Transportation Research Part C 19 (2011) 723–740

Contents lists available at ScienceDirect

Transportation Research Part C

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

A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows

Andrea Bettinelli, Alberto Ceselli, Giovanni Righini *

Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Via Bramante 65, 26013 Crema, Italy

ARTICLE INFO

Article history:
Received 4 December 2009
Received in revised form 27 July 2010
Accepted 30 July 2010

Keywords:
Vehicle routing
Column generation
Cutting planes

ABSTRACT

We present a branch-and-cut-and-price algorithm for the exact solution of a variation of the vehicle routing problem with time windows in which the transportation fleet is made by vehicles with different capacities and fixed costs, based at different depots. We illustrate different pricing and cutting techniques and we present an experimental evaluation of their combinations. Computational results are reported on the use of the algorithm both for exact optimization and as a heuristic method.

1. Introduction

There is a considerable economic interest around algorithmic techniques for optimization in logistics, since transportation represents a relevant factor in the final cost of goods.

Many real applications require to find a set of routes of minimum length for a fleet of vehicles with limited resources, in order to serve a given set of customers. When routes start and end at a unique depot, each customer has a given demand which must be served by a single vehicle, and the fleet is composed by identical vehicles with limited service capacity, the problem turns out to be a classical vehicle routing problem (VRP) (Golden et al., 2008).

However these applications often require to take into account several additional constraints (Santos et al., 2008; Prindezis and Kiranoudis, 2005). For instance, in several applications involving interaction with human personnel or customers, visiting certain locations is allowed only in certain time windows; furthermore, large companies or consortia of small transportation companies usually have more than one depot and they often manage a fleet made of vehicles with different capacities and operating costs (Ceselli et al., 2009).

Therefore, in this paper we consider three major extensions of the basic VRP model, namely time windows, heterogeneous vehicles and multiple depots. The resulting problem is called Multi-Depot Heterogeneous Vehicle Routing Problem with Time Windows (MDHVRPTW): we want to simultaneously determine the optimal composition, placement and routing of a heterogeneous fleet of capacitated vehicles in order to satisfy the delivery demands of a set of customers under time constraints.

We refer the reader to the recent book by Golden et al. (2008) for an up-to-date survey of VRP models and methods. State-of-the-art exact algorithms can solve instances with up to 135 customers (Fukasawa et al., 2006; Baldacci et al., 2007).

Several variants of the VRP involving subsets of the features of the MDHVRPTW have been studied in the literature.

State-of-the-art methods for the VRP with Time Windows (VRPTW) using either branch-and-cut (Kallehauge et al., 2007) or column generation (Chabrier, 2006; Desaulniers et al., 2005) are able to solve instances with up to some hundreds customers (Golden et al., 2008; Kallehauge, 2008).

* Corresponding author.
E-mail addresses: andrea.bettinelli@unimi.it (A. Bettinelli), alberto.ceselli@unimi.it (A. Ceselli), giovanni.righini@unimi.it (G. Righini).

0968-090X/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.trc.2010.07.008

A branch-and-cut-and-price algorithm for the Heterogeneous fleet Vehicle Routing Problem (HVRP), able to solve instances with up to 75 customers to proven optimality, has recently been proposed in Pessoa et al. (2009). The Heterogeneous Vehicle Routing Problem with Time Windows (HVRPTW) in the variant without any limit on the number of available vehicles was first addressed by Liu and Shen (1999) under the denomination of Fleet Size and Mix VRP with Time Windows; the authors proposed a number of insertion-based parallel savings heuristics, performing tests on instances with up to 100 customers. Later, Dullaert et al. (2002) developed three insertion-based heuristics, that are an extension of Solomon's (1987) sequential insertion heuristic combined with ideas of Golden et al.'s (1984). The authors tested their algorithms using a slightly different objective function. More recently Belfiore and Fávero (2007) and Dell'Amico et al. (2007) respectively proposed scatter search and "ruin-and-recreate" approaches to this problem. State-of-the-art heuristics are able to provide good quality approximations for problems with 100 customers in a few minutes.

Polacek et al. (2004) studied a Variable Neighborhood Search for the Multi-Depot Vehicle Routing Problem with Time Windows (MDVRPTW). A cluster-based optimization approach to the MDHVRPTW has been proposed by Dondo and Cerdá (2007), who also developed a large-scale neighborhood method to improve the solutions found by their previous heuristic (Dondo and Cerdá, 2009). In this way, instances with 100 customers can be heuristically solved in a time ranging from some seconds to 1 h.

Baldacci and Mingozzi (2009) proposed a unified exact method, which is able to solve different classes of vehicle routing problems, including heterogeneous fleets and multiple depots (MDHVRP), without time windows; this is based on the exact solution of an integer linear programming problem and on dual heuristics. Instances involving up to 100 customers were solved to optimality in some hours of computing time.

Finally, in Ceselli et al. (2009) a rich vehicle routing problem was considered: it involves multiple depots, heterogeneous vehicles, time windows and other features. The authors proposed a column generation heuristic yielding good results on instances with up to few hundred customers in some hours.

To the best of our knowledge, no exact algorithms have been proposed for the MDHVRPTW so far. In this paper we present a branch-and-cut-and-price algorithm for the exact optimization of the MDHVRPTW. The algorithm extends some recent ideas, such as bi-directional dynamic programming and decremental state space relaxation already applied to the VRPTW and other versions of the VRP. In particular, in Section 2 we introduce an extended formulation for the problem; in Section 3 we illustrate our column generation routines and show how the repeated execution of the dynamic programming-based pricing algorithm can be made more efficient; in the same section we also discuss the addition of strengthening inequalities and branching strategies. In Section 4 we report the results of an extensive experimental campaign, evaluating the effectiveness of our algorithm both as an exact and as a heuristic optimization method. Section 5 ends the paper with concluding remarks.

2. Problem formulation

The MDHVRPTW can be formulated as follows. Let $\mathscr{G} = (\mathscr{N} \cup \mathscr{H}, \mathscr{A})$ be a directed graph whose node set is the union of a set $\mathscr{N}$ of customers and a set $\mathscr{H}$ of depots.

Non-negative costs d_{ij} and t_{ij} are associated with each arc $(i,j) \in \mathscr{A}$, representing respectively the travel cost and the travel time to reach $j \in \mathscr{N}$ starting from $i \in \mathscr{N}$. Each customer $i \in \mathscr{N}$ has a delivery demand q_i, a service time s_i and a delivery time window $[a_i, b_i]$.

We consider a set $\mathscr{K}$ of vehicle types, with known capacities w_k and fixed costs f_k $\forall k \in \mathscr{K}$. We assume that the number u_k of available vehicles of each type $k \in \mathscr{K}$ is limited. The number of routes associated with each depot $h \in \mathscr{H}$ is limited by a given value g_h. This represents the number of drivers living close to each depot, who are able to operate routes from it; we assume drivers to be able to use any type of vehicle.

In the version of the problem we study, we assume that all vehicles can be freely associated with any depot: this corresponds to the situation in which the location of the vehicles at the depots is a decision, not a datum.

Finally, vehicles do not necessarily leave their depots at time 0: due to time windows, it may be convenient to delay the departure time to reduce waiting time at customer locations. Following (Liu and Shen, 1999) we indicate as *duration* the difference between the arrival and the starting time of the route, that is the actual time spent in delivery operations. This cannot exceed a given limit D, representing the length of driver shifts.

The objective is to minimize the sum of vehicles fixed costs and routing costs, satisfying the following conditions:

I. all customers must be served;
II. each customer must be visited by only one vehicle;
III. the service at each customer must start within the customer time window;
IV. each route begins at a depot and ends at the same depot;
V. the sum of the demands of the customers served in each route must not exceed the capacity of the associated vehicle;
VI. the duration of each route must not be greater than D;
VII. the number of available vehicles for each type must not be exceeded;
VIII. the number of allowed routes for each depot must not be exceeded.

Our framework allows to easily consider also the version in which the location of vehicles is fixed, i.e., in each depot a given number of vehicles of each type are pre-positioned; in this case a bound u_{hk} is imposed on the number of routes for each depot $h \in \mathscr{H}$ and for each vehicle type $k \in \mathscr{K}$, but the algorithm does not change. Nevertheless, in the remainder we discuss only the former version, because it is the only one having terms of comparison in the literature.

We introduce a set covering formulation of the MDHVRPTW as follows. We indicate as feasible any route satisfying conditions II, III, IV, V and VI. We indicate as Ω_{hk} the set of all feasible routes using a vehicle of type $k \in \mathscr{K}$ from depot $h \in \mathscr{H}$.

We associate a binary variable x_r with each feasible route: x_r takes value 1 if and only if route r is selected. We also use binary coefficients a_{ir} with value 1 if and only if customer $i \in \mathscr{N}$ is visited by route r. We indicate by c_r the cost of route r; it is equal to the sum of the vehicle fixed cost f_k and the routing costs, i.e., the optimal value of the Traveling Salesman Problem with Time Windows (TSPTW) on the customer set $\{i \in \mathscr{N} | a_{ir} = 1\}$ starting and ending at a particular depot. With these definitions we obtain the following integer linear programming model:

$$\text{minimize} \quad \sum_{h\in\mathscr{H}} \sum_{k\in\mathscr{K}} \sum_{r\in\Omega_{hk}} c_r x_r, \tag{1}$$

$$\text{s.t.} \quad \sum_{h\in\mathscr{H}} \sum_{k\in\mathscr{K}} \sum_{r\in\Omega_{hk}} a_{ir} x_r \geqslant 1 \quad \forall i \in \mathscr{N}, \tag{2}$$

$$\sum_{h\in\mathscr{H}} \sum_{r\in\Omega_{hk}} x_r \leqslant u_k \quad \forall k \in \mathscr{K}, \tag{3}$$

$$\sum_{k\in\mathscr{K}} \sum_{r\in\Omega_{hk}} x_r \leqslant g_h \quad \forall h \in \mathscr{H}, \tag{4}$$

$$x_r \in \{0,1\} \quad \forall r \in \bigcup_{k\in\mathscr{K}, h\in\mathscr{H}} \Omega_{hk}. \tag{5}$$

Constraints (2) are standard set covering constraints, modeling condition I, while (3) and (4) impose limits on the maximum number of available vehicles of each type and the maximum number of allowed routes for each depot, modeling conditions VII and VIII. The objective is to minimize the overall cost of the selected routes. In the remainder we indicate this formulation as *Master Problem* (MP).

