Math. Prog. Comp. (2014) 6:171–197
DOI 10.1007/s12532-014-0064-0

FULL LENGTH PAPER

A branch-and-price algorithm for the multi-depot heterogeneous-fleet pickup and delivery problem with soft time windows

Andrea Bettinelli · Alberto Ceselli · Giovanni Righini

Received: 26 March 2012 / Accepted: 10 January 2014 / Published online: 1 February 2014

Abstract The growing cost of transportation and distribution pushes companies, especially small and medium transportation enterprises, to form partnership and to exploit economies of scale. On the other hand, to increase their competitiveness on the market, companies are asked to consider preferences of the customers as well. Therefore, tools for logistics management need to manage collective resources, as many depots and heterogeneous fleets, providing flexible preference handling at the same time. In this paper we tackle a pickup and delivery vehicle routing problem involving such aspects; customers place preferences on visiting time (represented as soft time windows), and their violation is allowed at a price. Our interest in this problem stems from an ongoing industrial project. First we propose an exact branch-and-price algorithm, having as a core advanced dynamic programming techniques. Then we analyze through a computational campaign the impact of soft time windows management on the optimal solution in terms of both routing and overall distribution costs. Our experiments show that our approach can solve instances of real size, and clarify the practical usefulness of soft time windows management.

Mathematics Subject Classification 90C27 · 90C90 · 90B06 · 65K05

1 Introduction

Transportation and distribution are major cost factors in the supply chain. In recent years the introduction of decentralized production and delivery schemes yielded systems of growing complexity, and consequently calls for the development of more and

A. Bettinelli (✉)
DEI, Universitá di Bologna, Viale del Risorgimento 2, 40136 Bologna, Italy
e-mail: andrea.bettinelli@unibo.it

A. Ceselli · G. Righini
Dipartimento di Informatica, Polo Didattico e di Ricerca di Crema, Università degli Studi di Milano, Via Bramante 65, 26013 Crema, Italy

more effective computational tools for logistics management. As highlighted in [23], such an issue arises in many diverse application scenarios, such as urban courier services, less-than-truckload transportation, or door-to-door transportation services [28].

The Capacitated Vehicle Routing Problem (CVRP) [15] plays a fundamental role among classical optimization models in transportation. It deals with the delivery of goods by a fleet of trucks from a central depot to many customer locations. The main goal is to find a set of minimum cost routes for the vehicle fleet in order to meet customers demands and without violating capacity constraints. The Pickup and Delivery Problem (PDP) [5] is a generalization of the CVRP in which each customer requires to pickup a certain amount of goods from an origin location and to deliver it to a destination location. Each route must satisfy pairing constraints (origin and destination must be visited by the same vehicle, without any transshipment) and precedence constraints (the origin location must precede the destination in the route). For comprehensive reviews on routing problems involving pickup and delivery, the reader is referred to the papers by Savelsbergh and Sol [26], Cordeau et al. [11], Parragh et al. [19] and Berbeglia et al. [5].

The interest in basic CVRPs and PDPs is more academic than practical, as real world applications often require to take into account several additional details. In fact, in a typical industrial scenario, small and medium companies tend to aggregate with each other, in order to be more competitive and to exploit economies of scale. As a consequence there is the need to manage several depots and a fleet of heterogeneous vehicles, with different capacities and operating costs; it is also common that customers do not accept vehicles visiting them for pickup or delivery outside some working time windows [7].

Coping with these constraints is often not enough for a transportation company to gain competitiveness on the market. To increase the level of service also *preferences* of the customers must be considered. The most typical one is on the pickup and delivery time: inside their working time windows, customers prefer to be served in very narrow time slots, matching their internal schedule for the flow of goods. Trying to satisfy all these preferences is often not possible. Therefore, models in the literature introduce the following feature: the service is always performed in the working time slots, but the preference time slots can be violated by paying a suitable penalty [29]. Working and preference time slots respectively represent *hard* and *soft* time windows.

Several heuristic approaches have been proposed for the VRP with soft time windows: Koskosidis et al. [17] developed an optimization-based heuristic; Balakrishnan [2] proposed several constructive heuristics, while Taillard et al. [30] and Chiang and Russell [8] presented tabu search heuristics; Ibaraki et al. [16] studied acceleration techniques for local search algorithms in the case of multiple soft time windows.

Recently, Liberatore et al. [18] and Qureshi et al. [21] proposed branch-and-price approaches for the VRP with soft and semi-soft time windows respectively.

The PDP with hard time windows has been widely studied, especially in the version in which a single depot and a homogeneous fleet are available. Several exact approaches based on branch-and-cut [10,24] and branch-and-price [3,13,23,27] have been proposed.

The PDP with soft time windows is much less studied. It was first addressed in an early work by Sexton and Choi [29], where the authors developed a heuristic algorithm based on Benders' decomposition.

Original contribution. In this paper we consider all the above features at once. According to standard vehicle routing taxonomy, we refer to this problem as the Multi-Depot Heterogeneous-fleet Pickup and Delivery Problem with Soft Time Windows (MDHPDPSTW). To the best of our knowledge no algorithm is proposed in the literature, that is able to fully cope with all MDHPDPSTW features. Our interest in the MDHPDPSTW is motivated by an ongoing industrial project, whose aim is to develop an integrated intelligent system for small transportation companies operating in the urban area of Milan (Italy).

First, we propose an exact branch-and-price algorithm for the MDHPDPSTW. The core of our method is a pricing routine based on advanced dynamic programming techniques. The adaptation of known algorithmic paradigms, such as column generation and dynamic programming, to the features of this particular pick-up and delivery problem is not straightforward; our main original contributions are outlined in Sect. 3.2. We also report on the outcome of an extensive experimental campaign, designed to evaluate the impact of soft time windows both in terms of routing and overall costs.

The paper is organized as follows: in Sect. 2 we describe a set covering formulation for the problem; in Sect. 3 a branch-and-price algorithm is proposed and Sect. 4 is devoted to the experimental analysis; finally, in Sect. 5, we draw some conclusions.

2 Problem formulation

Given a set $\mathcal{K}$ of vehicle types, a set $\mathcal{H}$ of depots and a set $\mathcal{N}$ of customer requests, the problem can be defined on a directed graph $G = (V, A)$, where V contains a vertex for each depot $h \in \mathcal{H}$ and for the pickup and delivery locations of each customer $i \in \mathcal{N}$. Let $\mathcal{P}$ and $\mathcal{D}$ be the sets of vertices corresponding to all pickup and delivery locations respectively. We denote with $\xi^+(i)$ and $\xi^-(i)$ the pickup and delivery vertices of customer $i \in \mathcal{N}$ and with $\phi(j)$ the customer to whom vertex $j \in \mathcal{P} \cup \mathcal{D}$ belongs to. Non negative weights d_{ij}^k and t_{ij}^k are associated with each arc $(i, j) \in A$: they represent the transportation cost and the traveling time for each vehicle $k \in \mathcal{K}$ respectively. With each vertex $j \in V$ are associated a load variation q_j (positive for pickup and negative for delivery operations), a service time s_j, a hard time window $[A_j, B_j]$ and a soft time window $[a_j, b_j]$; if the service at location j starts inside its soft time window no penalty is incurred, otherwise a linear penalty must be paid, proportional to the anticipation or delay through non-negative coefficients α_j and β_j respectively. Let T_j be the starting time of service at location j; the penalty term $\pi(T_j)$ is defined as follows:

$$\pi(T_j) = \begin{cases} \alpha_j(a_j - T_j) & \text{if } T_j < a_j \\ 0 & \text{if } a_j \leq T_j \leq b_j \\ \beta_j(T_j - b_j) & \text{if } T_j > b_j. \end{cases}$$

Each vehicle type $k \in \mathcal{K}$ has given capacity w_k and fixed cost f_k. A limited number of vehicles is available: at most u_{hk} vehicles of type $k \in \mathcal{K}$ can be based at depot $h \in \mathcal{H}$.

This represents the situation where different companies (each of them with its own fleet and depot) share their resources.

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

I all customers are served,
II each customer is visited by one vehicle,
III each route begins at a depot and ends at the same depot,
IV the capacity of the associated vehicle is not exceeded,
V pickup and delivery operations for each customer are performed in the same route (pairing constraints),
VI the pickup vertices are visited before the corresponding delivery vertices (precedence constraints),
VII hard time windows constraints must be satisfied along each route,
VIII the number of available vehicles of each type for each depot is not exceeded.

We consider a set covering formulation of the MDHPDPSTW. We say that a route is feasible if it visits some customers and satisfies conditions III, IV, V, VI and VII. Let Ω_{hk} be the set of all feasible routes using a vehicle of type $k \in \mathcal{K}$ from depot $h \in \mathcal{H}$ and let $\Omega = \cup_{h\in\mathcal{H},k\in\mathcal{K}}\Omega_{hk}$ be their union. We associate a binary variable z_r with each feasible route $r \in \Omega_{hk}$, which takes value 1 if and only if route r is selected. Let e_{ir} be a binary coefficient with value 1 if and only if customer $i \in \mathcal{N}$ is visited by route r. Let c_r be the overall cost of route r, taking into account vehicle fixed cost f_k, routing costs and penalties for soft time windows violations. With these definitions we obtain the following integer linear programming model:

$$\min \sum_{h\in\mathcal{H}} \sum_{k\in\mathcal{K}} \sum_{r\in\Omega_{hk}} c_r z_r \tag{1}$$

$$\text{s.t.} \sum_{h\in\mathcal{H}} \sum_{k\in\mathcal{K}} \sum_{r\in\Omega_{hk}} e_{ir} z_r \geq 1 \qquad \forall i \in \mathcal{N} \tag{2}$$

$$\sum_{r\in\Omega_{hk}} z_r \leq u_{hk} \qquad \forall h \in \mathcal{H},\ k \in \mathcal{K} \tag{3}$$

$$z_r \in \{0, 1\} \qquad \forall h \in \mathcal{H},\ k \in \mathcal{K},\ r \in \Omega_{hk} \tag{4}$$

Constraints (2) are standard set covering constraints, modeling condition I, while constraints (3) impose limits on the maximum number of available vehicles of each type at each depot, modeling condition 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-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 $\mathcal{N}$, thus we use a column generation approach.

