Transportation Research Part B 100 (2017) 115–137

Contents lists available at ScienceDirect

Transportation Research Part B

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

# A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations

Gizem Ozbaygin[a,b,*], Oya Ekin Karasan[b], Martin Savelsbergh[a], Hande Yaman[b]

[a] H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332, USA
[b] Department of Industrial Engineering, Bilkent University, Ankara 06800, Turkey

A R T I C L E   I N F O

*Article history:*
Received 31 August 2016
Revised 12 January 2017
Accepted 8 February 2017

*Keywords:*
Vehicle routing
Trunk delivery
Roaming delivery locations
Branch-and-price
Resource-constrained shortest path

A B S T R A C T

We study the vehicle routing problem with roaming delivery locations in which the goal is to find a least-cost set of delivery routes for a fleet of capacitated vehicles and in which a customer order has to be delivered to the trunk of the customer's car during the time that the car is parked at one of the locations in the (known) customer's travel itinerary. We formulate the problem as a set-covering problem and develop a branch-and-price algorithm for its solution. The algorithm can also be used for solving a more general variant in which a hybrid delivery strategy is considered that allows a delivery to either a customer's home or to the trunk of the customer's car. We evaluate the effectiveness of the many algorithmic features incorporated in the algorithm in an extensive computational study and analyze the benefits of these innovative delivery strategies. The computational results show that employing the hybrid delivery strategy results in average cost savings of nearly 20% for the instances in our test set.

## 1. Introduction

A recent survey revealed that in 2016, for the first time, online purchases have surpassed in-store purchases (Farber, 2016). The business-to-consumer retail segment continues to grow year-over-year with an ever-increasing push towards online shopping. As an example, Amazon, one of the e-commerce giants received 398 orders per second and 34.4 million orders in total during its Prime Day event on July 15th, 2015 (Garcia, 2015). A year later, on Prime Day 2016, the number of orders increased by more than 60%, making it the biggest sales day in the history of the company (Gustafson, 2016). Due to the sheer volume, the huge number of delivery locations, and the aggressive service levels promised, last-mile logistics is a huge challenge for Amazon. Amazon's first-quarter shipping costs in 2016 hit $3.27 billion, a 42% increase from the same period in 2015 (Solomon, 2016).

Not surprisingly, Amazon is seeking and exploring innovative ideas to improve the efficiency of last-mile delivery operations. Among these innovative ideas is trunk delivery, where the orders of customers are delivered to the trunk of their cars, and introduced by Amazon in collaboration with Audi and DHL in certain areas (Popken, 2015; Geuss, 2015; Audi, 2015). Volvo too is offering the required technology and has launched an in-car delivery service, initially limited to Stockholm, Sweden (Volvo, 2015; 2016). Recently, Smart has partnered with DHL to start providing trunk delivery service to customers in Germany who order merchandise from Amazon in September 2016 (Behrmann and Weiss, 2016).

* Corresponding author.
*E-mail address:* ozbaygin@bilkent.edu.tr (G. Ozbaygin).

http://dx.doi.org/10.1016/j.trb.2017.02.003
0191-2615/© 2017 Elsevier Ltd. All rights reserved.

Motivated by the interest in trunk delivery, we study the variant of the vehicle routing problem (VRP) in which each customer has an itinerary specifying one or more locations with corresponding time windows where the customer's order can be delivered to the trunk of his/her car (the car will be parked at these locations during the given time windows). This problem is known as the vehicle routing problem with roaming delivery locations (VRPRDL) and was introduced recently by Reyes et al. (2016), who developed various construction and improvement heuristics for the problem.

We consider the static and deterministic version of the VRPRDL, i.e., we assume complete knowledge of the itinerary of each customer and deterministic travel times. It is not unrealistic, when planning delivery routes, to assume that knowledge of a customer's itinerary is available, as many companies, e.g., Roadie roadie.com, are already using analytics to predict the travel patterns of people using the location data collected from the GPS tracking technology in their smart phones. However, given that the actual travel times and customer itineraries may differ when the planned delivery routes are executed, in practice, technology to dynamically adjust routes will also be needed.

The VRPRDL combines two well-studied problems, namely, the VRP with time windows (VRPTW) and the generalized VRP (GVRP). The VRPTW is the problem of determining an optimal set of delivery routes serving the demand of a set of customers within their respective time windows. There is a vast body of literature on the VRPTW and its variants (see, for example, Savelsbergh, 1985; Solomon, 1987; Desrochers et al., 1988; 1992; Dabia et al., 2013; Agra et al., 2013; Schneider et al., 2014; Taş et al., 2014; Koç et al., 2015). The GVRP was introduced by Ghiani and Improta (2000) and it is another generalization of the VRP in which the set of delivery locations is partitioned into clusters and exactly one location from each cluster has to be visited in a solution. Despite its relatively recent introduction, the GVRP has already attracted the attention of many researchers, in part because it has many real-life applications (Baldacci et al., 2010; Bektas et al., 2011; Kovacs et al., 2014; Afsar et al., 2014; Quttineh et al., 2015; Biesinger et al., 2016; Louati et al., 2016).

The integration of the features of these two problems leads to the generalized vehicle routing problem with time windows (GVRPTW). To the best of our knowledge, the first study on the GVRPTW is due to Moccia et al. (2012), who present an incremental tabu search algorithm to solve the problem. The VRPRDL can be seen as a special case of the GVRPTW in which the sets of delivery locations for the customers form the clusters. However, the time windows exhibit a special structure, as the time windows of the locations in a cluster, i.e., the time windows of the delivery locations for a single customer, are non-overlapping. Our computational experiments reveal that it is this special structure that allows us to solve large instances. We note, however, that our solution approach does not explicitly exploit the time window structure and can, thus, also be used to solve instances of the GVRPTW.

The main contribution of our paper is that it presents an effective branch-and-price algorithm for the VRPRDL. Branch-and-price (Barnhart et al., 1998) has established itself as an effective solution methodology for a variety of vehicle routing and scheduling problems, e.g., the VRPTW (Desrochers et al., 1992), the VRP with stochastic demands (Christiansen and Lysgaard, 2007), the VRP with pickup and deliveries (Dell'Amico et al., 2006), and the VRP with split deliveries (Salani and Vacca, 2011). As far as we know, this is the first study in which an exact solution approach for the VRPRDL, or, more generally, for the GVRPTW, is developed. The novelty of our branch-and-price algorithm lies in the fact that it works with location clusters instead of locations. This requires modification of existing techniques for solving the pricing problem and modification of the implementation of certain branching rules. Furthermore, since the GVRPTW generalizes several variants of the VRP, our branch-and-price algorithm can be used to solve instances of these variants as well.

An extensive computational study shows that our branch-and-price algorithm is capable of solving instances of up to 120 customers. We also demonstrate that the algorithm can be used to solve instances of the variant in which a customer order can be delivered either to the customer's home or to the customer's car, although only instances of up to 60 customers. In Reyes et al. (2016), this problem variant is called the VRP with home and roaming delivery locations (VRPHRDL). Finally, we use our algorithm to analyze the cost savings that can be achieved when making optimal use of the option to deliver to the trunk of a car.

The remainder of this paper is organized as follows. Section 2 presents a mixed-integer programming formulation as well as a set-partitioning model for the VRPRDL, describes the pricing problem and how to solve it. Section 3 discusses the techniques employed in our branch-and-price algorithm to increase its efficiency and gives some implementation details. Section 4 explains how the VRPHRDL can be solved with our algorithm. Section 5 provides the results of an extensive computational study, demonstrating the efficacy of the branch-and-price algorithm and analyzing the benefits of trunk delivery. Section 6 concludes with some final remarks.

## 2. Problem definition and formulations

The VRPRDL is formally defined as follows. Let $G = (N, A)$ with $N = \{0, 1, \dots, n\}$ be a complete directed graph in which node 0 corresponds to the depot and the remaining nodes correspond to the locations of interest. Each arc $(i, j) \in A$ has an associated travel time $t_{ij}$ and cost $w_{ij}$ both satisfying the triangle inequality. The set of customers that require a delivery during the planning period $[0, T]$ is represented by $C$. The delivery for a customer $c \in C$ is characterized by a demand quantity $d_c$ and a geographic profile which specifies where and when a delivery can be made. Let $N_c \subseteq N$ denote the set of locations that customer $c$ will visit during the planning horizon. By duplicating locations, we may assume $N_c \cap N_{c'} = \emptyset$ for different customers $c, c' \in C$. Note that we can express the set of nodes as $N = N_0 \cup \{i \in N_c \mid c \in C\}$, where $N_0 = \{0\}$. The locations $i \in N_c$ for $c \in C$ have non-overlapping time windows $[e_i, l_i]$ during which the delivery can take place and correspond to the customer's vehicle itinerary during the planning horizon. We use $c(i)$ to denote the customer associated with location $i$ and we let $c(0) = 0$. A fleet of $m$ homogeneous vehicles, each with capacity $Q$, is available to make deliveries; vehicles start and end their delivery routes at the depot. The goal is to find a set of delivery routes visiting each customer at one of the locations in the customer's itinerary, during the time that the customer is at that location, and such that the demand delivered on a route is no more than $Q$, the duration of a route does not exceed $T$, and the total cost is minimized.

The basic version of the VRPRDL is static and deterministic, i.e., it is assumed that all customer locations and the time spent at these locations are known with certainty for the entire planning horizon. Let $x_{ij}$ for be a binary variable indicating whether arc $(i, j) \in A$ is used or not. We write $x(A') = \sum_{(i,j)\in A'} x_{ij}$ for $A' \subseteq A$ and we use $\delta^-(i)$ and $\delta^+(i)$ to denote the sets of incoming and outgoing arcs of node $i$, respectively. The basic version of the VRPRDL can be formulated as follows:

$$\min \sum_{(i,j)\in A} w_{ij} x_{ij}$$

$$x(\delta^-(i)) - x(\delta^+(i)) = 0 \qquad \forall i \in N, \tag{1}$$

$$\sum_{i\in N_c} x(\delta^-(i)) = 1 \qquad \forall c \in C, \tag{2}$$

$$x(\delta^+(0)) \le m, \tag{3}$$

$$s_j \ge s_i + t_{ij} x_{ij} + (e_j - l_i)(1 - x_{ij}) \qquad \forall (i, j) \in A, j \ne 0, \tag{4}$$

$$s_i + t_{i0} x_{i0} \le \min\{l_i + t_{i0}, T\} \qquad \forall (i, 0) \in A, \tag{5}$$

$$e_i \le s_i \le l_i \qquad \forall i \in N, \tag{6}$$

$$y_j \ge y_i + d_{c(j)} - Q(1 - x_{ij}) \qquad \forall (i, j) \in A, j \ne 0, \tag{7}$$

$$d_{c(i)} \le y_i \le Q \qquad \forall i \in N, \tag{8}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A,$$

where $s_i$ is the arrival time at location $i$ and $y_i$ is the cumulative quantity delivered by a vehicle making a delivery at location $i$ when it is leaving location $i$. Note that the values of the $y_i$ variables are relevant only for those locations that are actually visited. The objective is to minimize the total cost of the delivery routes. Vehicle flow conservation at every location is captured by Constraints (1). Constraints (2) guarantee that each customer receives a delivery at exactly one of the locations in his/her itinerary. Constraints (3) limit the number of vehicle departures from the depot to at most $m$. Constraints (4)–(6) determine the arrival times at each of the locations while ensuring that all vehicles return to the depot by time $T$ and the time windows at the locations are respected. At the same time, these constraints prevent subtours from occurring. Constraints (7) and (8) ensure that the required quantities are delivered to the customers (in combination with Constraints (2)) and that the capacity of the vehicles is respected. Note that this formulation is slightly different from the one presented in Reyes et al. (2016).