3. Branch-and-cut-and-price

We solve the linear relaxation of the MP to obtain a lower bound which is used in a tree search algorithm. The number of variables is exponential in the cardinality of the customer set $\mathscr{N}$, thus we use a column generation approach (Desaulniers et al., 2005). In this section we describe the main components of the resulting branch-and-cut-and-price algorithm, namely pricing, cut generation and branching. We also discuss some implementation issues.

3.1. Column generation

We initially consider only a small subset of the variables in the MP. Such initial restricted master problem (RMP) includes

- a set of $|\mathscr{N}|$ columns, one for each customer, representing the optimal paths serving one customer at a time,
- a set of columns representing a feasible MDHVRPTW solution, found using a straightforward greedy policy,
- an additional *dummy column* $\bar{r}$ of very high cost, having $a_{i\bar{r}} = 1 \forall i \in \mathscr{N}$ and every other coefficient set to 0.

The aim of the dummy column is to ensure feasibility at each node of the search tree.

We solve the linear relaxation of the RMP, and we search for columns which are not in the RMP, but have negative reduced cost. If no such column exists, the solution is optimal for the MP linear relaxation as well, and thus yields a valid lower bound to the problem. On the opposite, if any negative reduced cost column is found, it is added to the RMP, and the process is iterated.

Let λ, μ and σ be the nonnegative dual vectors corresponding respectively to constraints (2), and to constraints (3) and (4) rewritten as $\geqslant$ inequalities.

The reduced cost of a column encoding route r has the following form:

$$c_r - \sum_{i\in\mathscr{N}} \lambda_i a_{ir} + \mu_k + \sigma_h.$$

As stressed before, the routes generated must comply with conditions II, III, IV, V and VI: it is important to note that the same sequence of customers may correspond to a feasible or to an infeasible route according to the type of vehicle and the depot it is associated with. For instance, different types of vehicles imply different capacities. Moreover, cost c_r and dual variables μ_k and σ_h depend on both k and h. Therefore, at each column generation iteration we have to solve $|\mathscr{K}| \cdot |\mathscr{H}|$ pricing problems. We also remark that it is never convenient to perform cycles in a feasible solution, although the prize structure given by dual variables can make it appealing; therefore, a search must be performed for elementary routes only.

Hence, given a particular depot h and vehicle type k, the problem of finding the most negative reduced cost column encoding a route for vehicle k using depot h turns out to be a Resource Constrained Elementary Shortest Path Problem (RCESPP).

The RCESPP is NP-hard (Dror, 1994); we solve it using three pricing algorithms: a greedy one, a heuristic dynamic programming one and an exact dynamic programming one. In fact, dynamic programming techniques as those described in Righini and Salani (2006, 2008), show to be particularly effective in solving RCESPPs, provided careful design choices are made for each problem at hand. In particular, coping with condition VI is a critical issue in our case, and requires special adaptations to the standard framework.

Following (Salani, 2006), the three pricing algorithms are called in sequence, only if the previous pricing algorithm cannot find any column with negative reduced cost. In the remainder we give a description of the pricing problem and of the three algorithms. For the purpose of better illustrating them, we follow the reverse order with respect to their execution.

3.1.1. Exact dynamic programming

Let us consider first the case of a single depot $h \in \mathcal{H}$ and a single vehicle type $k \in \mathcal{K}$: let s and v be the two distinct copies of depot h, representing the departure and arrival node and let $w = w_k$ be the capacity of vehicle k.

For the exact solution of the RCESPP we use the technique proposed in Righini and Salani (2006), which consists of a bi-directional extension of node labels. It associates labels with each node $i \in \mathcal{N}$ of the graph: forward labels represent paths from s to i and backward labels represent paths from i to v. Each label is iteratively considered and the corresponding path is extended to adjacent nodes.

3.1.1.1. Label structure. Each label has the form $(S, \chi, \tau, \delta, \rho, C, i)$. This tuple represents the state of the path associated with the label; C is the cost of the path, i is the last reached node, S, χ, τ, δ and ρ are either resources or resource consumptions with the following meaning: S is a set indicating which nodes have already been visited, χ indicates the amount of capacity consumed, τ indicates the elapsed time from time 0 up to the earliest point in time at which service at i can start, δ indicates the minimum duration of the path from the departure time up to the point in time at which service at i starts.

The value ρ is needed to update δ at each extension; it represents a time buffer, indicating the maximum amount of time by which the path defined so far can be delayed still remaining feasible with respect to the time windows of the visited nodes.

Fig. 1 represents a sample instance of two nodes i and j on a time-line. The bold lines correspond to time spent in travel and service operations, the dashed lines correspond to waiting time either at an intermediate node or at the initial depot; the dotted line represents the time buffer ρ. By leaving the depot at time 0 and visiting node i (top figure), the vehicle has to wait until the time window of node j allows the service to start. Instead, such waiting can be avoided by delaying the departure from the depot (bottom figure). The time buffer ρ is spent to reduce the overall duration δ of the path.

3.1.1.2. Extension. In forward extensions the set S is initialized at $\emptyset$, all resource consumptions are initialized at 0 at the depot nodes, while ρ is initialized at a very large value.

The search is restricted to elementary paths by discarding extensions to any node $j \in S$. When a label $(S, \chi, \tau, \delta, \rho, C, i)$ is extended forward to a node j, a new label $(S', \chi', \tau', \delta', \rho', C', j)$ is computed using the following rules.

First, resources S, χ, τ and C are updated as follows:

$$S' = S \cup \{j\}, \tag{6}$$

$$\chi' = \chi + q_j, \tag{7}$$

$$\tau' = \max\{\tau + s_i + t_{ij}, a_j\}, \tag{8}$$

$$C' = C - \frac{1}{2}\lambda_i + d_{ij} - \frac{1}{2}\lambda_j, \tag{9}$$

in the updating formula for cost, we consider $\lambda_s = \lambda_v = 0$.

In order to update duration δ and time buffer ρ we have to consider a special case: if the vehicle arrives at j early, it should wait until a_j. Let us define the waiting time at j as $\omega = \max\{a_j - (\tau + s_i + t_{ij}), 0\}$. Hence the earliest starting time of service at j is equal to the earliest arrival time τ plus ω. Also the duration of the route is increased by the service time at i and the travel

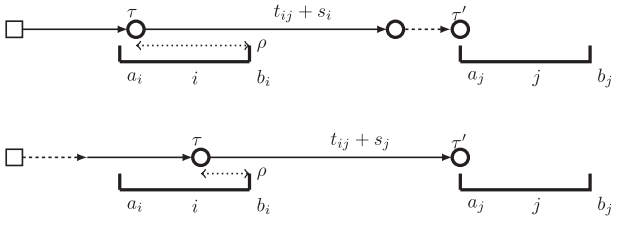

Fig. 1. Label resources in the dynamic programming RCESPP algorithm.

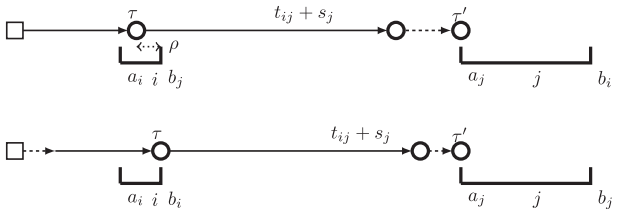

Fig. 2. A time window limiting the time buffer ρ.

time from i to j, but the time buffer can be spent by delaying the departure from the depot to reduce the waiting time, as in the sample instance reported in Fig. 1. It might not be possible to fully avoid waiting: the departure delay is limited by the amount ρ, which takes into account feasibility with respect to the time windows of the customers visited up to i, and by the waiting time, that is $\min\{\rho,\omega\}$; for example, in Fig. 2 the time window of node i is tight (top figure), and the time buffer ρ is not enough to compensate the waiting time at node j (bottom figure).

Besides traveling and service time, the actual increase in duration is therefore $\omega - \min\{\rho,\omega\}$, that is $-\min\{\rho-\omega,0\}$. By substitution with the expression of ω, we obtain the following update formula for δ':

$$\delta' = \delta + (s_i + t_{ij}) - \min\{\rho - \max\{a_j - (\tau + s_i + t_{ij}), 0\}, 0\}.$$

The remaining amount of possible delay to absorb future waiting times is ρ', which is limited from above both by the remaining buffer due to the time windows of the customers in S, that is $\rho - \min\{\rho,\omega\}$, and by the constraint imposed by the time window of customer j, that is $b_j - \tau'$. The update formula for ρ' is therefore

$$\rho' = \min\{\rho - \min\{\rho, \max\{a_j - (\tau + s_i + t_{ij}), 0\}\}, b_j - \tau'\}.$$

The new state is feasible if and only if

$$\chi' \leqslant w \text{ (capacity constraint)}, \tag{10}$$
$$\tau' \leqslant b_j \text{ (time window constraint)}, \tag{11}$$
$$\delta' \leqslant D \text{ (duration constraint)}, \tag{12}$$

otherwise it is fathomed.

The updating rules and feasibility tests for backward extension are symmetrical. Backward extensions start from time $T = \max_{i\in\mathcal{N}}\{b_i + s_i + t_{iv}\}$ that is the latest possible arrival time at the final depot.

In particular the value $T - \tau$ in a backward label at a generic node j indicates the latest point in time at which service at the node j can end.

The algorithm iteratively extends each forward and backward label to all possible successors or predecessors respectively.

In order to limit the extension of forward and backward labels and to reduce useless duplication of paths, we impose that each partial path can use at most half of a critical resource whose consumption is monotone along the paths. The best critical resource is the tightest one. In our case we have two meaningful choices: either time or duration.

Since in the instances from the literature it is often $D = +\infty$, we chose time as the critical resource.