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 $|\mathcal{N}|$ columns, one for each customer, representing the optimal paths serving one customer at a time,
- a set of columns representing a feasible MDHPDPSTW solution, found using a greedy procedure, described in Sect. 3.3,
- an additional *dummy column* $\bar{r}$ of very high cost, having $e_{i\bar{r}} = 1 \forall i \in \mathcal{N}$ 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 negative reduced cost columns are found, they are inserted into the RMP.

Let λ and μ be the non-negative dual vectors corresponding respectively to constraints (2), and to constraints (3) rewritten as $\geq$ inequalities. The reduced cost of a column encoding route r has the following form:

$$c_r - \sum_{i \in \mathcal{N}} \lambda_i e_{ir} + \mu_{hk}. \tag{5}$$

The routes generated must comply with conditions III, IV, V, VI and VII and they must be elementary according to condition II: 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_{hk} depend on both k and h. Therefore, at each column generation iteration we have to solve $|\mathcal{K}| \cdot |\mathcal{H}|$ 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); in particular we have capacity, pickup and delivery, hard and soft time windows constraints.

We stop the multiple pricing procedure as soon as negative reduced cost columns are found for some depot and vehicle type. This allows to keep the dual variables more stable and to save time during the earliest column generation iterations.

The RCESPP is NP-hard [12]; we solve it using a bi-directional dynamic programming method, which is based on the algorithm proposed by Liberatore et al. [18] for the VRP with soft time windows, modified to deal with pickup and delivery constraints.

The main difficulty when dealing with soft time windows in column generation algorithms is that the possibility of trading time versus cost generates an infinite number of feasible solutions of the pricing problem that do not dominate one another.

In algorithmic terms this means that the pricing algorithm must take into account an infinite number of Pareto-optimal states. In the algorithm by Liberatore et al. [18] different states are grouped into *labels* by means of a cost function that gives the cost of the corresponding path for each feasible service starting time at the current vertex.

We now describe the modified version of the dynamic programming procedure we use in our algorithm.

3.1.1 Exact dynamic programming

Let us consider first the case of a single depot and a single vehicle type: let o and v be the two distinct copies of the depot, representing the departure and arrival node and let w be the capacity of the vehicle. In the remainder we drop the vehicle type index k when we indicate symbols d_{ij} and t_{ij}.

The solving technique consists of a bi-directional extension of node labels. We propagate forward labels describing partial paths from the starting depot vertex o and backward labels corresponding to partial paths ending in v. Then pairs of labels are joined to obtain complete routes.

Modified costs. In principle, the route corresponding to the column of minimum reduced cost (5) can be found by solving a RCESPP on a graph having costs d_{ij} on edges, and prizes λ_l on vertices. However, it is computationally convenient to work with a cost matrix that satisfies the *delivery triangle inequalities*, i.e. $\bar{d}_{ij} + \bar{d}_{jk} \geq \bar{d}_{ik}$ for all $j \in \mathcal{D}$ [23]. Since we use a bi-directional dynamic programming method, for backward search we enforce the same property on the pickup vertices: $\bar{d}_{ij} + \bar{d}_{jk} \geq \bar{d}_{ik}$ for all $j \in \mathcal{P}$. Therefore, in the following we consider modified traveling costs $\bar{d}_{ij}$ and modified vertex prizes $\bar{\lambda}_l$; we defer the discussion on how to compute these modified costs and prizes to Sect. 3.2.

States and labels. In bi-directional dynamic programming we associate *forward states* and *backward states* to the vertices of the graph. A forward state associated with vertex $i \in V$ represents a path from the depot o to i. Different states associated with the same vertex correspond to different feasible paths reaching that vertex. When a vehicle reaches a vertex it can start the service immediately or it can wait and start the service at a later time in order to reduce penalties in case of early arrival. Therefore from each feasible state an infinite number of non-dominated states can be generated. The dynamic programming algorithm deals with them by grouping states into labels.

Each label represents all the states generated by the same path. A label associated with vertex $i \in V$ is a tuple $L_i = (S, \chi, \Psi, C(T_i), i)$, where i is the last reached vertex, S is the set of customers visited along the path, χ is the load of the vehicle after visiting vertex i, Ψ is the set of "open" customers (the ones for which the pickup operation has been performed but not yet the delivery), C is the cost of the path, T_i is the time at which the service at vertex i begins. In each label the continuous function $C(T_i)$ describes the trade-off between cost and time. This function is piecewise linear and convex, because it is the sum of piecewise linear and convex functions, like the one shown in Fig. 1, one for each visited vertex.

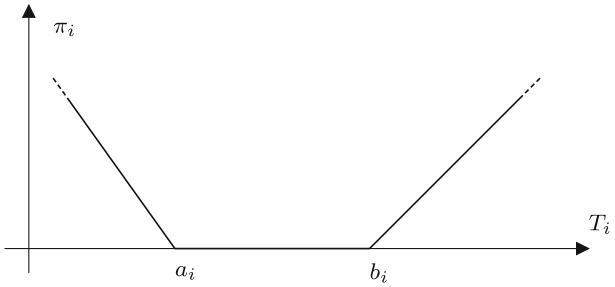

Fig. 1 A soft time window at a generic vertex $i \in V$: a linear penalty π_i is incurred depending on the service starting time T_i

In the same way we define *backward states*, corresponding to paths from vertex i to the final depot v represented by labels $(S, \chi, \Psi, C(T_i), i)$, where T_i is the time at which the service at vertex i ends. Of course here Ψ contains the customers for which the delivery vertex is in the path, but the pickup is not.

Extension. The dynamic programming algorithm iteratively extends all feasible forward and backward labels to generate new ones. The extension of a forward label corresponds to appending an additional arc (i, j) to a path from o to i, obtaining a path from o to j, while the extension of a backward label corresponds to appending an additional arc (j, i) to a path from i to v, obtaining a path from j to v.

In forward extensions the sets S and Ψ are initialized at $\emptyset$, the load χ is set to 0 and the cost function is $C(T_s) = 0\ \forall T_s \geq 0$ at the starting depot vertex o.

The search is restricted to elementary paths by discarding extensions to any vertex j corresponding to the pickup of a customer already in S. To satisfy precedence constraints we also discard extensions to any vertex j corresponding to the delivery location of a customer not in Ψ. Further, we cannot exceed the capacity of the vehicle, thus we must have

$$\chi + q_j \leq w.$$

When a label $(S, \chi, \Psi, C(T_i), i)$ is extended forward to a vertex j, a new label $(S', \chi', \Psi', C(T_j)', j)$ is computed using the following rules.
If j is a pickup vertex

$$S' = S \cup \{\phi(j)\} \tag{6}$$
$$\chi' = \chi + q_j \tag{7}$$
$$\Psi' = \Psi \cup \{\phi(j)\} \tag{8}$$
$$C'(T_j) = C(T_j - (s_i + t_{ij})) + \bar{d}_{ij} + \bar{\lambda}_j + \pi_j(T_j). \tag{9}$$

while if j is a delivery vertex

$$S' = S \tag{10}$$

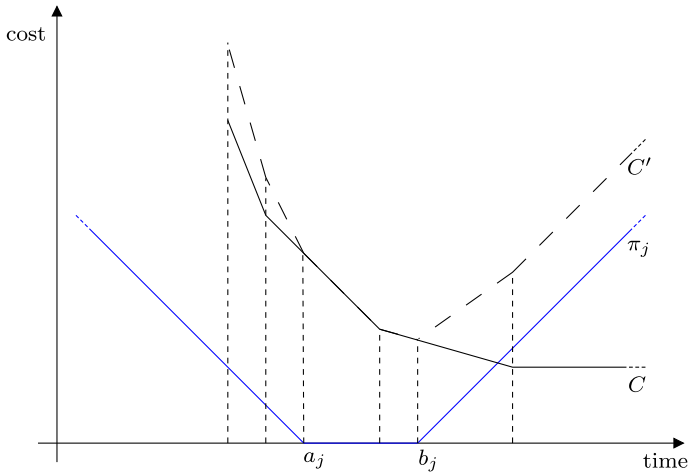

Fig. 2 Forward extension of a label to vertex j. The C' function resulting from the extension is the sum of the C function of the source label suitably shifted and the penalty function π_j

$$\chi' = \chi + q_j \tag{11}$$
$$\Psi' = \Psi \backslash \{\phi(j)\} \tag{12}$$
$$C'(T_j) = C(T_j - (s_i + t_{ij})) + \bar{d}_{ij} + \pi_j(T_j). \tag{13}$$

In this expression the cost function of the predecessor is evaluated at $T_i = T_j - (s_i + t_{ij})$, which is the latest time instant at which the service at vertex i should begin to allow starting the service at vertex j at time T_j. Figure 2 shows an example of forward extension. In graphical terms, the cost function $C(T_i)$ is shifted (black line) to the right by the service time at vertex i, that is s_i, plus the traveling time t_{ij} spent to reach vertex j, and it is shifted vertically by the traveling cost $\bar{d}_{ij}$. Then it is summed to the penalty term π_j, which depends on the arrival time T_j. If $C(T_i)$ is continuous and convex, then the resulting function $C(T_j)$ preserves both these properties. The number of segments in these piecewise linear functions is increased by at most two at every extension.