The VRPRDL can also be formulated as a set partitioning problem by applying Dantzig-Wolfe decomposition to the formulation above. Let $R$ denote the set of all feasible delivery routes (i.e., respecting capacity and time window constraints), let $w_r$ be the cost of route $r \in R$, and let $a_{ir}$ for every $i \in N$ and $r \in R$ indicate whether location $i$ is visited on route $r$ ($a_{ir} = 1$) or not ($a_{ir} = 0$). The set partitioning formulation of the VRPRDL is as follows:

$$\min \sum_{r\in R} w_r z_r \tag{9}$$

$$\sum_{r\in R} \sum_{i\in N_c} a_{ir} z_r = 1 \qquad \forall c \in C, \tag{10}$$

$$\sum_{r\in R} z_r \le m, \tag{11}$$

$$z_r \in \mathbb{Z}_+ \qquad \forall r \in R, \tag{12}$$

where $z_r$ is the number of times route $r$ is used. This formulation has exponentially many variables as the number of routes is exponential in the size of $N$. Therefore, we use column generation to solve its LP relaxation, which will be referred to as the master problem from now on. Note that since the arc costs satisfy the triangle inequality, we can replace $\sum_{r\in R} \sum_{i\in N_c} a_{ir} z_r = 1,\ c \in C$ with $\sum_{r\in R} \sum_{i\in N_c} a_{ir} z_r \ge 1,\ c \in C$ and still obtain a solution in which each customer is visited exactly once. This restricts the associated dual variable to be nonnegative, which typically leads to faster convergence of the column generation procedure.

### 2.1. Pricing problem

Let $\bar{R} \subset R$ be such that there exists a feasible solution to the master problem when $z_r = 0$ for all $r \in R \setminus \bar{R}$. A formulation involving only routes in $\bar{R}$ is called a restricted master problem (RMP). Once RMP is solved to optimality, we check if there exists a column with negative reduced cost with respect to the original master problem. Such a column is a route $r$ for which the following condition is satisfied:

$$w_r - \lambda_0^* - \sum_{c\in C}\sum_{i\in N_c} a_{ir}\lambda_c^* < 0. \tag{13}$$

Note that since $w_r = \sum_{(i,j)\in r} w_{ij}$ and $\lambda_0^* + \sum_{c\in C}\sum_{i\in N_c} a_{ir}\lambda_c^* = \sum_{i\in N} a_{ir}\lambda_{c(i)}^*$, we can rewrite the condition as:

$$\sum_{(i,j)\in r} w_{ij} - \sum_{i\in N} a_{ir}\lambda_{c(i)}^* < 0. \tag{14}$$

Thus, the pricing problem is an elementary shortest path problem with time window and capacity constraints (ESPPTWCC), where the cost of arc $(i, j)$ is set to $w_{ij} - \lambda_{c(i)}^*$ for $i \in N_c$ and $j \in N\backslash N_c$, i.e., the goal is to find an elementary path of shortest length starting and ending at the depot and respecting capacity and time window constraints. The ESPPTWCC can be formulated as follows:

$$\min \sum_{(i,j)\in A} (w_{ij} - \lambda_{c(i)}^*)x_{ij}$$

$$x(\delta^-(0)) = 1, \tag{15}$$

$$x(\delta^+(0)) = 1, \tag{16}$$

$$x(\delta^-(i)) - x(\delta^+(i)) = 0 \qquad \forall i \in N \setminus \{0\}, \tag{17}$$

$$\sum_{i\in N_c} x(\delta^-(i)) \le 1 \qquad \forall c \in C, \tag{18}$$

$$\sum_{c\in C} d_c \sum_{i\in N_c} x(\delta^+(i)) \le Q, \tag{19}$$

$$s_j \ge s_i + t_{ij}x_{ij} + (e_j - l_i)(1 - x_{ij}) \qquad \forall (i, j) \in A, j \ne 0, \tag{20}$$

$$s_i + t_{i0}x_{i0} \le \min\{l_i + t_{i0}, T\} \qquad \forall (i, 0) \in A, \tag{21}$$

$$e_i \le s_i \le l_i \qquad \forall i \in N, \tag{22}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A.$$

Constraints (15)–(17), together with subtour elimination Constraints (20), define a path starting from and ending at the depot. Constraints (18) force this path to be elementary, i.e., they guarantee that each customer is visited at most once. Constraints (19)–(22) enforce the capacity and time window restrictions.

### 2.2. Solving the pricing problem

Solving a mixed integer programming problem using an off-the-shelf solver at every pricing iteration is computationally expensive. Hence, we adopt the iterative label setting algorithm proposed by Boland et al. (2006) to solve the ESPPTWCC. Given a directed graph with arbitrary arc lengths and a specified pair of source and sink nodes $s$ and $t$, ESPPTWCC is the problem of finding the shortest resource-feasible path from $s$ to $t$. As implied by its name, we have two different resources in ESPPTWCC arising from time window and capacity restrictions. Hence, a resource-feasible path in our context is one that respects the availability of time and capacity resources. Moreover, this path should also be elementary, i.e., free of cycles. A common way to ensure that an elementary resource-feasible path is obtained at the end of a label setting algorithm is to use node-visit resources, which were introduced by Feillet et al. (2004). A node-visit resource is a resource with a capacity of one unit, which is consumed when the associated node is visited. Note, however, that the definition of an elementary path is slightly different in the case of the VRPRDL, because multiple locations are associated with a single customer. If nodes $i$ and $j$ are both locations associated with customer $c \in C$, then a path containing both $i$ and $j$ should not be considered elementary because it visits customer $c$ at least twice. Therefore, instead of using node-visit resources, we maintain a customer-visit resource for each customer, which is consumed when any one of his associated locations is included in the path.

Label setting is a dynamic programming approach which constructs resource-feasible paths originating at a source node $s$ and ending at a sink node $t$. Starting with the trivial path containing only the source node $s$, the label setting procedure extends unprocessed partial paths along all feasible arcs to create new (partial) paths. A state is associated with each partial path capturing the cost and the resource consumptions along the path. The extension of a partial path along an arc is infeasible if the resulting path would not be resource-feasible or if it cannot be augmented to reach the sink node within the available resource limits.

The efficiency of a label setting algorithm depends on the dominance relation that is used to eliminate partial paths. Let $P_{si}$ and $P'_{si}$ be two distinct paths from the source node $s$ to node $i$ with their respective states given by the label vectors $\mathbf{U}$ and $\mathbf{U}'$. Suppose that the first element of the label vector represents the cost and the remaining elements represent the resource consumptions (in the same order for both label vectors) along the corresponding path. Path $P_{si}$ is said to dominate path $P'_{si}$ if $\mathbf{U} \neq \mathbf{U}'$ and $U_k \leq U'_k$ for $k = 1, \ldots, K+1$, where $K$ is the number of resources.

To be able to eliminate additional partial paths using the above dominance relation, Feillet et al. (2004) introduce the notion of *unreachable nodes*. When creating a new partial path, the idea is to consume not only the node-visit resources for the nodes in the partial path itself, but also the node-visit resources of nodes that are not in the path, but cannot be reached by any feasible extension of the current partial path. Because we use customer-visit resources (rather than node-visit resources), we consider the unreachability of customers and say that customer $c$ is unreachable if every node $i \in N_c$ is unreachable. Thus, the following resource consumption rule is adopted in our algorithm when consuming customer-visit resources. The resource corresponding to customer $c$ in a partial path is consumed either when a node $i \in N_c$ is in the path, or else when none of the nodes from $N_c$ can be contained in any feasible extension of the current partial path.

As the label associated with a partial path is a vector representing the state of the path, i.e., its cost and resource consumptions, the size of the state-space increases with the number of resources. In order to limit the size of the state-space and to accelerate the label-setting algorithm, Boland et al. (2006) suggest to start the solution procedure with a state-space relaxation in which node-visit resources are initially not present. When the label setting procedure ends, the multiplicity of each node (number of times each node is visited) in the optimal path is computed and node-visit resources are introduced for some or all of the nodes with multiplicity greater than one. The label setting procedure is iteratively executed, each time starting with the original resources and the set of all node-resources introduced during the previous iterations, until the optimal path returned at the end of an iteration is elementary. The advantage of this state-space augmenting approach is that the number of node-visit resources introduced to obtain a least-cost elementary path is usually much smaller than the number of nodes.

To be able to provide details of the state-space augmenting approach of Boland et al. (2006), we introduce some additional notation. Let $S$ be the set of critical customers, i.e., the customers for which a customer-visit resource is maintained. We denote the multiplicity of customer $c$ on path $p$, i.e., the number of times a location $i \in N_c$ is visited on path $p$, by $M_c(p)$. We start with $S = \emptyset$ and update $S$ at the end of each label setting iteration that did not return an elementary optimal path by adding the customer with the highest multiplicity. In case of ties, we add the customer with the smallest index. Now suppose that the state corresponding to a given path $p$ is described by the label vector $\mathbf{U} = (U_0, \ldots, U_{|S|+2})$. We assume that $U_0$, $U_1$, and $U_2$ specify the cost of $p$, and the consumption of the time and capacity resource along $p$, respectively. The remaining elements of $\mathbf{U}$ indicate the consumption of customer-visit resources in the order that they were added to $S$. For example, if $c_k$ is the $k$th customer added to $S$, then $U_{k+2}$ shows whether customer $c_k$ is included in the path $p$ or not. To keep track of the resource consumption, we define $h^r_{ij}$ to be the amount of resource $r$ consumed when arc $(i, j)$ is used ($h^1_{ij} = t_{ij}$ and $h^2_{ij} = d_j$). When extending a label $\mathbf{U^i}$ through arc $(i, j)$ to create a new label $\mathbf{U^j}$, we set $U^j_1 = max\{U^i_1 + t_{ij}, e_j\}$, because if the delivery vehicle arrives at location $j$ before $e_j$, then it has to wait for customer $c(j)$, and $U^j_2 = U^i_2 + d_j$. We define $R^r_i$ to be the limit of resource $r$ at node $i$. The need to have a node index in our resource limit definition is the presence of time windows, since the closing time of a window is different for each node and we have to respect these limits while constructing resource-feasible paths. Even though the capacity resource limit is independent of the node and is equal to $Q$, we keep the node index for generality.