3.1.1.3. Join. Forward and backward paths must be joined together to produce complete $s - v$ paths. The result of the join is a feasible solution if all the resource constraints are satisfied. When a forward path $(S^{fw}, \chi^{fw}, \tau^{fw}, \delta^{fw}, \rho^{fw}, C^{fw}, i)$ is joined with a backward path $(S^{bw}, \chi^{bw}, \tau^{bw}, \delta^{bw}, \rho^{bw}, C^{bw}, j)$, the feasibility conditions for a vehicle type k are:

$$S^{fw} \bigcap S^{bw} = \emptyset$$
$$\chi^{fw} + \chi^{bw} \leqslant w$$
$$\tau^{fw} + s_i + t_{ij} + s_j + \tau^{bw} \leqslant T$$

Furthermore, the constraint on the maximum duration of the whole path has to be satisfied. We remark that in backward labels, paths are supposed to end at time T, but can be shifted up to ρ^{bw} time units without violating time windows of the visited customers. Therefore $T - \tau^{bw} - \rho^{bw} - s_j$ represents the earliest point in time at which service at the node j can start in the backward path. The way in which backward paths are built already ensures $T - \tau^{bw} - \rho^{bw} - s_j \geqslant a_j$.

Let us consider the sample three-nodes instance depicted in Fig. 3. The structure of the figure is similar to the previous ones: the bolded lines correspond to time spent in travel and service operations, the dashed lines correspond to late departure and early arrival at the depot, the dotted line represents waiting time. During the join on arc (i,j), the time window of node k prevents the backward path to be shifted by more than ρ^{bw} units in time. In the same way, the time window of node i prevents the forward path to be shifted by more than ρ^{fw} units in time.

The waiting time needed to join the forward and backward paths is therefore $\omega = T - \tau^{bw} - \rho^{bw} - s_j - (\tau^{fw} + \rho^{fw} + s_i + t_{ij})$, the actual waiting time at node j is $\max\{\omega,0\}$ and the duration of the overall path is

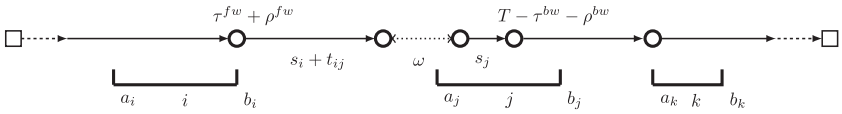

Fig. 3. Computation of the waiting time during the join of two labels.

$$\delta^{fw} + s_i + t_{ij} + \max\{\omega, 0\} + s_j + \delta^{bw}.$$

The feasibility conditions on the overall duration of the path is thus the following:

$$\delta^{fw} + s_i + t_{ij} + \max\{T - \tau^{bw} - \rho^{bw} - s_j - (\tau^{fw} + \rho^{fw} + s_i + t_{ij}), 0\} + s_j + \delta^{bw} \leqslant D.$$

The reduced cost of the resulting $s - v$ path is

$$C^{fw} - \frac{\lambda_i}{2} + d_{ij} - \frac{\lambda_j}{2} + C^{bw} + f_k + \mu_k + \sigma_h.$$

3.1.1.4. Dominance test. During the extension of labels, a dominance test is done to fathom labels that cannot lead to an optimal solution. Let $l' = (S', \chi', \tau', \delta', \rho', C', i)$ and $l'' = (S'', \chi'', \tau'', \delta'', \rho'', C'', i)$ be two labels associated with node i. Then the former dominates the latter if the following conditions are satisfied

(a) $S' \subseteq S''$
(b) $\chi' \leqslant \chi''$
(c) $\tau' \leqslant \tau''$
(d) $\delta' \leqslant \delta''$
(e) $\rho' \geqslant \rho''$
(f) $C' \leqslant C''$

Condition (e) is too restrictive and can be relaxed into the surrogate inequality

(g) $\delta' - \rho' \leqslant \delta'' - \rho''$

without losing the optimality guarantee. It is not hard to show by counter examples that inequalities (a)–(d), (f) and (g) represent a set of necessary dominance conditions: dropping any of them optimal labels might be discarded.

Further, as shown in Feillet et al. (2004), it is sometimes possible to identify some node $u \in \mathcal{N}$ that cannot be reached by any feasible extension of a given label, because of resource limitations.

In this case it is useful to insert u in the set S of that label: it is easy to check that enlarging set S'' helps satisfying condition (a); at the same time, if a node cannot be reached by extending label l' due to resource limitations, it cannot be reached by extending label l'' either, since resource consumption in l'' is not lower. Therefore, enlarging each set S allows dynamic programming fathoming a larger number of labels and hence reducing the computation time.

3.1.1.5. Decremental state space relaxation. The dynamic programming algorithm is executed iteratively applying decremental state space relaxation (Righini and Salani, 2008). The idea is to project the state space of the problem to a smaller one, by removing some elementarity constraints. From an algorithmic point of view, this amounts to identify a set of *critical nodes* $\widetilde{\mathcal{N}}$, and to replace extension rule (6) as

$$S' = (S \cup \{j\}) \cap \widetilde{\mathcal{N}}.$$

This relax ed problem can be solved more efficiently, since more labels can be compared in the dominance test.

In order to identify a good critical node set, we initialize $\widetilde{\mathcal{N}} = \emptyset$; then we iteratively solve the state space relaxation of the pricing problem and insert in the set $\widetilde{\mathcal{N}}$ all the nodes visited more than once in the optimal path, until we find an elementary one.

3.1.1.6. Bounding. Using ideas similar to Christofides et al. (1981) and Baldacci et al. (2007), we compute lower bounds in a preprocessing phase; these are checked during the dynamic programming iterations to detect suboptimal labels that can be fathomed.

Consider forward labels first: we disregard elementarity constraints, time windows and duration constraints, and for each node $i \in \mathcal{N}$ we compute the least cost path from i to v that uses exactly p capacity units, for all values of p in the range $[q_i, \ldots, w]$.

Such computation requires the search for a capacity constrained shortest path; this is a weakly $\mathcal{NP}$-hard problem, and can be solved effectively in a single run for all vertices i and for all values $p \in [0, \ldots, w]$ as follows. We build an auxiliary graph with $w + 1$ layers, one for each value of p in the range, each containing a copy of the original graph. Then, each copy of node i in layer p' is connected to a copy of node j in layer p'' if and only if $p'' = p' + q_j$, and the cost of the arc connecting them is

$d_{ij} - \lambda_i/2 - \lambda_j/2$; each copy of i in layer p is also connected to the copy of the same node in layer $p+1$ with a zero-cost arc. Since this auxiliary graph is acyclic, one can use standard shortest path algorithms, and solve the problem of finding the optimal path starting from node i by finding the optimal shortest path tree rooted at node i in layer 0: the optimal path using at most p capacity units is the one connecting node i in layer 0 with node v in layer p. We note that the same computation can be done by reversing each arc and finding an optimal shortest path tree rooted in node v for each layer p. The overall complexity of this procedure is therefore $O(\min\{|\mathcal{N}|, w\} \cdot (|\mathcal{N}| \cdot w)^2)$.

Let us indicate as F_{ip} the value of the optimal path from node i in layer p to node v in layer w; since elementarity, time windows and duration constraints have been relaxed, this is a valid lower bound to the cost of any backward label associated with node i with $\chi \leqslant p$. When the exact dynamic programming algorithm is executed, if a forward label $(S, \chi, \tau, \delta, \rho, C, i)$ is such that

$$C + F_{i\chi} + f_k + \mu_k + \sigma_h \geqslant 0, \tag{13}$$

the label cannot lead to a negative reduced cost path and, therefore, it can be discarded.

If the instances are asymmetric, we pre-compute an analogous lower bound to possibly fathom backward labels applying the same idea in the other direction. This pre-computation is done once for each column generation iteration, once the dual values have been obtained from the linear relaxation of the restricted master problem.

Preliminary computational results showed that this technique considerably reduces the overall number of labels explored, with a negligible additional computational effort.

3.1.1.7. Aggregated pricing algorithm. One of the main difficulties with the heterogeneous fleet and multi-depot version of the VRP is that the reduced cost of the routes depends on the depot chosen and the vehicle type. Hence in principle it would be necessary to execute the pricing algorithm for each combination of $k \in \mathcal{K}$ and $h \in \mathcal{H}$. One of the main features of our algorithm is that multiple execution of the pricing algorithm is avoided.

First, in order to optimize all vehicle types at once, we extend the labels only once at each column generation iteration, using the capacity of the largest vehicle. Indeed, feasibility rules for time and duration are satisfied for each vehicle type, since travel time does not change from one type to another. Hence, during the join phase a column is generated for each vehicle type $k \in \mathcal{K}$ such that $w_k \geqslant \chi^{fw} + \chi^{bw}$. A similar approach can be applied to speed-up the bounding technique described above: the values F can be computed just once, considering only the largest capacity value. The drawback of this approach is that it weakens the bounding condition (13); in fact, in order to perform a valid check, we have to consider at the same time the capacity of the largest vehicle, and the smallest fixed costs $\min_{k\in\mathcal{K}}\{f_k + \mu_k\}$.

Furthermore it is possible to deal with all the depots in a single run as follows. We enlarge the state space by introducing different reduced cost (C_h), time consumption (τ_h), duration (δ_h) and time buffer (ρ_h) resources for each depot $h \in \mathcal{H}$. When a forward label is extended from a node i to a node j and $\tau_{\overline{h}} + t_{ij} + s_i > b_j$ for a certain $\overline{h} \in \mathcal{H}$, the path is not feasible for depot $\overline{h}$: thus we set $\tau'_{\overline{h}} = +\infty$. A similar argument applies to δ_h resources.

For what concerns domination rules, whenever two labels l' and l'' satisfy conditions (a)–(d), (f) and (g) for a particular depot $\overline{h}$, resources $\tau''_{\overline{h}}$, $\delta''_{\overline{h}}$ and $C''_{\overline{h}}$ are set to $+\infty$, as label l'' can never yield an optimal path for depot $\overline{h}$.

A forward label is fathomed when, due either to extensions, feasibility checks or dominance, it has $\tau_h > T/2$ and $\delta_h > D$ for each $h \in \mathcal{H}$.

Finally, we observe that in each label reduced costs C_h and resources τ_h,δ_h and ρ_h for each $h \in \mathcal{H}$ only differ for an offset given by the first and last arc in the path. We exploit this observation by avoiding to store a value for each $h \in \mathcal{H}$, and saving memory by computing such offset at each check.

