédé, F., Péton, O., 2014. The Dial-A-Ride Problem with Transfers. Computers & Operations Research 41, 12–23.

Parragh, S. N., 2011. Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. Transportation Research Part C: Emerging Technologies 19 (5), 912–930.

Parragh, S. N., Doerner, K. F., Hartl, R. F., 2008. A survey on pickup and delivery problems. Journal für Betriebswirtschaft 58 (2), 81–117.

Parragh, S. N., Doerner, K. F., Hartl, R. F., 2010. Variable neighborhood search for the dial-a-ride problem. Computers & Operations Research 37 (6), 1129–1138.

Parragh, S. N., Schmid, V., 2013. Hybrid column generation and large neighborhood search for the dial-a-ride problem. Computers & Operations Research 40 (1), 490–497.

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

Qu, Y., Bard, J. F., 2013. The heterogeneous pickup and delivery problem with configurable vehicle capacity. Transportation Research Part C: Emerging Technologies 32, 1–20.

Qu, Y., Bard, J. F., 2015. A Branch-and-Price-and-Cut Algorithm for Heterogeneous Pickup and Delivery Problems with Configurable Vehicle Capacity. Transportation Science 49 (2), 254–270.

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

Ropke, S., Cordeau, J.-F., 2009. Branch and cut and price for the pickup and delivery problem with time windows. Transportation Science 43 (3), 267–286.

Ropke, S., Cordeau, J.-F., Laporte, G., 2007. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks 49 (4), 258–272.

Ropke, S., Pisinger, D., Nov. 2006. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. Transportation Science 40 (4), 455–472.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G., 2000. Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics 159 (2), 139–171.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: International Conference on Principles and Practice of Constraint Programming. Springer, pp. 417–431.

Subramanian, A., Uchoa, E., Ochi, L. S., 2013. A hybrid algorithm for a class of vehicle routing problems. Computers & Operations Research 40 (10), 2519 – 2531.

Taillard, E. D., 1999. A heuristic column generation method for the heterogeneous fleet VRP. RAIRO-Operations Research 33 (1), 1–14.

Tang, J., Kong, Y., Lau, H., Ip, A. W. H., 2010. A note on "Efficient feasibility testing for dial-a-ride problems". Operations Research Letters 38 (5), 405–407.

Toth, P., Vigo, D., 1997. Heuristic algorithms for the handicapped persons transportation problem. Transportation Science 31 (1), 60–71.