Algorithm 1 represents a straightforward implementation of the state-space augmenting approach that solves the ESPPTWCC optimally. Details of the label setting subroutine can be found in Appendix A. Despite being more efficient than using off-the-shelf software to solve the integer programming formulation of the pricing problem, employing a straightforward implementation of the state-space augmenting algorithm to solve the pricing problem may still be (too)

**Algorithm 1:** State-space augmenting algorithm.

```
Set $S = \emptyset$
do
    $\mathcal{P}^*$ = labelSetting(S)
    Find $p^* \in \mathcal{P}^*$ with shortest length
    Find the customer $c$ having the highest multiplicity on path $p^*$
    if $M_c(p^*) > 1$ then
        Set $S \leftarrow S \cup \{c\}$
while $p^*$ is not elementary
Return $p^*$
```

time-consuming. However, to solve the master problem there is no need to identify a most negative reduced cost column at every pricing iteration; identifying any negative reduced cost column suffices. Finding a most negative reduced cost column is typically needed only towards the end of the column generation process, because few, if any, negative reduced cost columns exist at that time. Consequently, we invoke Algorithm 1 in our implementation only if no negative reduced cost columns can be detected heuristically, and even in that case we terminate the algorithm as soon as a pre-specified number of negative reduced cost columns is constructed by the algorithm.

## 3. A branch-and-price algorithm

We develop a branch-and-price algorithm to solve the VRPRDL, i.e., a branch-and-bound algorithm in which at each node of the search tree the LP relaxation is solved using column generation. The most time consuming component of a branch-and-price algorithm is typically the solution of the pricing problem. Therefore, the efficient detection of negative reduced cost columns is critical to the performance of any branch-and-price algorithm. In the following, we present the techniques we employ to speed up the solution of the pricing problem, we describe the adopted branching scheme, and we discuss how we deal with the tailing-off effect.

### *3.1. Heuristic pricing*

It is not necessary to return a most-negative reduced cost column in each pricing iteration, it suffices to return (at least one) negative reduced cost column, if one exists. Even though the change in the value of the solution to the restricted master problem, when adding any negative reduced cost column rather than a most-negative reduced cost column, may be smaller, and more pricing iterations may have to be performed to reach an optimal solution, the reduction in solution time in each pricing step typically reduces the overall computation time.

A simple application of the above idea is based on the observation that usually many elementary negative reduced cost paths are found during the first few label setting iterations. Therefore, in our implementation of the state-space augmentation algorithm, we collect elementary negative reduced cost paths found in each iteration and will sometimes terminate the algorithm prematurely, i.e., before a most-negative reduced cost column has been found, and return all elementary negative reduced cost columns collected.

Another observation leads to a second useful idea: as the dual values are updated after each pricing iteration, some elementary paths with a non-negative reduced cost in one pricing iteration may have a negative reduced cost in a subsequent iteration. Therefore, at the end of a pricing iteration, non-dominated elementary paths with non-negative reduced costs found in the last label setting iteration are kept in a column pool. At the start of each pricing iteration, the columns in the pool are evaluated to see if they (now) have a negative reduced cost. To ensure that it does not become too costly to explore the column pool, we limit its size, i.e., we keep a maximum number of columns. At the end of a pricing iteration, if we add elementary non-negative reduced cost columns to the pool, we check its size, and, if the maximum pool size is exceeded, the oldest columns are removed until the desired pool size is reached.

Another strategy to detect negative reduced cost columns quickly is to make use of heuristics before invoking the exact pricing algorithm. To this end, we implement a truncated search version of the state-space augmenting approach. Instead of maintaining all non-dominated labels and their associated partial paths, we store only a small number of such labels at each node to speed up the search procedure. More specifically, we keep only a pre-specified number of efficient labels per node, and each time a new label is added to the list of efficient labels of a node, we discard the one with the largest cost if the number of labels exceeds the limit. In this way, fewer labels are treated at each label setting iteration which can facilitate faster detection of negative cost elementary paths. Again, excluding a part of the solution space may come at the expense of performing more iterations, i.e., there is a trade-off between the number of efficient labels maintained and the number of label setting iterations performed by the state-space augmenting approach.

Briefly, we try to identify negative reduced cost columns first by exploring the column pool, then by invoking the truncated-search version of the state-space augmentation algorithm, and finally by invoking the full-search version of the state-space augmentation algorithm when the other approaches fail. To control the time spent in pricing iterations even further, we terminate any pricing iteration as soon as a predetermined number of elementary negative reduced cost columns has been found.

Finally, we keep track of the minimum cost $\gamma$ of the elementary paths detected during the search, which gives an upper bound on the optimal value of the ESPPTWCC, and the cost $\eta$ of the optimal path obtained at the end of each label setting iteration, which gives a lower bound on the optimal value of the ESPPTWCC. Once the gap between these bounds is closed, we can conclude that the pricing problem has been solved optimally.

As mentioned earlier, the state-space augmenting approach starts without any customer-visit resources. However, the customers added to the set $S$ of critical customers during a pricing iteration are likely to be visited more than once in subsequent pricing iterations, and clearing $S$ at the end of each pricing iteration may therefore result in an increase in the number of label setting iterations. Consequently, rather than initializing the algorithm with $S = \emptyset$ each time, we retain the set of critical customers throughout the column generation process at a node of the search tree. We set $S = \emptyset$ only at the start of processing a node in the search tree.

## *3.2. Bidirectional search*

A common technique used to speed up the solution of the pricing problem is bidirectional search, in which paths, consuming around half of the resources available, are constructed from both the source and the sink and then merged to obtain complete paths. Even though our implementation of bidirectional search provided efficiency gains when solving the pricing problem exactly, the overall performance of the branch-and-price algorithm deteriorated. We speculate that the reason is that we infrequently solve the pricing problem exactly and that the columns returned by the truncated search, which can be substantially different when deploying bidirectional search, are not as effective.

## *3.3. Branching*

When the optimal solution to the master problem is fractional, we have to perform branching to find integer solutions. We branch on the arc variables, $x_{ij}$, in the original formulation. If an arc variable has a fractional value, we force the arc to be a part of the solution in one branch while prohibiting routes containing that arc in the other branch. Branching on variables in the original formulation has become a standard feature of many branch-and-price algorithms (Feillet, 2010).

Each variable of the master problem corresponds to the number of times a particular route is used in the optimal delivery plan. Therefore, we compute the value of an arc $(i, j)$ in an optimal solution to the master problem by summing the optimal values of the routes that include the arc $(i, j)$. Once a branching decision is enforced, we have to ensure that the columns in the restricted master problem and the ones that will be returned by the pricing subroutine are compatible with this decision. To guarantee compatibility at a child node, first we filter the columns coming from the restricted master problem associated with its parent node to exclude those that are in conflict with the branching decision and then update the pricing problem by forbidding the proper arc(s).

Ensuring compatibility is rather straightforward when the branching decision requires prohibiting a certain arc $(i, j)$, i.e., the columns containing this arc are removed from the restricted master problem and the arc $(i, j)$ is ignored while solving the pricing problem. On the other hand, to ensure that $(i, j)$ is included in the solution, we omit all columns (routes) containing any arc from the set $(\delta^+(i) \setminus (i, j)) \cup (\delta^-(j) \setminus (i, j)) \cup (\delta^-(l) : l \in (N_{c(i)} \setminus \{i\}) \cup (N_{c(j)} \setminus \{j\}))$ from the restricted master problem, and discard all the arcs in this set while solving the pricing problem (i.e., we do not extend partial paths through these arcs). Note that as a result of column filtering, the master problem may become infeasible. Therefore, we always maintain a set of artificial columns with a high cost that guarantee the feasibility of the master problem. More specifically, we store the columns used to initialize the restricted master problem, and introduce these columns as “artificial” columns prior to starting column generation at a node of the search tree. By assigning the sum of the costs of these columns to each one of them individually, we ensure that these columns will not be in the optimal basis when column generation completes.

We have considered three strategies for selecting the arc to branch on. The first one is conventional branching, denoted by *CB*, where we select an arc with value closest to 0.5. The second one is to choose an arc that appears in the greatest number of routes in the solution to the master problem, denoted by *MF*. The last one is to choose an arc with a fractional value that occurs the earliest in time, denoted by *ED*. In particular, for an arc $(i, j)$, we determine the time at which a vehicle using this arc in its route departs from node $i$. There may be multiple routes containing $(i, j)$, in this case, we take the minimum of the departure times from node $i$ over all such routes. In all of the arc selection strategies, we have the additional restrictions that the value of the chosen arc should be less than one and both of its endpoints should correspond to customer locations.

In *CB*, if there is more than one arc whose value is closest to 0.5, then we pick the first one encountered. In *MF*, we break ties by choosing either the arc encountered first during the search (*MF*) or the arc whose value is closest to 0.5 (*MFC*). Similarly, for *ED*, we either select the arc encountered first during the search (*ED*) or the arc whose value is closest to 0.5 (*EDC*).

## *3.4. Initial set of columns and feasible solutions*

The column generation method starts by solving a restriction of the master problem. Therefore, we need to provide an initial set of columns that guarantees the feasibility of the master problem, i.e., a feasible starting solution. The quality of this solution can have a significant impact on the performance of the branch-and-price algorithm, especially for large problem instances. In our experiments, we use the feasible solution found by the heuristic of Reyes et al. (2016). The heuristic constructs a feasible solution within a few seconds for small and medium size instances and within a few minutes for large instances. The time spent by the heuristic on improving the initial feasible solution depends on the quality of that solution, but is small compared to the time spent by our branch-and-price algorithm.

The value of any feasible solution provides an upper bound on the optimal objective function value and can thus be used to fathom nodes by bound. Of course, the higher the quality of the feasible solution, the more effective the fathoming becomes. Therefore, we also embed the following commonly used heuristic for producing a, hopefully high-quality, feasible solution. After solving the master problem at the root node, we solve the restricted master problem, i.e., including initial as well as generated columns, as an integer program (simply handing it to an off-the-shelf software package). This heuristic has proven to be quite successful in a number of different applications, and initial experimentation in our setting revealed that the resulting integer programs could be solved quickly, and, thus, did not impede the overall solution process.

## 3.5. Handling the tailing-off effect

Many pricing iterations may have to be performed with little or no improvement in the objective function value towards the end of the column generation process at a node in the search tree. This situation is quite common in branch-and-price algorithms and is known as the tailing-off effect. Below, we discuss two approaches for dealing with the tailing-off effect.