3.1.2. Heuristic dynamic programming

The heuristic dynamic programming pricing algorithm uses basically the same bi-directional label extension described above, but dominance condition (a) on the set S of visited nodes is dropped.

Since in this way several labels representing promising paths are fathomed, an additional dominance condition is enforced, as explained hereafter.

For each label, we define a *potential* $R(S, \chi)$ which is an upper bound on the amount of dual prizes λ that can be collected by completing the path.

In order to get a better evaluation of this potential, we reduce each coefficient λ_i by $\min_{j\in\mathcal{N}\setminus S}\{d_{ij}\}$, since it is always needed to select an outgoing arc from each visited node, and cycles are not allowed.

Then we consider the problem of selecting a subset of nodes which are not yet visited, whose overall demand still fits in the residual capacity of the vehicle and whose potential is maximum. For a given vehicle type k, such problem can be formulated as follows:

$$R(S, \chi) = \max \quad \sum_{i\in\mathcal{N}\setminus S} \lambda_i z_i, \tag{14}$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{N}\setminus S} q_i z_i \leqslant w_k - \chi, \tag{15}$$

$$z_i \in \{0, 1\} \quad \forall i \in \mathcal{N} \setminus S, \tag{16}$$

where S represents the set of visited customers and χ is the sum of their demands; binary variables z_i are 1 if node i is selected, 0 otherwise; constraint (15) is a capacity constraint. The objective (14) is to maximize the sum of the potential corresponding to the selected nodes.

Computing each $R(S,\chi)$ value requires to solve a 0–1 Knapsack problem (Martello and Toth, 1990; Kellerer et al., 2005); therefore, to speed-up the computation we approximate the potential by computing its continuous relaxation $\bar{R}(S,\chi)$: this can be done in $O(|\mathcal{N}|\log|\mathcal{N}|)$ time; furthermore, the computation of $\bar{R}(S,\chi)$ after each label extension can be done in linear time through very efficient re-optimization techniques.

Therefore, in the heuristic pricing algorithm a label $l' = (S',\chi',\tau',\delta',\rho',C',i)$ dominates a label $l'' = (S'',\chi'',\tau'',\delta'',\rho'',C'',i)$ if the following conditions are met:

$$
\begin{aligned}
&(\tilde{a})\ \bar{R}(S',\chi') \geqslant \bar{R}(S'',\chi''),\\
&(\tilde{b})\ \chi' \leqslant \chi'',\\
&(\tilde{c})\ \tau' \leqslant \tau'',\\
&(\tilde{d})\ \delta' \leqslant \delta'',\\
&(\tilde{f})\ C' \leqslant C'',\\
&(\tilde{g})\ \delta' - \rho' \leqslant \delta'' - \rho'',
\end{aligned}
$$

and at least one of these inequalities is strict. This results in a much larger number of dominations and reduces computational time; on the other hand, these conditions are necessary but not sufficient to allow dominance, and therefore we loose optimality guarantees on the best path found.

Finally, by changing in sign each value $\bar{R}(S,\chi)$ we also obtain a lower bound on the improvement one can get by extending each label. Therefore, given the reduced cost ϕ^* of the best $s-v$ path found so far, any label having

$$C - \bar{R}(S,\chi) + \min_{k\in\mathcal{K}}\{\mu_k\} + \min_{h\in\mathcal{H}}\{\sigma_h\} - \frac{\lambda_i}{2} \geqslant \phi^*,$$

is fathomed, because it cannot provide better solutions.

3.1.3. Greedy

In the greedy pricing algorithm a single label $(S,\chi,\tau,\delta,\ \rho,C,i)$ is considered and iteratively extended to a single node with a nearest neighbor policy.

Given a vehicle type k, we first compute the set $\mathcal{R}$ of reachable customers, that is the set of nodes $j\in\mathcal{N}$ satisfying the following conditions

$$
\begin{aligned}
&j \notin S,\\
&\chi + q_j \leqslant w_k,\\
&\max\{\tau + s_i + t_{ij}, a_j\} \leqslant T,\\
&\delta + (s_i + t_{ij}) - \min\{\rho - \max\{a_j - (\tau + s_i + t_{ij}), 0\}, 0\} \leqslant D.
\end{aligned}
$$

If $\mathcal{R} = \emptyset$ we close the path going back to the depot and we stop the search. Otherwise, among the customers in $\mathcal{R}$, we select the one that minimizes the path cost, that is

$$\bar{j} \in \operatorname{argmin}_{j\in\mathcal{R}}\{d_{ij} - \lambda_j\},$$

we extend the label to node $\bar{j}$ using the same update rules described for the exact pricing algorithm, and we iterate.

It is still possible to simultaneously find routes for the different vehicle types: if the value $\chi + q_j$ exceeds the capacity w_k of some vehicle types, we generate a valid route for these types by closing the path with an arc from i to v. Then, if the capacity of the largest vehicle is not reached, we add $\bar{j}$ to the path and iterate.

This greedy algorithm is repeated for each depot $h\in\mathcal{H}$.

We tried different policies for selecting the nearest neighbor at each iteration, without observing substantial differences.

3.2. 2-Path inequalities

To strengthen the lower bound we add violated 2-path inequalities. These cuts have been introduced by Kohl et al. (1999) for the CVRPTW, extending an idea of Laporte et al. (1985). In this paper we use the same heuristic separation algorithm described in Kohl et al. (1999). We search for a subset $\mathcal{S}\subset\mathcal{N}$ such that the customers in $\mathcal{S}$ are served by less than two vehicles in the fractional solution of the master problem, but require at least two vehicles in any integer solution. In particular, this property holds for a set $\mathcal{S}$ in the following two cases: (a) if the capacity of the largest vehicle is not sufficient to satisfy the demand of all the customers in $\mathcal{S}$, (b) if there is no tour including all customers in $\mathcal{S}$ that respects their time windows, and therefore does not exceed the maximum length T. While property (a) can simply be checked by inspection, property (b) requires to compute a feasible Hamiltonian tour of customers in $\mathcal{S}$, that is to solve an instance of TSP with Time Windows.

However, when $|\mathscr{S}|$ is small the TSPTW feasibility problem can be easily solved by dynamic programming (see Dumas et al. (1994)). We remark that the property holds in other cases as well, but these two are the only easily computable ones.

A 2-path inequality can take two forms. Both have been used in the literature (Baldacci and Mingozzi, 2009; Pessoa et al., 2009), but to the best of our knowledge no comparison between them has ever been presented. In the following we perform such a comparison, while in Section 4 we discuss different ways of handling 2-path inequalities in our algorithm.

The first form is the following:

$$\sum_{h\in\mathscr{H}}\sum_{k\in\mathscr{K}}\sum_{r\in\Omega_{hk}}\alpha_{r\mathscr{S}}x_r \geqslant 2, \tag{17}$$

where the coefficient $\alpha_{r\mathscr{S}}$ is equal to the number of arcs (i,j) used by the route r such that $i \in \mathscr{S}$ and $j \notin \mathscr{S}$. Coefficients $\alpha_{r\mathscr{S}}$ might be larger than 1, since it is not needed to the nodes in $\mathscr{S}$ to be visited consecutively.

The second form is:

$$\sum_{h\in\mathscr{H}}\sum_{k\in\mathscr{K}}\sum_{r\in\Omega_{hk}|\sum_{i\in\mathscr{S}}a_{ir}\geqslant 1} x_r \geqslant 2. \tag{18}$$

Since $\sum_{i\in\mathscr{S}}a_{ir} \geqslant 1$ for each column encoding a path visiting a customer in $\mathscr{S}$, the inequality forces to select at least two such paths. One can show that any fractional solution satisfying inequalities (18) also satisfies inequalities (17); in fact, each column r whose variable does not appear in inequality (17) has no incoming arcs in $\mathscr{S}$, hence no node in $\mathscr{S}$ is visited, and the variable does not appear in inequality (18) either; at the same time whenever a variable appears in (18) it also appears in (17) with a coefficient not smaller than (18), because it is always needed to select at least an incoming arc in $\mathscr{S}$ to visit its nodes. In the same way, it is easy to show that there exist fractional solutions satisfying inequalities (17) but violating inequalities (18). Therefore, the bound obtained by introducing (18) can be stronger than that found after introducing (17).

For example, let us consider Fig. 4, where three routes (bolded, dotted and dashed arcs), visit a subset $\mathscr{S}$ made by gray nodes. Bolded and dashed routes enter $\mathscr{S}$ once, and have therefore coefficient $\alpha = 1$, while dotted route interleaves nodes inside and outside of $\mathscr{S}$, entering $\mathscr{S}$ twice and having thus $\alpha = 2$. Each route is fractionally selected with value 0.5: the corresponding amount of flow is reported next to each arc. The total incoming flow in $\mathscr{S}$ is 2.0, but the fractional number of routes serving $\mathscr{S}$ is 1.5. Hence an inequality in the form (17) is respected, while the corresponding inequality in form (18) is violated.

We embed the dynamic generation of these inequalities in a column generation scheme separating them at the end of the column generation iterations; the separation algorithm described in Kohl et al. (1999) is used in any case. Whenever violated cuts are found, column generation is executed again. This price-and-cut loop is repeated until neither negative reduced cost columns nor violated cuts are found.

