OR Spectrum (2012) 34:593–633
DOI 10.1007/s00291-010-0229-9

REGULAR ARTICLE

Models and algorithms for the heterogeneous dial-a-ride problem with driver-related constraints

Sophie N. Parragh · Jean-François Cordeau · Karl F. Doerner · Richard F. Hartl

Published online: 9 November 2010

Abstract This paper introduces models and algorithms for a static dial-a-ride problem arising in the transportation of patients by non-profit organizations such as the Austrian Red Cross. This problem is characterized by the presence of heterogeneous vehicles and patients. In our problem, two types of vehicles are used, each providing a different capacity for four different modes of transportation. Patients may request to be transported either seated, on a stretcher or in a wheelchair. In addition, some may require accompanying persons. The problem is to construct a minimum-cost routing plan satisfying service-related criteria, expressed in terms of time windows, as well as driver-related constraints expressed in terms of maximum route duration limits and mandatory lunch breaks. We introduce both a three-index and a set-partitioning formulation of the problem. The linear programming relaxation of the latter is solved by a column generation algorithm. We also propose a variable neighborhood search heuristic. Finally, we integrate the heuristic and the column generation approach into a collaborative framework. The column generation algorithm and the collaborative framework provide tight lower bounds on the optimal solution values for small-to-medium-sized

S. N. Parragh (✉) · K. F. Doerner · R. F. Hartl
Department of Business Administration, University of Vienna,
Bruenner Strasse 72, 1210 Vienna, Austria
e-mail: Sophie.Parragh@univie.ac.at

K. F. Doerner
e-mail: Karl.Doerner@univie.ac.at

R. F. Hartl
e-mail: Richard.Hartl@univie.ac.at

J.-F. Cordeau
Canada Research Chair in Logistics and Transportation and CIRRELT, HEC Montréal,
3000, chemin de la Côte-Sainte-Catherine, Montréal H3T 2A7, Canada
e-mail: Jean-Francois.Cordeau@hec.ca

instances. The variable neighborhood search algorithm yields high-quality solutions for realistic test instances.

Keywords Passenger transportation · Dial-a-ride-problem · Column generation · Variable neighborhood search

1 Introduction

Our study is motivated by a static problem faced by non-profit organizations such as the Austrian Red Cross (ARC) in the field of patient transportation. These organizations have to devise daily routing plans for their ambulances to serve a set of transportation requests that have been formulated in advance of the planning. Each request is defined by origin and destination locations, a number of passengers to be transported, and a time at which the passengers should either be picked-up at the origin or dropped-off at the destination.

The ARC disposes of vehicles of two different types, each type providing a different capacity for four modes of transportation: staff seat, patient seat, stretcher and wheelchair. Each patient may demand to be transported either seated, on a stretcher, or in a wheelchair. Accompanying persons may also be present and seating room for them has to be provided on the vehicle. Accompanying persons are allowed to use a staff seat. They may also use a patient seat or sit on the stretcher. Patients that can be transported seated are not allowed to use a staff seat. They can, however, be transported on the stretcher in the case where there are no more patient seats available. Patients that have to be transported on a stretcher or in a wheelchair can only be transported in the corresponding mode. Finally, some users require the presence of a second staff member aboard the vehicle.

The ambulance dispatcher has to assign drivers to vehicles (there are usually fewer drivers than available vehicles). Thus, also driver-related constraints have to be taken into account in the planning process. These constraints refer to maximum shift lengths and mandatory breaks. Non-profit organizations such as the ARC have to ensure that their expenses are kept low while providing a reasonably high service level. Thus, the aim is to construct a routing plan that is of minimum routing cost. This plan has to respect service-related criteria, expressed in terms of time windows, as well as labor regulations. The resulting problem can be defined as a heterogeneous dial-a-ride-problem with driver-related constraints (HDARPD).

To address this complex static routing problem, we propose a Variable Neighborhood Search (VNS) heuristic relying on several neighborhood operators. We also introduce a column generation algorithm based on a set-partitioning formulation of the problem and we discuss how the heuristic and the column generation approach can be integrated into a collaborative framework. The lower bounds computed by means of the latter two methods are used to assess the quality of the solutions produced by the heuristic for small- and medium-sized instances.

The contributions of this work are fourfold. First, we introduce a formulation for the HDARPD that incorporates routing and driver deployment decisions. This integration is made possible by the fact that the time frame of daily regular patient transportation corresponds to the drivers' work shifts. Second, a column generation algorithm is proposed. The central new aspect lies in the way the maximum route duration limit together with a non-empty time window at the start depot is treated in the dynamic programming part. Third, an efficient meta-heuristic algorithm tailored to the problem is presented. Fourth, the integration of the two methods into a collaborative scheme is investigated.

In the collaborative scheme, the proposed VNS supports the column generation algorithm in finding additional columns. Every ten column generation iterations, the VNS tries to improve the current best lower bound solution. If this solution is fractional, it is first transformed into an integer solution before running the VNS. When an improved solution is found by the VNS, the associated columns are transferred to the column generation algorithm. In contrast to the pricing heuristics used in the column generation framework, which use reduced costs to generate additional columns, the VNS uses the actual routing costs.

The remainder of the paper is organized as follows. Section 2 discusses related literature and Sect. 3 introduces some notation and two mathematical formulations of the problem. In Sect. 4, we then explain how the column generation subproblem can be solved both by exact and heuristic pricing procedures. These procedures are used within a column generation framework which is described in Sect. 5. This is followed by the description of the VNS heuristic in Sect. 6, by computational experiments in Sect. 7, and by the conclusion.

2 Related work

Dial-a-ride problems (DARP) are generalizations of pickup and delivery problems with time windows (PDPTW). In the DARP, people are being transported instead of goods. This gives rise to the issue of service quality which can be ensured either through additional constraints or with extra terms in the objective function. The latter approach is followed, e.g., by Parragh et al. (2009), while we adopt the first approach in this paper.

Like the majority of the works published on the DARP, we address here its static version. Although the DARP often is partly dynamic, a large portion of the requests are usually known in advance. In addition, a good algorithm for the static case can often be used as a basis to develop an algorithm for the dynamic case (Berbeglia et al. 2010).

Previous publications considering heterogeneous versions of the DARP involve, e.g., the work of Toth and Vigo (1997). The heterogeneity considered by these authors refers to two modes of transportation (seated passengers and passengers in wheelchairs) and to several different types of vehicles. The authors have devised a parallel insertion heuristic and a tabu thresholding algorithm for this problem. Another heterogeneous version of the DARP has been described by Melachrinoudis et al. (2007). They developed a tabu search heuristic for a problem with several different types of vehicles in terms of capacity limits but only one mode of transportation. Heterogeneous vehicles in terms of capacity are also considered by Rekiek et al. (2006) who introduced a grouping genetic algorithm.

In the context of a dynamic environment, Beaudry et al. (2009) have adapted the tabu search heuristic of Cordeau and Laporte (2003) to solve a heterogeneous DARP that arises in large hospitals. It involves transportation requests requiring three different modes of transportation (seated, on a bed, or in a wheelchair) and several different types of vehicles. Hanne et al. (2009) report on a computer-based planning system for a dynamic problem in a large German hospital. They consider hospital-specific constraints such as multi-dimensional capacities.

For overviews of the DARP and the closely related PDPTW, we refer the reader to Berbeglia et al. (2007), Cordeau and Laporte (2007) and Parragh et al. (2008a,b).

In related work, Xu et al. (2003) have addressed a practical pickup and delivery problem with multiple vehicle types, multiple time windows, and compatibility constraints between requests as well as between requests and vehicles. In addition, first-in-first-out loading requirements and driver work rules are taken into account. This problem is solved by a heuristic column generation algorithm in which several pricing heuristics are employed. In their experiments, the solution obtained at the root node is often integer. If not, a Mixed Integer Program (MIP) is solved on the set of generated columns. Column generation integrated into a branch-and-cut framework was also successfully applied to the standard PDPTW by Ropke and Cordeau (2009). This algorithm outperforms an earlier branch-and-cut algorithm by Ropke et al. (2007) in terms of the maximum problem size that can be solved. These results indicate that the use of a column generation-based algorithm is a promising direction. Finally, a combination of column generation with a local search method has recently been studied by Danna and Lepape (2005) in the form of a so-called cooperation scheme.

In an earlier study, Parragh (2010) has analyzed and solved a restricted version of the HDARPD, focusing solely on heterogeneous users and fleet aspects. For this simpler variant, state-of-the-art branch-and-cut algorithms for the DARP (Cordeau 2006; Ropke et al. 2007) were successfully adapted. In the present work, maximum user ride times are not considered explicitly. They are considered implicitly in terms of time windows at both the pick-up and the drop-off locations. This modification makes the application of column generation possible and allows us to solve much larger instances.

3 Problem formulation

In the following the basic notation needed to formulate the HDARPD is given. Thereafter, two different problem formulations are presented: a three-index model and a more compact set-partitioning formulation. The latter will serve as the basis for the proposed column generation framework.

3.1 Notation

The HDARPD is modeled on a complete directed graph $G = (V, A)$, where V is the set of vertices and A the set of arcs. The vertices correspond to the depot and the pickup and delivery locations. To each arc, (i, j) are associated a non-negative travel cost c_{ij} and a non-negative travel time t_{ij}. Arc costs and times are based on road network data.

A set of n customer requests, each consisting of a pickup and delivery location pair $\{i, n+i\}$, has to be served. The set of pickup locations is denoted by $P = \{1, \dots, n\}$ and the set of delivery locations by $D = \{n+1, \dots, 2n\}$.

At every pickup vertex one patient has to be transported and this patient may demand one of three different transportation modes. For every transportation mode $r \in R$, where $R = \{0, 1, 2, 3\}$ denotes the set of transportation modes, let q_i^r denote the load of passenger i with respect to mode r. Passengers may have to be transported seated ($q_i^1 = 1$), on a stretcher ($q_i^2 = 1$), or in a wheelchair ($q_i^3 = 1$). Each patient may also require an accompanying person ($q_i^0 = 1$). The demand at every delivery vertex is equal to $q_{n+i}^r = -q_i^r$ for all $r \in R$. In addition to the driver, a patient may need a second staff member to be on the vehicle. These are referred to as "attendants" in the following. The binary parameter a_i represents this requirement. It takes value 1 if an attendant is needed, and 0 otherwise. Furthermore, let V_a be the set of all vertices demanding an attendant on board the vehicle. Attendants occupy staff seats when they are on the vehicle.

Every user specifies a time window $[e_i, l_i]$ either for the pick-up (origin) or for the drop-off (destination) location, and the beginning of service has to start within this time window. If a vehicle arrives too early, it has to wait until service is possible. A maximum passenger ride time L is implicitly considered in order to provide reasonable service quality. This is done by artificially constructing a time window at the origin (resp. destination) relative to the time window given at the corresponding destination (resp. origin): in the case of an outbound request (a time window is given for the destination), the time window at the origin i is set to $e_i = \max\{0, e_{n+i} - L - d_i\}$ and $l_i = \min\{l_{n+i} - t_{i,n+i} - d_i, \bar{H}\}$; $\bar{H}$ denotes the end of the planning horizon. In the case of an inbound request (a time window is given for the origin), the destination time window is set to $e_{n+i} = e_i + d_i + t_{i,n+i}$ and $l_{n+i} = \min\{l_i + d_i + L, \bar{H}\}$. The parameter d_i denotes the service time at vertex i.

A set K of m heterogeneous vehicles is available to serve the transportation requests. Each vehicle $k \in K$ is associated with constants $C^{r,k}$. They give the amount of resource r available on the vehicle. The ARC disposes of two basic vehicle types. Type 1 (T1) provides one staff seat, six patient seats, and one wheelchair place. Type 2 (T2) provides two staff seats, one patient seat, one stretcher, and one wheelchair place. Patients demanding to be transported seated may use a patient seat or the stretcher. Patients demanding a stretcher can only be transported on a stretcher. The same applies to wheelchair passengers. Accompanying persons, however, may use a staff seat or a patient seat. They may also use the stretcher if no other transportation mode is available.

Each route starts at the depot location 0 within a prespecified time window and it finishes at the end depot $2n+2$, respecting a route duration limit T. This limit is based on Austrian labor regulations. Driver working shifts are limited to 8.5 h per day including a (lunch) break of $H = 30$ min. The lunch break has to start within a given time window $[e_H, l_H]$. It can be held at every vertex. In addition, only a certain number of drivers m_d (usually $m_d < m$) and only a limited number of attendants (in Austria these are employees serving their alternative service) m_a are available. An attendant can only work during morning or afternoon periods on a vehicle. An attendant working in the morning has to return to the noon depot $2n+1$ within a certain time window $[e_{2n+1}, l_{2n+1}]$. An attendant working in the afternoon has to be picked-up at the noon depot within the time window. If there are more drivers available than actually needed to serve all requests, the excess drivers may be employed for attendant duties. In this case, they only serve half of the day on a vehicle.