### 3.5.1. Using lagrangian dual bounds for early pruning

The occurrence of tailing-off at a node is especially unfortunate if the node is fathomed by bound once the master problem has been solved, because in that case much time may have been "wasted" by "unnecessarily" solving pricing problems. This situation can, possibly, be prevented if an alternative lower bound can be computed that can be used to fathom the node before the column generation process at the node completes. Such a bound can be computed using concepts from Lagrangian relaxation. Note, however, that this may also mean that fewer columns are added to the column pool, which can have a negative impact on solution efficiency.

Dualizing the covering constraints in the master problem, we obtain the Lagrangian relaxation

$$\min \sum_{r\in R} w_r z_r + \sum_{c\in C} \lambda_c \Big(1 - \sum_{r\in R}\sum_{i\in N_c} a_{ir} z_r\Big)$$

$$\text{s.t.} \sum_{r\in R} z_r \le m, \tag{23}$$

$$z_r \ge 0, \quad r\in R, \tag{24}$$

which provides a lower bound on the optimal value of the master problem for any nonnegative vector of multipliers $(\lambda_1, \dots, \lambda_{|C|})$. Rearranging gives

$$\sum_{c\in C} \lambda_c + \left\{ \begin{array}{l} \min \sum_{r\in R} (w_r - \sum_{i\in N\setminus\{0\}} a_{ir}\lambda_{c(i)}) z_r \\ \text{s.t. } (23), (24) \end{array} \right\},$$

which, when taking the values of the optimal dual solution to RMP, gives

$$\sum_{c\in C} \lambda_c + \left\{ \begin{array}{l} \min \sum_{r\in R} (\bar{w}_r + \lambda_0) z_r \\ \text{s.t. } (23), (24) \end{array} \right\},$$

where $\bar{w}_r$ is the reduced cost of route $r$ and $\lambda_0$ is the value of the dual variable corresponding to (23). This quantity can be bounded from below by

$$\sum_{c\in C} \lambda_c + m(\bar{w}_{min} + \lambda_0) = \sum_{c\in C} \lambda_c + m\lambda_0 + m\bar{w}_{min} = \theta^*_{RMP} + m\bar{w}_{min},$$

where $\bar{w}_{min}$ is the minimum reduced cost and $\theta^*_{RMP}$ is the optimal value of RMP.

This lower bound can be computed easily at the end of a column generation iteration if the optimal value of the pricing problem is known. (Note that $\theta^*_{RMP}$ always provides an upper bound on the optimal value of the master problem, and that when $\bar{w}_{min} = 0$ it provides a lower bound as well, in which case the master problem is solved optimally.) Observe that it is possible to obtain a lower bound on the optimal value of the master problem as soon as a complete label setting iteration is performed, because the state-space augmenting algorithm yields a lower bound on the optimal value of the pricing problem at the end of each label setting iteration. Therefore, whenever the full-search version of the state-space augmentation algorithm is used in a pricing iteration, we compute a lower bound for the master problem at the end of each label setting iteration and see if it can be used to prune the node.

### 3.5.2. Early branching

Another strategy to deal with the tailing-off effect is to prematurely terminate the column generation procedure and perform branching early. To accomplish this, one can stop generating columns and apply branching, for example, when the improvement in the objective function value is less than $\epsilon$ in the last $\Delta$ iterations. Note that the values of $\epsilon$ and $\Delta$ need to be set carefully, because branching too aggressively may grow the search tree significantly and may be counterproductive.

## 3.6. Implementation details

There are many algorithmic choices and parameter settings that impact the computational performance of the branch-and-price algorithm. In the following, we first summarize our implementation of a straightforward branch-and-price algorithm and then describe the algorithmic choices and parameter settings for the enhanced version that incorporates the ideas discussed above.

### 3.6.1. *Straightforward branch-and-price*

In order to evaluate the improvements achieved by our enhanced branch-and-price algorithm, we consider a straightforward approach, in which out-and-back routes from the depot to the first location of every customer are used to define the initial set of columns (these routes always correspond to a feasible solution because $m = |C|$ in VRPRDL instances), the state-space augmenting algorithm is initialized with $S = \emptyset$ at every pricing iteration, only a single column with the most negative reduced cost is added to the restricted master problem (i.e., the pricing problem is solved to optimality), and the conventional branching strategy ($CB$) is employed; there is no early pruning, no early branching, and the restricted master problem at the end of the column generation process at the root node is not used to seek a, possibly improved, feasible solution.

### 3.6.2. *Enhanced branch-and-price*

In our enhanced branch-and-price algorithm, the following parameter settings control the solution of the pricing problem: the column generation process terminates as soon as $\beta$ elementary negative reduced cost columns are identified. The truncated-search version of the state-space augmentation algorithm restricts the number of non-dominated labels at each node to $\alpha$. In the truncated-search version of the state-space augmentation algorithm, we also monitor the cost $\gamma$ of the shortest elementary path detected so far, which yields an upper bound on the optimal ESPPTWCC value, and the cost $\eta$ of the shortest path obtained at the end of each truncated label setting iteration, which provides a lower bound on the cost of the shortest elementary path that can be obtained by truncated-search, and terminate the truncated search when these bounds at the end of a label setting iteration are equal. The size of the column pool is 1000, i.e., at most 1000 columns are stored at any one time. Only when neither the exploration of the column pool nor the truncated search produces any elementary negative reduced cost columns do we invoke exact pricing. The set $S$ of critical customers used in the state-space augmentation algorithm is retained throughout the column generation process at each node in the search tree, and it is cleared right before column generation starts at another node.

In some of our computational experiments and for some values of $\alpha$ and $\beta$, we also consider forcing the truncated search to stop upon completing a single label setting iteration even when the number of negative reduced cost columns detected is less than $\beta$ and the gap between $\gamma$ and $\eta$ is not closed at the end of this label setting iteration. We use $T$ to denote that we adopt this termination criterion, i.e., stop at the end of the first label setting iteration, instead of iterating until $\gamma$ and $\eta$ become equal, which is denoted by $F$.

Furthermore, the solution obtained by the heuristic of Reyes et al. (2016) is used to initialize the algorithm, the restricted master problem is solved as an integer program upon completing column generation at the root node in the hope of finding an improved feasible solution, early pruning is active, and the nodes in the branch-and-price tree are evaluated in a depth-first order (best-bound and depth-first search produce similar results on small and medium size instances, the latter performs better for large size instances). Early branching is not used as computational experiments revealed that the benefits are negligible.

Finally, we want to point out that our branch-and-price algorithm can also solve instances where arc costs or travel times do not satisfy the triangle inequality. In the former case, one can simply use the partitioning constraints given by (10) when defining the master problem. In the latter case, one has to compute the shortest travel time between each pair of locations prior to executing the branch-and-price algorithm, because this information is used in certain steps of the state-space augmenting method when solving the pricing problems and in preprocessing the problem graph to reduce its size.

## 4. Incorporating a home delivery option

Trunk delivery was introduced in the hope that it would create cost-saving opportunities compared to making home deliveries. It is easy to construct examples where this is indeed the case. However, it is also easy to construct examples where this is not the case and the cost actually increases. Thus, companies that are considering trunk delivery will likely deploy a hybrid model in which deliveries can either be made at the home location of the customer (during the entire planning horizon) or to trunk of the customer's car at one of the locations in the car's itinerary, if this creates cost savings.

Fortunately, our branch-and-price algorithm can be used for solving this hybrid model by simply making slight changes in the instance data. In particular, we replace the time windows associated with the home location of each customer, which, in the instances of VRPRDL, are the first and last location of the car's itinerary by a single location with time window [0, $T$], where $T$ is the length of the planning horizon. Note that this means that the time windows of the locations visited by a customer are no longer non-overlapping. However, the branch-and-price algorithm has no components that exploit or rely on this non-overlapping property and thus it can be applied without any modification. Of course the performance may (and, as we will see, will) deteriorate.

## 5. Computational study

We conduct a computational study to (1) evaluate the performance of our branch-and-price algorithm, and to (2) assess the benefits of employing trunk delivery services.

### 5.1. Instances and preprocessing

In our computational experiments, we use two sets of VRPRDL instances.

The first set consists of slightly modified versions of the 40 random instances introduced in Reyes et al. (2016), in which the travel times have been adjusted to ensure that they satisfy the triangle inequality and, when necessary, the time windows at locations have been adjusted accordingly. These instances range in size from 15 to 120 customers, each with up to 5 roaming delivery locations. This set of instances is well-suited to assess the impact of the number of customers and roaming delivery locations on the performance of the branch-and-price algorithm.

The second set contains two variations of 10 medium-size instances generated in order to investigate the impact of the distance of the roaming delivery locations to the depot on the performance of the branch-and-price algorithm. The distance of the roaming delivery locations to the depot may impact the performance of the branch-and-price algorithm for several reasons. When the roaming delivery locations are closer to the depot, this typically implies that the customers spend more time traveling, which in turn implies that there is less time available for making deliveries, i.e., the time windows at the roaming delivery locations are narrower. Narrower time windows may lead to more effective preprocessing, i.e., elimination of more variables. On the other hand, when the roaming delivery locations are closer to the depot, this typically implies that the roaming delivery locations of different customers are closer together, which in turn implies that there are more feasible solutions. More feasible solutions not only implies possibly lower costs, but may also lead to less effective preprocessing, i.e., elimination of fewer variables. The first variation of each instance is created using the instance generator described in Reyes et al. (2016), which chooses the roaming delivery locations for a customer uniform randomly in a circle with radius $sT/2\rho$ and center at the customer's home location, where $s$ is the (constant) vehicle speed and $\rho$ is the maximum number of locations per customer. In the second variation of each instance, the home location and the fraction of time spent at every roaming delivery location are the same, but the roaming delivery locations themselves tend to be closer to the depot (which implies that the associated time window widths are usually different). More precisely, in the second variation, the roaming locations are chosen uniform randomly in a circle with radius $sT/2\rho$ and center on the line connecting the home location and the depot, either at the midpoint of the line or at distance $sT/2\rho$ of the home location, whichever is closer to the home location. (As with the first set of instances, the travel times and the time windows are adjusted to ensure that the travel times satisfy the triangle inequality.)

In all the instances, the planning horizon is 12 hours, the vehicle capacity is 750, and the geographic profile of every customer consists of at least one and at most six locations (the first and last location always being the home location of the customer - in case there is only one location, it signals that the customer is at home all day).