The extension rules for backward propagation are similar, but the role played by pickup and delivery vertices is switched. Backward initialization consists of starting with a label corresponding to the final depot, with $S = \psi = \emptyset$, $\chi = 0$ and $C(T_v) = 0 \ \forall T_v \leq M$, where T_v represents the arrival time at the final depot v and M is a "big enough" value, i.e. a constant larger than $\max_{j \in V}\{b_j + t_{jv}\}$. When a label $(S, \chi, \Psi, C(T_i), i)$ is extended backward to a vertex j, a new label $(S', \chi', \Psi', C(T_j)', j)$ is computed using the following rules.
If j is a pickup vertex

$$S' = S \tag{14}$$
$$\chi' = \chi + q_j \tag{15}$$
$$\Psi' = \Psi \backslash \{\phi(j)\} \tag{16}$$

$$C'(T_j) = C(T_j + (s_i + t_{ji})) + \bar{d}_{ji} + \hat{\pi}_j(T_j). \tag{17}$$

while if j is a delivery vertex

$$S' = S \cup \{\phi(j)\} \tag{18}$$
$$\chi' = \chi + q_j \tag{19}$$
$$\Psi' = \Psi \cup \{\phi(j)\} \tag{20}$$
$$C'(T_j) = C(T_j + (s_i + t_{ji})) + \bar{d}_{ji} + \bar{\lambda}_j + \hat{\pi}_j(T_j). \tag{21}$$

The cost function of the successor is evaluated at $T_i = T_j + s_i + t_{ji}$ which is the earliest possible time at which the service at vertex i should end in order to end at time T_j the service at vertex j. Since in backward propagation times refer to the end of the service, we have to consider also shifted penalty functions $\hat{\pi}(T_i) = \pi(T_i - s_i)$.

The algorithm iteratively extends forward and backward labels 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 most useful critical resource is the tightest one. In our case the only meaningful choice is time.

Dominance rules. The ability to reduce the number of states generated plays a crucial role in the effectiveness of dynamic programming algorithms. We first remove from each label part of the cost function encoding feasible states that cannot lead to an optimal solution and then we apply dominance rules to fathom labels.

Let us consider a generic forward label, like the one in Fig. 3. Since it is possible to wait at no cost, all the states corresponding to value of service starting time for which the first derivative of the cost function is positive are dominated by states of the same label with smaller cost and time. In graphical terms the rightmost part of the piecewise linear function is always an unbounded horizontal half line, that replaces

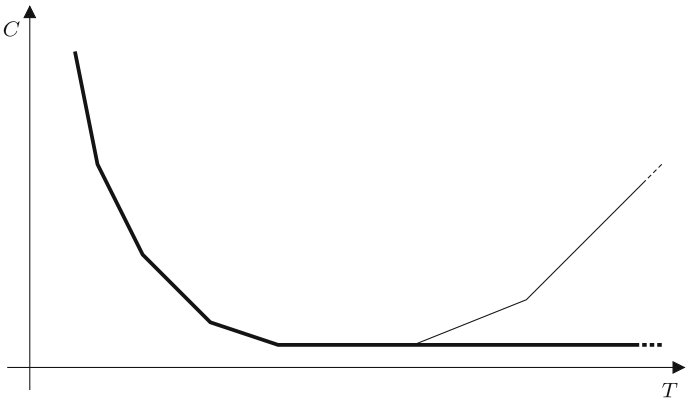

Fig. 3 A generic forward label cost function. The rightmost part of the function, with positive first derivative, is replaced by an horizontal segment

all the segments with positive first derivative. Further, any service starting time T_i that falls outside the hard time window of the last visited customer is infeasible. Hence, the domain of the cost function is restricted to the interval $[A_i, B_i]$. Symmetric arguments can be applied to backward labels: an unbounded horizontal half line replace the leftmost segments, with negative slope, and the domain of the function is bounded by the hard time window of the current vertex.

The second strategy to reduce the number of explored states consists in a test, based on a set of dominance rules, that allows to identify states that cannot lead to an optimal solution of the RCESPP. Those states can of course be discarded without losing the optimality guarantee.

Let $l' = (S', \chi', \Psi', C'(T_i'), i)$ and $l'' = (S'', \chi'', \Psi'', C''(T_i''), i)$ be two forward or backward states associated with node i. Then the former dominates the latter if the following conditions are satisfied

$$
\begin{aligned}
&(a) \quad S' \subseteq S'' \\
&(b) \quad \chi' \leq \chi'' \\
&(c) \quad \Psi' = \Psi'' \\
&(d) \quad T_i' \leq T_i'' \\
&(e) \quad C'(T_i') \leq C''(T_i'').
\end{aligned}
$$

Since for any label the load of the vehicle is uniquely determined by the open customer set, condition (b) is implied by condition (c) and it is, therefore, redundant. Further, as shown in Feillet et al. [14], it is sometimes possible to identify some vertices $u \in \mathcal{N}$ 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 more labels and hence helps to reduce the computational time.

As shown in [23], if the delivery triangle inequalities hold in the modified cost matrix, for forward labels the dominance condition (c) can be relaxed to

$$
(c') \quad \Psi' \subseteq \Psi''
$$

since we have the guarantee that visiting a delivery node cannot make the path cheaper. This weaker dominance rule allows to fathom a larger number of states. The same consideration is true for backward labels if we substitute the delivery triangle inequalities with the pickup triangle inequalities. In Sect. 3.2 we describe a technique for generating suitable modified costs, respecting triangle inequalities for both forward and backward propagation at the same time.

When dominance is applied to single states and the tests succeed, the result is to remove the dominated one. In our case we have labels each representing an infinite number of states. The effect of the dominance test is to delete some parts of the cost functions (see Fig. 4). The states surviving the test are those of minimum cost for

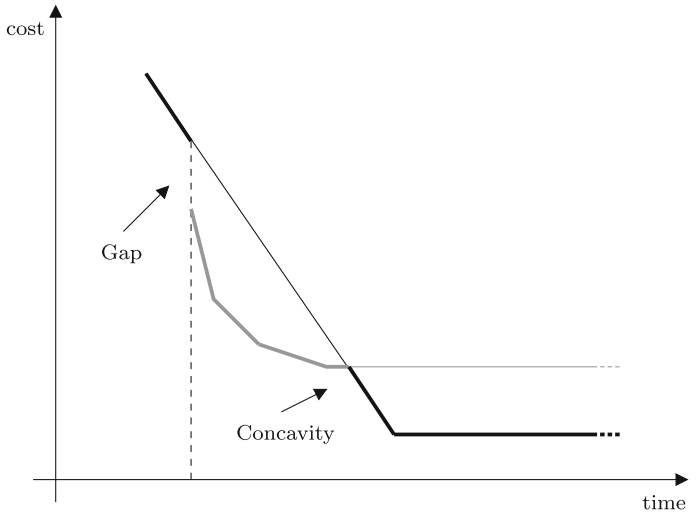

Fig. 4 As an effect of the dominance test, some parts of the piecewise linear functions are deleted. Only the states drawn with heavy lines are non-dominated and survive the test

each feasible starting service time. As a consequence the resulting cost function may contain gaps and it is not convex in general.

When a new label is generated, it is tested against all the other labels referred to the same vertex. If $S' = S''$ and $\Psi' = \Psi''$ the two labels are merged into one. This way at most one piecewise linear function is kept for each combination of S, Ψ and i. In this case the new function may contain vertical gaps and concavities, like the one obtained considering the heavy lines (both black and gray) in Fig. 4. Instead, if $S' \subset S''$ or $\Psi' \subset \Psi''$ only the states in label l'' can be dominated while l' is left unchanged. The new cost function C'' may contain horizontal gaps (black heavy lines in Fig. 4):

Join. Forward and backward paths must be joined together to produce complete o–v paths. The result of the join is a feasible solution if all the resource constraints are satisfied. When a forward path $(S^{fw}, \chi^{fw}, \Psi^{fw}, C^{fw}, i)$ is joined with a backward path $(S^{bw}, \chi^{bw}, \Psi^{bw}, C^{bw}, j)$, the feasibility conditions are:

$$\Psi^{fw} = \Psi^{bw}$$
$$S^{fw} \bigcap S^{bw} = \Psi^{fw}$$
$$T_j^{bw} - T_i^{fw} \geq s_i + t_{ij} + s_j$$

If the join is feasible, then the cost of the resulting path can be obtained as the minimum of the function

$$C(T) = C^{fw}\left(T - s_i - \frac{t_{ij}}{2}\right) + \bar{d}_{ij} + \sum_{l \in \Psi^{fw}} \bar{\lambda}_l + C^{bw}\left(T + s_j + \frac{t_{ij}}{2}\right).$$

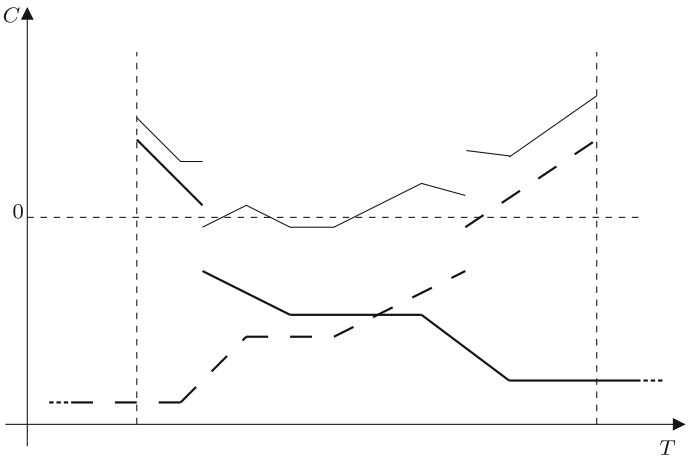

Fig. 5 When a forward and a backward labels are joined together, the two corresponding cost functions (*bold lines*) are summed up. The resulting piecewise linear function (*thin line*) may have multiple local minima. All its points below 0 correspond to negative reduced cost routes

The term $\sum_{l \in \Psi^{fw}} \bar{\lambda}_l$ is needed because the prizes of the customers in $\Psi^{fw} = \Psi^{bw}$ are counted twice, once in the forward and once in the backward label. This function may have several local minima, as shown in Fig. 5. The reduced cost of the global minimum of this function is checked: if it is negative a new column is generated. The detection of the global minimum takes time linear in the number of discontinuity points of the two piecewise linear functions, since it requires a merge operation between two sorted lists (see below).