Thus, the set of all vertices is given by $V = P \cup D \cup \{0, 2n+1, 2n+2\}$, and the set of all arcs by $A = \{(i, j) : i \in V \backslash \{2n+2\}, j \in V \backslash \{0\}, i \neq j\}$.

3.2 A three-index formulation

We now introduce a three-index formulation based on the following binary decision variables:

$$x_{ij}^k = \begin{cases} 1, & \text{if arc } (i, j) \text{ is traversed by vehicle } k, \\ 0, & \text{otherwise,} \end{cases}$$

$$z_0^k = \begin{cases} 1, & \text{if a driver is assigned to vehicle } k, \\ 0, & \text{otherwise,} \end{cases}$$

$$z_1^k = \begin{cases} 1, & \text{if an attendant is assigned to vehicle } k \text{ in the morning,} \\ 0, & \text{otherwise,} \end{cases}$$

$$z_2^k = \begin{cases} 1, & \text{if an attendant is assigned to vehicle } k \text{ in the afternoon,} \\ 0, & \text{otherwise,} \end{cases}$$

$$v_i^k = \begin{cases} 1, & \text{if the lunch break is held at vertex } i, \\ 0, & \text{otherwise.} \end{cases}$$

In addition, we let $u \in \{0, \ldots, m_d\}$ denote the number of drivers that serve as additional attendants; B_i^k denotes the beginning of service of vehicle k at vertex i and $Q_i^{r,k}$ denotes the load of the vehicle with respect to resource r when leaving vertex i. Finally, let W_H^k represent the waiting time until the lunch break on vehicle k.

The objective considered is the minimization of total routing costs:

$$\min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}^k. \tag{1}$$

For ease of exposition, we introduce the constraints of the model in several groups. The first group contains the demand constraints and defines the basic structure of the routes:

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1 \quad \forall i \in P, \tag{2}$$

$$\sum_{j \in V} x_{ij}^k - \sum_{j \in V} x_{n+i,j}^k = 0 \quad \forall i \in P, k \in K, \tag{3}$$

$$\sum_{i\in V} x_{ij}^k - \sum_{i\in V} x_{ji}^k = 0 \quad \forall j \in P \cup D \cup \{2n+1\}, k \in K, \tag{4}$$

$$\sum_{j\in V} x_{0j}^k = z_0^k \quad \forall k \in K, \tag{5}$$

$$\sum_{i\in V} x_{i,2n+2}^k = z_0^k \quad \forall k \in K, \tag{6}$$

$$x_{ij}^k \in \{0,1\} \quad \forall i, j \in V, k \in K, \tag{7}$$

$$z_0^k \in \{0,1\} \quad \forall k \in K. \tag{8}$$

Constraints (2) ensure that each request is served exactly once while (3) ensure that each origin–destination pair is visited by the same vehicle. Flow conservation is imposed by equalities (4). The subsequent equalities (5) and (6) guarantee that, if a driver is assigned to a vehicle, the vehicle starts at and returns to the depot.

Attendant-related conditions form another block of constraints:

$$\sum_{i\in V} x_{i,2n+1}^k = \max\left\{z_1^k, z_2^k\right\} \quad \forall k \in K, \tag{9}$$

$$Q_0^{0,k} \geq z_1^k \quad \forall k \in K, \tag{10}$$

$$x_{i,2n+1}^k = 1 \Rightarrow Q_{2n+1}^{0,k} \geq Q_i^{0,k} - z_1^k + z_2^k \quad \forall i \in V, k \in K, \tag{11}$$

$$x_{i,2n+1}^k = 1 \Rightarrow Q_{2n+1}^{r,k} \geq Q_i^{r,k} \quad \forall i \in V k \in K, r \in R\setminus\{0\}, \tag{12}$$

$$z_1^k + z_2^k \geq a_i \sum_{j\in V} x_{ij}^k \quad \forall i \in V, k \in K, \tag{13}$$

$$z_1^k, z_2^k \in \{0,1\} \quad \forall k \in K. \tag{14}$$

Equalities (9) make sure that the noon depot is used if an attendant is assigned to the corresponding vehicle for morning or afternoon periods. This is necessary to ensure either the appropriate pick-up or drop-off of the attendant at the beginning or at the end of the shift. An attendant assigned to a vehicle requires a staff seat and this is modeled in inequalities (10)–(12). Furthermore, a user demanding an attendant aboard the vehicle can only be visited if an attendant is on the vehicle, which is reflected by constraints (13).

Consistency of resource and load variables is guaranteed by the following constraints:

$$x_{ij}^k = 1 \Rightarrow Q_j^{r,k} \geq Q_i^{r,k} + q_j^r \quad \forall i \in V, j \in V\setminus\{2n+1\}, k \in K, r \in R, \tag{15}$$

$$\sum_{r'=r}^{2} Q_i^{r',k} \leq \sum_{r'=r}^{2} C^{r',k} \quad \forall i \in V, k \in K, r \in R\setminus\{3\}, \tag{16}$$

$$Q_i^{3,k} \leq C^{3,k} \quad \forall i \in V, k \in K, \tag{17}$$

$$Q_i^{r,k} \geq 0 \quad \forall i \in V, k \in K, r \in R. \tag{18}$$

Inequalities (15) ensure load propagation from one vertex to the next. Upgrading constraints for resources 0, 1, and 2 are given by (16). They guarantee that capacity restrictions regarding these resources are not violated. They also ensure that each patient demanding resource 0, 1 or 2 can only be loaded if there is either enough capacity of the resource demanded or of another one with a higher number (0 = staff seat, 1 = patient seat, 2 = stretcher). Finally, constraints (17) guarantee that patients demanding resource 3 (wheelchair place) can only be transported if there is enough capacity of resource 3.

Suppose an empty T2 vehicle (2 staff seats, 1 patient seat, 1 stretcher, 1 wheelchair place) visits two origin locations in a sequence. At each location a seated patient with an accompanying person has to be picked up. The first seated passenger fills the patient seat, the accompanying person the first staff seat. If it was not possible to have seated patients sit on the stretcher, the second origin location could not be visited. However, the given upgrading conditions allow seated passengers to sit on the stretcher. Therefore, origin location two can be visited: the seated passenger uses the stretcher and the accompanying person the second staff seat. Suppose the same two origin locations are visited by an empty T1 vehicle (1 staff seat, 6 patient seats, 1 wheelchair place). In this case the first seated patient again uses a patient seat and the accompanying person the staff seat. At the second location there is no more empty staff seat available. However, again due to the given upgrading possibilities, the accompanying person can use a patient seat. Thus, after visiting location two, three patient seats and the staff seat are occupied.

The next inequalities define the beginning of service for each vertex:

$$x_{ij}^k = 1 \wedge v_i^k = 0 \ \Rightarrow B_j^k \geq B_i^k + d_i + t_{ij} \quad \forall i, j \in V, k \in K, \tag{19}$$

$$x_{ij}^k = 1 \wedge v_i^k = 1 \ \Rightarrow B_j^k \geq B_i^k + d_i + t_{ij} + W_H^k + H \quad \forall i, j \in V, k \in K, \tag{20}$$

$$B_{n+i}^k \geq B_i^k \quad \forall i \in P, k \in K, \tag{21}$$

$$v_{2n+2}^k = 0 \ \Rightarrow B_{2n+2}^k - B_0^k \leq T \quad \forall k \in K, \tag{22}$$

$$v_{2n+2}^k = 1 \ \Rightarrow B_{2n+2}^k - B_0^k + W_H^k + H \leq T \quad \forall k \in K, \tag{23}$$

$$\sum_{i \in V} v_i^k \geq z_0^k \quad \forall k \in K, \tag{24}$$

$$v_i^k \in \{0, 1\} \quad \forall i \in V, k \in K. \tag{25}$$

If vertex i is chosen for the lunch break ($v_i^k = 1$), in addition to the service time associated with this vertex, the vehicle waits (see variable W_H^k) at this vertex until the lunch break time window starts. Then, it stays until the lunch break is concluded. Note that these constraints also take care of subtour elimination given that $t_{ij} + d_i > 0$ for all $i, j \in V : i \neq j$. Inequalities (21) ensure that every user's pick-up location is visited before the corresponding drop-off location. Total route duration is limited by (22) and (23). We distinguish two cases. In the first case (22), the end depot does not serve as the lunch break location. In the second case (23), the lunch break is held at the end depot; therefore, the respective route duration is increased by a possible waiting time until the lunch break and the duration of the lunch break itself. If a driver is assigned to a vehicle, then constraints (24) ensure that the associated route contains a break.

The different limits on the beginning of service are given in the following:

$$e_i \leq B_i^k \leq l_i \quad \forall i \in V, k \in K, \tag{26}$$

$$z_1^k = 1 \wedge z_2^k = 0 \Rightarrow B_0^k \leq B_i^k \leq B_{2n+1}^k \quad \forall i \in V_a, k \in K, \tag{27}$$

$$z_1^k = 0 \wedge z_2^k = 1 \Rightarrow B_{2n+1}^k \leq B_i^k \leq B_{2n+2}^k \quad \forall i \in V_a, k \in K, \tag{28}$$

$$v_i^k = 1 \Rightarrow e_H \leq B_i^k + d_i + W_H^k \leq l_H \quad \forall i \in V, k \in K, \tag{29}$$

$$W_H^k \geq 0 \quad \forall k \in K. \tag{30}$$

Standard time window constraints are modeled by (26). Additional time-related constraints provide new bounds on the beginning of service at those vertices where an attendant is required. Bounds for morning periods are given by (27) and for afternoon periods in (28). If an attendant is present during both periods, no additional bounds are needed. Furthermore, constraints (29) guarantee that the lunch break starts within the lunch break time window $[e_H, l_H]$.

Finally, inequalities (31) and (32) limit the number of drivers and attendants that can be assigned to vehicles. Each driver who is appointed to attendant duties, instead of driving, can be employed for either morning or afternoon periods. Thus, the number of attendants available is increased and the number of drivers is decreased by the same amount u:

$$\sum_{k \in K} z_0^k \leq m_d - u, \tag{31}$$

$$\sum_{k \in K} z_1^k + \sum_{k \in K} z_2^k \leq m_a + u, \tag{32}$$

$$u \in \{0, \ldots, m_d\}. \tag{33}$$

3.3 A set-partitioning formulation

The HDARPD can be reformulated in a more compact way. Let $\mathcal{T}$ denote the set of available vehicle types and Ω_t the set of feasible routes for vehicles of type $t \in \mathcal{T}$. Let also Ω be the set of all feasible routes, i.e., $\Omega = \bigcup_{t \in \mathcal{T}} \Omega_t$. Furthermore, let m_t denote the number of available vehicles of type t. For each route $\omega \in \Omega$, let c_ω be the cost of the route. The constants $b_{i\omega}$ and g_ω represent the number of times vertex $i \in P$ is traversed by ω and the number of attendants needed by route ω, respectively. Finally, variable y_ω takes value 1 if and only if route ω is used in the solution. The problem can thus be formulated as the following set-partitioning problem (SP):

$$\min \sum_{\omega \in \Omega} c_\omega y_\omega \tag{34}$$

subject to

$$\sum_{\omega \in \Omega} b_{i\omega} y_\omega = 1 \quad \forall i \in P, \tag{35}$$

$$\sum_{\omega \in \Omega_t} y_\omega \leq m_t \quad \forall t \in \mathcal{T}, \tag{36}$$

$$\sum_{\omega \in \Omega} y_\omega \leq m_d - u, \tag{37}$$

$$\sum_{\omega \in \Omega} g_\omega y_\omega \leq m_a + u, \tag{38}$$

$$u \geq 0, \tag{39}$$

$$y_\omega \in \{0, 1\} \quad \forall \omega \in \Omega. \tag{40}$$

The objective function (34) minimizes the total cost of the selected routes. Constraints (35) guarantee that every request is served exactly once. Inequalities (36) limit the number of vehicles of type t that can be used in the solution. Constraint (37) ensures that at most as many vehicles are used as there are drivers. Drivers that are not needed in routes may be employed as additional attendants, thus increasing the number of attendants that can be used by u in constraint (38).

To compute a lower bound, the linear programming relaxation of SP (LSP) can be solved, where LSP is obtained by replacing (40) with

$$y_\omega \geq 0 \quad \forall \omega \in \Omega. \tag{41}$$

Due to the large size of Ω the above formulation will not be solved directly. Instead, a restricted version of this problem, considering only a small subset of columns $\Omega' \subset \Omega$, will be solved. The set Ω' is generated by solving LSP using column generation. In column generation LSP decomposes into a (restricted) master problem and $|\mathcal{T}|$ subproblems, one for each vehicle type. Let π_i, σ_t, λ, and ϕ be the dual variables associated with constraints (35) for index i, with constraints (36) for index t, and with constraint (37) and (38), respectively. After having solved the restricted master problem, their values can be retrieved and used to compute the reduced cost of a given route. A negative reduced cost indicates that the corresponding column may improve the current solution if added to the restricted master problem. In our case, the reduced cost of column y_ω corresponding to route $\omega \in \Omega_t$ is given by

$$\bar{c}_\omega = c_\omega - \sum_{i \in P} b_{i\omega} \pi_i - \sigma_t - \lambda - g_\omega \phi, \tag{42}$$

where c_ω gives the actual routing cost of route ω. These costs are reduced by the dual variable values π_i of all requests i which are part of ω. In addition, they are reduced by the value of σ_t, which is associated with the vehicle type t of route ω (see constraints (36)). Finally, c_ω is also reduced by the values of λ and ϕ, where ϕ has to be multiplied by the number of attendants employed on route ω (see constraint (38)).