Although there is no restriction on the fleet size in the original instances, we assume that $m$, the number of vehicles available for making the deliveries, is equal to the number of routes in the solution provided by the heuristic of Reyes et al. (2016) plus one when solving VRPRDL and VRPHRDL instances with our enhanced algorithm. We impose this restriction, because smaller values of $m$ results in stronger lower bounds on the optimal value of the master problem, and preliminary experiments using small to medium sized instances revealed that the optimal costs remain unchanged even if we set $m = |C|$. Characteristics of the first and second sets of instances are provided in Tables 1 and 2, respectively. For each instance, we present the number of customers, the average number of locations per customer, and, considering all customers, the minimum, average, and maximum distance from home location to the depot, the minimum, average, and maximum fraction of time available for making a delivery, and the average width of time windows. Note that for the second set of instances, the number of customers is not specified in the table as all instances in this set contain 40 customers. Also in Table 2 is an additional column presenting the average distance from the roaming delivery locations to the depot. We observe that there are noticeable differences between instances, with some instances having, on average, only 2.47 locations per customer whereas others have, on average, 4.33 locations per customer, and some instances having, on average, only 49% of the planning horizon available to make deliveries whereas others have, on average, 90% of the planning horizon available.

We compute the Euclidean distance between each pair of locations based on their coordinates and then round this distance to the nearest integer. In order to ensure that arc costs satisfy the triangle inequality, we assign the shortest distance between each pair of nodes to the cost of the arc connecting these two nodes. Furthermore, we reduce the size of the network by eliminating nodes and arcs that cannot be a part of any feasible solution. Our preprocessing steps are as follows:

1. Eliminate node $i$ and all arcs incident to this node if $t_{0i} > l_i$ or $e_i + t_{i0} > T$; and
2. Eliminate arc $(i, j)$ if $\max\{t_{0i}, e_i\} + t_{ij} > l_j$ or $\max\{t_{0i}, e_i\} + t_{ij} + t_{j0} > T$

Essentially, we eliminate a node $i$ if an out-and-back route from the depot to the node, i.e., route $0 - i - 0$, leads to a time window violation. Similarly, we eliminate an arc $(i, j)$ if route $0 - i - j - 0$ is not time-feasible. This simple preprocessing is very effective, it reduces the number of nodes and arcs substantially: the minimum, average, and maximum reduction in the number of arcs across the VRPRDL instances are 72.53%, 87.49% and 93.1%, respectively. The reduction is smaller for the VRPHRDL instances as the home locations now have wide time windows, and thus fewer nodes and arcs are eliminated.

The algorithm is implemented in Java using the branch-and-price framework of Java OR library (*jORLib*) and Java graph theory library (*JGraphT*), and CPLEX 12.6.3 is employed for solving the restricted master problems through Concert Technology. All experiments are performed on a 64-bit machine with Intel Xeon E5-2650 v3 processor at 2.30 GHz. The time limit is set to two hours for instances with up to 60 customers and to six hours for instances with 120 customers. In the

**Table 1**
Characteristics of the instances in the first set.

| Instance | Number of customers | Avg number of customer locations | Avg width of time windows | Distance to depot (from home location) | | | Fraction of time available for delivery | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Min | Avg | Max | Min | Avg | Max |
| 1 | 15 | 4.07 | 102.93 | 3 | 67.07 | 166 | 0.05 | 0.58 | 1 |
| 2 | 15 | 3.73 | 125.25 | 28 | 83.73 | 148 | 0.31 | 0.65 | 1 |
| 3 | 15 | 3.40 | 139.61 | 7 | 72.33 | 167 | 0.08 | 0.66 | 1 |
| 4 | 15 | 3.27 | 130.96 | 3 | 70.80 | 159 | 0.08 | 0.59 | 1 |
| 5 | 15 | 3.40 | 142.96 | 13 | 101.07 | 174 | 0.14 | 0.68 | 1 |
| 6 | 20 | 3.25 | 148.23 | 2 | 76.45 | 152 | 0.16 | 0.67 | 1 |
| 7 | 20 | 3.35 | 138.49 | 15 | 76.25 | 174 | 0.08 | 0.64 | 1 |
| 8 | 20 | 3.95 | 102.09 | 6 | 87.10 | 161 | 0.04 | 0.56 | 1 |
| 9 | 20 | 3.75 | 94.65 | 3 | 73.35 | 171 | 0.01 | 0.49 | 1 |
| 10 | 20 | 3.10 | 154.32 | 0 | 80.55 | 169 | 0.14 | 0.66 | 1 |
| 11 | 30 | 3.40 | 130.90 | 2 | 85.43 | 176 | 0.03 | 0.62 | 1 |
| 12 | 30 | 3.73 | 103.13 | 20 | 78.03 | 174 | 0.01 | 0.53 | 1 |
| 13 | 30 | 3.90 | 99.88 | 3 | 71.27 | 171 | 0.04 | 0.54 | 1 |
| 14 | 30 | 3.53 | 124.70 | 5 | 73.07 | 171 | 0.12 | 0.61 | 1 |
| 15 | 30 | 4.10 | 94.59 | 3 | 81.70 | 176 | 0.04 | 0.54 | 1 |
| 16 | 30 | 3.93 | 107.95 | 2 | 72.77 | 160 | 0.03 | 0.59 | 1 |
| 17 | 30 | 4.30 | 83.51 | 2 | 85.23 | 165 | 0.08 | 0.50 | 1 |
| 18 | 30 | 3.50 | 120.30 | 17 | 89.70 | 154 | 0.01 | 0.58 | 1 |
| 19 | 30 | 3.23 | 139.63 | 12 | 82.47 | 179 | 0.08 | 0.63 | 1 |
| 20 | 30 | 2.47 | 223.78 | 12 | 75.10 | 172 | 0.05 | 0.77 | 1 |
| 21 | 60 | 3.73 | 115.26 | 4 | 78.88 | 173 | 0.04 | 0.60 | 1 |
| 22 | 60 | 3.53 | 117.18 | 1 | 80.07 | 170 | 0.01 | 0.58 | 1 |
| 23 | 60 | 3.90 | 100.76 | 2 | 89.03 | 179 | 0.04 | 0.55 | 1 |
| 24 | 60 | 3.80 | 118.83 | 8 | 93.12 | 175 | 0.03 | 0.63 | 1 |
| 25 | 60 | 3.88 | 108.87 | 3 | 79.60 | 165 | 0.06 | 0.59 | 1 |
| 26 | 60 | 3.73 | 108.36 | 1 | 89.77 | 178 | 0.01 | 0.56 | 1 |
| 27 | 60 | 3.45 | 131.47 | 8 | 90.27 | 176 | 0.03 | 0.63 | 1 |
| 28 | 60 | 3.63 | 111.70 | 1 | 75.07 | 180 | 0.04 | 0.56 | 1 |
| 29 | 60 | 3.75 | 112.79 | 2 | 91.95 | 179 | 0.02 | 0.59 | 1 |
| 30 | 60 | 3.95 | 108.27 | 4 | 93.97 | 177 | 0.01 | 0.59 | 1 |
| 31 | 120 | 3.83 | 112.43 | 0 | 76.23 | 178 | 0.01 | 0.60 | 1 |
| 32 | 120 | 3.67 | 116.42 | 0 | 83.98 | 174 | 0.04 | 0.59 | 1 |
| 33 | 120 | 3.92 | 104.71 | 0 | 69.69 | 179 | 0.02 | 0.57 | 1 |
| 34 | 120 | 3.78 | 112.51 | 2 | 79.45 | 176 | 0.01 | 0.59 | 1 |
| 35 | 120 | 3.75 | 107.41 | 1 | 77.35 | 174 | 0.07 | 0.56 | 1 |
| 36 | 120 | 3.51 | 127.94 | 0 | 89.57 | 178 | 0.05 | 0.62 | 1 |
| 37 | 120 | 3.88 | 102.49 | 3 | 83.14 | 177 | 0.02 | 0.55 | 1 |
| 38 | 120 | 3.51 | 126.62 | 0 | 86.11 | 177 | 0.03 | 0.62 | 1 |
| 39 | 120 | 3.56 | 119.54 | 0 | 89.30 | 178 | 0.03 | 0.59 | 1 |
| 40 | 120 | 3.84 | 103.44 | 0 | 79.73 | 171 | 0.07 | 0.55 | 1 |

computational results tables, we will indicate that a time limit was reached with TL. Note that the solution times reported in the tables do not include the time spent by the heuristic of Reyes et al. (2016) to find the initial solution.

## 5.2. Evaluating the performance of the branch-and-price algorithm

To be able to evaluate the benefits of the various techniques introduced in the branch-and-price algorithm, we start by solving the first set of instances with the straightforward implementation described in Section 3.6.1. The results can be found in Table 3. For each instance, we report the name of the instance, the cost of the initial solution, the cost of the best solution, the integrality gap, the solution time (in seconds), the total number of pricing iterations performed during the execution of the algorithm, and the number of nodes evaluated during the search, respectively. When the algorithm terminates within the time limit, the best solution is optimal, otherwise it may or may not be optimal. The integrality gap is computed as the ratio $(\theta^*_{IP} - \theta^*_{LP})/\theta^*_{LP}$, where $\theta^*_{LP}$ is the optimal value of the master problem at the root node of the search tree and $\theta^*_{IP}$ is the cost of the best solution found during the execution of the algorithm. When the master problem at the root node cannot be solved within the time limit, it is not possible to compute an integrality gap, which is indicated with a dash. If an instance is solved to optimality (i.e., the algorithm terminates within the time limit), the integrality gap provides some additional insight into the relative difficulty of the instance. If, on the other hand, an instance cannot be solved within the time limit, then the integrality gap provides a quality guarantee of the best solution found.