Also we try to join labels before the end of the extension phase: after generating a bunch of new labels, the Join operation is tentatively performed in search for negative reduced cost columns. If it succeeds, then pricing is stopped; otherwise a new bunch of new labels is generated. In our experiments each bunch included 50,000 new labels.

Implementation details. Cost functions are encoded as a list of segments sorted by increasing values of time. For each segment the start point, the end point, the slope and the intercept of the line containing the segment are stored.

The propagation of a cost function requires, as explained above, a shift operation and a sum between the cost function of the label and the penalty function of the vertex. The shift operation is trivial: we scan all the segments in the line and we increase the coordinates of the extreme points by the required amount. To compute the sum of two cost functions, we scan their lists simultaneously starting from the leftmost segments (i.e., smallest time). At each iteration a segment corresponding to each common time interval is created. The cost value of the extreme points of the new segment is the sum of the values of the two functions at that point in time. Then we move to the next segment of the function with the smallest starting time. The computational complexity is linear in the number of segments of the two functions.

When two labels are merged, the cost function of the resulting label is the minimum of the cost functions of the original labels. As for the extension, a parallel scan of the two functions is performed. At each iteration the next breakpoint is computed. A

breakpoint might be an extreme point of a segment or the intersection point of two segments. For each breakpoint the minimum cost among the two original functions is selected. The computational complexity is linear in the number of segments of the two functions.

A similar operation is performed during pairwise dominance check. Let us assume we have to check if a label L'' is dominated by a label L'. We scan the two cost functions to identify the minimum for each time value. In this case the new segments are added to the resulting function only if they were part of the potentially dominated label L''. If the resulting function does not contain any segment, then L'' is fully dominated and can be discarded. If the resulting function is non-empty, it becomes the new cost function of L''. The computational complexity is again linear in the number of segments of the two functions.

It is of course necessary to store in the labels also the information to get the associated partial path. Merge operations generate labels where a different path might be associated with each segment of the cost function. Thus, this information has to be stored for each segment.

In principle it would be sufficient to store a pointer to the predecessor. Unfortunately, this is not possible in our configuration: in order to save memory, dominated labels are deleted.

Decremental state space relaxation. The dynamic programming algorithm is executed iteratively applying decremental state space relaxation [6,22]. 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 customers* $\tilde{\mathcal{N}}$, and to replace extension rule (6) as

$$S' = (S \cup \{\phi(j)\}) \cap \tilde{\mathcal{N}}$$

This relaxed problem remains an ESPPRC but it 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 $\tilde{\mathcal{N}} = \emptyset$; then we iteratively solve the state space relaxation of the pricing problem and insert in the set $\tilde{\mathcal{N}}$ all the nodes visited more than once in the optimal path, until we find an elementary one.

Other acceleration techniques for this class of labeling algorithms have been recently proposed in the literature. Baldacci et al. [4] proposed a new state-space relaxation, called *ng-route* relaxation for the RCESPP; Pecin et al. [20] developed an efficient ng-route pricing in which a DSSR technique is embedded into the ng-route relaxation. Contardo et al. [9] introduced the so-called strong degree constraints, a new family of valid inequalities that impose partial elementarity.

3.1.2 Heuristic pricing

As stated before, the pricing subproblem is difficult to solve. Following [23], we construct a graph G_ρ in which each vertex $i \in V$ is incident with at most ρ outgoing arcs reaching a pickup vertex and ρ outgoing arcs reaching a delivery vertex. We choose the arcs that are cheapest with respect to $\bar{d}_{ij}$. Then we add the arcs connecting

the starting depot vertex o with the pickup vertices and the delivery vertices with the ending depot v. We then call the dynamic programming method on G_ρ. The described algorithm will be called Restricted Graph Dynamic Programming (RGDP). In our experiments we set $\rho = 10$.

3.2 Costs redistribution

As shown by Ropke and Cordeau [23], it is computationally convenient to work with a cost matrix that satisfies the *delivery triangle inequalities*, i.e. $\bar{d}_{ij} + \bar{d}_{jk} \geq \bar{d}_{ik}$ for all $j \in \mathcal{D},\ \forall i, k \in \mathcal{P} \cup \mathcal{D}$, because it allows to apply weaker dominance rules. The authors proposed a method to transform an arbitrary cost matrix into a cost matrix that satisfies the delivery triangle inequality while maintaining the optimal solution of the pricing problem. However their method does not enforce the pick-up triangle inequality and therefore they could not successfully apply bi-directional dynamic programming. Hereafter we propose a method to redistribute and possibly modify the graph weights in the pricing problem, allowing for bi-directional dynamic programming. For this purpose we need the triangle inequality to hold on the pickup vertices: $\bar{d}_{ij} + \bar{d}_{jk} \geq \bar{d}_{ik}$ for all $j \in \mathcal{P},\ \forall i, k \in \mathcal{P} \cup \mathcal{D}$. To achieve this, we solve the following linear program

$$
\begin{aligned}
&\min\ y \\
&\text{s.t. } \bar{d}_{ij} = d_{ij} + \gamma_i^{out} + \gamma_j^{in} && \forall i, j \in \mathcal{P} \cup \mathcal{D} \cup \mathcal{H} \\
&\quad \bar{d}_{ij} + \bar{d}_{jk} \geq \bar{d}_{ik} && \forall i, j, k \in \mathcal{P} \cup \mathcal{D} \\
&\quad \bar{d}_{ij} + \bar{d}_{jk} \geq \bar{d}_{ik} && \forall i \in \mathcal{H},\ \forall j, k \in \mathcal{P} \cup \mathcal{D} \\
&\quad \bar{d}_{ij} + \bar{d}_{jk} \geq \bar{d}_{ik} && \forall i, j \in \mathcal{P} \cup \mathcal{D},\ \forall k \in \mathcal{H} \\
&\quad \bar{\lambda}_l = \lambda_l + \gamma_{\xi^+(l)}^{in} + \gamma_{\xi^+(l)}^{out} + \gamma_{\xi^-(l)}^{in} + \gamma_{\xi^-(l)}^{out} && \forall l \in \mathcal{N} \\
&\quad \gamma_i^{in} = \gamma_i^{out} && \forall i \in \mathcal{H} \\
&\quad y \geq \bar{\lambda}_l && \forall l \in \mathcal{N} \\
&\quad \bar{\lambda}_l \geq 0 && \forall l \in \mathcal{N}
\end{aligned}
$$

where variables γ_i^{in} and γ_i^{out} represent the cost which is shifted from each vertex to its incident edges. We obtain a modified cost matrix $\bar{d}$ that satisfies the triangular inequalities and a set of prizes $\bar{\lambda}_l$ associated with the customers. It is easy to see that a feasible solution always exists, for large enough values of the variables γ. In case many such matrices exist, the linear program identifies one minimizing the magnitude of vertex prizes. In the dynamic programming procedure, we assign to a path the prize of a customer when the pickup vertex is reached during forward propagation, or when the delivery vertex is reached during backward propagation. In this way we ensure delivery triangle inequalities in the former case and pickup triangle inequalities in the latter. Moreover we have the guarantee that the cost of any o–v path does not change if computed with $\bar{d}$ and $\bar{\lambda}$ instead of d and λ.

3.3 Primal heuristic

We use a simple greedy heuristic to obtain an initial primal bound. We start with an empty route and a randomly selected vehicle. We select the customer (among the ones not yet covered) which can be inserted into the route with the smallest cost increase and we add it. We recall that each customer requires to visit two vertices. Thus, we evaluate their insertion in all the feasible positions along the route. If no customer can be feasibly inserted, we start a new route. The process is iterated until all customer has been satisfied. If the limit on the number of available vehicles is tight the heuristic may fail in finding a complete solution. Nevertheless the columns identified by the algorithm can be used to initialize the restricted master problem, as mentioned earlier.

3.4 Branching strategy

We use two branching policies, described hereafter; they are similar to those described in [25] and widely used in the literature for the VRP.