Thus, subproblem t corresponds to finding a single vehicle route $\omega \in \Omega_t$ for a vehicle of type t such that its reduced cost $\bar{c}_\omega$ is minimum. It is subject to constraints (3)-(30), omitting superscript k, setting $z_0 = 1$, and replacing k by t in the case of the capacities $C^{r,t}$. The index t refers to the vehicle type.

In the next section, we explain how columns of negative reduced costs can be identified by pricing algorithms. The overall column generation framework is then described in Sect. 5.

4 Solving the column generation subproblem

In order to find negative reduced cost columns we implement a label setting shortest path algorithm and several heuristics. The use of heuristics, aiding the exact procedure in finding negative reduced cost paths, may yield significant run time reductions (see Savelsbergh and Sol 1998). Following the findings of Ropke and Cordeau (2009) in the context of the PDPTW, our label setting algorithm considers only elementary paths. This implies that every vertex can be visited at most once in a path. It requires storing and processing additional information during the solution of the subproblem. However, usually it yields better results than the use of a non-elementary shortest path problem because it provides stronger lower bounds.

In the following the elementary shortest path algorithm will first be described in detail. Thereafter, the heuristics will be briefly discussed.

4.1 The label setting shortest path algorithm

Label setting shortest path algorithms are used to compute the shortest paths between source and sink nodes. On every arc that is used on the path from the source to the sink, resources are consumed. A label stores the following information: the node it is associated with, the resource consumption until that node, and a pointer to its parent label. It thus represents a path starting at the source node and ending at the node it is associated with, characterized by a certain resource consumption. The labeling algorithm implemented here is based on the one described by Ropke and Cordeau (2009) for solving the elementary shortest path problem with time windows, capacity, and pickup and delivery. The subproblem we have to solve is also a constrained shortest path problem. It can be described as an elementary shortest path problem with time windows, multiple capacities, pickup and delivery, and route duration constraints.

4.1.1 Lunch break and attendant requirements

To properly treat the lunch break requirements and the presence of attendants on the vehicle, three additional artificial vertices are introduced in the graph: one denoted by $2n + 3$ for the morning attendant, one denoted by $2n + 4$ for the afternoon attendant, and one denoted by $2n + 5$ for the lunch stop. If a path contains the morning attendant vertex, an attendant is aboard the vehicle during the morning shift. Similarly, if the afternoon attendant vertex is part of the path, an attendant is assigned to the vehicle in the afternoon. Every path has to contain the lunch vertex, with one exception: if the vehicle returns to the end depot before the end of the lunch time window and the lunch has not yet taken place, it is assumed that it is held at the end depot. The vertex that is followed by the lunch vertex on the constructed graph is the one where the lunch break will be held.

In the graph, vertex $2n+3$ can only be visited from the origin depot 0 and $2n+4$ only from the noon depot $2n+1$. Travel times for these vertices are set to $t_{0,2n+3} = t_{2n+1,2n+4} = 0$, $t_{2n+3,j} = t_{0,j}$ and $t_{2n+4,j} = t_{2n+1,j}$ for all j. In the case of the lunch vertex, travel times from all vertices to this vertex are set to $t_{i,2n+5} = 0$ for all i. Let now i_{2n+5} denote the vertex visited directly before the lunch vertex. The travel times from the lunch vertex are dynamically set to $t_{2n+5,j} = t_{i_{2n+5},j}$. The time windows of the attendant pickup vertices are set to the beginning and the end of the planning horizon, respectively. In the case of the lunch vertex a time window is given, i.e., $e_{2n+5} = e_H$ and $l_{2n+5} = l_H$ (see above). The service times at the attendant vertices are set to $d_{2n+3} = d_{2n+4} = 0$ and to $d_{2n+5} = H$ in the case of the lunch vertex. The load is set to $q^0_{2n+3} = q^0_{2n+4} = 1$ at the attendant vertices and to zero for all other resources of artificial vertices.

4.1.2 Label management

For each label the following data are stored: η the vertex of the label, δ the departure time at η, Q^r_{cum} the cumulative load of resource r when leaving vertex η, c_{cum} the accumulated cost up to vertex η, $b \in \{0, 1\}$ whether a lunch stop has already taken place or not, $\alpha \in \{0, 1\}$ whether an attendant is aboard the vehicle or not, $o \in \{0, 1\}$ whether the noon depot has already been visited or not, $\mathcal{V} \subseteq \{0, \ldots, 2n+5\}$ the set of vertices visited along the path, $\mathcal{O} \subseteq \{1, \ldots, n\}$ the set of open requests, f the forward time slack, w_{cum} the accumulated waiting time, and a pointer to its parent label. The resources f and w_{cum} are needed to check whether the route duration limit can be respected; f gives the maximum time the departure at the noon depot can be shifted forward in time. Section 4.1.4 will explain these resources in further detail.

The extension of a label κ along an arc $(\eta(\kappa), j)$ is only possible if the following holds:

$$\delta(\kappa) + t_{\eta(\kappa),j} \leq l_j, \tag{43}$$

$$Q^r_{\text{cum}}(\kappa) + q^r_j + \sum_{r'=r+1}^{2} q^{r'}_j \leq C^{r,t}_{\text{cum}} \quad \forall r \in R, \tag{44}$$

$$\alpha(\kappa) \geq a_j, \tag{45}$$

$$(1 - b(\kappa)) \left(\max\left\{\delta(\kappa) + t_{\eta(\kappa),j}, e_j\right\} + d_j\right) \leq l_H, \tag{46}$$

$$\max\left\{\max\left\{\delta(\kappa)+t_{\eta(\kappa),j}, e_j\right\}+d_j, (1 - b(\kappa))e_H\right\}+(1 - b(\kappa))H-e_0-F^0_j \leq T, \tag{47}$$

$$j \notin \mathcal{V}(\kappa). \tag{48}$$

Here, $C^{r,t}_{\text{cum}}$ is set to $C^{r,t} + \sum_{r'=r+1}^{2} C^{r',t}$. The final time slack F^0_j at vertex j is given by $F^0_j = \min\{\min[f(\kappa), l_j - (\delta(\kappa) + t_{\eta(\kappa),j}) + w_{\text{cum}}(\kappa)], w_{\text{cum}}(\kappa) + \max[0, e_j -$

$(\delta(\kappa) + t_{\eta(\kappa),j})]\}$, i.e., the minimum over all forward time slacks and the total waiting time until vertex j (see Sect. 4.1.4 for further details). Condition (43) ensures time window feasibility: the departure from the previous vertex $\eta(\kappa)$ plus the travel time from $\eta(\kappa)$ to j has to be smaller than or equal to the end of the time window at vertex j. According to condition (44) a path can only be extended along arc $(\eta(\kappa), j)$ if all loading restrictions are satisfied. Condition (45) states that if vertex j demands an attendant aboard the vehicle ($a_j = 1$), it can only be visited if an attendant is currently on the vehicle ($\alpha(\kappa) = 1$). If the lunch node is not yet part of the path, extension along arc $(\eta(\kappa), j)$ is only possible if it can still be feasibly inserted after vertex j. This is taken care of by (46). Feasibility with respect to route duration is guaranteed by condition (47). For further details on this issue we refer to Sect. 4.1.4. Finally, elementarity is ensured by (48).

Moreover, κ and j must comply with the following conditions:

$$j \in D \Rightarrow j - n \in \mathcal{O}(\kappa), \tag{49}$$
$$j = 2n + 2 \Rightarrow \mathcal{O}(\kappa) = \emptyset \wedge \alpha - o \leq 0. \tag{50}$$

Condition (49) ensures that if j is a delivery, it can only be visited if the request is open, i.e., the corresponding pickup has already been visited. Condition (50) ensures that a label can only be extended to the end depot $2n + 2$ if there are no more open requests. In addition, the noon depot must have been visited if an attendant is currently on the vehicle.

If a label can feasibly be extended along arc $(\eta(\kappa), j)$, a new label κ' is generated at vertex j:

$$\eta(\kappa') = j, \tag{51}$$
$$\delta(\kappa') = \max\left\{\delta(\kappa) + t_{\eta(\kappa),j}, e_j\right\} + d_j, \tag{52}$$
$$Q^0_{\text{cum}}(\kappa') = \begin{cases} Q^0_{\text{cum}}(\kappa) - 1 & \text{if } j = 2n+1 \wedge \alpha(\kappa) = 1, \\ Q^0_{\text{cum}}(\kappa) + q^0_j + \sum_{r'=1}^{2} q^{r'}_j & \text{otherwise,} \end{cases} \tag{53}$$
$$Q^r_{\text{cum}}(\kappa') = Q^r_{\text{cum}}(\kappa) + q^r_j + \sum_{r'=r+1}^{2} q^{r'}_j \quad \forall r \in R \backslash \{0\}, \tag{54}$$
$$c_{\text{cum}}(\kappa') = c_{\text{cum}}(\kappa) + \bar{c}_{\eta(\kappa),j}, \tag{55}$$
$$b(\kappa') = \begin{cases} 1 & \text{if } j = 2n+5, \\ b(\kappa) & \text{otherwise,} \end{cases} \tag{56}$$
$$\alpha(\kappa') = \begin{cases} 1 & \text{if } j \in \{2n+3, 2n+4\}, \\ 0 & \text{if } j = 2n+1, \\ \alpha(\kappa) & \text{otherwise,} \end{cases} \tag{57}$$
$$o(\kappa') = \begin{cases} 1 & \text{if } j = 2n+1, \\ o(\kappa) & \text{otherwise,} \end{cases} \tag{58}$$
$$\mathcal{V}(\kappa') = \mathcal{V}(\kappa) \cup \{j\}, \tag{59}$$

$$\mathcal{O}(\kappa') = \begin{cases} \mathcal{O}(\kappa) \cup \{j\} & \text{if } j \in P, \\ \mathcal{O}(\kappa) \setminus \{j-n\} & \text{if } j \in D, \\ \mathcal{O}(\kappa) & \text{otherwise,} \end{cases} \tag{60}$$

$$w_{\text{cum}}(\kappa') = w_{\text{cum}}(\kappa) + \max\left\{0, e_j - (\delta(\kappa) + t_{\eta(\kappa),j})\right\} \tag{61}$$

$$f(\kappa') = \min\left\{f(\kappa), w_{\text{cum}}(\kappa) + l_j - (\delta(\kappa) + t_{\eta(\kappa),j})\right\}. \tag{62}$$

At the origin depot (the first vertex along each path) these labels are initialized by setting $\delta(0) = e_0$, $Q^r_{\text{cum}}(0) = 0$ for all $r \in R$, $c_{\text{cum}} = 0$, $b(0) = 0$, $\alpha(0) = 0$, $o = 0$, $f(0) = l_0 - e_0$, and $w_{\text{cum}}(0) = 0$.

4.1.3 Dominance

The dominance criterion used here is similar to the one denoted by DOM1' by Ropke and Cordeau (2009). Let $\mathcal{U}(\kappa)$ denote the set of unreachable requests of label κ, where $\mathcal{U}(\kappa) = \mathcal{V} \cup \left\{i \in P : \delta(\kappa) + t_{\eta(\kappa),i} > l_i\right\}$. According to this criterion a label κ dominates another label κ' if

$$\eta(\kappa) = \eta(\kappa'), \delta(\kappa) \leq \delta(\kappa'), c_{\text{cum}}(\kappa) \leq c_{\text{cum}}(\kappa'), \mathcal{U}(\kappa) \subseteq \mathcal{U}(\kappa'), \mathcal{O}(\kappa) \subseteq \mathcal{O}(\kappa'). \tag{63}$$

In addition, in our case, the following has to hold:

$$f(\kappa) \geq f(\kappa'), w_{\text{cum}}(\kappa) \geq w_{\text{cum}}(\kappa'), b(\kappa) = b(\kappa'), \alpha(\kappa) = \alpha(\kappa'), o(\kappa) = o(\kappa'). \tag{64}$$

This implies that the amount of time by which the departure from the origin depot can be shifted forward in time in label κ has to be at least as large as in label κ'. If the lunch node or the noon depot have already been visited by the partial path represented by label κ the same has to be true for κ'. This also applies to whether an attendant is aboard the vehicle or not. Furthermore, we only apply the dominance check to labels with $\eta \in V$.