When we add 2-path inequalities to the master problem, they induce a new set of dual variables, which must be taken into account in the solution of the pricing problem. Let $\pi \geqslant 0$ be the dual vector associated with these inequalities. When the weak form (17) is used, the expression of the reduced costs of the routes must be modified by subtracting a term $\alpha_{r\mathscr{S}}\pi(\mathscr{S})$ for each 2-path inequality. This can simply be obtained considering modified arc costs

$$\hat{d}_{ij} = d_{ij} - \sum_{\mathscr{S}:i\in\mathscr{S},j\in\mathscr{N}\setminus\mathscr{S}}\pi(\mathscr{S}).$$

This does not change the structure of the pricing problem. On the contrary, when the strong form (18) is used each inequality (18) requires to introduce a new resource in the state of the dynamic programming algorithm. Such resource is initialized at 1; after the first visit to a customer in $\mathscr{S}$, the resource is set to 0, and the reduced cost of the label is decreased

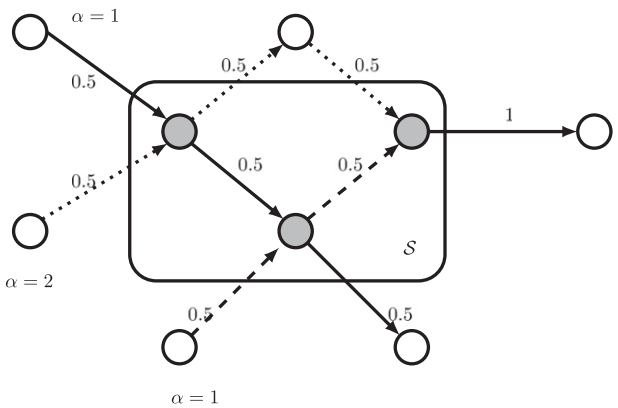

Fig. 4. Partial fractional solution covering eight nodes of a sample instance.

by the value of the corresponding dual variable. No label having one of these resources at 0 can dominate a label having the corresponding resource at 1. Hence, the pricing problem becomes more and more time consuming as these inequalities are introduced.

3.3. Branching strategy

We use two branching policies, described hereafter; these are analogous to those described in Salani (2006) and widely used in the literature for the classical VRP. The tree search policy is discussed in Section 4.

3.3.1. Branching on the number of vehicles

Let $\bar{x}_r$ be the (possibly fractional) value of each variable x_r in the optimal MP fractional solution and let $y_k = \sum_{r\in\Omega_k}\bar{x}_r$ be the (possibly fractional) number of vehicles of type k in such solution. We select the vehicle type $\bar{k}$ whose corresponding y_k variable has its fractional part closest to 0.5; then we perform binary branching, imposing to use at least $\lceil y_{\bar{k}}\rceil$ vehicles of type $\bar{k}$ in one child node and no more than $\lfloor y_{\bar{k}}\rfloor$ in the other.

These branching decisions are handled as follows: first, we modify constraints (3) as follows:

$$l_k \leqslant \sum_{r\in\Omega_{hk}} x_r \leqslant u_k \quad \forall k \in \mathscr{K}, \tag{19}$$

then we set $l_k = \lceil y_{\bar{k}}\rceil$ in one child node, $u_k = \lfloor y_{\bar{k}}\rfloor$ in the other.

The advantage of this branching technique is to leave the pricing subproblem unchanged: dual variables μ_k still appear as constants in the objective function of the pricing problem, but they are now unrestricted in sign. On the other hand constraints (19) belong to the MP, which is solved as a linear programming problem, and tightening them usually results in rather weak improvements in the lower bound.

3.3.2. Branching on arcs

When y_k is integer for all vehicle types $k \in \mathscr{K}$, we apply a different branching rule which forbids the use of some arcs. We choose the customer $i \in \mathscr{N}$ that is split among the largest number of routes in the optimal fractional solution of MP and we forbid half of its outgoing arcs to be used in the first child node, and the other half to be used in the second child node. To handle these branching decisions in the pricing problem it is enough to set travel time of forbidden arcs to $+\infty$.

4. Experimental analysis

4.1. Implementation and hardware

Our algorithms have been implemented in C++, using SCIP 1.1 (Achterberg, 2007) as a branch-and-cut-and-price framework. Our version of SCIP embeds the CPLEX 11 implementation of the simplex algorithm to solve LP subproblems, and automatically switches between primal and dual simplex, depending on the characteristics of each instance. SCIP performs advanced management of the column and row pools, including their removal and re-insertion. We turned off preprocessing and automatic cut generation features, but kept all other parameters at their default values, including the decision tree search policy.

We also experimented on introducing stabilization techniques and using the barrier algorithm to solve the LP subproblems.

In both cases we observed a reduction on the number of column generation iterations needed to converge, but the overall performance improvement was negligible. This is because stabilization and the barrier methods help to reduce the number of initial and useless column generation iterations, but in our case most of the CPU time was spent during few final column generation iterations, when the exact dynamic programming algorithm is called. The use of our pricing heuristics can be considered itself as a useful tool to overcome stability problems.

The results reported in this section are obtained using a single core of an Intel Core 2 Duo 3 GHz workstation, equipped with 2 GB of RAM, running Linux OpenSuse 11.

4.2. Benchmark instances

To test our algorithm we used three datasets.

- Dataset 1 is composed by instances for the HVRPTW proposed by Liu and Shen (1999), derived from the VRPTW instances of Solomon (1987). The original dataset contains 56 problems. Each problem has 100 customers, whose position are generated in the Euclidean plane, and the travel time between nodes is equal to their Euclidean distance. These instances are divided in six classes, depending on the geographical distribution of the customers: random (classes R1 and R2), clustered (C1 and C2) or semi-clustered (RC1 and RC2). Sets R1, C1 and RC1 have a short scheduling horizon and allow for a small number of customers per route. Sets R2, C2 and RC2 have a long scheduling horizon and allow for a large number of cus-

tomers per route. Liu and Shen adapted Solomon's problems to the heterogeneous fleet version by considering vehicles with different capacities. For each of the six classes, three sets of fixed costs for the vehicles were considered, obtaining a total of 168 instances. Since several of these instances are too hard to be solved to proven optimality, we created two new sets of instances by restricting them respectively to the first 50 and 25 customers in the data files. All these instances have a single depot, no constraints on maximum route duration and no limits on the number of vehicles. Therefore, our Dataset 1 is composed by a total of 504 instances.

- Dataset 2 includes 504 more instances, obtained by simply adding a second depot to each instance in Dataset 1. The coordinates of this depot are randomly generated with uniform probability distribution in the smallest rectangle containing all the customers.
- Dataset 3 contains four instances for the MDVRPTW from Cordeau et al. (2001). These are the smallest in their dataset and include only one type of vehicle. In Table 1 we report for each instance in the dataset the number of customers, the number of depots, the maximum allowed duration of the route and the capacity of the vehicle.

All instances considered are symmetric.

Furthermore, we considered the MDHVRPTW instance discussed in Dondo and Cerdá (2009). As presented in Bettinelli et al. (2009), this instance turned out to be very easy to solve using our algorithms, and therefore we omit further detail.

In the remainder we present the outcome of three different sets of experiments. In Section 4.3 we studied the trade-off between the computing time and the tightness of the lower bounds obtained at the root nodes; in Section 4.4 we present the performance of the overall branch-and-cut-and-price algorithm for the exact optimization of the three datasets; in Section 4.5 we discuss the effectiveness of the same algorithm when it is used as a column generation-based heuristic technique.

4.3. Lower bounds

It is known that by dropping elementary conditions on the path, that is running the exact pricing algorithm with decremental state space relaxation and fixing the core $\widetilde{\mathcal{N}} = \emptyset$, the RCESPP becomes more tractable (Righini and Salani, 2008). At the same time, paths containing cycles are super-optimal, and by introducing the corresponding infeasible columns in the RMP the bound is weakened. Nevertheless, the generation of valid inequalities can help in filling the same duality gap, and there is a trade-off between the hardness of the pricing problem, the feasibility of the columns which populate the RMPs and the strength of the cutting planes introduced.

Therefore, in the first set of experiments we compared the dual bounds that can be obtained at the root node of the branch-and-bound tree with different combinations of pricing and cutting strategies.

We considered two pricing strategies: (StrongPrice) to generate columns using the greedy pricing algorithm, the heuristic dynamic programming pricing algorithm and the exact dynamic programming pricing algorithm in sequence, as described in Section 3.1; (WeakPrice) to generate columns by relaxing the elementary path conditions, and using the relaxed pricing procedure described above.

At the same time, we considered three cut generation strategies: (NoCut) without 2-path inequalities; (WeakCut) cuts in the weak form (17); (StrongCut) cuts in the strong form (18). In strategies WeakCut and StrongCut we used the heuristic described in Kohl et al. (1999) to perform cut separation.

A time limit of one hour was imposed to each test.

Tables 2–4 show the results we obtained on the instances of Dataset 1 using respectively NoCut, WeakCut or StrongCut. Each table has one row for each class of instances and their columns indicate the class name, the size and the total number of instances for which the time limit was not exceeded, the average number of column generation iterations needed to converge, the average number of generated cuts, the average duality gap with respect to the best known integer solution and the average time spent during column generation for the two pricing strategies.

As expected, the CPU time increases as the size of the instance increases; the duality gap is not always small: since it is computed using best known primal solutions, this might also depend on poor primal bounds.

We noticed that even finding a valid bound within the time limit is sometimes a difficult task; this becomes more evident as more aggressive cut handling strategies are applied. For instance, a valid bound is obtained on 13 C2 instances with $|\mathcal{N}| = 100$ using StrongPrice, but the price-and-cut loop terminates only 12 times using WeakCut and only 10 times using StrongCut (see Tables 2–4).

No relevant difference can be observed by varying the fixed costs of the vehicles (instances *a*, *b* and *c* in each class). The most noticeable performance gap on Dataset 1 is between Type 1 (R1,C1,RC1) and Type 2 (R2,C2,RC2) instances: the

Table 1
Details of instances in Dataset 3.

Instance	Customers	Depots	Duration	Capacity
pr01	48	4	500	200
pr02	96	4	480	195
pr07	72	6	500	200
pr08	144	6	475	190

Table 2
Lower bounds without cut generation.

Set	Size	StrongPrice				WeakPrice			
		Solved	Iterations	LB (%)	Time	Solved	Iterations	LB (%)	Time
R1	25	36	40.33	0.80	0.11	36	76.42	2.04	0.16
	50	36	71.36	2.36	2.08	36	179.69	3.72	1.71
	100	35	145.91	4.69	287.94	36	441.89	5.74	41.96
C1	25	27	47.26	6.12	0.31	27	71.74	6.84	0.13
	50	27	86.15	5.97	2.86	27	149.48	6.59	0.89
	100	26	168.65	6.68	167.42	27	294.50	7.20	6.01
RC1	25	24	42.88	3.86	0.38	24	73.54	8.52	0.16
	50	24	90.04	9.68	7.37	24	156.04	12.33	1.03
	100	24	168.38	7.35	427.75	24	412.54	9.52	35.17
R2	25	22	124.00	8.36	232.30	33	140.77	26.86	11.47
	50	7	241.29	10.15	301.01	24	381.57	22.73	32.91
	100	2	521.00	8.28	1248.17	3	2276.50	14.35	635.39
C2	25	20	187.70	19.61	234.50	24	136.55	26.60	3.96
	50	15	315.20	14.31	397.24	21	297.53	18.53	24.97
	100	13	1025.00	9.96	1103.55	20	1198.92	11.98	231.87
RC2	25	19	51.26	2.36	161.35	24	114.37	20.48	1.69
	50	13	99.54	0.50	222.90	22	330.85	16.26	95.97
	100	5	208.00	4.70	1098.81	15	553.20	9.90	97.81

Table 3
Lower bounds with WeakCut.