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

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

$$l_k \leq \sum_{r \in \Omega_{hk}} z_r \leq u_k \quad \forall h \in \mathcal{H}, k \in \mathcal{K} \tag{22}$$

then we set $l_{\bar{h}\bar{k}} = \lceil y_{\bar{h}\bar{k}} \rceil$ in one child node, $u_{\bar{h}\bar{k}} = \lfloor y_{\bar{h}\bar{k}} \rfloor$ 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 (22) 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.

Branching on arcs. When y_{hk} is integer for all combinations of depots $h \in \mathcal{H}$ and vehicle types $k \in \mathcal{K}$, we apply a different branching rule which forbids the use of some arcs. We choose the customer $i \in \mathcal{R}$ 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. Fixing an arc variable to 0 is equivalent to assign the arc an infinite cost. Therefore it is no longer possible to have a modified cost matrix that satisfy pickup (delivery) triangle inequalities. As a consequence, we cannot use dominance rule (c') after branching on arcs and therefore in this case we switch to rule (c).

Table 1 Characteristics of the RCL07 instances

Class	Q	W
AA	15	60
BB	20	60
CC	15	120
DD	20	120

4 Experimental analysis

Implementation and hardware. Our algorithms have been implemented in C++, using SCIP 2.0.1 [1] as a branch-and-cut-and-price framework. Our version of SCIP embeds the CPLEX 12.4 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 we kept all other parameters at their default values, including the decision tree search policy.

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 RAM, running Linux OpenSuse 11. The time limit was set to two hours in all tests.

Benchmark instances. To test our algorithm we considered a set of instances (RCL07) proposed by Ropke and Cordeau [23] for the pickup and delivery problem with time windows. The instances were produced with a generator similar to that initially proposed by Savelsbergh and Sol [27]. In all instances, the coordinates of each pickup and delivery location are chosen randomly, according to a uniform distribution over the $[0, 50] \times [0, 50]$ square; the values of d_{ij}^k and t_{ij}^k are given by Euclidean distances and they do not depend on vehicle types. The load q_i of request i is selected randomly from the interval $[5, Q]$, where Q is the vehicle capacity. A planning horizon of length $T = 600$ is considered, and each time window has width W. In all instances the primary objective consists of minimizing the number of vehicles and a fixed cost of 10^4 is thus imposed on each vehicle type. The instances are grouped in four classes according to the different values of Q and W. The characteristics of these classes are summarized in Table 1. There are ten instances with $30 \leq n \leq 75$ for each class. The name of each instance (e.g., AA50) indicates the class to which it belongs and the number of requests it contains.

Instances RCL07 have hard time windows. In order to test our algorithm, we introduced three soft time windows patterns:

$$\begin{aligned}
&\text{I} \quad a_i = A_i, b_i = B_i,\ \alpha_i = \beta_i = \infty\ \forall i \in \mathcal{N}\ ;\\
&\text{II} \quad a_i = A_i + \tfrac{B_i - A_i}{3}, b_i = B_i - \tfrac{B_i - A_i}{3},\ \alpha_i = \beta_i = 1.5\ \forall i \in \mathcal{N};\\
&\text{III} \quad a_i = A_i + \tfrac{B_i - A_i}{2}, b_i = B_i - \tfrac{B_i - A_i}{2},\ \alpha_i = \beta_i = 1\ \forall i \in \mathcal{N}.
\end{aligned}$$

The corresponding penalty functions are depicted in Fig. 6.

Finally, we derived a forth data-set (RCL07_H) with two depots and three vehicle types with capacities 15, 20 and 25 and fixed costs 100, 150, and 200. Only type II

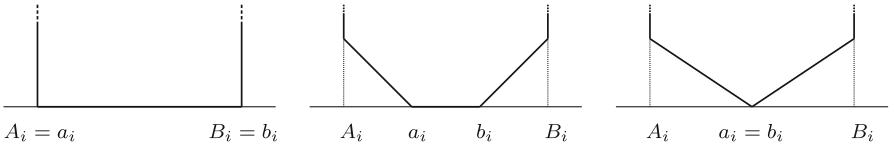

Fig. 6 Penalty functions corresponding to the different soft time windows types

time windows were considered for this data-set, being the most generic and realistic version of soft time windows.

We also considered another data-set (LI_LIM) which includes a subset of the instances proposed by Li and Lim for the PDPTW. These instances originate from the Solomon VRPTW instances and they involve approximately 100 customer requests.

In Sect. 4.1 we present the results of the overall branch-and-price algorithm; in Sect. 4.2 we discuss the impact of the soft time windows on the optimal solutions of the pickup and delivery problem.

4.1 Branch-and-price

The first set of experiments concerns the branch-and-price algorithm. It uses the heuristic pricer with the restricted graph (RGDP) combined with the exact dynamic programming. We consider data-set RCL07 with the three different types of time windows (type I, II and III), where type I coincides with hard time windows, and the other two types correspond to soft time windows. In Tables 2, 3 and 4 we show the results obtained

Table 2 BCP average results, data-set RCL07, type I

Class	Solved	Avg. gap %	Avg. time	Avg. nodes
AA	10/10	0.0	355.9	128.2
BB	9/10	0.1	72.1	60.3
CC	7/10	15.8	1,006.8	244.3
DD	6/10	26.6	1,488.7	209.8

Table 3 BCP average results, data-set RCL07, type II

Class	Solved	Avg. gap %	Avg. time	Avg. nodes
AA	10/10	0.0	114.1	21.4
BB	9/10	0.0	41.0	16.8
CC	7/10	15.8	258.3	35
DD	5/10	9.4	290.1	19.2

Table 4 BCP average results, data-set RCL07, type III

Class	Solved	Avg. gap %	Avg. time	Avg. nodes
AA	10/10	0.0	135.2	35.4
BB	9/10	0.0	54.3	35.6
CC	7/10	20.8	516.9	83.6
DD	7/10	23.4	661.5	27.6

by the branch-and-price algorithm on instances featuring time windows of type I, II, and III, respectively. We show average results for each class of instances: in particular, we present the number of solved instances (out of 10), the average percentage gap (for the instances not solved to optimality), the average computing time and average number of explored nodes (for the instances solved to optimality). Detailed results are reported in Tables 5, 6, and 7. For each instance the upper bound, the lower bound and the computing time for the root node, the lower bound, the percentage gap between LB and UB (Gap %), the computational time in seconds and the number of explored nodes for the full branch-and-price method are reported (Table 8). Table 5 has also a column (Z^*) with the best known solutions. As one can see, the approach is robust with respect to the specific penalty function considered. On instances of class CC and DD more severe penalty functions yield shorter computing times; a similar effect is observed on instances of class AA and BB introducing type II time windows. We tried to explain this phenomenon as follows: when hard time windows are wide the pricing algorithm must explore a large search space; introducing penalties allows to consider a smaller number of promising solutions. When time windows are not so wide and type III time windows are introduced, however, such an advantage is lost due to the complexity of label cost functions.

For a few instances the remaining gap at the end of computation is still high. We observed that, in general, tight lower bounds are either obtained at the root or after exploring very few nodes by branching on the number of vehicles. Therefore, we conjecture that large gaps are due to poor upper bounds. Indeed, no sophisticated primal heuristic has been introduced, as this was not the main focus of this work. In addition, these instances have very high vehicles fixed cost: as a consequence, a difference of a single vehicle between the primal and the dual solutions is enough to produce a gap as large as 20–30 %.

In Table 9 and 10 the same information is shown for instances RCL07_H. As mentioned before, these instances have 2 depots and 3 vehicle types. Our algorithm solved, within the time limit, all the instances with narrow time windows (classes AA and BB) and 12 instances out of 20 with broad time windows (classes CC and DD).

We observe that, when tested on instances with time windows of type I (only hard time windows), single depot and homogeneous fleet, the number of instances closed within two hours by our method is similar to that of Ropke and Cordeau [23] and Baldacci et al. [3], even these algorithms are especially tailored to solve a simpler problem.

We have also tested the branch-and-price procedure on data-set LI_LIM. Results are reported in Table 11. These instances proved much more difficult and in several cases our algorithm was unable to complete the evaluation of the root node within the time limit. This is due to the large size size of the instances and to the the width of the time windows. This difficulty in solving LI_LIM instances confirms the experiments of [23].

4.2 Impact of soft time windows

The last set of experiments was conducted in order to understand the impact of the soft time windows on the optimal solutions. In the first test we analyze how the routing cost changes when soft time windows are introduced with respect to a scenario where