4.1.4 Time windows at the start depot and minimum route duration

Imposing a route duration limit together with a non-empty time window at the origin depot in a label setting algorithm requires further adjustments. One option, introduced by Desaulniers and Villeneuve (2000) and reviewed by Irnich (2008), consists in appending two resources to each label, coupled by a max-term, denoted as q and z. These are extended as follows:

$$q(\kappa') = t_{\eta(\kappa),j} + \max\left\{q(\kappa) - (t_{\eta(\kappa),j} + d_{\eta(\kappa)}), z(\kappa) - l_j\right\} \tag{65}$$

$$z(\kappa') = t_{\eta(\kappa),j} + \max\left\{z(\kappa), q(\kappa) - (t_{\eta(\kappa),j} + d_{\eta(\kappa)}) + e_j\right\}. \tag{66}$$

Here, we show that this is equivalent to using the forward time slack notion developed by Savelsbergh (1992).

The idea of Desaulniers and Villeneuve (2000) can be explained as follows. Let $U = \{0, 1, 2, \ldots, \tilde{n}\}$ denote a feasible path. The earliest possible departure time $\tilde{e}_j$ at node j when traveling along U can be calculated as

$$\tilde{e}_0 = e_0, \tag{67}$$
$$\tilde{e}_j = \max\left\{\tilde{e}_{j-1} + (d_{j-1} + t_{j-1,j}), e_j\right\} \quad \forall j \in \{1, \ldots, \tilde{n}\}. \tag{68}$$

Similarly, the latest arrival time $\tilde{l}_j$ at node j for which waiting can be avoided is computed by setting

$$\tilde{l}_0 = l_0, \tag{69}$$
$$\tilde{l}_j = \min\left\{\tilde{l}_{j-1} + (d_{j-1} + t_{j-1,j}), l_j\right\} \quad \forall j \in \{1, \ldots, \tilde{n}\}. \tag{70}$$

Let now U be a feasible (partial) path starting at the origin depot 0 and ending at node i. Then, let $s_i = \sum_{(k,l)\in U}(t_{kl} + d_k)$ denote the sum of actual travel times along U (including service time but excluding waiting time), the following two parameters can be defined:

$$\tilde{q}_i = s_i - \tilde{l}_i, \tag{71}$$
$$\tilde{z}_i = \max\{s_i, \tilde{q}_i + \tilde{e}_i\}. \tag{72}$$

Then the minimum duration of path U (ending at i) is equal to $\max\{\tilde{z}_i, \tilde{q}_i + \delta_i\}$, where δ_i denotes the departure time from vertex i.

Now it can be shown that $\tilde{q}_i$ is equivalent to the forward time slack defined by Savelsbergh (1992). Let $\tilde{f}_i^0$ denote the forward time slack from the origin depot to the end of the path U (here node i). It is the maximum amount of time by which the departure at the depot can be shifted forward without violating any other time window constraint. It is computed as follows (where B_j denotes the beginning of service at node j):

$$\tilde{f}_i^0 = \min_{0\le j\le i}\left\{l_j - \left[B_0 + \sum_{p=1}^{j}(d_{p-1} + t_{p-1,p})\right]\right\}. \tag{73}$$

Let us assume that $e_0 = 0$ and therefore $B_0 = 0$. We obtain,

$$\tilde{f}_i^0 = \min_{0\le j\le i}\left\{l_j - \sum_{p=1}^{j}(d_{p-1} + t_{p-1,p})\right\}. \tag{74}$$

Let us now rewrite $\tilde{l}_i$ as

$$\tilde{l}_i = \min_{0 \leq j \leq i} \left\{ l_j + \sum_{p=j+1}^{i} (d_{p-1} + t_{p-1,p}) \right\}, \tag{75}$$

and replace $\tilde{l}_i$ by this formulation in $\tilde{q}_i$. We obtain

$$\tilde{q}_i = \sum_{p=1}^{i} (d_{p-1} + t_{p-1,p}) - \min_{0 \leq j \leq i} \left\{ l_j + \sum_{p=j+1}^{i} (d_{p-1} + t_{p-1,p}) \right\} \tag{76}$$

$$= - \min_{0 \leq j \leq i} \left\{ l_j - \sum_{p=1}^{j} (d_{p-1} + t_{p-1,p}) \right\}. \tag{77}$$

This shows that $\tilde{q}_i = -\tilde{f}_i^0$. Furthermore, let us examine the minimum route duration time as defined by Desaulniers and Villeneuve (2000). It is given by $\max\{\tilde{z}_i, \tilde{q}_i + \delta_i\}$. We still assume that $B_0 = e_0 = 0$. By substituting $\tilde{z}_i$ this term can be rewritten as $\max\{s_i, \tilde{q}_i + \tilde{e}_i, \tilde{q}_i + \delta_i\} = \max\{s_i, \tilde{q}_i + \max(\tilde{e}_i, \delta_i)\}$. If δ_i is computed correctly it will always be greater than or equal to $\tilde{e}_i$ in our case. Therefore, $\max\{s_i, \tilde{q}_i + \delta_i\}$ is equivalent to the above expression. Now if we denote by $\tilde{w}_i$ the accumulated waiting time until node i, it is easy to see that $\delta_i - \tilde{w}_i = s_i$. Thus, $\max\{s_i, \tilde{q}_i + \delta_i\} = \delta_i - \min(-\tilde{q}_i, \tilde{w}_i) = \delta_i - \min(\tilde{f}_i^0, \tilde{w}_i)$. This corresponds to what Cordeau and Laporte (2003) use to compute the minimum duration of a given route.

Thus, to handle a time window at the start depot together with a route duration limit, the notion of forward time slack defined by Savelsbergh (1992) can be used. In order to do so, we use the additional resources w_{cum} and f (see above). To generalize what has been shown to the case of $e_0 > 0$, these resources are initialized, as already pointed out, with $f(0) = l_0 - B_0$ ($B_0 = e_0$) and $w_{\text{cum}}(0) = 0$.

4.1.5 Label elimination

Labels that are currently in the queue of labels to be processed can be eliminated if the deliveries of open requests cannot be reached in a feasible path. As in Ropke (2005) and Ropke and Cordeau (2009) we consider sets of one and two deliveries and one set of three deliveries. The last set consists of the following three vertices: the delivery that is farthest away from the current vertex; the delivery that is farthest away from these two; and the delivery that is farthest away from the current vertex and the two previously selected deliveries. If for one of these sets no path can be found that serves all deliveries in the set in a feasible way, the label can be eliminated.

In addition to these sets of deliveries, we check whether there is an attendant on the vehicle and if the noon depot has not been visited yet. If this is the case and the noon depot cannot be reached in a feasible way the label can also be eliminated.

4.2 Heuristic algorithms

To accelerate the column generation process several heuristic procedures are used to generate negative reduced cost columns. These can be divided into two classes; those based on the labeling algorithm, and those that rely on simple construction and improvement principles.

4.2.1 Heuristics based on the labeling algorithm

Two of the heuristic algorithms used to generate columns are based on the exact labeling algorithm. Both are also used by Ropke and Cordeau (2009) in the context of the PDPTW. The first heuristic (*LimLabels*) simply limits the number of labels that can be in the queue of unprocessed labels at any time. At first the limit is set to 500. If no negative reduced cost column can be found with this limit, it is increased to 1,000. If again no negative reduced cost columns are generated the limit is set to 2,000. The second heuristic (*LimGraph*) applies the exact labeling algorithm on a reduced graph. Two reduced graphs are used. In Graph G_5 every pickup vertex and every delivery vertex is only connected to the five closest pickup vertices and the five closest delivery vertices. In addition, if not already present, connections between each pickup vertex and its corresponding delivery vertex are added. Furthermore, the start depot is connected to all pickup vertices and the morning attendant node. The noon depot is connected to the afternoon attendant node. All delivery vertices are connected to the end depot. All vertices except the end depot are connected to the lunch node and the noon depot. The morning attendant node, the afternoon attendant node and the lunch node are connected to all vertices except the start depot. The second reduced Graph G_{10} is constructed in the same way. However, instead of the five closest pickups and deliveries, it considers the ten closest. Using the five or ten closest pickups and the five or ten closest deliveries seems to provide a good balance between computing time and the quality of the columns identified by the heuristic.

4.2.2 Construction/improvement based heuristics

The four remaining pricing heuristics use construction or improvement algorithms. Like the labeling heuristics described above, they are based on those used by Ropke and Cordeau (2009) to solve the PDPTW. *ConstrHeur* is a simple construction heuristic that starts from every pickup and delivery vertex pair and iteratively adds requests by means of a best insertion criterion regarding reduced cost. *RandConstrHeur* is also a construction heuristic but here randomized best insertion is used to insert additional requests. The randomization process favors requests that least increase the reduced costs of the partial route. In both construction heuristics, every time a new request is inserted, the resulting route is checked to see whether it has a negative reduced cost. If this is the case, the corresponding column is generated. After the check, the route undergoes local search based improvement (see Sect. 6.3). The local search algorithm minimizes the actual routing cost and it only considers moves that yield a feasible route. If this results in another negative reduced cost route, the corresponding column is again added to the pool. This additional local search phase has not been used by

Ropke and Cordeau (2009). Preliminary experiments showed that it is beneficial with respect to solution quality. Heuristic *LNSCurrBasis* applies Large Neighborhood Search (LNS) (Shaw 1998) on the routes in the current basis. It works as follows. In a removal step, up to 50% of the requests forming the route are randomly removed from it. In an insertion step, requests are reinserted using randomized best insertion as described above. These two steps are repeated until no further improvement can be found. Between 15 and 20 non-improving steps are performed. Finally, *LNSRandConstr* simply improves all solutions obtained by the randomized construction algorithm with LNS.

5 The column generation framework

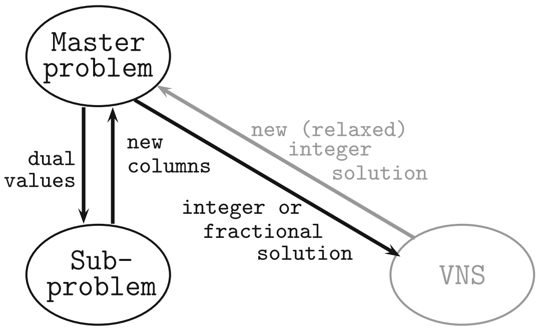
The overall column generation framework that we use to solve the HDARPD is summarized in Algorithm 1. During the initialization phase initial columns are generated (Sect. 5.1) and added to Ω'. In addition, a number of pre-processing steps (Sect. 5.2) are applied. Then, LSP is solved on Ω' and the dual variable values associated with the different constraints are retrieved. Based on these values, negative reduced cost columns are generated using the different heuristics. They are invoked in a certain sequence (Sect. 5.3). If all fail, then the exact dynamic programming procedure is called. All new negative reduced cost columns are added to Ω' and LSP is solved again. This is repeated until no new negative reduced cost column can be identified. In this case, the optimal solution to LSP has been found. Additionally, every ten iterations the proposed VNS heuristic (Sect. 6) is applied to the current solution of LSP. This process is represented in Fig. 1. The so-called collaborative scheme is described in further detail in Sect. 5.4.

Algorithm 1 The column generation framework

```
initialization generate initial columns by VNS (Sect. 6), introduce artificial columns and add them to the
current set of routes or columns $\Omega'$, do pre-processing (graph pruning, time window tightening; Sect. 5.2)
repeat
  solve linear relaxation of set-partitioning problem (LSP) on $\Omega'$ (Sect. 3.3)
  collaborative scheme: every 10 iterations apply VNS to the current LSP solution (Sect. 5.4)
  generate new negative reduced cost columns heuristically (Sect. 4.2)
  if no negative reduced cost columns are found then
    run exact label setting algorithm to find negative reduced cost columns (Sect. 4.1)
  end if
until no more negative reduced cost columns can be found
return optimal solution of LSP
```

5.1 Initial columns

An initial set of columns is generated by means of a heuristic algorithm, namely a VNS. The VNS is described in detail in Sect. 6. A limit of 2×10^4 iterations is applied regardless of whether a feasible solution can be found within this time limit or not. Here, ascending moves are not allowed. In addition, one artificial column for each $i \in P$ is generated, having a coefficient of 1 in the row corresponding to the



Master problem
dual values
new columns
Sub-problem
new (relaxed) integer solution
integer or fractional solution
VNS

Fig. 1 The collaborative scheme

demand constraint for request i and zeros in all other rows. These columns are given a sufficiently large cost of M.

5.2 Pre-processing

Before starting the column generation process, several pre-processing steps are performed. These are mostly based on time window tightening and graph pruning techniques. They are based on those described in detail by Cordeau (2006). In addition, we use the cyclic time window tightening steps described by Desrochers et al. (1992) and Kallehauge et al. (2005):

$$e_k := \max\left\{e_k, \min[l_k, \min_{(i,k)}(e_i + (t_{ik} + d_i))]\right\}, \tag{78}$$

$$e_k := \max\left\{e_k, \min[l_k, \min_{(k,j)}(e_j - (t_{kj} + d_k))]\right\}, \tag{79}$$

$$l_k := \min\left\{l_k, \max[e_k, \max_{(i,k)}(l_i + (t_{ik} + d_i))]\right\}, \tag{80}$$

$$l_k := \min\left\{l_k, \max[e_k, \max_{(k,j)}(l_j - (t_{kj} + d_k))]\right\}. \tag{81}$$

These steps are repeated until no further time window tightening is possible. Note that only arcs are considered that have not been eliminated in previous pre-processing steps. Arcs that would lead to an infeasible solution regarding vehicle capacity restrictions are also removed from the graph.