Set	Size	StrongPrice					WeakPrice				
		Solved	Iterations	Cuts	LB (%)	Time	Solved	Iterations	Cuts	LB (%)	Time
R1	25	36	41.00	1.17	0.61	0.11	36	79.22	1.89	1.81	0.14
	50	36	73.22	0.81	2.29	1.58	36	187.00	4.22	3.49	1.81
	100	35	141.34	0.97	4.68	192.56	36	448.49	4.57	5.67	53.35
C1	25	27	45.74	0.07	6.12	0.38	27	74.70	0.74	6.83	0.11
	50	27	83.44	0.04	5.97	1.64	27	151.59	2.78	6.58	0.84
	100	26	162.85	0.04	6.68	98.51	27	300.04	31.23	7.17	7.02
RC1	25	24	42.92	0.38	2.90	0.50	24	86.38	11.00	6.98	0.18
	50	24	89.58	2.92	8.35	8.64	24	187.63	31.17	10.36	1.46
	100	24	166.04	5.46	7.12	428.47	24	471.67	61.38	8.98	65.01
R2	25	22	117.95	0.05	8.35	173.62	33	146.23	0.14	27.58	16.99
	50	7	241.29	0.00	10.15	301.01	24	441.29	0.14	22.74	41.76
	100	2	459.00	0.00	8.28	1378.22	3	2225.00	16.00	14.14	954.97
C2	25	20	169.40	0.10	18.04	232.46	24	142.50	0.10	25.06	4.27
	50	15	315.40	1.47	13.97	397.25	22	314.27	2.27	18.17	22.33
	100	12	1195.42	0.00	10.28	1442.24	19	1233.50	8.92	12.39	237.01
RC2	25	19	49.32	0.00	7.62	66.35	24	122.15	2.40	19.60	2.30
	50	13	99.54	0.00	0.50	222.90	22	365.31	14.23	19.31	153.80
	100	5	203.60	0.00	4.70	1055.61	13	641.80	22.80	9.62	167.56

computation successfully terminates on almost all Type 1 instances, with both pricing strategies and with all cut handling strategies, while timeout expires more often on Type 2 instances as the size of the instances increases. In fact, Type 2 instances have loose capacity constraints, long time horizons and large time windows; this means longer feasible routes and far more numerous labels generated by the pricing algorithm.

Instances in Class C2 show a particular characteristic: they require more column generation iterations and less computing time, and the final duality gap is larger. This can be explained because fractional solutions can exploit both the cluster structure and the capacity of small vehicles, while integer solutions in general cannot.

Comparing the results reported on the same rows in each table (different pricing strategies, same cutting strategy) one can see that WeakPrice is much faster: on average more column generation iterations are needed to converge, but the required CPU time is a small fraction of that required by StrongPrice. In fact, more columns are feasible using WeakPrice: a larger number of columns has to be explored before reaching convergence, even though each of them is easier to generate. However, when a valid bound is found using StrongPrice, the duality gap is significantly smaller. This behavior is observed for all cut strategies.

Table 4
Lower bounds with StrongCut.

Set	Size	StrongPrice					WeakPrice				
		Solved	Iterations	Cuts	LB (%)	Time	Solved	Iterations	Cuts	LB (%)	Time
R1	25	36	41.00	1.17	0.61	0.12	36	78.94	1.75	1.78	0.14
	50	36	73.22	0.81	2.29	1.58	36	187.00	3.89	3.43	1.85
	100	35	141.34	1.03	4.68	193.34	36	451.83	4.49	5.62	59.04
C1	25	27	45.74	0.07	6.12	0.38	27	75.15	0.81	6.81	0.12
	50	27	83.44	0.04	5.97	1.64	27	152.22	3.41	6.55	0.95
	100	26	162.85	0.04	6.68	103.35	27	303.81	25.77	7.12	44.55
RC1	25	24	42.92	0.38	2.90	0.50	24	87.63	8.29	5.97	0.39
	50	24	89.58	2.92	7.38	8.64	24	190.67	29.58	8.28	8.70
	100	23	165.35	5.70	7.00	376.51	24	492.43	45.74	8.34	165.24
R2	25	22	117.91	0.05	8.33	172.23	33	144.91	0.09	26.90	17.50
	50	7	241.29	0.00	10.15	301.86	24	446.00	0.14	22.64	43.45
	100	2	459.00	0.00	8.28	1421.31	3	2254.50	9.50	14.03	1384.21
C2	25	20	180.15	0.10	18.04	229.87	24	155.95	0.10	25.06	4.41
	50	15	315.33	1.47	13.97	400.77	22	324.80	2.53	18.03	26.00
	100	10	1176.60	0.00	10.40	1039.42	19	1213.20	4.20	12.20	225.33
RC2	25	19	49.32	0.00	2.36	67.20	24	125.68	2.16	19.04	3.11
	50	13	99.54	0.00	0.50	225.78	22	418.69	12.54	12.83	506.07
	100	5	201.25	0.00	4.76	871.94	10	722.50	17.75	9.52	731.18

Comparing the results reported in the same block in different tables (same pricing strategy, different cutting strategy), one can see that few cuts are generated with StrongPrice; however, as expected, many more are found with WeakPrice. Cuts are especially effective for WeakPrice in class RC; StrongCut is useful in particular in small R2 and C2 instances, when it is combined with StrongPrice.

For some instances the use of cuts improves the bound and reduces the time as well. A preprocessing rule was responsible for this counter intuitive result: using NoCut when a dual variable λ_i is 0, the corresponding customer i is removed from the pricing problem; when cuts are introduced, π dual variables can make it appealing to visit such customers: this helps heuristic pricing in finding "good" columns earlier, and therefore in speeding up the computation.

We chose to use StrongPrice with WeakCut as the best trade-off. In this way we were able to complete the price-and-cut loop at the root node within the time limit for all instances but one, still obtaining tight dual bounds and allowing the heuristic generation of good columns.

4.4. Branch-and-cut-and-price

In the second set of experiments we let our algorithm perform an exact optimization of each instance in the three Datasets. For these tests we used StrongPrice, heuristic separation of inequalities (17), WeakCut and the branching policy described in Section 3.3. We turned off all SCIP general purpose heuristics, but *fracdiving* and *RENS* (Timo Berthold, 2006) which proved effective in preliminary tests.

A time limit of one hour was imposed to each test.

Tables 5–7 show the results we obtained on the instances of Dataset 1, Dataset 2 and Dataset 3 respectively. As before, each table has one row for each class of instances. In the columns of each table we report in turn the class name, the number of instances solved to proven optimality within the time limit, the average optimality gap on the instances for which optimality was not proved and the average solution time for the solved instances. We remark that for some instances the time limit did not allow even to find an initial dual bound, as reported in Tables 2–4. These have been excluded from this test.

Type 2 classes confirm to be much harder than Type 1. The exact algorithm is fully effective only on small instances: almost all Type 1 classes with 25 customers, 1/3 of those with 50 customers and only 5 of the largest ones are solved to proven optimality. Much less Type 2 instances are solved. On small instances the algorithm is also fast: few seconds are enough to complete each computation. An unexpected bad behavior is observed in class RC1a. By comparing Table 5 with Tables 3 and 8a we notice that the difficulty consists in finding a good primal bound.

Comparing Tables 7, 5 and 6, we conclude that handling multiple depots is not an issue in our algorithm. In fact, even the instance with 72 customers and 6 depots of Dataset 3, involving identical vehicles, can be easily solved. Indeed, our feeling is that managing a heterogeneous fleet is what really complicates the problem: multiple depots naturally favor geometric partitions of customers, and therefore the generation of less overlapping routes, without complicating the pricing algorithm. This is not the case for heterogeneous vehicles: good routes for vehicles with different capacity can be substantially different, and there are no geometric properties to exploit.

Table 5
Results of the exact algorithm on instances in Dataset 1.

Class	Size	Solved	Gap (%)	Time	Set	Size	Solved	Gap (%)	Time
R1a	25	12	0.00	3.59	R2a	25	4	40.35	265.71
	50	7	2.61	452.34		50	1	0.00	1201.98
	100	0	5.04	–		100	0	–	–
R1b	25	11	0.55	1.42	R2b	25	5	4.21	569.78
	50	7	2.91	186.69		50	1	18.99	182.77
	100	1	8.28	271.15		100	0	19.88	
R1c	25	12	0.00	1.88	R2c	25	9	6.59	433.17
	50	8	4.11	631.20		50	2	18.14	486.85
	100	2	10.50	394.75		100	0	9.83	–
R1	25	35	0.55	2.32	R2	25	18	13.84	433.90
	50	22	3.15	432.85		50	4	18.71	589.61
	100	3	7.76	353.55		100	0	14.86	–
C1a	25	9	0.00	7.91	C2a	25	5	65.18	24.35
	50	0	0.67	–		50	3	36.21	801.53
	100	0	2.22	–		100	0	4.77	–
C1b	25	7	0.79	118.76	C2b	25	7	0.00	312.46
	50	1	1.83	0.00		50	4	26.20	162.30
	100	0	3.69	–		100	0	16.41	–
C1c	25	8	3.32	5.44	C2c	25	7	0.00	30.37
	50	1	3.32	647.69		50	4	0.26	151.48
	100	2	1.27	1453.95		100	0	12.05	–
C1	25	24	1.63	39.42	C2	25	19	65.18	132.71
	50	2	1.89	323.85		50	11	15.73	332.70
	100	2	2.48	1453.95		100	0	12.65	–
RC1a	25	1	3.23	120.25	RC2a	25	3	5.36	167.13
	50	0	10.35	–		50	3	3.38	683.29
	100	0	6.01	–		100	0	9.30	–
RC1b	25	8	0.00	6.27	RC2b	25	6	0.00	80.47
	50	2	6.86	78.23		50	4	0.00	95.47
	100	0	12.37	–		100	0	9.90	–
RC1c	25	8	0.00	0.87	RC2c	25	7	0.00	356.01
	50	3	6.66	55.15		50	5	0.00	339.04
	100	0	10.25	–		100	0	9.40	–
RC1	25	17	3.23	10.43	RC2	25	16	5.36	217.26
	50	5	8.27	64.38		50	12	3.38	343.91
	100	0	9.54	–		100	0	9.44	–

4.5. Column generation-based heuristics

While, to the best of our knowledge, no exact algorithm for the MDHVRP has been proposed so far, in the literature a few algorithms showed to perform very well in heuristically solving the problem.