We observe that the straightforward implementation is able to solve all instances with 15, 20, and 30 customers, but fails to find an optimal solution or prove its optimality for three of the instances with 60 customers. In fact, the first pricing

**Table 2**
Characteristics of the instances in the second set.

| Instance | Avg number of customer locations | Avg width of time windows | Avg distance from roaming locations to depot | Distance to depot (from home location) | | | Fraction of time available for delivery | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Min | Avg | Max | Min | Avg | Max |
| 41_v1 | 3.98 | 159.60 | 101.71 | 16 | 24.57 | 176 | 0.75 | 0.88 | 1 |
| 41_v2 | | 149.79 | 80.72 | 16 | 24.50 | 175 | 0.66 | 0.83 | 1 |
| 42_v1 | 3.93 | 161.66 | 91.10 | 19 | 23.60 | 167 | 0.72 | 0.88 | 1 |
| 42_v2 | | 151.25 | 68.85 | 18 | 23.59 | 167 | 0.70 | 0.82 | 1 |
| 43_v1 | 4.05 | 158.07 | 88.51 | 4 | 21.43 | 171 | 0.76 | 0.89 | 1 |
| 43_v2 | | 144.85 | 69.19 | 4 | 21.44 | 171 | 0.57 | 0.81 | 1 |
| 44_v1 | 3.53 | 181.38 | 93.58 | 2 | 24.65 | 174 | 0.68 | 0.89 | 1 |
| 44_v2 | | 173.61 | 72.11 | 2 | 24.61 | 174 | 0.62 | 0.85 | 1 |
| 45_v1 | 3.60 | 180.12 | 67.59 | 4 | 22.67 | 177 | 0.79 | 0.90 | 1 |
| 45_v2 | | 166.60 | 48.31 | 4 | 22.69 | 177 | 0.59 | 0.83 | 1 |
| 46_v1 | 4.33 | 143.42 | 98.54 | 1 | 20.57 | 176 | 0.77 | 0.86 | 1 |
| 46_v2 | | 131.63 | 79.35 | 1 | 20.52 | 176 | 0.65 | 0.79 | 1 |
| 47_v1 | 4.18 | 145.56 | 96.31 | 11 | 22.85 | 174 | 0.70 | 0.84 | 1 |
| 47_v2 | | 139.58 | 77.20 | 11 | 22.87 | 173 | 0.67 | 0.81 | 1 |
| 48_v1 | 3.83 | 167.82 | 97.96 | 9 | 25.11 | 178 | 0.72 | 0.89 | 1 |
| 48_v2 | | 153.76 | 76.83 | 9 | 25.08 | 178 | 0.53 | 0.82 | 1 |
| 49_v1 | 3.80 | 165.56 | 100.17 | 12 | 26.97 | 175 | 0.74 | 0.87 | 1 |
| 49_v2 | | 148.48 | 72.55 | 12 | 26.96 | 175 | 0.51 | 0.78 | 1 |
| 50_v1 | 4.03 | 155.06 | 78.11 | 2 | 20.42 | 176 | 0.75 | 0.87 | 1 |
| 50_v2 | | 146.07 | 60.85 | 2 | 20.48 | 177 | 0.68 | 0.82 | 1 |

**Table 3**
Results with the straightforward BAP.

| Instance | Initial solution | Best solution | Integrality gap (%) | Solution time (s) | Iterations | Nodes |
|---|---|---|---|---|---|---|
| 1 | 2074 | 901 | 0 | 0.58 | 39 | 1 |
| 2 | 2316 | 1286 | 0 | 0.27 | 29 | 1 |
| 3 | 2234 | 991 | 0 | 0.43 | 30 | 1 |
| 4 | 1982 | 1062 | 0 | 0.27 | 30 | 1 |
| 5 | 3322 | 1832 | 0 | 0.04 | 26 | 1 |
| 6 | 3328 | 1294 | 1.73 | 2.30 | 245 | 31 |
| 7 | 3204 | 1155 | 1.29 | 30.47 | 773 | 70 |
| 8 | 3170 | 1455 | 0 | 0.18 | 39 | 1 |
| 9 | 2838 | 1260 | 1.86 | 3.32 | 307 | 32 |
| 10 | 3270 | 1684 | 0 | 0.55 | 33 | 1 |
| 11 | 4932 | 1922 | 0.31 | 14.28 | 283 | 27 |
| 12 | 4610 | 2324 | 2.52 | 120.38 | 16914 | 2707 |
| 13 | 4868 | 1747 | 0 | 23.33 | 101 | 1 |
| 14 | 4084 | 1273 | 0 | 30.83 | 100 | 1 |
| 15 | 4656 | 1694 | 0 | 22.78 | 97 | 1 |
| 16 | 4770 | 1938 | 0 | 57.13 | 82 | 1 |
| 17 | 4502 | 1965 | 0.10 | 5.75 | 117 | 3 |
| 18 | 5392 | 1827 | 0 | 2.10 | 83 | 1 |
| 19 | 5286 | 2083 | 2.46 | 93.55 | 4140 | 413 |
| 20 | 4236 | 1822 | 0 | 78.70 | 103 | 1 |
| 21 | 10374 | 3761 | 0 | 606.60 | 216 | 1 |
| 22 | 9316 | 2828 | 0 | 272.43 | 357 | 3 |
| 23 | 10326 | 4440 | 0.01 | 238.27 | 208 | 3 |
| 24 | 10536 | 3378 | 0 | 382.37 | 250 | 1 |
| 25 | 9784 | 9784 | – | TL | 1 | 0 |
| 26 | 10822 | 4536 | 0 | 78.61 | 158 | 1 |
| 27 | 10634 | 2865 | 0 | 296.23 | 242 | 1 |
| 28 | 9450 | 9450 | – | TL | 1 | 0 |
| 29 | 10988 | 3964 | 0.73 | TL | 82536 | 11593 |
| 30 | 11840 | 4107 | 0 | 45.22 | 177 | 1 |
| 31 | 18142 | 18142 | – | TL | 1 | 0 |
| 32 | 19514 | 19514 | – | TL | 1 | 0 |
| 33 | 18008 | 18008 | – | TL | 4 | 0 |
| 34 | 19880 | 19880 | – | TL | 2 | 0 |
| 35 | 19196 | 19196 | – | TL | 1 | 0 |
| 36 | 21772 | 21772 | – | TL | 3 | 0 |
| 37 | 20010 | 20010 | – | TL | 2 | 0 |
| 38 | 20032 | 20032 | – | TL | 2 | 0 |
| 39 | 20136 | 20136 | – | TL | 480 | 0 |
| 40 | 20042 | 20042 | – | TL | 1 | 0 |

problem cannot be solved within two hours for Instances 25 and 28. None of the instances with 120 customers can be solved within six hours.

Next, we solve Instances 1–30 using the enhanced implementation introduced in Section 3.6.2 with different combinations of control parameters for the solution of the pricing problem and with different branching schemes. More specifically, six configurations for solving the pricing problem were evaluated: (5, 5, *F*), (5, 10, *F*), (10, 10, *F*), (15, 10, *F*), (10, 10, *T*) and (15, 10, *T*), where the first parameter specifies the number of labels kept in the truncated-search ($\alpha$), the second parameter specifies the number of negative reduced cost columns required before terminating the solution of the pricing problem early ($\beta$), and the third parameter specifies whether or not the state-space augmenting algorithm is terminated after a single label setting iteration (*T* or *F*). Furthermore, all branching schemes were evaluated: *CB, MF, MFC, ED*, and *EDC*. Thus, in total, the 30 instances were solved 30 times.

We found that 19 of the instances can be solved optimally without branching by each of the tested configurations. The solution to the master problem at the root node is always integer in 18 cases. In the other case (for Instance 21), some settings required solving the restricted master problem at the end of the column generation process at the root node as an integer program in order to produce an integer solution with cost equal to the optimal objective value of the master problem. Furthermore, we found that the branching schemes *ED* and *EDC* performed poorly compared to the other branching schemes. Therefore, in Table 4, we present the results for six configurations and three branching schemes for 12 instances. We report the solution time (in seconds), the number of pricing iterations, and the number of nodes evaluated.

To analyze the results and to choose a default branching scheme and default control parameters for the solution of the pricing problem, we will focus on the solution times. First, we observe that both the *MF* and the *MFC* branching schemes outperform the *CB* branching scheme. Second, when comparing the different pricing parameter configurations used in the experiments, we see that the pricing problem is solved to optimality more often when multiple label setting iterations are not allowed during truncated search. This is expected given the fact that the column generation procedure is initialized with $S = \emptyset$ at each branch-and-price tree node and no customers are added to *S* by truncated-search when it is terminated during or at the end of the first label setting iteration. Invoking the full-search provides a means to update *S* and to identify the negative reduced cost columns that would not otherwise be encountered during truncated-search. However, this usually increases the total number of column generation iterations as well as the number of nodes in the branch-and-price tree, and results in longer solution times. Hence, we conclude that the configurations ($\alpha$, $\beta$, *F*) lead to better performance in general. Although the choice between the different configurations of control parameters for the solution of the pricing problem is not obvious, schemes *MF*(5, 5, *F*), *MFC*(5, 5, *F*), and *MFC*(10, 10, *F*) appear to be most "robust", in the sense that they lead to the minimum total solution time across the first set of instances (with up to 60 customers).

We also examined how often a pricing iteration completes after (1) exploring the column pool, (2) applying truncated-search, and (3) applying full-search. For all branching schemes and all parameter configurations, most pricing iterations complete after truncated-search, which implies that truncated-search returns at least one elementary negative reduced cost column. Although rare, pricing iterations sometimes complete after exploring the column pool, especially when many pricing iterations have to be performed, which is not surprising since the column pool fills up gradually and the more columns, the better the chance of having negative reduced cost columns in the column pool in subsequent iterations.

We use the three most promising settings, i.e., *MF*(5, 5, *F*), *MFC*(5, 5, *F*), and *MFC*(10, 10, *F*) to solve the 120 customer instances. The results can be found in Table 5. For each instance, the first four columns correspond to the name of the instance, the cost of the initial solution, the cost of the solution obtained by solving the restricted master problem at the root node as an integer program, and the cost of the best solution found, respectively. The remaining columns provide the integrality gap, the solution time, the total number of pricing iterations, the average time per pricing iteration, and finally, the number of nodes evaluated. The best solution for each instance is highlighted in bold.

We observe that the algorithm is able to find an optimal solution to three instances with the settings *MF*(5, 5, *F*), *MFC*(5, 5, *F*) and *MFC*(10, 10, *F*). Considering the number of best solutions obtained with each of these settings, *MF*(5, 5, *F*) is superior to the others as it produces the best solution for seven instances, whereas six best solutions are found by the other two schemes. Based on the solution times for the instances that are solved to optimality, *MFC*(5, 5, *F*) results in the minimum total solution time. Hence, no setting clearly dominates the others.

We also observe that solving the restricted master problem at the root node as an integer program is quite effective. With the setting *MF*(5, 5, *F*), the solution value improves, on average, by almost 5%, and for Instance 39 the improvement exceeds 10%. The integrality gap values reveal that high-quality solutions are produced; with the setting *MF*(5, 5, *F*), the average integrality gap is only 1.31%. Finally, we observe that although the total number of pricing iterations significantly decreases for most instances with a larger value of $\beta$, the time per pricing iteration increases when the number of labels kept at a node during truncated-search is larger (compare the time per iterations for *MFC*(5, 5, *F*) and *MFC*(10, 10, *F*)).

To evaluate the benefits of the various techniques introduced in the branch-and-price algorithm to improve its performance, we next compare the straightforward implementation with the enhanced implementation with the setting *MF*(5, 5, *F*). The results can be found in Table 6. A dash in the column "Root IP solution" means that the master problem has an integer optimal solution.