5.3 Sequence of pricing heuristics

Instead of the fixed pricing heuristic sequence used by Ropke and Cordeau (2009) and which always invokes *ConstrHeur* first, we use a roulette wheel selection mechanism. Initially, only *ConstrHeur* can be chosen. Its score is set to one, while the score of all others is set to zero. Thereafter, in every column generation iteration, the score of the heuristic that obtained one or more new negative reduced cost columns is increased by one. Thus, its probability of being chosen as the first heuristic is increased. The heuristics are ordered as in Ropke and Cordeau (2009): *ConstrHeur–LNSCurrBasis–RandConstrHeur–LNSRandConstr–LimLabels– LimGraph*. If *ConstrHeur* is chosen as the first heuristic but it fails to yield a new negative reduced cost column for either vehicle types, we switch to *LNSCurrBasis*. If *LNSCurrBasis* fails we switch to *RandConstrHeur* and so on. If *LNSCurrBasis* is chosen as the first heuristic and fails to obtain negative reduced cost columns, only the heuristics following *LNSCurrBasis* in the list are tried in the above order. This applies to all heuristics. If also *LimGraph* (the last one on the list) fails to generate negative reduced cost columns, the exact label setting algorithm is started. If this procedure also fails to generate negative reduced cost columns, the optimal solution of the relaxed problem (LSP) has been found. If it is also integer the optimal solution of SP has been found. In all labeling algorithms we stop as soon as 50 negative reduced cost columns have been generated. All labeling algorithms use a sorted queue of unprocessed labels. Labels are ordered according to increasing reduced cost.

LSP is solved again on Ω' every time at least one new negative reduced cost column for either vehicle type could be generated. Every heuristic tries to find negative reduced cost columns for T1 and T2 vehicles in alternating order. If new negative reduced cost columns for T1 vehicles are generated their validity for T2 vehicles is checked. If they are valid, they are added to the column pool for T2 vehicles. Only then LSP is solved. This is not done if one of the heuristics yields new negative reduced cost columns for T2 vehicles; it is not very likely that columns for T2 vehicles are also valid for T1 vehicles.

5.4 Collaborative scheme

For every ten iterations (one iteration corresponds to finding one or more negative reduced cost columns, adding these columns to Ω', and solving LSP) the optimal solution of the current LSP is passed to the VNS. If there is still an artificial column in the basis the VNS resumes the search using the last incumbent of the previous run. If there are no more artificial columns in the basis, we distinguish between an integer and a fractional solution. In the former case, it is passed to the VNS as it is. In the latter case, all duplicate requests are removed before passing it on to the VNS. Duplicate requests are kept on the route that is associated with the y_ω closest to one. Empty routes are eliminated. In this case there can be more vehicles in use of a certain type than actually available. However, it is still passed to the VNS keeping all increased limits on the number of routes per vehicle type. The number of drivers available is set to $\bar{m}_d := \max(m_d, m_{cg} - 1)$ (m_{cg} gives the number of routes used after having removed duplicate requests). If $m_{cg} - 1 > m_d$, as many drivers as there are currently needed minus one are available. Otherwise, at most the original number of drivers m_d can be used. A new best solution generated by the VNS might thus be infeasible regarding two aspects: the number of vehicles of a certain vehicle type employed and the total number of drivers. All other constraints are respected. This entails that the resulting columns might not be combinable as such but they are all feasible. If the solution obtained by the VNS is of lower cost (actual costs not reduced costs) than the current solution of the master problem without duplicate requests, the corresponding routes are transformed into columns and added to the master. Here, the VNS is run for 10^4 iterations and ascending moves are not considered, i.e., a deteriorating solution cannot become a new incumbent solution (see Sect. 6.4). This collaborative scheme is inspired by Danna and Lepape (2005). In contrast to Danna and Lepape (2005), however, we do not only use the collaborative local search method to improve the best integer solution found so far. We use it to improve the current optimal solution, regardless of whether it is integer of fractional.

6 Variable neighborhood search

This section introduces the VNS heuristic that we have developed to solve the HDARPD. VNS was introduced by Mladenovic and Hansen (1997) and, as summarized in Algorithm 2, it consists of four major design elements: the initialization phase, the shaking operators, the iterative improvement mechanism, and the decision whether to move to the new solution or not.

Algorithm 2 Variable neighborhood search

```
**initialization** {determine an initial solution $s$ and set $h \leftarrow 1$}
**repeat**
  *shaking* {determine a solution $s'$ in the neighborhood $h$ of $s$}
  *iterative improvement* {apply local search to $s'$ yielding $s''$}
  *move or not* {if $s''$ meets the acceptance requirements the incumbent solution $s$ is replaced by $s''$ and
  $h \leftarrow 1$, otherwise $h \leftarrow (h \mod h_{\max}) + 1$; if $s''$ is feasible and better than $s_{\text{best}}$, set $s_{\text{best}} \leftarrow s''$}
**until** some stopping criterion is met
**return** $s_{\text{best}}$
```

In the initialization phase a first incumbent solution s is generated. In the repeat loop the algorithm iterates through neighborhoods of different sizes. At each iteration a solution s' is generated at random in the neighborhood of the current incumbent solution s (shaking phase). The new solution s' may be improved by means of an iterative improvement procedure (local search) yielding s''. If s'' meets the acceptance criteria (move or not) it replaces s and becomes the new incumbent. In this case or whenever the largest neighborhood size is reached, the search continues with the smallest neighborhood. If s'' does not constitute a new incumbent, the size of the neighborhood is increased.

Infeasibilities are allowed during the search. They are penalized through the following evaluation function:

$$\hat{f}(s) = c(s) + \sum_{r \in R} \hat{\alpha}_r q_r(s) + \hat{\beta} d(s) + \hat{\gamma} w(s) + \hat{\zeta} a(s). \tag{82}$$

The term $c(s)$ gives the routing cost of solution s. The terms $q_r(s)$, $d(s)$, $w(s)$, and $a(s)$ represent load violations ($\forall r \in R$), duration violations, time window violations, and attendant violations (if there are more attendants needed in s than there are available), respectively. The associated penalty parameters $\hat{\alpha}_r$, $\hat{\beta}$, $\hat{\gamma}$, and $\hat{\zeta}$ are dynamically adjusted throughout the search (see Parragh et al. (2010) and Parragh (2009) for details). Note that a solution s can only become a new best solution s_{best} if $q_r(s) = d(s) = w(s) = a(s) = 0$ for all $r \in R$.

This VNS is based on the ones developed by Parragh et al. (2009); Parragh et al. (2010) and Parragh (2009) for the standard DARP and the multi-objective DARP: it also uses a simulated annealing type mechanism (Kirkpatrick et al. 1983) in order to decide whether or not an ascending move shall be carried out. In addition, the employed iterative improvement phase is largely based on the one employed by Parragh et al. (2009); Parragh et al. (2010). However, in the current implementation, different neighborhood operators are employed. Instead of the zero split neighborhood, we introduce a second move operator. It randomly relocates requests to other vehicles, including all currently unused vehicles. The first move operator may move requests to one additional, currently unused, vehicle and also the chain operator may involve currently empty vehicles. These mechanisms aim at changing the current fleet configuration. However, if it results in a solution that employs more drivers than are available, the solution has to be repaired. Therefore, a repair mechanism is introduced. In addition, a new route evaluation scheme and a noon depot insertion algorithm are designed. The former determines the best location for the lunch break. The latter determines the best insertion position for the noon depot. In the following, the different design elements of the VNS for the HDARPD are described in further detail.

6.1 Initialization

The initialization procedure generates an initial solution which will often be infeasible. It is constructed as follows. In a first step the average number of requests per vehicle is computed and rounded to the next integer. Then, all requests are inserted into routes in the order in which they appear in the instance file. We start with the first route of the first type, opening the next route as soon as the average number of requests per vehicle has been reached. If a request does not fit into a route due to a lack of the demanded resource on the current vehicle, it is inserted into the next vehicle route with this resource available. The procedure ends as soon as all requests have been inserted into some route. Finally, all routes are checked to see whether a request demands an attendant aboard the vehicle. If this is the case, the noon depot is inserted at the best possible position and an attendant is assigned to the corresponding vehicle for the corresponding shift (morning or afternoon).

6.2 Shaking

During the shaking phase four different neighborhood operators are employed: the first swaps two sequences of vertices, the next two are based on the move operator, and the last one makes use of the ejection chain idea described by Glover (1996). A swap, a move, and an ejection chain neighborhood operator in combination with a "zero split" neighborhood are successfully employed by Parragh et al. (2009) in a solution framework for the multi-objective DARP. Two of the neighborhoods are also employed by Parragh et al. (2010) in the context of the standard DARP. The neighborhood operators employed in the current work are based on those. They are adapted, however, in order to cope with the special characteristics of the HDARPD.

Swap neighborhood In the swap neighborhood (S) two sequences of vertices are exchanged. Two routes are chosen at random from all routes currently in use (vehicles not assigned to a driver are not considered). Then, the starting points and the lengths of the sequences to be moved are randomly selected. Insertion into the new routes is conducted in a one by one fashion: considering each request at a time and inserting it into its best position. The notion of *critical vertices* as described by Cordeau and Laporte (2003) is employed. Note that if a sequence of vertices is moved their corresponding origins (destinations) have to be moved as well. The neighborhood size in this context refers to the maximum length (number of vertices) of the two sequences.

Move neighborhoods The first move neighborhood (M) consists in moving requests from their current routes to other routes. First, the requests to be moved are randomly chosen from the set of all requests. Second, for each request to be moved an insertion route is determined across all routes, excluding the request's original route. At most one additional route that is currently not assigned to a driver can be considered. As described in further detail in Parragh (2009), insertion routes are either selected randomly or the "closest" route in terms of spatial distance is taken. Random and "closeness" selection are chosen with a probability of 0.5 each. Third, the selected requests are moved to the best position in their new routes (again using the notion of critical vertices). The size of this neighborhood structure is defined by the maximum number of requests moved.

The second move neighborhood (Mx) distinguishes itself from the first one in the way the insertion routes are selected. Here, only random selection is employed. Furthermore, all routes, also those lacking a driver, are eligible for selection.

Chain neighborhood The third neighborhood is referred to as chain neighborhood (C). It works in a similar way as the ejection chain neighborhood defined by Glover (1996). In contrast to its original version, the number of routes affected is a parameter. From the first route a sequence of requests (selected as in the swap neighborhood) is moved to the second route. In a second step, the sequence that decreases the evaluation function value of the second route by the most is moved to a third route (it may also be the first route). The second step is repeated until the maximum number of sequences moved has been reached. All insertions of sequences into their new routes are done one by one in the best possible way. Again the notion of critical vertices is employed. All routes are selected randomly. Here, the neighborhood size represents at the same time the number of sequences moved and the maximum sequence length. Again, empty routes may be selected.

Repair function Both of the move operators as well as the chain neighborhood operators may construct a solution with more vehicles in use than drivers available. Such a solution has to be "repaired". The repair procedure employed here simply chooses one route at random out of all non-empty routes and redistributes the requests forming this route to other non-empty routes. Request insertion is done one by one in the best possible way, again using the notion of critical vertices. The repair procedure is repeated until the number of vehicles in use meets the number of available drivers.

Neighborhood sequence The shaking operators described above are applied in the following order: S1–M1–C1–S2–M2–C2–S3–M3–C3–S4–M4–C4–Mx4 (the number given in addition to the neighborhood abbreviation indicates the neighborhood size). This means that the first neighborhood ($h = 1$) corresponds to applying the shaking operator S1, whereas, e.g. in the case of $h = 6$ operator C2 will be used. In Parragh et al. (2009); Parragh et al. (2010), the last neighborhood is the parameterless zero split neighborhood which distributes requests forming a natural sequence to other routes. Here, Mx4 is the last neighborhood. In combination with the repair function it is the strongest diversification mechanism in place. It may relocate the largest number of requests and it may easily change the number of vehicles of each vehicle type employed.

6.3 Iterative improvement

At every iteration, after moving requests and repairing the resulting solution, the obtained solution may undergo local search-based improvement. As in Parragh et al. (2010), if $c(s') < 1.02c(s)$, s is automatically subject to local search-based improvement, otherwise it has a 1% chance to undergo local search. If s' meets the acceptance criteria to become the new incumbent solution it is also subject to local search based improvement, given that $c(s') > 1.05c(s)$. In any case only those routes that are affected by the shaking step are improved. Here again the notion of critical vertices is used. The search moves along each of these routes. At every origin it removes the corresponding request and reinserts it at every possible position. Tentative re-insertion always starts at the beginning of the route. If an improving position is found, the request is kept at its new position and the search restarts at the first origin on the route (first improvement). If no improving position is encountered, the request is kept at its original position and the search continues with the next origin on the route. The improvement heuristic terminates as soon as the last origin on the route is reached and no improving position has been found.