Hence, in the third set of experiments we tuned our algorithm to produce good integer solutions faster: we truncated the optimization process at the root node of the branching tree; then, we tried to combine the columns in the final RMP in a good integer solution. In particular, we considered two adaptations.

- In Algorithm iRMP, the column generation process is stopped after a given time limit α, or when no negative reduced cost column is found; then, integrality conditions on the variables of the RMP are restored, and CPLEX 11 ILP solver is invoked to optimize the ILP problem obtained in this way. CPLEX optimization is stopped after a time limit β, or when optimality is proved.
- In Algorithm Fix&Gen, column generation is performed for up to a given time limit γ; then, each variable taking value 1 in the final RMP fractional solution, and the remaining variable with highest fractional value, are fixed to 1. The reduced problem obtained in this way is re-optimized by performing again column generation for up to a time limit γ. This column generation and fixing process is iterated, until a full integer solution is found.

We compared the results of Algorithms iRMP and Fix&Gen with those obtained by the algorithms of Liu and Shen (LS) (Liu and Shen, 1999), Belfiore and Fávero (BF) (Belfiore and Fávero, 2007), Dell'Amico et al. (DAMPV) (Dell'Amico et al., 2007). We restricted our test to Dataset 1, since algorithms LS, BF and DAMPV have been tested on these instances only.

Table 6
Results of the exact algorithm on instances in Dataset 2.

Class	Size	Solved	Gap (%)	Time	Class	Size	Solved	Gap (%)	Time
R1a	100	0	4.90	–	R2a	100	0	–	–
	50	4	1.92	351.58		50	0	11.09	–
	25	12	0.00	11.71		25	3	31.85	145.67
R1b	100	1	9.19	52.55	R2b	100	0	–	–
	50	5	3.06	455.72		50	0	20.55	–
	25	10	0.04	2.87		25	4	11.90	207.95
R1c	100	1	7.06	127.23	R2c	100	0	9.67	–
	50	7	4.68	599.23		50	1	18.91	139.44
	25	11	0.00	1.06		25	6	12.13	170.72
R1	100	2	6.84	89.89	R2	100	0	9.67	–
	50	16	2.97	492.47		50	1	18.42	139.44
	25	33	0.02	5.48		8	13	14.51	176.39
C1a	100	0	3.45	–	C2a	100	0	24.37	–
	50	0	0.92	–		50	2	15.38	180.24
	25	8	0.29	445.40		25	6	0.00	100.98
C1b	100	0	5.21	–	C2b	100	0	22.03	–
	50	0	3.32	–		50	4	30.71	818.75
	25	7	2.14	245.84		25	7	0.00	134.60
C1c	100	2	1.78	1948.98	C2c	100	0	14.86	–
	50	0	4.13	–		50	3	11.62	1095.53
	25	7	0.31	305.59		25	7	0.00	59.73
C1	100	2	3.69	1948.98	C2	100	0	19.93	–
	50	0	2.79	–		50	9	16.06	769.12
	25	22	1.04	337.42		25	20	0.00	98.31
RC1a	100	0	7.03	–	RC2a	100	0	9.88	–
	50	0	11.16	–		50	1	1.59	114.18
	25	1	4.50	998.31		25	4	7.35	164.88
RC1b	100	0	11.35	–	RC2b	100	0	9.85	–
	50	2	9.65	189.42		50	4	0.00	592.67
	25	8	0.00	351.47		25	5	0.78	39.45
RC1c	100	0	9.86	–	RC2c	100	0	3.49	–
	50	4	10.57	285.75		50	5	0.00	619.54
	25	8	0.00	34.08		25	7	0.00	314.71
RC1	100	0	9.53	–	RC2	100	0	8.28	–
	50	6	10.53	253.64		50	10	1.59	558.26
	25	17	4.50	240.16		25	16	5.71	191.24

Table 7
Results of the exact algorithm on instances in Dataset 3.

Instance	Customers	Depots	Duration	Capacity	UB	LB	Gap (%)	Time
pr01	48	4	500	200	1074.21	1074.21	0.00	2.21
pr07	72	6	500	200	1418.22	1418.22	0.00	834.94
pr02	96	4	480	195	1851.99	1743.53	5.86	3600.02
pr08	144	6	475	190	–	–	–	–

We remark that, although no a-priori optimality guarantee is given by this procedure, the dual bound obtained by column generation allows us to control the quality of the final solution. This cannot be done in any of the LS, BF and DAMPV algorithms.

In Table 8a we report the results of our experiments. The table contains a row for each class of instances in the dataset, and is composed by different blocks, one for each algorithm, as indicated in the leading row. The columns in the first block report the average value of the best solution found by LS and the average CPU time spent; each subsequent block includes the percentage improvement with respect to the value of the LS solution and the average CPU time spent. The CPU time values for algorithms LS, BF and DAMPV are taken from the original papers: according to Dongarra (2009) the CPU time reported in LS, BF, and DAMPV should be scaled respectively by a factor 20, 2 and 10 in order to be equivalent to CPU time in our computer.

First, in order to get roughly the same computing time of DAMPV, we tried to use iRMP algorithm by fixing α to 10 min and β to 5 min. In Type 1 instances iRMP is consistently the best, especially in terms of average solutions cost. In classes R2a and C2a, however, iRMP performs poorly.

Table 8
Evaluation of heuristic algorithms for the MDHVRP.

Set	LS		BF		DAMPV		iRMP(10 + 5 min)		iRMP(1 h + 1 h)		Fix&Gen(1 min)		Fix&Gen(3 min)		Fix&Gen(1 h + 3 min)	
	Value	Time	Δ (%)	Time	Δ (%)	Time	Δ (%)	Time	Δ (%)	Time	Δ (%)	Time	Δ (%)	Time	Δ (%)	Time
(a) Dataset 1																
R1a	4398		−4.27	531	−5.14	921.95	−4.64	106.04	−4.64	134.87	−6.35	139.89	−6.33	161.75	−6.33	112.64
R1b	2054		−5.67	586	−6.23	921.57	−6.83	264.49	−6.83	424.63	−7.75	179.69	−7.48	425.02	−7.27	859.00
R1c	1700		−3.49	605	−5.65	921.53	−6.64	244.65	−6.64	318.56	−6.95	190.41	−6.53	333.61	−6.60	502.51
R1	2717	531	−4.48	574	−5.51	921.68	−6.04	205.06	−6.04	292.68	−6.81	170.00	−6.64	306.79	−6.61	491.38
C1a	8007		−7.31	644	−11.12	921.67	−9.25	32.47	−9.25	25.96	−11.22	18.99	−11.22	18.53	−11.22	20.03
C1b	2485		−3.63	629	−4.00	921.27	−2.48	98.39	−2.48	314.08	−2.51	44.06	−2.66	52.05	−2.57	394.95
C1c	1705		−2.00	665	−4.88	921.28	−2.87	105.67	−2.87	629.88	−1.20	112.50	−1.26	180.39	−1.23	632.06
C1	4065.67	435	−4.32	646	−8.70	921.40	−4.87	78.84	−4.87	323.30	−7.86	58.52	−7.90	83.65	−7.88	349.01
RC1a	5184		−1.03	619	−1.61	921.36	−3.25	370.77	−3.25	904.39	−1.94	218.93	−2.58	519.56	−2.64	841.51
RC1b	2235		−2.00	591	−3.04	920.86	−1.88	272.02	−1.88	454.32	0.57	167.07	0.42	272.36	0.31	378.42
RC1c	1849		−3.18	621	−3.66	921.09	−2.50	183.93	−2.50	194.78	−0.71	129.49	−0.11	199.52	−0.03	251.25
RC1	3089.33	470	−2.07	610	−2.35	921.10	−2.54	275.57	−2.54	517.83	−1.08	171.83	−1.34	330.48	−1.39	490.39
R2a	3809		−2.63	672	−5.29	921.24	45.64	859.36	32.35	4659.52	6.85	300.52	0.44	836.42	0.06	4318.98
R2b	1797		−3.89	806	−4.78	896.98	−7.69	901.42	−16.40	5618.96	−11.82	306.99	−17.44	919.37	−22.39	4371.23
R2c	1513		−4.87	794	−3.96	921.03	−16.27	924.39	−18.21	5700.76	−18.51	313.73	−22.89	854.65	−26.46	4307.33
R2	2373	529	−3.80	757	−4.87	913.08	7.23	895.06	−0.75	5326.41	−2.10	307.08	−8.07	870.15	−9.93	4332.51
C2a	6717		−5.39	672	−10.69	921.30	34.21	766.75	12.40	5118.5	−4.57	225.50	−7.21	582.97	−14.48	3324.05
C2b	1970		−3.26	686	−5.57	921.15	0.35	730.78	−9.44	4035.4	−4.82	175.26	−10.31	584.03	−11.46	2680.96
C2c	1288		−0.91	633	−1.82	683.24	−2.50	557.16	−4.80	2913.88	2.24	176.52	−1.66	434.03	−1.87	2945.29
C2	3325	444	−3.19	664	−8.43	841.90	10.69	684.90	−0.61	4022.59	−3.69	192.43	−7.05	533.67	−12.09	2983.43
RC2a	5273		−8.76	834	−12.42	921.18	−18.64	819.13	−18.99	3566.34	−21.60	601.90	−22.16	1478.85	−22.40	4703.58
RC2b	2324		−5.78	870	−6.64	921.14	−22.89	778.20	−23.72	3582.67	−28.14	372.07	−28.24	1118.27	−28.84	4199.70
RC2c	1978		−4.22	910	−14.09	920.90	−26.05	768.77	−28.01	3642.37	−32.23	362.44	−37.35	1052.91	−39.03	3992.84
RC2	3191.67	533	−6.25	871	−11.29	921.07	−22.52	788.70	−23.57	3597.13	−25.23	445.47	−26.51	1216.67	−27.08	4298.71