We observe that the benefits of implementing the techniques described earlier are significant. Instances that can be solved to optimality are solved orders of magnitude faster, and for the instances that cannot be solved to optimality, solutions of much better quality are obtained. Specifically, for the instances that can be solved optimally by both approaches, the reduction in average solution time is 97%. Furthermore, when the straightforward implementation fails to find an optimal

**Table 4**
Results with *CB*, *MF*, and *MFC* arc selection rules.

| | (5, 5, *F*) | | | (5, 10, *F*) | | | (10, 10, *F*) | | | (15, 10, *F*) | | | (10, 10, *T*) | | | (15, 10, *T*) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Iters. | Nodes | Time | Iters. | Nodes | Time | Iters. | Nodes | Time | Iters. | Nodes | Time | Iters. | Nodes | Time | Iters. | Nodes |
| **Select the arc for branching using the *CB* rule** | | | | | | | | | | | | | | | | | | |
| 6 | 1.06 | 193 | 47 | 0.75 | 84 | 21 | 0.96 | 100 | 25 | 0.99 | 106 | 29 | 1.16 | 164 | 33 | 1.21 | 198 | 45 |
| 7 | 1.52 | 217 | 20 | 1.48 | 150 | 16 | 1.67 | 139 | 19 | 2.27 | 169 | 23 | 2.85 | 217 | 23 | 3.04 | 216 | 23 |
| 9 | 0.67 | 143 | 25 | 0.52 | 103 | 22 | 0.53 | 76 | 17 | 0.57 | 88 | 19 | 0.73 | 125 | 25 | 0.88 | 138 | 25 |
| 11 | 0.87 | 119 | 11 | 0.68 | 73 | 7 | 0.82 | 82 | 13 | 1.17 | 93 | 15 | 0.85 | 85 | 7 | 1.45 | 109 | 11 |
| 12 | 96.68 | 16833 | 4765 | 82.10 | 12567 | 3957 | 53.76 | 9881 | 3195 | 101.41 | 15435 | 4981 | 57.07 | 9877 | 2971 | 95.59 | 14331 | 4243 |
| 17 | 0.64 | 73 | 3 | 0.59 | 60 | 7 | 0.76 | 77 | 7 | 0.59 | 44 | 3 | 0.52 | 45 | 3 | 0.57 | 67 | 3 |
| 19 | 5.76 | 523 | 73 | 14.14 | 1464 | 293 | 8.58 | 586 | 105 | 7.47 | 399 | 79 | 8.51 | 557 | 89 | 11.78 | 712 | 111 |
| 23 | 1.06 | 72 | 1 | 2.11 | 54 | 1 | 2.60 | 58 | 1 | 3.98 | 73 | 3 | 5.25 | 95 | 2 | 2.10 | 53 | 1 |
| 25 | 883.02 | 12085 | 899 | 158.57 | 1810 | 171 | 852.88 | 8379 | 962 | 832.75 | 7832 | 1027 | 389.69 | 3544 | 333 | 857.31 | 7307 | 699 |
| 28 | 43.52 | 572 | 31 | 241.30 | 2022 | 251 | 30.89 | 161 | 5 | 89.42 | 441 | 35 | 194.78 | 1119 | 113 | 106.05 | 656 | 61 |
| 29 | 1,714.91 | 42775 | 8557 | 1,713.56 | 39998 | 9471 | 2,407.46 | 43979 | 11197 | 1,461.10 | 20813 | 5207 | 2,776.68 | 51304 | 10841 | 1,570.03 | 25831 | 5477 |
| **Select the arc for branching using the *MF* rule** | | | | | | | | | | | | | | | | | | |
| 6 | 1.08 | 193 | 47 | 0.82 | 84 | 21 | 0.98 | 100 | 25 | 0.98 | 106 | 29 | 1.10 | 164 | 33 | 1.13 | 198 | 45 |
| 7 | 2.87 | 391 | 57 | 1.91 | 222 | 25 | 2.23 | 203 | 27 | 4.16 | 360 | 81 | 2.94 | 222 | 29 | 3.62 | 251 | 29 |
| 9 | 0.52 | 125 | 21 | 0.50 | 98 | 21 | 0.63 | 98 | 23 | 0.68 | 102 | 25 | 0.72 | 134 | 23 | 0.85 | 143 | 27 |
| 11 | 1.13 | 151 | 25 | 0.94 | 98 | 17 | 0.85 | 82 | 13 | 1.14 | 93 | 15 | 1.22 | 122 | 19 | 1.34 | 109 | 11 |
| 12 | 14.61 | 2174 | 579 | 9.25 | 1277 | 325 | 6.44 | 762 | 221 | 12.00 | 1389 | 401 | 17.59 | 2261 | 607 | 14.06 | 1889 | 489 |
| 17 | 0.73 | 89 | 7 | 0.57 | 60 | 7 | 0.74 | 77 | 7 | 0.55 | 44 | 3 | 0.44 | 45 | 3 | 0.93 | 99 | 7 |
| 19 | 11.13 | 1216 | 201 | 16.19 | 1881 | 379 | 10.41 | 844 | 173 | 6.79 | 524 | 113 | 10.10 | 860 | 153 | 15.82 | 1180 | 209 |
| 23 | 1.10 | 72 | 1 | 2.13 | 54 | 1 | 2.61 | 58 | 1 | 4.12 | 73 | 3 | 5.35 | 95 | 2 | 2.13 | 53 | 1 |
| 25 | 643.79 | 8695 | 687 | 451.87 | 6236 | 761 | 1,394.33 | 15068 | 2063 | 2,277.97 | 22493 | 3319 | 1,282.84 | 15608 | 1642 | 2,340.25 | 21952 | 2516 |
| 28 | 43.90 | 572 | 31 | 116.01 | 1027 | 131 | 30.48 | 161 | 5 | 103.08 | 502 | 45 | 175.97 | 1030 | 101 | 103.48 | 619 | 52 |
| 29 | 38.25 | 789 | 155 | 330.35 | 6644 | 1689 | 367.58 | 6145 | 1635 | 174.20 | 2215 | 555 | 127.94 | 1825 | 371 | 115.17 | 1643 | 281 |
| **Select the arc for branching using the *MFC* rule** | | | | | | | | | | | | | | | | | | |
| 6 | 1.19 | 193 | 47 | 0.79 | 84 | 21 | 0.97 | 100 | 25 | 1.02 | 106 | 29 | 1.10 | 164 | 33 | 1.16 | 198 | 45 |
| 7 | 2.11 | 264 | 37 | 1.83 | 212 | 23 | 1.92 | 179 | 21 | 4.19 | 329 | 65 | 3.10 | 236 | 29 | 3.38 | 242 | 27 |
| 9 | 0.63 | 143 | 25 | 0.52 | 98 | 21 | 0.65 | 90 | 21 | 0.79 | 124 | 31 | 0.71 | 134 | 23 | 0.87 | 138 | 25 |
| 11 | 0.97 | 125 | 17 | 0.95 | 98 | 17 | 0.81 | 82 | 13 | 1.134 | 93 | 15 | 0.86 | 85 | 7 | 1.42 | 109 | 11 |
| 12 | 13.06 | 2022 | 523 | 9.21 | 1244 | 321 | 12.42 | 1479 | 419 | 11.089 | 1277 | 361 | 11.77 | 1644 | 461 | 14.11 | 1736 | 453 |
| 17 | 0.53 | 70 | 3 | 0.57 | 60 | 7 | 0.74 | 77 | 7 | 0.58 | 44 | 3 | 0.449 | 45 | 3 | 0.58 | 67 | 3 |
| 19 | 8.72 | 894 | 155 | 11.25 | 1172 | 229 | 11.53 | 900 | 177 | 8.44 | 641 | 147 | 9.15 | 817 | 145 | 14.96 | 1095 | 195 |
| 23 | 1.08 | 72 | 1 | 2.13 | 54 | 1 | 2.62 | 58 | 1 | 4.26 | 73 | 3 | 5.22 | 95 | 2 | 2.13 | 53 | 1 |
| 25 | 63.17 | 713 | 25 | 376.50 | 3901 | 471 | 119.80 | 830 | 66 | 503.35 | 4230 | 556 | 1,179.33 | 14473 | 1510 | 1,500.08 | 12833 | 1286 |
| 28 | 44.93 | 572 | 31 | 283.52 | 2221 | 289 | 31.46 | 161 | 5 | 88.14 | 441 | 35 | 172.70 | 1027 | 99 | 103.78 | 656 | 61 |
| 29 | 163.76 | 3744 | 818 | 145.53 | 2618 | 587 | 83.35 | 1127 | 261 | 189.05 | 2736 | 653 | 98.72 | 1644 | 318 | 174.85 | 2847 | 551 |

**Table 5**
Results obtained with the selected settings on the large instances.

| Instance | Initial solution | Root IP solution | Best solution | Integrality gap (%) | Solution time (s) | Iterations in total | Time per iteration (s) | Nodes |
|---|---|---|---|---|---|---|---|---|
| **(5, 5, *F*) with *MF* branching rule** | | | | | | | | |
| 31 | 5186 | **4935** | **4935** | 0.01 | 1,629.82 | 1375 | 1.18 | 1 |
| 32 | 5685 | **5278** | **5278** | 2.82 | TL | 56702 | 0.38 | 5518 |
| 33 | 5156 | 5091 | 5091 | 3.06 | TL | 48765 | 0.44 | 3564 |
| 34 | 5486 | 5219 | **5218** | 0.19 | 8,547.16 | 5747 | 1.48 | 305 |
| 35 | 5685 | 5536 | 5530 | 1.70 | TL | 39837 | 0.54 | 3708 |
| 36 | 7088 | **6498** | **6498** | 0 | 168.13 | 599 | 0.28 | 1 |
| 37 | 4967 | **4845** | **4845** | 0.55 | TL | 48336 | 0.44 | 3305 |
| 38 | 5745 | 5610 | **5608** | 0.99 | TL | 112974 | 0.19 | 12000 |
| 39 | 6552 | 5878 | 5878 | 1.37 | TL | 114894 | 0.19 | 12733 |
| 40 | 5265 | 5056 | **5048** | 2.43 | TL | 57739 | 0.37 | 3677 |
| **Average** | **5681.5** | **5394.6** | **5392.9** | **1.31** | **3448.37** | **48696.8** | **0.55** | **4481.2** |
| **(5, 5, *F*) with *MFC* branching rule** | | | | | | | | |
| 31 | 5186 | **4935** | **4935** | 0.01 | 1,645.31 | 1375 | 1.19 | 1 |
| 32 | 5685 | **5278** | **5278** | 2.82 | TL | 20252 | 1.06 | 1561 |
| 33 | 5156 | 5091 | **5083** | 2.90 | TL | 39177 | 0.55 | 2154 |
| 34 | 5486 | 5219 | **5218** | 0.19 | 4,618.77 | 3114 | 1.48 | 125 |
| 35 | 5685 | 5536 | 5528 | 1.67 | TL | 41013 | 0.52 | 2749 |
| 36 | 7088 | **6498** | **6498** | 0 | 183.85 | 599 | 0.31 | 1 |
| 37 | 4967 | **4845** | **4845** | 0.55 | TL | 51382 | 0.42 | 3966 |
| 38 | 5745 | 5610 | 5609 | 1.01 | TL | 122279 | 0.17 | 9513 |
| 39 | 6552 | 5878 | 5878 | 1.37 | TL | 88890 | 0.24 | 10250 |
| 40 | 5265 | 5056 | 5056 | 2.59 | TL | 52736 | 0.41 | 3217 |
| **Average** | **5681.5** | **5394.6** | **5392.8** | **1.31** | **2149.31** | **42081.7** | **0.64** | **3353.7** |
| **(10, 10, *F*) with *MFC* branching rule** | | | | | | | | |
| 31 | 5186 | **4935** | **4935** | 0.01 | 1,513.15 | 622 | 2.43 | 1 |
| 32 | 5685 | **5278** | **5278** | 2.82 | TL | 23568 | 0.91 | 2479 |
| 33 | 5156 | 5093 | 5085 | 2.94 | TL | 40682 | 0.53 | 3111 |
| 34 | 5486 | 5221 | **5218** | 0.19 | 12,159.45 | 5605 | 2.16 | 501 |
| 35 | 5685 | 5548 | **5519** | 1.50 | TL | 21859 | 0.98 | 2234 |
| 36 | 7088 | - | **6498** | 0 | 213.30 | 364 | 0.58 | 1 |
| 37 | 4967 | 4854 | 4854 | 0.74 | TL | 34600 | 0.62 | 3641 |
| 38 | 5745 | 5616 | 5610 | 1.02 | TL | 66327 | 0.32 | 7344 |
| 39 | 6552 | 5882 | **5849** | 0.87 | TL | 51237 | 0.42 | 7256 |
| 40 | 5265 | 5050 | 5050 | 2.47 | TL | 28476 | 0.75 | 3448 |
| **Average** | **5681.5** | **5275.22** | **5389.6** | **1.26** | **4628.63** | **27334** | **0.97** | **3001.6** |