6.4 Move or not

The decision as to whether the search moves to the new solution s'' or not is based on a simulated annealing acceptance criterion (Černy 1985; Kirkpatrick et al. 1983). As in Parragh et al. (2010), if $\hat{f}(s'') < \hat{f}(s)$, s'' is always accepted. If $\hat{f}(s'') \geq \hat{f}(s)$, s is accepted with a probability equal to $\exp(-\frac{\hat{f}(s'')-\hat{f}(s_{\text{best}})}{\hat{t}})$. The temperature $\hat{t}$ is set such that as soon as the first feasible solution has been identified, a solution that is 0.5% worse than the current best solution is accepted with probability 0.2. Before the first feasible solution has been identified every solution that has an evaluation function value $\hat{f}(s'') \leq 1.05\hat{f}(s)$ is accepted with a probability of 1%.

6.5 Route evaluation

Every time a route is modified its routing cost and constraint violations have to be evaluated. Here, the notion of forward time slack, as for the label setting algorithm in Sect. 4.1.4, is applied. In a first step the lunch location is determined. Thereafter, the forward time slack is calculated. In a separate procedure the noon depot is either inserted in the best possible way or removed, if there are no more requests on the route that demand an attendant. In the local search step, the noon depot insertion/removal procedure is only invoked at the very end. The same applies to the request insertion routine. Whenever a request is removed from a route it is also invoked.

7 Computational experiments

All programs are implemented in C++. In the column generation framework, the LP solver of CPLEX 11.1 is used together with Concert Technology 2.6. All experiments are carried out on a 3.2 GHz Pentium D computer with a memory of 4 GB. All solution procedures are tested on three artificial data sets and on real-world data. In the following we first describe the characteristics of the different instances and we then discuss the results obtained.

7.1 Artificial instances

For each instance of the data set "A" introduced by Cordeau (2006) three instances with different degrees of heterogeneity are generated. The original instances contain between 16 and 48 requests and between 2 and 4 vehicles. In all new instances the vehicle fleet is increased, containing 2 T1 and 4 T2 vehicles. The number of available drivers is set to the original number of vehicles (between 2 and 4). The choice of the fleet configuration reflects the real-world situation: In reality, only some T1 vehicles and many (more than drivers) T2 vehicles are available. However, at most as many T2 vehicles as there are drivers available can be used. Therefore, it does not make sense to consider a larger vehicle fleet. The characteristics of the new instances are summarized in Table 1. Setting "X" is the most homogeneous one: 50% of the original

Table 1 Artificial instances: data

Data set	Probability for patient to be			Probability	Probability	m_a	Fleet
	Seated	On stretcher	In wheelchair	for AP	AT demanded		
X	0.50	0.25	0.25	0.00	0.25	$\lceil m(0.5+\rho)\rceil$	2 T1, 4 T2
Y	0.25	0.25	0.50	0.10	0.50	$\lceil m(1+\rho)\rceil$	2 T1, 4 T2
Z	0.83	0.11	0.06	0.50	0.50	$\lceil m(1+\rho)\rceil$	2 T1, 4 T2

ρ randomly chosen in [0, 1]
T1: 1 staff seat, 6 patient seats, 1 wheelchair place
T2: 2 staff seats, 1 patient seat, 1 stretcher, 1 wheelchair place
AP accompanying person, *AT* attendant

users are converted into seated passengers; 25% into patients on stretchers; and 25% into persons needing a wheelchair. The probability that an accompanying person (AP) is present is set to zero. The number of attendants available is randomly set to between 0.5 of and 1.5 times the number of drivers available (rounded up to the next integer). The probability for an attendant to be demanded by a patient is set to 25%. For setting "Y", 25% of the original users are transformed into seated patients, 25% into patients on stretchers and 50% into wheelchair patients; 10% of all patients are assumed to be accompanied by someone. The number of attendants is randomly set to at least the number of drivers available and at most to twice the number of drivers. The probability for an attendant to be demanded by a patient is set to 50%. In the third setting, denoted by "Z", 83% of the patients are assumed to be seated, 11% are assumed to be on a stretcher, and 6% are assumed to be in a wheelchair. This setting is based on the data provided by the ARC. Finally, the probability for a patient to be accompanied by someone is set to 50%. All other settings are equal to those of data set "Y". In all instances, a 60-min time window is associated with the start depot. The maximum route duration is reduced by 60 min with respect to the original data. As before, it is equal for all vehicles. The time window at the noon depot is set to $e_{2n+1} = e_0 + T/2$ and $l_{2n+1} = e_{2n+1} + 15$. The lunch time window is set to $e_H = 180$ and $l_H = 360$ and its duration to $H = 30$ min.

7.2 Real-world instances

We also apply the different solution algorithms to 15 real-world instances from the ARC. They possess the following characteristics. As in data set "Z" 83% of the passengers are seated patients, 11% have to be transported on a stretcher, and 6% in a wheelchair; 50% of all these passengers take an accompanying person with them and about 40% demand additional personnel (an attendant) on the vehicle. Three T1 vehicles and 31 T2 vehicles are available. The maximum route duration (driver working hours) is 510 min. The lunch break has to start between 11 am and 2 pm. It lasts 30 min. Every driver starts working between 6:30 and 8:30 am. These two points in time give the time window at the start depot. The time window at the noon depot lasts from 12:30 to 1:00 pm. Users specify a time window for either the pick-up or the drop-off location. Time window width is equal to 30 min. Maximum user ride times have been set to $L_i = t_{i,n+i} + 30$ for all $i \in P$. As mentioned above, ride time limits are not explicitly considered; depending on which time window has been specified by the user, the time window for the corresponding location without time window is set relative to the existing time window; in the case of a time window at the destination, it is set to $e_i = e_{n+i} - (L_i + d_i)$ and $l_i = l_{n+i} - (t_{i,n+i} + d_i)$ at the origin; in the case of a time window at the origin, it is set to $e_{n+i} = e_i + d_i + t_{i,n+i}$ and $l_{n+i} = l_i + d_i + L_i$ at the corresponding destination.

7.3 Column generation results

In a first step two versions of the column generation framework are tested on the artificial data sets. The first version only uses pure column generation, while in the second version the collaborative scheme is employed. Both procedures are first compared within a restrictive time limit of 1 h. Within this time limit, pure column generation solves 24 out of 36 instances. With the collaborative scheme in place, one additional instance (i.e., 25 instances) can be solved. Within an increased time limit of 5 days, pure column generation solves 32 out of 36 instances and the collaborative scheme 31 out of 36 instances. Tables 2 and 3 contain the results obtained with the increased limit. Table 2 gives the results for pure column generation while Table 3 gives the results for the collaborative scheme. The following information is provided. First, the time needed to compute the initial VNS solution and the total run time, excluding the initial VNS, of the respective program is given. Then, the lower bounds and the best integer solutions found throughout the search are presented. In the case where the time limit is reached, the current "best" lower bound is given in brackets; it may not be the optimal one. If the obtained lower bound is integer, then the best integer solution corresponds to the optimal integer solution. Otherwise, it corresponds to the solution obtained from solving SP on the set of generated columns within a run time limit of 10 min. If the corresponding instance is identified as infeasible, neither a lower bound nor an integer solution is provided. The status of the obtained lower bound is indicated in the next column (integer (int.), fractional (frac.), or infeasible (inf.)). This is followed by the total number of columns generated, and by the number of times the different pricing procedures found at least one new negative reduced cost column. Rows $\overline{X}$, $\overline{Y}$, and $\overline{Z}$ give the average values for the respective data set. Row $\overline{XYZ}$ gives the total average values across all data sets.

In both cases, pure column generation and the collaborative approach, the two heuristic pricing procedures which prove to be the most useful are *ConstrHeur* and *LimLabels*. Also, all other heuristic pricing procedures contribute a number of negative reduced cost columns and thus should not be left out. When comparing the two tables, the following differences can be observed. In the case of the smallest instances, pure column generation is faster than the collaborative approach. This is due to the fact that, in the case of the smallest instances, the initial VNS quite often already finds the optimal solution. Thus, intermediate calls to the VNS do not improve this solution. They only increase computation times. This relation changes in the case of medium-sized and larger instances. Here, in some cases the collaborative approach is faster while in others, lower computation times are due to pure column generation. The intuition is that calls to the intermediate VNS are not always useful although in many cases they are. When comparing the "best" lower bound (i.e., either the lower bound or the best objective value obtained, if the time limit is reached), the collaborative approach is slightly better than pure column generation on average. When comparing the best integer solution found during the search, the collaborative approach outperforms pure column generation on average within the 1 h time limit and also within the 5-day time limit. Comparison of the results given in the two tables on a per instance level shows that the collaborative scheme is better or equal in all but three cases. Finally, in the case of instance a4-48 of data set Y (marked by an asterisk) the provided lower bound computed by the collaborative approach is optimal. However, the optimality check of the collaborative scheme (i.e., the last call to the pricing routines) ends after the run time limit. Therefore, it is not counted to the number of instances solved within 5 days but it is used for comparison purposes for the heuristic method.

Table 2 Artificial instances: pure column generation

	CPU		LB	Best int.	Stat.	Cols	Number of times new cols found by					
	Init. VNS	Gen. cols					*ConstrHeur*	*LNSCurrBasis*	*RandConstrHeur*	*LNSRandConstr*	*LimLabels*	*LimGraph*
X												
a2–16	17.50	4.69	299.37	299.37	int.	658	6	1	3	2	5	0
a2–20	64.64	184.36	376.70	376.70	int.	2,295	41	7	25	5	24	1
a2–24	105.83	53,143.70	461.66	461.66	int.	4,729	29	72	12	14	54	0
a3–18	14.41	12.36	291.68	291.68	int.	1,175	9	5	9	4	2	0
a3–24	38.37	485.27	351.19	365.59	frac.	3,051	58	18	30	11	34	1
a3–30	89.60	4,155.15	510.79	510.79	int.	5,924	83	36	42	24	78	25
a3–36	–	–	(602.52)	602.52	–	15,020	116	222	5	8	62	40
a4–16	16.39	3.97	–	–	inf.	711	12	8	4	1	2	0
a4–24	27.82	39.49	388.95	392.37	frac.	2,364	27	5	9	0	10	0
a4–32	80.55	523.40	504.79	509.23	frac.	4,951	41	28	8	4	17	0
a4–40	–	–	(610.15)	610.15	–	12,423	142	109	0	10	45	30
a4–48	132.73	249,301.00	665.11	671.58	frac.	20,552	118	85	94	19	86	102
$\overline{X}$	58.78	30,785.34	460.27	462.88		6,154	57	50	20	9	35	17

Table 2 Continued

	CPU		LB	Best int.	Stat.	Cols	Number of times new cols found by					
	Init. VNS	Gen. cols					*ConstrHeur*	*LNSCurrBasis*	*RandConstrHeur*	*LNSRandConstr*	*LimLabels*	*LimGraph*
Y												
a2–16	43.86	5.83	–	–	inf.	1,164	17	5	11	0	4	0
a2–20	63.18	24.89	371.62	371.62	int.	1,928	46	13	38	9	16	0
a2–24	184.80	1,912.23	–	–	inf.	3,516	18	54	27	12	43	0
a3–18	20.11	3.76	295.39	295.39	int.	609	9	6	3	1	2	0
a3–24	72.67	76.62	353.13	355.47	frac.	3,021	32	17	31	13	34	0
a3–30	87.82	3,748.05	487.13	487.13	int.	7,586	88	11	30	16	41	20
a3–36	–	–	(601.30)	607.08	–	14,138	176	67	34	14	54	74
a4–16	16.51	1.95	–	–	inf.	544	8	2	4	0	4	0
a4–24	48.25	63.12	–	–	inf.	2,009	28	26	17	1	14	0
a4–32	77.73	500.74	504.91	507.29	frac.	4,984	51	38	2	3	32	2
a4–40	117.62	13,804.90	587.56	591.21	frac.	8,200	63	52	12	6	46	29
a4–48	–	–	(717.95)	717.95	–	21,775	154	48	105	33	230	91
$\overline{Y}$	73.26	2014.21	489.87	491.64		5,790	58	28	26	9	43	18