Table 5 Data-set RCL07, soft time windows type I

Instance	Z^*	Root node			Branch-and-price			
		UB	LB	Time (s)	LB	Gap %	Time (s)	Nodes
aa30	31,119.1	31,119.1	26,171.0	2.2	31,119.1	0.0	3.3	5
aa35	31,299.8	31,299.8	26,361.3	4.7	31,299.8	0.0	21.5	56
aa40	31,515.9	31,515.9	31,506.6	6.8	31,515.9	0.0	33.7	44
aa45	31,759.8	31,759.8	31,759.8	11.6	31,759.8	0.0	11.6	1
aa50	41,775	41,775	34,092.9	20.5	41,775.0	0.0	83.9	75
aa55	41,907.8	41,907.8	36,988.9	18.2	41,907.8	0.0	37.6	8
aa60	42,140.7	42,140.7	38,887.7	24.3	42,140.7	0.0	117.5	38
aa65	42,250.2	42,250.2	40,017.2	25.2	42,250.2	0.0	131.0	26
aa70	42,452.3	42,452.3	41,239.3	28.1	42,452.3	0.0	2,296.6	714
aa75	52,461.6	52,461.6	43,163.7	55.3	52,461.6	0.0	822.8	315
bb30	31,086.3	31,086.3	22,628.2	2.3	31,086.3	0.0	2.8	3
bb35	31,281.2	31,281.2	27,057.8	3.0	31,281.2	0.0	5.0	9
bb40	31,493.4	31,493.4	30,303.7	4.8	31,493.4	0.0	11.3	8
bb45	41,555.1	41,555.1	32,753.3	10.9	41,555.1	0.0	16.9	14
bb50	41,701	41,701	35,928.4	10.5	41,701.0	0.0	19.3	16
bb55	41,885.7	41,885.7	39,781.5	18.9	41,885.7	0.0	21.7	3
bb60	62,420.1	62,420.1	54,925.4	17.5	62,420.1	0.0	76.4	71
bb65	62,639.1	62,639.1	55,910.3	14.7	62,639.1	0.0	61.4	58
bb70	62,951	62,951	61,532.9	16.1	62,951.0	0.0	434.4	361
bb75	63,127.5	63,127.5	62,229.8	23.1	63,083.8	0.1	t.l.	3,641
cc30	31,087.7	31,087.7	22,810.1	7.5	31,087.7	0.0	8.5	5
cc35	31,230.6	31,230.6	24,248.1	23.0	31,230.6	0.0	45.7	40
cc40	31,358.5	31,358.5	25,289.4	67.0	31,358.5	0.0	175.3	154
cc45	31,509.1	31,509.1	28,939.1	151.2	31,509.1	0.0	481.4	57
cc50	41,685.3	41,685.3	34,058.2	60.0	41,685.3	0.0	419.0	131
cc55	41,836.3	41,836.3	36,425.0	73.1	41,836.3	0.0	1912.9	632
cc60	42,009.3	42,009.8	37,838.7	123.6	42,000.0	0.0	t.l.	1,961
cc65	42,164	52,434.1	39,480.2	194.2	42,141.9	24.3	t.l.	650
cc70	52,201.7	52,240.1	42,116.0	322.8	42,384.2	23.2	t.l.	239
cc75	52,359	52,359	43,562.3	418.1	52,359.0	0.0	4,005.4	691
dd30	21,133.3	21,133.3	18,702.2	24.7	21,133.3	0.0	47.1	21
dd35	31,210.9	31,210.9	21,524.9	32.0	31,210.9	0.0	64.9	46
dd40	31,352.2	31,352.2	23,138.2	77.5	31,352.2	0.0	167.1	52
dd45	31,483.9	31,483.9	24,872.1	193.5	31,483.9	0.0	658.6	200
dd50	31,600.9	31,600.9	26,587.2	227.3	31,600.9	0.0	3,313.1	687
dd55	31,743.3	31,743.9	28,831.3	364.9	31,743.3	0.0	4,681.5	253
dd60	32,069.2	42,559.7	31,458.3	303.1	31,981.4	33.4	t.l.	95
dd65	42,107.3	52,230	35,313.1	249.7	42,087.0	24.3	t.l.	898
dd70	42,214.2	52,604.8	36,690.6	688.4	42,195.8	24.4	t.l.	431
dd75	42,359.9	52,812.8	38,759.5	546.2	42,340.3	24.5	t.l.	302

Table 6 Data-set RCL07, soft time windows type II

Instance	Root node			Branch-and-price			
	UB	LB	Time (s)	LB	Gap %	Time (s)	Nodes
aa30	31,245.1	26,537.8	2.1	31,245.1	0.0	2.7	3
aa35	31,558.1	27,162.7	4.7	31,558.1	0.0	6.2	3
aa40	32,107.0	32,107.0	6.5	32,107.0	0.0	6.5	
aa45	32,838.5	32,799.5	14.0	32,838.5	0.0	37.6	11
aa50	42,055.8	35,272.9	21.7	42,055.8	0.0	48.4	19
aa55	42,224.7	38,014.4	34.1	42,224.7	0.0	63.3	10
aa60	42,761.8	40,069.2	23.6	42,761.8	0.0	120.4	48
aa65	43,125.8	41,388.9	35.1	43,125.8	0.0	199.4	37
aa70	43,776.2	43,169.9	45.2	43,776.2	0.0	320.7	58
aa75	52,864.7	45,104.6	77.5	52,864.7	0.0	335.7	24
bb30	31,183.2	23,336.0	2.3	31,183.2	0.0	3.1	3
bb35	31,481.5	27,888.4	4.5	31,481.5	0.0	6.4	3
bb40	32,027.1	31,215.4	5.9	32,027.1	0.0	8.2	3
bb45	41,732.6	33,716.4	12.0	41,732.6	0.0	15.1	3
bb50	41,989.1	37,068.0	12.1	41,989.1	0.0	16.3	3
bb55	42,468.0	40,926.6	29.7	42,468.0	0.0	41.8	7
bb60	62,898.7	55,987.7	17.6	62,898.7	0.0	25.7	5
bb65	63,235.5	57,199.7	19.8	63,235.5	0.0	43.1	17
bb70	63,944.1	62,945.8	23.9	63,944.1	0.0	209.3	107
bb75	64,455.3	63,788.0	33.5	64,433.6	0.0	t.l.	3,407
cc30	31,380.1	23,959.1	10.9	31,380.1	0.0	11.9	3
cc35	31,576.0	25,602.8	78.7	31,576.0	0.0	85.5	5
cc40	31,878.7	27,074.1	89.4	31,878.7	0.0	244.6	119
cc45	32,640.8	30,955.9	279.0	32,640.8	0.0	570.5	30
cc50	42,326.7	36,354.2	78.7	42,326.7	0.0	322.7	68
cc55	42,983.7	39,133.2	135.1	42,983.7	0.0	239.1	17
cc60	43,363.2	40,940.5	283.4	43,363.2	0.0	333.8	3
cc65	54,363.7	42,752.7	234.2	44,136.3	23.2	t.l.	344
cc70	56,811.3	45,705.9	316.5	45,822.2	24.0	t.l.	102
cc75	53,578.7	47,350.0	526.9	53,474.4	0.2	t.l.	611
dd30	21,629.2	20,294.2	50.0	21,629.2	0.0	57.4	3
dd35	31,491.7	23,427.8	62.9	31,491.7	0.0	96.7	22
dd40	31,684.6	25,208.7	94.1	31,684.6	0.0	104.4	5
dd45	31,971.8	27,262.9	248.3	31,971.8	0.0	265.5	3
dd50	32,292.9	29,054.2	206.7	32,292.9	0.0	926.5	63
dd55	33,141.3	31,549.5	604.2	32,980.1	0.5	t.l.	146
dd60	42,494.3	34,623.6	452.5	34,740.0	22.3	t.l.	104
dd65	42,907.9	38,547.2	483.4	42,889.7	0.0	t.l.	655
dd70	43,256.6	40,156.8	838.7	43,198.0	0.1	t.l.	112
dd75	54,352.8	42,269.9	899.9	43,885.9	23.9	t.l.	48

Table 7 Data-set RCL07, soft time windows type III

Instance	Root node			Branch-and-price			
	UB	LB	Time (s)	LB	Gap %	Time (s)	Nodes
aa30	31,470.9	26,791.3	2.6	31,470.9	0.0	3.3	3
aa35	31,848.6	27,470.6	4.7	31,848.6	0.0	7.2	5
aa40	32,498.0	32,498.0	5.4	32,498.0	0.0	5.4	1
aa45	33,256.1	33,242.5	12.8	33,256.1	0.0	31.4	10
aa50	42,478.3	35,685.4	18.8	42,478.3	0.0	48.2	19
aa55	42,802.2	38,525.1	34.2	42,802.2	0.0	85.3	19
aa60	43,358.7	40,647.7	21.9	43,358.7	0.0	281.5	103
aa65	43,790.4	42,009.7	31.5	43,790.4	0.0	250.8	68
aa70	44,450.7	43,784.3	35.4	44,450.7	0.0	232.4	54
aa75	53,621.6	45,777.8	66.4	53,621.6	0.0	406.9	72
bb30	31,348.7	23,562.9	2.1	31,348.7	0.0	2.9	3
bb35	31,755.7	28,158.6	4.1	31,755.7	0.0	5.5	3
bb40	32,365.2	31,534.2	5.5	32,365.2	0.0	8.1	3
bb45	42,058.0	34,070.4	10.4	42,058.0	0.0	14.1	3
bb50	42,387.8	37,477.0	12.7	42,387.8	0.0	16.8	3
bb55	42,982.8	41,385.6	24.6	42,982.8	0.0	61.0	25
bb60	63,322.7	56,426.1	17.1	63,322.7	0.0	27.6	8
bb65	63,745.2	57,688.7	15.6	63,745.2	0.0	24.6	5
bb70	64,522.4	63,479.8	23.0	64,522.4	0.0	328.2	267
bb75	65,000.0	64,328.1	26.8	64,985.9	0.0	7,200.1	3,040
cc30	31,836.3	24,426.2	9.8	31,836.3	0.0	12.0	5
cc35	32,207.1	26,164.3	22.5	32,207.1	0.0	28.1	3
cc40	32,636.3	27,810.1	85.2	32,636.3	0.0	117.2	15
cc45	33,510.5	31,741.9	152.5	33,510.5	0.0	2,266.7	417
cc50	43,217.7	37,189.9	46.4	43,217.7	0.0	152.6	46
cc55	43,928.2	39,970.1	131.8	43,928.2	0.0	373.1	61
cc60	44,470.7	41,925.7	212.2	44,470.7	0.0	668.6	38
cc65	55,543.0	43,830.1	207.5	45,325.5	22.5	7,200.3	459
cc70	65,682.8	46,913.6	388.6	46,972.5	39.8	7,203.3	107
cc75	54,858.5	48,622.5	737.3	54,824.5	0.1	7,200.1	848
dd30	22,262.1	20,826.9	47.8	22,262.1	0.0	51.6	3
dd35	32,049.4	24,080.3	48.9	32,049.4	0.0	59.2	3
dd40	32,379.1	25,954.1	81.5	32,379.1	0.0	96.0	9
dd45	32,774.4	28,082.6	188.3	32,774.4	0.0	210.1	3
dd50	33,281.4	29,960.9	151.4	33,281.4	0.0	462.3	30
dd55	34,114.7	32,542.7	421.2	34,114.7	0.0	3,051.8	116
dd60	43,374.5	35,707.4	337.1	35,841.4	21.0	7,210.7	75
dd65	44,121.6	39,704.4	340.8	44,121.6	0.0	699.1	29
dd70	55,426.3	41,364.4	873.8	44,681.7	24.1	7,200.4	145
dd75	56,839.0	43,630.2	1,142.6	45,459.0	25.0	7,202.5	72