File	Best known	iRMP(10 + 5 min)		iRMP(1 h + 1 h)		Fix&Gen(1 min)		Fix&Gen(3 min)		Fix&Gen(1 h + 3 min)	
		Δ	Time	Δ	Time	Δ	Time	Δ	Time	Δ	Time
(b) *Dataset 3*											
pr01	1074.12	0.00	2.22	0.00	2.22	0.00	2.70	0.00	2.75	0.00	2.75
pr02	1762.21	0.03	437.82	0.03	437.82	0.02	215.27	0.01	445.19	0.01	696.14
pr07	1418.22	0.00	23.88	0.00	23.88	0.02	40.27	0.02	37.95	0.02	33.63
pr08	2096.73	0.04	640.82	0.03	3600.20	0.13	589.50	0.04	1106.45	0.06	4504.50

We found out that limit α affects the performance of iRMP more than limit β: since columns in iRMP are generated to be good for fractional solutions, their quality for integer solutions is not guaranteed, especially during the first column generation iterations. Therefore, we also tested iRMP by increasing both α and β to 60 min. No substantial difference was observed in Type 1 classes, since the time limit was almost never exceeded; iRMP improved solutions on Type 2 classes, but still performed worse than other methods in classes R2a and C2a.

Then, we tested Fix&Gen setting time limit γ to 1 min. Indeed, this heuristic is both faster and more accurate, especially in Type 2 classes. We also tried to increase the time limit γ up to 3 min, and we verified that the quality of the solutions keeps improving.

Finally, we tested Fix&Gen by fixing γ to 60 min for the first iteration and to 3 min for the subsequent iterations. On the average the quality of solutions improved, especially in difficult classes (i.e., R2 and C2), even though in some classes (i.e., R1b, C1b, C1c and RC1c) this is not the case. This result is obtained at the expense of a significant increase in computing time.

We also tested our algorithms on Dataset 3, which is difficult because all vehicles have to be used in any feasible integer solution. This makes, in fact, local search more appropriate than iterative route generation. Furthermore, the fixing of a column is likely to affect the overall solution, or even to make it impossible to obtain a feasible one; this effect is more evident when time limit γ is small, and is therefore needed to fix one of the columns produced in early column generation iterations. Nevertheless, we still obtain good results, matching the best known solution in 2 instances and obtaining solutions whose value is within 4% of the best known.

The choice of the right heuristic is obviously instance-dependent. However, Fix&Gen seems to give the best trade-off between quality of the solution and computational effort, especially setting γ between 1 and 3 min.

5. Conclusions

In this paper we have exploited the intrinsic flexibility of a known mathematical programming framework to solve an important optimization problem in distribution logistics, namely the MDHVRPTW.

Besides being flexible the method allows for different combinations of cutting and pricing strategies. We have presented an experimental evaluation of these strategies in terms of lower bound tightness and computing time. This suggests to investigate mixed strategies: for instance it may be appealing to initialize the RMP using WeakPrice, repair the columns in the RMP by removing cycles, then switch to StrongPrice, including WeakCut in the price-and-cut loop. A switch to StrongCut has to be considered only at the end of this process, if the duality gap is still large. Since the number of column generation iterations using WeakPrice is sometimes large, the introduction of a stabilization technique seems appropriate (Rousseau et al., 2007).

When the size of the instance prevents the algorithm to find provably optimal solutions, it is still possible to use the algorithm as a column generation-based heuristic, which is competitive and often better than constructive or local search-based heuristics from the literature, with the additional advantage of providing also dual bounds.

Finally our experimental results reveal that what makes the MDHVRPTW difficult is the number of different vehicle types rather than the presence of multiple depots. The tightness of the time windows also plays an important role as already highlighted in the literature.

Acknowledgements

The authors are grateful to Jacques Desrosiers for his useful comments, following the presentation of a preliminary version of this paper at Odysseus 2009 conference in Izmir (Turkey).

References

Achterberg, T., 2007. Constraint Integer Programming. PhD thesis, Technische Universität Berlin.

Baldacci, R., Christofides, N., Mingozzi, A., 2007. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. Mathematical Programming 115, 351–385.

Baldacci, R., Mingozzi, A., 2009. A unified exact method for solving different classes of vehicle routing problems. Mathematical Programming 120 (2), 347–380.

Belfiore, P.P., Fávero, L.P.L., 2007. Scatter search for the fleet size and mix vehicle routing problem with time windows. Central European Journal of Operations Research 15, 351–368.

Timo Berthold, 2006. Primal Heuristics for Mixed Integer Programs. Master's thesis, Technische Universität Berlin.

Bettinelli, A., Ceselli, A., Righini, G., 2009. A Branch-and-price Algorithm for the Multi-depot Heterogeneous Fleet Vehicle Routing Problem with Time Windows, AIRO Winter.

Ceselli, A., Righini, G., Salani, M., 2009. A column generation algorithm for a rich vehicle-routing problem. Transportation Science 43 (1), 56–69.

Chabrier, A., 2006. Vehicle routing problem with elementary shortest path based column generation. Computers and Operations Research 33 (10), 2972–2990.

Christofides, N., Mingozzi, A., Toth, P., 1981. State-space relaxation procedures for the computation of bounds to routing problems. Networks 11, 145–164.

Cordeau, J.F., Laporte, G., Mercier, A., 2001. A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society 52, 928–936.

Dell'Amico, M., Monaci, M., Pagani, C., Vigo, D., 2007. Heuristic approaches for the fleet size and mix vehicle routing problem with time windows. Transportation Science 41 (4), 516–526.

Desaulniers, G., Desrosiers, J., Solomon, M.M., 2005. Column Generation. Springer.

Dondo, R., Cerdá, J., 2007. A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. European Journal of Operational Research 176, 1478–1507.
Dondo, R., Cerdá, J., 2009. A hybrid local improvement algorithm for large-scale multi-depot vehicle routing problems with time windows. Computers & Chemical Engineering. 33 (2), 513–530.
Dongarra, J.J., 2009. Performance of Various Computers using Standard Linear Equations Software. Technical Report, University of Tennessee. <http://www.netlib.org/benchmark/performance.ps>.
Dror, M., 1994. Note on the complexity of the shortest path models for column generation in VRPTW. Operations Research 42, 977–978.
Dullaert, W., Janssens, G.K., Sorensen, K., Vernimmen, B., 2002. New heuristics for the fleet size and mix vehicle routing problem with time windows. Journal of the Operational Research Society 53, 1232–1238.
Dumas, Y., Desrosiers, J., Gelinas, E., Solomon, M.M., 1994. An optimal algorithm for the travelling salesman problem with time windows. Operations Research 42, 626–642.
Feillet, D., Dejax, P., Gendreau, M., Gueguen, C., 2004. An exact algorithm for the elementary shortest path with resource constraints: application to some vehicle routing problems. Networks 44, 216–229.
Fukasawa, R., Longo, H., Lysgaard, J., Poggi de Aragão, M., Reis, M., Uchoa, E., Werneck, R.F., 2006. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Mathematical Programming 106, 491–511.
Golden, B., Raghavan, S., Wasil, E. (Eds.), 2008. The Vehicle Routing Problem: Latest Advances and New Challenges. Springer.
Golden, B.L., Assad, A., Levy, L., Gheysens, F., 1984. The fleet size and mix vehicle routing problem. Computers and Operations Research 11, 49–65.
Kallehauge, B., 2008. Formulations and exact algorithms for the vehicle routing problem with time windows. Computers and Operations Research 35 (7), 2307–2330.
Kallehauge, B., Boland, N., Madsen, O.B.G., 2007. Path inequalities for the vehicle routing problem with time windows. Networks 49 (4), 273–293.
Kellerer, H., Pferschy, U., Pisinger, D., 2005. Knapsack Problems. Springer Verlag.
Kohl, N., Desrosiers, J., Madsen, O.B.G., Solomon, M.M., Soumis, F., 1999. 2-path cuts for the vehicle routing problem with time windows. Transportation Science 33, 101–116.
Laporte, G., Norbert, Y., Desrochers, M., 1985. Optimal routing under capacity and distance restrictions. Operations Research 33, 1050–1073.
Liu, F.H., Shen, S.Y., 1999. The fleet size and mix vehicle routing problem with time windows. Journal of the Operational Research Society 50, 721–732.
Martello, S., Toth, P., 1990. Knapsack Problems: Algorithms and Computer Implementations. Wiley, New York.
Pessoa, A., Poggi de Aragão, M., Uchoa, E., 2009. A robust branch-and-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. Networks 54 (4), 167–177.
Polacek, M., Hartl, Richard F., Doerner, K., Reimann, M., 2004. A variable neighborhood search for the multidepot vehicle routing problem with time windows. Journal of Heuristics 10, 613–627.
Prindezis, N., Kiranoudis, C.T., 2005. An internet-based logistics management system for enterprise chains. Journal of Food Engineering 70 (3), 373–381.
Righini, G., Salani, M., 2006. Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optimization 30 (3), 255–273.
Righini, G., Salani, M., 2008. New dynamic programming algorithms for the resource constrained elementary shortest path problem. Networks 51, 155–170.
Rousseau, L.M., Gendreau, M., Feillet, D., 2007. Interior point stabilization for column generation. Operations Research Letters 35 (5), 660–668.
Salani, M., 2006. Branch-and-Price Algorithms for Vehicle Routing Problems. PhD thesis, Universita degli Studi di Milano.
Santos, L., Coutinho-Rodrigues, J., Current, J.R., 2008. Implementing a multi-vehicle multi-route spatial decision support system for efficient trash collection in Portugal. Transportation Research Part A: Policy and Practice 42 (6), 922–934.
Solomon, M.M., 1987. Algorithms for the vehicle routing and scheduling problems with time windows constraints. Operations Research 35, 254–265.