Table 2 Continued

	CPU		LB	Best int.	Stat.	Cols	Number of times new cols found by					
	Init. VNS	Gen. cols					*ConstrHeur*	*LNSCurrBasis*	*RandConstrHeur*	*LNSRandConstr*	*LimLabels*	*LimGraph*
Z												
a2–16	34.06	8.27	308.30	308.30	int.	1,086	23	2	10	3	6	0
a2–20	78.44	1,702.77	398.65	398.65	int.	2,640	42	17	58	17	34	0
a2–24	100.13	944.26	423.05	423.05	int.	3,658	90	31	44	10	24	0
a3–18	21.02	4.14	297.24	297.24	int.	785	13	6	2	1	2	0
a3–24	43.93	92.72	354.19	355.15	frac.	2,057	34	12	34	13	18	0
a3–30	96.62	1,289.59	495.70	498.43	frac.	5,485	89	48	33	20	33	8
a3–36	135.26	107,372.00	569.36	573.55	frac.	12,497	114	12	55	36	84	88
a4–16	20.86	2.88	–	–	inf.	697	11	6	10	0	2	0
a4–24	31.24	53.96	388.69	388.69	int.	1,970	24	9	14	8	13	0
a4–32	96.75	562.22	498.34	514.28	frac.	4,118	68	45	9	5	23	0
a4–40	95.91	68,132.30	581.86	601.92	frac.	9,535	79	38	43	11	32	31
a4–48	202.89	229,265.00	679.21	679.21	int.	13,249	103	33	61	28	133	61
$\overline{Z}$	79.76	34,119.18	454.05	458.04		4,815	58	22	31	13	34	16
$\overline{XYZ}$	71.17	23,044.55	465.88	468.77		5,586	57	33	26	10	37	17

Cols columns, *frac*. fractional, *Gen*. generating, *inf*. infeasible, *Init*. initial, *int*. integer, *LB* lower bound, *Stat*. status

Table 3 Artificial instances: collaborative scheme

	CPU		LB	Best int.	Stat.	Cols	Number of times new cols found by					
	Init. VNS	Gen. cols					*ConstrHeur*	*LNSCurrBasis*	*RandConstrHeur*	*LNSRandConstr*	*LimLabels*	*LimGraph*
X												
a2–16	17.50	15.99	299.37	299.37	int.	675	7	2	8	0	7	0
a2–20	64.17	228.98	376.70	376.70	int.	1,775	24	11	22	9	17	0
a2–24	108.21	18,519.80	461.66	461.66	int.	4,726	29	74	13	4	58	0
a3–18	14.42	15.24	291.68	291.68	int.	1,130	9	3	5	0	4	0
a3–24	38.09	598.28	351.19	356.30	frac.	2,948	52	11	24	16	33	3
a3–30	91.41	3,047.32	510.79	510.79	int.	5,565	99	14	50	23	63	20
a3–36	–	–	(602.52)	602.52	–	15,459	125	173	28	12	75	72
a4–16	16.34	5.56	–	–	inf.	665	9	4	7	1	2	0
a4–24	28.13	32.78	388.95	390.96	frac.	2,155	16	1	7	0	8	0
a4–32	80.47	646.83	504.79	509.98	frac.	4,692	38	14	8	6	19	1
a4–40	–	–	(609.02)	614.75	–	13,554	172	53	46	30	50	24
a4–48	–	–	(665.11)	669.62	–	18,972	109	53	52	39	87	87
$\overline{X}$	50.97	2,567.86	460.16	462.21		6,026	57	34	23	12	35	17

Table 3 Continued

	CPU		LB	Best int.	Stat.	Cols	Number of times new cols found by					
	Init. VNS	Gen. cols					*ConstrHeur*	*LNSCurrBasis*	*RandConstrHeur*	*LNSRandConstr*	*LimLabels*	*LimGraph*
Y												
a2–16	43.85	21.37	–	–	inf.	1,210	14	3	13	1	5	0
a2–20	63.11	95.05	371.62	371.62	int.	1,894	30	7	37	3	15	0
a2–24	187.98	1,064.16	–	–	inf.	3,450	19	49	29	4	36	0
a3–18	20.02	11.37	295.39	295.39	int.	720	9	1	5	0	5	0
a3–24	73.05	209.39	353.13	355.47	frac.	3,038	43	19	36	11	26	0
a3–30	87.93	4,065.66	487.13	487.13	int.	7,776	71	5	38	14	45	32
a3–36	–	–	(597.64)	599.14	–	14,461	204	32	48	12	85	59
a4–16	16.39	2.69	–	–	inf.	565	10	1	2	1	3	0
a4–24	48.08	110.58	–	–	inf.	2,113	26	25	17	4	10	0
a4–32	76.36	824.18	504.91	509.03	frac.	4,831	44	39	2	4	15	5
a4–40	118.68	11,365.80	587.56	591.15	frac.	7,987	68	38	12	6	56	33
a4–48[a]	–	–	(717.95)	717.95	int.	23,902	222	52	121	56	251	88
$\overline{Y}$	73.55	1,777.03	489.42	490.86		5,996	63	23	30	10	46	18

Table 3 Continued

	CPU		LB	Best int.	Stat.	Cols	Number of times new cols found by					
	Init. VNS	Gen. cols					*ConstrHeur*	*LNSCurrBasis*	*RandConstrHeur*	*LNSRandConstr*	*LimLabels*	*LimGraph*
Z												
a2–16	33.73	28.62	308.30	308.30	int.	972	24	4	14	2	3	0
a2–20	78.87	983.79	398.65	398.65	int.	2,653	61	23	55	7	26	0
a2–24	99.24	532.96	423.05	423.05	int.	3,748	91	34	41	15	23	0
a3–18	20.76	19.03	297.24	297.24	int.	841	13	8	4	6	2	0
a3–24	44.64	230.14	354.19	355.15	frac.	2,395	38	15	55	6	23	0
a3–30	97.60	1,659.25	495.70	498.29	frac.	5,304	72	39	51	18	46	4
a3–36	137.62	213,300.00	569.36	573.55	frac.	14,322	87	14	74	65	111	99
a4–16	20.61	6.87	–	–	inf.	630	11	11	2	1	3	0
a4–24	31.04	60.59	388.69	388.69	int.	1,848	19	6	7	5	10	1
a4–32	97.88	884.19	498.34	505.66	frac.	4,254	52	42	14	3	31	2
a4–40	96.42	63,289.30	581.86	600.83	frac.	9,599	73	55	21	22	36	30
a4–48	198.21	297,329.00	679.21	679.21	int.	13,536	104	34	61	36	146	62
$\overline{Z}$	79.72	48,193.65	454.05	457.15		5,009	54	24	33	16	38	17
$\overline{XYZ}$	69.38	19,974.35	465.72	467.99		5,677	58	27	29	12	40	17

Cols columns, *frac*. fractional, *Gen*. generating, *inf*. infeasible, *Init*. initial, *int*. integer, *LB* lower bound, *Stat*. status
[a] Optimality proven within a bit more than 5 days

As the original real-world instances are too large to be solved with either of the two exact procedures, results for these will only be provided for the heuristic method. However, in order to get an idea of how difficult practical instances are, we generated two additional real-world-based data sets containing eight instances each. In these two data sets, the number of requests in every instance is reduced by a factor of 5 or 3 with respect to the original data, respectively, and the time window length is decreased to 15 min. In Table 4, we compare the results of pure column generation to those of the collaborative scheme for those instances that are five times smaller than the original ones. Similar information as in the previous tables is given. In addition, we also provide the size of each instance in terms of the total number of requests n, the number of drivers m_d, and the number of attendants m_a available. Within a time limit of 1 h, the collaborative scheme solves four out of eight instances, while pure column generation only solves three instances. Within the increased time limit of 5 days, both procedures solve seven out of eight instances. However, considerable differences in terms of run times can be observed. For example, in the case of instance mai0605a the collaborative scheme is 2.3 times faster than pure column generation, while in the case of instance mai1805a, pure column generation is 2.7 times faster than the collaborative scheme. When comparing the best integer solutions obtained, the collaborative scheme again outperforms pure column generation on average. Only for instances aug0508a and mai2105a the integer solutions of pure column generation are better than those obtained by means of the collaborative scheme.

Given these results, it seems fair to say that the integration of a collaborative local search method into the column generation framework has a positive impact on the performance of the whole method but it does not outperform pure column generation in all cases.

We also applied both methods to those instances that have been reduced by a factor of 3. They comprise between 49 and 91 requests and between 6 and 10 drivers. Within a 1-h time limit none of the instances is solved with either of the two methods. With an increased time limit, both methods solve instance aug1208b (58 requests and 7 drivers) within a couple of hours of run time; in more than 3 days, pure column generation also solves instance aug1308b (55 requests and 7 drivers). All other instances remain unsolved. Therefore, we only provide heuristic results for this set of instances.

Summarizing the above results, both methods consistently solve both artificial and real-world-based instances with up to 34 requests and some instances with up to 58 requests.

7.4 Heuristic results

In Table 5, the results obtained by means of the proposed VNS for the artificial data sets are presented. A limit of 10^5 iterations is used as stopping criterion. For each data set the following information can be taken from the table: the name of the instance; the average solution values over five random runs; the best solution values out of these five runs; and the respective percentage deviations from the exact lower bounds, where known (see above). The average percentage gap between the lower bound and the obtained average solution value is less than 2% for all three data sets. Computing

Table 4 Real-world-based instances (5 times smaller): pure column generation and collaborative scheme

	n	m_d	m_a	Pure column generation						Collaborative scheme					
				CPU Init. VNS	CPU Gen. cols	LB	Best int.	St.	Cols	CPU Init. VNS	CPU Gen. cols	LB	Best int.	St.	Cols
aug0508a	29	4	6	45.78	128.95	377.32	377.46	frac.	2,701	45.93	190.34	377.32	379.28	frac.	2,552
aug1108a	30	4	6	36.17	2,148.14	449.74	449.74	int.	3,432	36.18	1,273.65	449.74	449.74	int.	3,298
aug1208a	34	4	6	62.71	3,904.88	426.01	434.50	frac.	6,900	62.31	3,130.95	426.01	429.20	frac.	7,067
aug1308a	33	4	6	54.00	12,935.90	473.26	491.87	frac.	5,912	54.13	13,844.80	473.26	486.99	frac.	6,562
mai0605a	41	5	7	55.11	60,776.40	471.47	485.97	frac.	8,669	55.14	26,015.50	471.47	478.55	frac.	9,318
mai0705a	42	5	7	–	–	(616.48)	639.47	–	7,279	–	–	(617.07)	636.51	–	7,115
mai1805a	54	6	9	82.15	85,068.90	582.38	618.22	frac.	10,713	83.55	233,089.00	582.38	601.32	frac.	11,057
mai2105a	28	4	5	32.65	104.25	390.65	391.10	frac.	3,524	32.69	195.58	390.65	391.22	frac.	3,793
Avg.				52.65	23,581.06	541.04	486.04		6,141	52.85	39,677.12	541.13	481.60		6345

Cols columns, *frac*. fractional, *Gen*. generating, *inf*. infeasible, *Init*. initial, *int*. integer, *LB* lower bound, *St*. status

Table 5 Artificial instances: VNS (10^5 iterations, 5 runs)

	Data set *X*					Data set *Y*					Data set *Z*				
	Avg.	%	Best	%	CPU	Avg.	%	Best	%	CPU	Avg.	%	Best	%	CPU
a2–16	299.37	0.00	299.37	0.00	201.34	–	–	–	–	–	308.30	0.00	308.30	0.00	270.89
a2–20	377.14	0.12	376.70	0.00	345.76	371.62	0.00	371.62	0.00	680.22	398.65	0.00	398.65	0.00	436.34
a2–24	461.66	0.00	461.66	0.00	685.8	–	–	–	–	–	426.08	0.72	425.30	0.53	633.21
a3–18	291.68	0.00	291.68	0.00	89.72	295.39	0.00	295.39	0.00	132.68	297.24	0.00	297.24	0.00	128.09
a3–24	359.72	2.43	356.30	1.45	247.14	361.66	2.42	360.78	2.17	314.13	361.00	1.92	355.15	0.27	280.76
a3–30	512.34	0.30	510.79	0.00	516.5	496.02	1.82	487.13	0.00	527.24	506.05	2.09	498.29	0.52	555.64
a3–36	614.31	–	604.35	–	987.97	617.55	–	596.61	–	1,108.15	580.01	1.87	572.55	0.56	942.36
a4–16	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
a4–24	402.44	3.47	394.34	1.38	169.8	–	–	–	–	–	396.92	2.12	393.13	1.14	180.79
a4–32	516.21	2.26	513.10	1.64	367.19	508.30	0.67	507.29	0.47	340.34	506.36	1.61	504.44	1.22	353.16
a4–40	623.57	–	613.33	–	753.95	595.90	1.42	592.55	0.85	644.27	600.02	3.12	595.72	2.38	617.07
a4–48	689.15	3.61	680.27	2.28	862.78	734.42	2.29	728.74	1.50	1,122.68	697.64	2.71	684.56	0.79	1,164.09
Avg.	467.96	1.35	463.81	0.75	475.27	497.61	1.23	492.51	0.71	608.71	461.66	1.47	457.58	0.67	505.67

Avg. average

Table 6 Real-world-based instances (5 times smaller): VNS (5 runs)

	n	m_d	m_a	VNS 10^5 iterations					VNS 2×10^5 iterations				
				Avg.	%	Best	%	CPU	Avg.	%	Best	%	CPU
aug0508a	29	4	6	384.96	2.02	383.10	1.53	274.54	383.64	1.68	378.89	0.42	519.25
aug1108a	30	4	6	458.75	2.00	451.03	0.29	229.47	459.30	2.13	453.15	0.76	445.85
aug1208a	34	4	6	431.95	1.39	428.99	0.70	313.55	436.23	2.40	428.99	0.70	671.22
aug1308a	33	4	6	480.80	1.59	479.66	1.35	316.00	483.59	2.18	479.66	1.35	651.54
mai0605a	41	5	7	488.11	3.53	485.94	3.07	287.16	483.61	2.58	473.38	0.41	590.89
mai0705a	42	5	7	664.30	–	654.11	–	471.91	640.08	–	627.32	–	769.51
mai1805a	54	6	9	623.57	7.07	603.78	3.67	495.44	622.99	6.97	615.81	5.74	937.00
mai2105a	28	4	5	409.49	4.82	391.62	0.25	190.51	399.35	2.23	391.10	0.12	375.15
Avg				492.74	3.21	484.78	1.55	322.32	488.60	2.88	481.04	1.36	620.05

Avg. average

times are not really low, but acceptable (less than 9 min on average). The rather long computing times with respect to the low total iteration limit are due to the complex evaluation procedure, including a possible repositioning of the noon depot and the appropriate choice of the lunch break location.