Table 8 Average results for soft, inner and outer time windows

	Soft TW value			Outer			Inner		
Class	Solved	Avg. time	Avg. nodes	Solved	Avg. time	Avg. nodes	Solved	Avg. time	Avg. nodes
AA	10/10	114.1	21.4	10/10	355.9	128.2	10/10	6.9	3.4
BB	9/10	41.0	16.8	9/10	72.1	60.3	10/10	3.1	1.7
CC	7/10	245.7	40.3	7/10	507.1	169.8	10/10	4.4	18.7
DD	5/10	290.1	19.2	6/10	850.1	201.2	10/10	3.5	3.0

To compute the average time and number of nodes, only the instances solved to optimality by all three configurations have been considered

Table 9 BCP average results, data-set RCL07_H

Class	Solved	Avg. gap %	Avg. time	Avg. nodes
AA	10/10	0.0	1,021.8	83.5
BB	10/10	0.0	536.1	275.3
CC	7/10	15.8	1,533.6	54.6
DD	5/10	6.1	1,016.9	81.4

only hard time windows are imposed. In Table 12, we show average results for the four classes of instances. Only the instances solved to optimality in all the configurations are considered. For each class, we present the routing cost for each type of time windows and, for type II and III (i.e. soft time windows), the percentage difference with respect to type I (i.e. hard time windows). The routing cost increases of up to 15 % when dealing with soft time windows, since in this case one is willing to travel longer distances in order to reduce the penalty cost. We recall that in this test instances soft time windows are cut inside the hard ones and, thus, they are narrower. Hence, in a context where soft time windows do not represent a real cost, but a customer preference, the penalty functions must be carefully weighted, since even moderate penalties for soft time windows violation, may produce significant increase in routing cost.

Let us now consider a different point of view: since tackling soft time windows requires to introduce non-trivial additional algorithms, it is tempting to approximate the problem using only hard time windows. We consider instances RCL07 with time windows type II: therefore each vertex $i \in V$ has a hard time window $[A_i, B_i]$ and a smaller soft time window $[a_i, b_i]$. First, we solve to optimality a PDP with only hard time windows $[A_i, B_i]$, which we call outer time windows; this corresponds to neglect the customer preference, optimizing only company costs. Second, we solve to optimality a PDP with only hard time windows $[a_i, b_i]$, which we call inner time window (see Fig. 7); this corresponds to treat user preferences as constraints, neglecting their impact on company costs.

Then we evaluate the solutions found with respect to the real penalty functions and we compare them to the optimal solutions obtained when solving the problem with soft time windows. The results are shown in Table 13. For each class of instances we show the optimal objective function value obtained for the problem with soft time windows, outer time windows and inner time windows. In addition we show, for outer and inner time windows, the increase of cost (loss) that is caused by neglecting the

Table 10 Data-set RCL07_H, soft time windows type II

Instance	Root node			Branch-and-price			
	UB	LB	Time (s)	LB	Gap %	Time (s)	Nodes
AA30	1,541.9	1,541.9	13.2	1,541.9	0.0	13.2	1
AA35	1,748.4	1,733.9	32.0	1,748.4	0.0	70.1	32
AA40	1,936.1	1,899.1	51.6	1,936.1	0.0	204.1	100
AA45	2,129.8	2,095.9	101.1	2,129.8	0.0	520.1	155
AA50	2,265.2	2,247.8	149.5	2,265.2	0.0	429.9	79
AA55	2,463.6	2,432.4	219.8	2,463.6	0.0	1,372.1	243
AA60	2,654.4	2,635.8	316.1	2,654.4	0.0	777.8	83
AA65	2,775.3	2,767.8	430.4	2,775.3	0.0	478.2	7
AA70	2,949.6	2,926.2	1,176.5	2,949.6	0.0	3,250.4	100
AA75	3,095.5	3,080.8	1,636.6	3,095.5	0.0	3,102.7	35
BB30	1,714.4	1,714.4	6.0	1,714.4	0.0	6.0	1
BB35	2,012.2	2,012.2	12.7	2,012.2	0.0	12.7	1
BB40	2,222.8	2,222.8	20.2	2,222.8	0.0	20.2	1
BB45	2,384.8	2,374.9	35.8	2,384.8	0.0	49.7	7
BB50	2,654.7	2,631.2	68.6	2,654.7	0.0	226.5	79
BB55	2,888.6	2,870.0	81.4	2,888.6	0.0	356.5	106
BB60	3,774.6	3,751.8	68.2	3,774.6	0.0	569.0	457
BB65	4,022.9	4,009.5	111.7	4,022.9	0.0	1,034.7	630
BB70	4,349.6	4,319.5	141.1	4,349.6	0.0	2,662.3	1,419
BB75	4,435.9	4,417.7	218.6	4,435.9	0.0	423.6	52
CC30	1,637.6	1,637.6	21.0	1,637.6	0.0	21.0	1
CC35	1,821.4	1,817.4	57.7	1,821.4	0.0	74.9	7
CC40	1,966.8	1,949.7	100.4	1,966.8	0.0	638.6	74
CC45	2,146.5	2,109.7	225.2	2,146.5	0.0	3,032.8	193
CC50	2,391.0	2,379.0	535.3	2,391.0	0.0	3,046.4	61
CC55	2,583.3	2,572.8	775.4	2,583.3	0.0	2,478.3	45
CC60	2,725.8	2,725.8	1,443.2	2,725.8	0.0	1,443.5	1
CC65	3,377.0	2,901.6	1,625.6	2,910.3	16.0	t.l.	51
CC70	3,798.8	3,099.1	2,104.8	3,106.3	22.3	t.l.	26
CC75	3,550.2	3,249.5	2,926.2	3,254.4	9.1	t.l.	21
DD30	1,664.2	1,662.1	22.2	1,664.2	0.0	23.9	5
DD35	1,884.5	1,880.4	61.8	1,884.5	0.0	71.0	3
DD40	2,076.6	2,066.9	103.0	2,076.6	0.0	207.8	19
DD45	2,275.7	2,262.9	146.3	2,275.7	0.0	665.7	48
DD50	2,488.5	2,432.9	255.5	2,462.1	1.1	t.l.	559
DD55	2,640.5	2,618.5	350.4	2,640.5	0.0	4,116.3	332
DD60	3,172.9	2,846.2	574.5	2,863.6	10.8	t.l.	253
DD65	3,226.7	3,189.3	641.0	3,202.7	0.8	t.l.	268
DD70	3,583.0	3,336.6	1,462.0	3,347.4	7.0	t.l.	84
DD75	3,905.3	3,511.4	2,773.5	3,517.1	11.0	t.l.	37

Table 11 Data-set LI_LIM

Instance	Root node			Branch-and-price			
	UB	LB	Time (s)	LB	Gap %	Time (s)	Nodes
LC1_2_1	2,704.6	2,704.6	9.8	2,704.6	0	9.78	1
LC1_2_2	2,764.6	2,764.6	t.l.	2,764.6	0	t.l.	1
LC1_2_3	10,067.7		t.l.			t.l.	1
LC1_2_4			m.o.			m.o.	0
LC1_2_5	2,702.0	2,702.0	28.4	2,702.0	0	28.39	1
LC1_2_6	2,701.0	2,701.0	108.6	2,701.0	0	108.55	1
LC1_2_7	2,701.0	2,701.0	162.7	2,701.0	0	162.66	1
LC1_2_8	2,689.8	2,689.8	642.9	2,689.8	0	642.87	1
LC1_2_9	10,322.2		t.l.			t.l.	1
LC1_2_10	9,005.7		t.l.			t.l.	1
LR1_2_1	4,819.1	4,819.1	7.2	4,819.1	0	7.16	1
LR1_2_2	4,166.2	4,166.2	2,264.89	4,166.2	0	2,264.89	1
LR1_2_3	7,458.1		t.l.			t.l.	1
LR1_2_4	7,944.1		t.l.			t.l.	1
LR1_2_5	4,221.6	4,220.7	26.7	4,221.6	0	27.45	3
LR1_2_6	7,905.1		t.l.			t.l.	1
LR1_2_7	5,278.9		t.l.			t.l.	1
LR1_2_8	8,171.6		t.l.			t.l.	1
LR1_2_9	3,978.6	3,927.5	1,292.0	3,935.4	1.1	t.l.	92
LR1_2_10	8,052.1		t.l.			t.l.	1
LRC1_2_1	3,606.1	3,603.2	23.9	3,606.1	0	28.7	3
LRC1_2_2	6,847.0		t.l.			t.l.	1
LRC1_2_3	6,023.5		t.l.			t.l.	1
LRC1_2_4	8,160.3		t.l.			t.l.	1
LRC1_2_5	7,867.3		t.l.			t.l.	1
LRC1_2_6	3,360.9	3,360.9	282.7	3,360.9	0	282.7	1
LRC1_2_7	6,648.3		t.l.			t.l.	1
LRC1_2_8	7,076.7		t.l.			t.l.	1
LRC1_2_9	7,287.2		t.l.			t.l.	1
LRC1_2_10	7,540.7		t.l.			t.l.	1

soft time windows. As it can be seen the loss is significant, especially for inner time windows, where it is often necessary to increase the number of vehicles used in order to satisfy the strict time constraints.

5 Conclusions

In this work we have proposed an exact method to solve an important optimization problem in distribution logistics, namely the MDHPDPSTW. Our experiments have

Table 12 Change in the routing costs of the optimal solutions values with different time windows configurations

Class	Type I	Type II		Type III	
	Routing cost	Routing cost	Avg $\Delta\%$	Routing cost	Avg $\Delta\%$
AA	1,691.30	1,786.71	5.34	1,786.90	5.35
BB	1,757.75	1,870.61	6.03	1,889.72	6.98
CC	1,439.69	1,631.44	11.75	1,626.18	11.47
DD	1,232.14	1,445.38	14.75	1,418.06	13.11

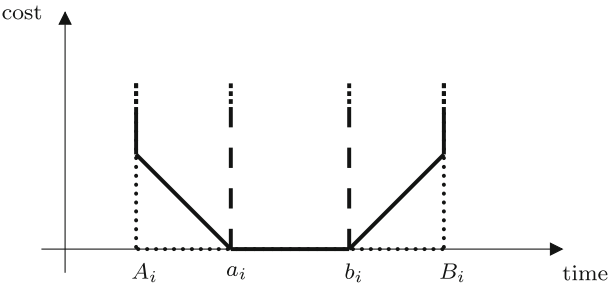

Fig. 7 Representation of soft time window (*black line*), outer hard time window (*dotted line*) and inner hard time window (*dashed line*) at a generic vertex $i \in V$

Table 13 Impact of soft time windows measured by the increase in the objective function when inner or outer hard time windows are used

Class	Soft TW value	Outer		Inner	
		Value	Loss	Value	Loss
AA	38,675.92	39,190.25	514.33	61,866.96	22,462.43
BB	40,540.02	41,107.18	567.16	65,354.56	19,692.36
CC	33,960.45	35,112.73	1,152.29	50,284.82	13,692.09
DD	28,268.47	29,447.99	1,179.52	39,052.05	9,777.47

shown that our approach is successful, solving to proven optimality instances with up to 75 customers, regardless of the penalty pattern and time windows width; when tested on the particular case in which a single depot, a homogeneous fleet and only hard time windows need to be considered, it also matches the performance of state-of-the-art algorithms from the literature, offering at the same time much more modeling flexibility.

Furthermore, we have analyzed the impact of managing customer preferences by soft time windows. Our experiments reveal that such a flexibility comes at the price of an increase of 5–15 % in routing costs; at the same time, disregarding customer preferences in the planning phase, or treating preferences as constraints yields solutions which are likely to be either rejected by the customers, or too expensive.

Future research on this topic can follow two main directions. On one side it would be interesting to extend the model including other real-world operational constraints (e.g. drivers rest periods). A second research direction consists in looking for a more efficient algorithm for the solution of the pricing subproblem, for example including recent ideas such as the above mentioned ng-route relaxation and strong degree constraints.

Acknowledgments This work has been supported by the program "Dote ricerca applicata" funded by Regione Lombardia and Pointcar Servizi Telematici s.r.l.. The authors also acknowledge the support of the Italian Ministry for Education, University and Research, through the project "Modelli e Algoritmi per Problemi di Ottimizzazione Combinatoria nella Gestione di Sistemi di Trasporto" (PRIN 2008).

References

1. Achterberg, T.: Constraint Integer Programming. PhD thesis, Technische Universität Berlin (2007)
2. Balakrishnan, N.: Simple heuristics for the vehicle routing problem with soft time windows. J. Oper. Res. Soc. **44**, 279–287 (1993)
3. Baldacci, R., Bartolini, E., Mingozzi, A.: An exact algorithm for the pickup and delivery problem with time windows. Oper. Res. **59**(2), 414–426 (2011)
4. Baldacci, R., Mingozzi, A., Roberti, R.: New route relaxation and pricing strategies for the vehicle routing problem. Oper. Res. **59**, 1269–1283 (2011)
5. Berbeglia, G., Cordeau, J.F., Gribkovskaia, I., Laporte, G.: Static pickup and delivery problems: a classification scheme and survey. Top **15**(1), 1–31 (2007)
6. Boland, B., Dethridge, J., Dumitrescu, I.: Accelerated label setting algorithms for the elementary resource constrained shortest path problem. Oper. Res. Lett. **34**, 58–68 (2006)
7. Ceselli, A., Righini, G., Salani, M.: A column generation algorithm for a rich vehicle-routing problem. Transp. Sci. **43**(1), 56–69 (2009)
8. Chiang, W.C., Russell, R.A.: A metaheuristic for the vehicle-routeing problem with soft time windows. J. Oper. Res. Soc. **55**, 1298–1310 (2004)
9. Contardo, C., Cordeau, J.-F., Gendron, B.: An exact algorithm based on cut-and-column generation for the capacitated location-routing problem. INFORMS J. Comput. 2013 (forthcoming)
10. Cordeau, J.-F.: A branch-and-cut algorithm for the dial-a-ride problem. Oper. Res. **54**, 573–586 (2006)
11. Cordeau, J.F., Laporte, G., Ropke, S.: Recent models and algorithms for one-to-one pickup and delivery problems. In: Golden, B.l., Raghavan, S., Wasil, E.A. (eds.) Vehicle Routing: Latest Advances and Challenges, pp. 327–357. Springer, Berlin (2008)
12. Dror, M.: Note on the complexity of the shortest path models for column generation in vrptw. Oper. Res. **42**(5), 977–978 (1994)
13. Dumas, Y., Desrosiers, J., Soumis, F.: The pickup and delivery problem with time windows. Eur. J. Oper. Res. **54**, 7–22 (1991)
14. Feillet, D., Dejax, P., Gendreau, M., Gueguen, C.: An exact algorithm for the elementary shortest path with resource constraints: application to some vehicle routing problems. Networks **44**, 216–229 (2004)
15. Golden, B.l., Raghavan, S., Wasil, E.A. (eds.): Vehicle Routing: Latest Advances and Challenges. Springer, Berlin (2008)
16. Ibaraki, T., Imahori, S., Kubo, M., Masuda, T., Uno, T., Yagiura, M.: Effective local search algorithms for routing and scheduling problems with general time-window constraints. Transp. Sci. **39**, 206–232 (2005)
17. Koskosidis, Y.A., Powell, W.B., Solomon, M.M.: An optimization based heuristic for vehicle routing and scheduling with soft time window constraints. Transp. Sci. **26**, 69–85 (1992)
18. Liberatore, F., Salani, M., Righini, G.: A pricing algorithm for the vehicle routing problem with soft time windows. In: Bertazzi, L., Speranza, M.G., van Nunen, J.A.E.E. (eds.) Proceedings of the International Workshop on Distribution Logistics 2006, vol. 619, pp. 251–266, Brescia (2009)
19. Parragh, S., Doerner, K., Hartl, R.: A survey on pickup and delivery problems: Part ii: Transportation between pickup and delivery locations. Journal für Betriebswirtschaft **58** (2008)
20. Pecin, D., Poggi, M., Martinelli, R.: Efficient elementary and restricted non-elementary route pricing. Technical Report 11/13, PUC-Rio (2013)

21. Qureshi, A.G., Taniguchi, E., Yamada, T.: An exact solution approach for vehicle routing and scheduling problems with soft time windows. Transp. Res. Part E Logist. Transp. Rev. **45**, 960–977 (2009)
22. Righini, G., Salani, M.: New dynamic programming algorithms for the resource constrained elementary shortest path problem. Networks **51**, 155–170 (2008)
23. Ropke, S., Cordeau, J.F.: Branch and cut and price for the pickup and delivery problem with time windows. Transp. Sci. **43**, 267–286 (2009)
24. Ropke, S., Cordeau, J.F., Laporte, G.: Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks **49**, 258–272 (2007)
25. Salani, M.: Branch-and-Price Algorithms for Vehicle Routing Problems. PhD thesis, Università degli Studi di Milano (2006)
26. Savelsbergh, M.W.P., Sol, M.: The general pickup and delivery problem. Transp. Sci. **29**, 17–29 (1995)
27. Savelsbergh, M.W.P., Sol, M.: Drive: Dynamic routing of independent vehicles. Oper. Res. **29**, 474–490 (1998)
28. Scott, C., Lundgren, H., Thompson, P.: Guide to Supply Chain Management. Springer, Berlin (2011)
29. Sexton, T.R., Choi, Y.M.: Pickup and delivery of partial loads with soft time windows. Am. J. Math. Manage. Sci. **6**, 369–398 (1986)
30. Taillard, E., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.Y.: A tabu search heuristic for the vehicle routing problem with soft time windows. Transp. Sci. **31**, 170–186 (1997)