Table 6 contains the results obtained for the 8 real-world instances that have been reduced by a factor 5 with respect to the original data. The following information is provided: the name of the instance; the size of the instance in terms of the total number of requests n; the number of drivers m_d; the number of attendants m_a available; and for each iteration limit, the average and the best solution value out of five random runs, the corresponding deviation from the lower bound (only where the optimal lower bound is known), and the total average run time in seconds. On average, the proposed VNS yields solutions within 3.21% of the lower bound, using 10^5 iterations. With an increased limit of 2×10^5 iterations, the average percentage gap is reduced to 2.88. In the case of instance mai1805a, the lower bound solution is highly fractional. This explains why the obtained gap between the heuristic upper bound and the column generation lower bound is so large. It can be assumed that it is at least partly due to a larger integrality gap with respect to the other instances. Total run times are below 16 min for all instances.

Table 7 provides similar information to Table 6. However, for this medium-sized data set, which is again based on the available real-world data, a lower bound can only be computed for two instances. Thus, we only provide average and best solution values for the proposed VNS within a limit of 10^5 and 2×10^5 iterations. The percentage deviations presented in Table 7 give the deviations from the best solutions encountered with the two iteration limits. As expected, more iterations lead, on average, to better solution values.

Finally, Table 8 provides the results for those data sets that are based on ARC data from 15 days in the city of Graz. As mentioned above, all of them are too large to be solved by means of column generation. Therefore, only the proposed VNS is applied. Because of the large size of these instances, two different iteration limits are used.

Table 7 Real-world-based instances (3 times smaller): VNS (5 runs)

	n	m_d	m_a	All best	VNS 10^5 iterations					VNS 2×10^5 iterations				
					Avg.	%	Best	%	CPU	Avg.	%	Best	%	CPU
aug0508b	49	6	9	667.90	676.06	1.22	667.90	0.00	461.64	676.76	1.33	670.34	0.36	833.46
aug1108b	51	6	9	693.31	712.22	2.73	693.31	0.00	428.37	706.65	1.92	696.80	0.50	802.34
aug1208b	58	7	10	718.02	733.79	2.20	718.02	0.00	404.47	735.34	2.41	730.46	1.73	788.95
aug1308b	55	7	10	643.30	653.10	1.52	643.30	0.00	355.91	650.33	1.09	644.15	0.13	731.29
feb0402b	60	7	11	684.51	703.89	2.83	695.09	1.55	444.74	697.84	1.95	684.51	0.00	847.73
mai0605b	69	8	11	764.79	785.08	2.65	764.79	0.00	498.42	777.71	1.69	769.87	0.66	969.59
mai1805b	91	10	15	1,107.04	1,167.19	5.43	1,140.29	3.00	743.60	1,129.74	2.05	1,107.04	0.00	1,362.91
nov0411b	79	9	13	867.97	902.28	3.95	878.21	1.18	598.93	882.35	1.66	867.97	0.00	1,173.83
Avg.				768.36	791.70	2.82	775.11	0.72	492.01	782.09	1.76	771.39	0.42	938.76

Avg. average

Table 8 Real-world instances: VNS (5 runs)

	n	m_d	m_a	All best	VNS 5×10^5 iterations					VNS 10^6 iterations				
					Avg.	%	Best	%	CPU	Avg.	%	Best	%	CPU
aug0508	147	17	26	1,374.33	1,406.03	2.31	1,379.42	0.37	6,797.44	1,392.12	1.29	1,374.33	0.00	12,241.06
aug1108	154	18	27	1,415.55	1,447.50	2.26	1,422.90	0.52	5,121.00	1,429.40	0.98	1,415.55	0.00	10,179.36
aug1208	174	19	29	1,632.16	1,688.72	3.47	1,673.86	2.55	7,854.12	1,666.13	2.08	1,632.16	0.00	15,394.96
aug1308	166	19	29	1,552.40	1,600.94	3.13	1,576.00	1.52	6,729.45	1,565.90	0.87	1,552.40	0.00	12,795.56
feb0202	233	20	30	2,087.65	2,144.81	2.74	2,103.59	0.76	20,555.08	2,113.63	1.24	2,087.65	0.00	37,997.24
feb0402	182	21	32	1,677.70	1,722.32	2.66	1,701.25	1.40	7,484.74	1,696.75	1.14	1,677.70	0.00	14,200.60
feb1002	186	18	27	1,804.26	1,854.35	2.78	1,829.44	1.40	12, 483.68	1,839.04	1.93	1,804.26	0.00	24,718.90
mai0605	208	22	33	1,760.82	1,851.03	5.12	1,833.69	4.14	10,917.44	1,779.48	1.06	1,760.82	0.00	20,268.88
mai0705	210	23	35	1,964.68	2,077.48	5.74	2,046.29	4.15	10, 119.45	1,997.39	1.66	1,964.68	0.00	18,350.90
mai1805	273	30	45	2,570.25	2,677.39	4.17	2,629.02	2.29	11,725.66	2,601.40	1.21	2,570.25	0.00	22,719.08
mai2105	140	16	24	1,457.36	1,494.56	2.55	1,476.78	1.33	5,745.56	1,470.55	0.91	1,457.36	0.00	10,813.82
nov0411	239	25	38	2,155.53	2,225.80	3.26	2,186.96	1.46	11,984.40	2,171.58	0.74	2,155.53	0.00	23,501.62
nov0911	247	24	36	2,207.46	2,303.18	4.34	2,269.98	2.83	15,720.66	2,248.11	1.84	2,207.46	0.00	29,666.94
nov1211	192	21	32	1,711.99	1,805.97	5.49	1,784.38	4.23	9,152.52	1,749.12	2.17	1,711.99	0.00	16,999.46
nov1611	219	21	32	2,097.98	2,236.66	6.61	2,212.37	5.45	15,687.24	2,158.05	2.86	2,097.98	0.00	30,054.52
Avg.				1,831.34	1,902.45	3.77	1,875.06	2.29	10,538.56	1,858.58	1.47	1,831.34	0.00	19,993.53

Avg. average

First, VNS is run for 5×10^5 iterations. This configuration results in solution values that are on average 3.77% worse than the best solution found during both experimental settings. On average, slightly less than 3 h of run time is needed. Then, the limit is increased to 10^6 iterations. In less than twice the time (on average 5.5 h), the average gap from the best known solution is reduced to 1.47%.

For the smallest instances containing up to 50 requests, 10^5 iterations will lead to solutions of high quality when compared to the lower bound. For medium size real-world instances more iterations are necessary to yield solutions of acceptable quality. In the case of the largest real-world instances, a limit of 5×10^5 or even 10^6 iterations, if time allows, should be used.

8 Conclusion

This paper has described a heterogeneous dial-a-ride problem with driver-related constraints and has introduced two new formulations of the problem. A column generation approach has been proposed to compute lower bounds based on the set-partitioning formulation. A variable neighborhood search heuristic has also been developed. Computational experiments show that on the artificial instances, high-quality solutions are obtained within rather short run times. A collaborative scheme, integrating the VNS heuristic into the column generation framework, has also been developed. Comparisons with the pure column generation show that the collaborative scheme improves the efficiency of the original method but does not outperform it in all cases. Finally, also realistic instances are solved by means of the proposed VNS algorithm. Given the fact that about 80% of the requests carried out by the ARC are already known the day before, the proposed algorithm can be run for several hours in order to provide a good initial routing plan. We thus hope that the methods proposed in this paper will serve as the basis for the development of a computer-aided routing tool that will support ambulance dispatchers at the Austrian Red Cross in their day-to-day work.

Acknowledgments This work was supported by the Special Translational Research Program of the Austrian Science Fund (FWF) under Grants #L286-N04 and #L362-N15, and by the Canadian Natural Sciences and Engineering Research Council under grant 227837-09. This support is gratefully acknowledged. We thank Fabien Tricoire for his suggestions and the Austrian Red Cross for providing the real-world data. We are also grateful to the referees for their valuable comments.

References

Beaudry A, Laporte G, Melo T, Nickel S (2009) Dynamic transportation of patients to hospitals. OR Spectrum 32:77–107

Berbeglia G, Cordeau J-F, Gribkovskaia I, Laporte G (2007) Static pickup and delivery problems: a classification scheme and survey. TOP 15:1–31

Berbeglia G, Cordeau J-F, Laporte G (2010) Dynamic pickup and delivery problems. Eur J Oper Res 202:8–15

Černy V (1985) Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. J Opt Theory Appl 45:41–51

Cordeau J-F (2006) A branch-and-cut algorithm for the dial-a-ride problem. Oper Res 54:573–586

Cordeau J-F, Laporte G (2003) A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transport Res B Meth 37:579–594

Cordeau J-F, Laporte G (2007) The dial-a-ride problem: models and algorithms. Ann Oper Res 153:29–46

Danna E, Lepape C (2005) Branch-and-price heuristics: a case study on the vehicle routing problem with time windows. In: Desaulniers G, Desrosiers J, Solomon MM (eds) Column generation. Springer, New York, pp 99–129

Desaulniers G, Villeneuve D (2000) The shortest path problem with time windows and linear waiting costs. Transport Sci 34(3):312–319

Desrochers M, Desrosiers J, Solomon M (1992) A new optimization algorithm for the vehicle routing problem with time windows. Oper Res 40:342–354

Glover F (1996) Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Appl Math 65:223–253

Hanne T, Melo T, Nickel S (2009) Bringing robustness to patient flow management through optimized patient transports in hospitals. Interfaces 39:241–255

Irnich S (2008) Resource extension functions: properties, inversion, and generalization to segments. OR Spectrum 30:113–148

Kallehauge B, Larsen J, Madsen OBG, Solomon MM (2005) Vehicle routing problems with time windows. In: Desaulniers G, Desrosiers J, Solomon MM (eds) Column generation. Springer, New York

Kirkpatrick S, Gelatt CD Jr, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680

Melachrinoudis E, Ilhan AB, Min H (2007) A dial-a-ride problem for client transportation in a health-care organization. Comput Oper Res 34:742–759

Mladenovic N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24:1097–1100

Parragh SN (2009) Ambulance routing problems with rich constraints and multiple objectives. PhD thesis, University of Vienna, Faculty of Business, Economics and Statistics

Parragh SN (2010) Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. Transp Res C Emer (to appear)

Parragh SN, Doerner KF, Gandibleux X, Hartl RF (2009) A heuristic two-phase solution method for the multi-objective dial-a-ride problem. Networks 54:227–242

Parragh SN, Doerner KF, Hartl RF (2008a) A survey on pickup and delivery problems. Part I: transportation between customers and depot. J Betriebswirtschaft 58:21–51

Parragh SN, Doerner KF, Hartl RF (2008b) A survey on pickup and delivery problems. Part II: transportation between pickup and delivery locations. J Betriebswirtschaft 58:81–117

Parragh SN, Doerner KF, Hartl RF (2010) Variable neighborhood search for the dial-a-ride problem. Comput Oper Res 37:1129–1138

Rekiek B, Delchambre A, Saleh HA (2006) Handicapped person transportation: an application of the grouping genetic algorithm. Eng Appl Artif Intel 19:511–520

Ropke S (2005) Heuristic and exact algorithms for vehicle routing problems. PhD thesis, Computer Science Department at the University of Copenhagen (DIKU)

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

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

Savelsbergh MWP (1992) The vehicle routing problem with time windows: minimizing route duration. ORSA J Comput 4:146–154

Savelsbergh MWP, Sol M (1998) DRIVE: dynamic routing of independent vehicles. Oper Res 46:474–490

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Proceedings CP-98 (fourth international conference on principles and practice of constraint programming)

Toth P, Vigo D (1997) Heuristic algorithms for the handicapped persons transportation problem. Transport Sci 31:60–71

Xu H, Chen Z-L, Rajagopal S, Arunapuram S (2003) Solving a practical pickup and delivery problem. Transport Sci 37:347–364