solution within the time limit, it is also unable to solve the root LP; this never happens with the enhanced implementation. Finally, we note that the average integrality gap, when using the enhanced implementation, is 1.85%, which demonstrates that high-quality solutions can be obtained for VRPRDL instances with up to 120 customers.

## 5.3. Impact of distance of roaming delivery locations to the depot on algorithm performance

In the previous experiments, we focused on identifying an effective parameter configuration for solving the pricing problems and an effective branching scheme. Next, we assess the performance of the branch-and-price algorithm (with configuration *MF*(5, 5, *F*)) on the second set of instances in order to evaluate whether the distance of the roaming delivery locations to the depot affects its performance. The results for both variations are reported in Table 7.

We observe that for each instance the cost of the solution to the second variation is smaller than the cost of the solution to the first variation, which is to be expected as it should be possible to better exploit the roaming delivery locations when they are closer to the depot. Furthermore, we see that all second variation instances are solved optimally, with a maximum solution time of 1005.36*s*, whereas instance 42_$v$1 cannot be solved within the time limit of two hours. Moreover, the total time spent by the algorithm in solving all second variation instances is considerably less than the total time required to solve all first variation instances. However, the algorithm does not perform uniformly better on second variation instances. The fact that there are likely fewer feasible solutions in the first variation instances may explain why more first variation instances can be solved at the root node of the search tree. That there are likely more feasible solutions in the second variation instances is reflected by the fact that the problem graph after preprocessing for the second variation is denser for all instances.

## 5.4. Assessing the benefits of employing trunk delivery services

To be able to comprehensively assess the benefits of trunk delivery services, we need to be able to solve instances of the VRPHRDL as well. Therefore, we start by investigating how well our branch-and-price algorithm performs on VRPHRDL

**Table 6**
Straightforward branch-and-price vs. the default branch-and-price with parameter configuration (5, 5, *F*) and the *MF* branching rule.

| Instance | Straightforward | | *MF*(5, 5, *F*) | | | | |
|---|---|---|---|---|---|---|---|
| | Best solution | Solution time (s) | Initial solution | Root IP solution | Best solution | Integrality gap (%) | Solution time (s) |
| 1 | 901 | 0.58 | 957 | – | 901 | 0 | 0.26 |
| 2 | 1286 | 0.27 | 1292 | – | 1286 | 0 | 0.04 |
| 3 | 991 | 0.43 | 1004 | – | 991 | 0 | 0.07 |
| 4 | 1062 | 0.27 | 1069 | – | 1062 | 0 | 0.04 |
| 5 | 1832 | 0.04 | 1832 | – | 1832 | 0 | 0.02 |
| 6 | 1294 | 2.30 | 1300 | 1300 | 1294 | 1.73 | 1.08 |
| 7 | 1155 | 30.47 | 1158 | 1158 | 1155 | 1.29 | 2.87 |
| 8 | 1455 | 0.18 | 1555 | – | 1455 | 0 | 0.07 |
| 9 | 1260 | 3.32 | 1272 | 1268 | 1260 | 1.86 | 0.52 |
| 10 | 1684 | 0.55 | 1733 | – | 1684 | 0 | 0.03 |
| 11 | 1922 | 14.28 | 1931 | 1922 | 1922 | 0.31 | 1.13 |
| 12 | 2324 | 120.38 | 2325 | 2325 | 2324 | 2.52 | 14.61 |
| 13 | 1747 | 23.33 | 1758 | – | 1747 | 0 | 0.68 |
| 14 | 1273 | 30.83 | 1281 | – | 1273 | 0 | 0.64 |
| 15 | 1694 | 22.78 | 1696 | – | 1694 | 0 | 0.50 |
| 16 | 1938 | 57.13 | 1941 | – | 1938 | 0 | 0.75 |
| 17 | 1965 | 5.75 | 1966 | 1965 | 1965 | 0.10 | 0.73 |
| 18 | 1827 | 2.10 | 1831 | – | 1827 | 0 | 0.23 |
| 19 | 2083 | 93.55 | 2121 | 2110 | 2083 | 2.46 | 11.13 |
| 20 | 1822 | 78.70 | 1889 | – | 1822 | 0 | 1.53 |
| 21 | 3761 | 606.60 | 3775 | – | 3761 | 0 | 4.13 |
| 22 | 2828 | 272.43 | 2877 | – | 2828 | 0 | 10.74 |
| 23 | 4440 | 238.27 | 4447 | 4440 | 4440 | 0.01 | 1.10 |
| 24 | 3378 | 382.37 | 3477 | – | 3378 | 0 | 11.62 |
| 25 | 9784 | TL | 3375 | 3169 | 3161 | 0.84 | 643.79 |
| 26 | 4536 | 78.61 | 4586 | – | 4536 | 0 | 1.87 |
| 27 | 2865 | 296.23 | 2877 | – | 2865 | 0 | 7.08 |
| 28 | 9450 | TL | 4220 | 4176 | 4173 | 0.08 | 43.90 |
| 29 | 3964 | TL | 4017 | 3979 | 3964 | 0.73 | 38.25 |
| 30 | 4107 | 45.22 | 4122 | – | 4107 | 0 | 1.80 |
| 31 | 18142 | TL | 5186 | 4935 | 4935 | 0.01 | 1629.82 |
| 32 | 19514 | TL | 5685 | 5278 | 5278 | 2.82 | TL |
| 33 | 18008 | TL | 5156 | 5091 | 5091 | 3.06 | TL |
| 34 | 19880 | TL | 5486 | 5219 | 5218 | 0.19 | 8547.16 |
| 35 | 19196 | TL | 5685 | 5536 | 5530 | 1.70 | TL |
| 36 | 21772 | TL | 7088 | 6498 | 6498 | 0.00 | 168.13 |
| 37 | 20010 | TL | 4967 | 4845 | 4845 | 0.55 | TL |
| 38 | 20032 | TL | 5745 | 5610 | 5608 | 0.99 | TL |
| 39 | 20136 | TL | 6552 | 5878 | 5878 | 1.37 | TL |
| 40 | 20042 | TL | 5265 | 5056 | 5048 | 2.43 | TL |

instances. We experimented with the three settings that proved most effective for the VRPRDL instances and found that although no setting clearly dominated, most best-known solutions are found with the setting *MFC*(10, 10, *F*). Therefore, we report the results obtained with this setting in Table 8, where, for completeness sake, we include also the value of the best solution found by any of the three settings (highlighting in bold the values of the solutions that have a smaller cost than the value of the best solution obtained with the setting *MFC*(10, 10, *F*)).

For the VRPHRDL instances with up to 60 customers, 24 of the 30 instances can be solved to optimality within two hours and optimal solutions to most of these instances are found at the root node. Note that, as in previous experiments, the initial solutions are obtained by the heuristic of Reyes et al. (2016).

For the VRPHRDL instances with 120 customers, the algorithm does not finish solving the root node LP within six hours, demonstrating that VRPHRDL instances are harder to solve than VRPRDL instances. Even when the time limit is reached before solving the root node LP, a large number of columns has been generated, which enables us to find an improved solution by solving the restricted master problem as an integer program when the time limit is reached. We observe that for these difficult instances, the branch-and-price algorithm finds solutions that, on average, improve the initial solution by approximately 4%. As mentioned at the start of this section, the primary reason that VRPHRDL instances are more difficult than VRPRDL instances is that time windows at the home locations are much wider, which reduces the effectiveness of the preprocessing, and, as a consequence, leads to denser graphs and more arc variables.

Having alternative locations to deliver a customer order, e.g., to the trunk of the customer's car when it is parked somewhere other than at home, provides additional flexibility to a delivery company. However, it is easy to construct examples

**Table 7**
Results for Instances 41–50 obtained by enhanced branch-and-price with *MF*(5, 5, *F*).

| Instance | Initial solution | Root IP solution | Best solution | Integrality gap (%) | Solution time (s) | Iterations | Nodes |
|---|---|---|---|---|---|---|---|
| 41_$v$1 | 3278 | 3208 | 3203 | 1.88 | 1249.35 | 33672 | 6147 |
| 41_$v$2 | 2147 | 2139 | 2133 | 2.33 | 854.47 | 7528 | 805 |
| 42_$v$1 | 2842 | – | 2799 | 0 | 3.00 | 58 | 1 |
| 42_$v$2 | 2032 | 2012 | 1946 | 4.12 | 1005.36 | 6472 | 743 |
| 43_$v$1 | 2614 | 2607 | 2607 | 3.68 | TL | 111845 | 14854 |
| 43_$v$2 | 1995 | 1966 | 1966 | 1.09 | 270.72 | 1644 | 135 |
| 44_$v$1 | 2344 | 2273 | 2261 | 2.03 | 98.52 | 2342 | 251 |
| 44_$v$2 | 1749 | – | 1610 | 0 | 41.59 | 222 | 1 |
| 45_$v$1 | 3220 | – | 3217 | 0 | 1.63 | 34 | 1 |
| 45_$v$2 | 2479 | – | 2478 | 0 | 9.76 | 80 | 1 |
| 46_$v$1 | 2806 | 2805 | 2805 | 0.91 | 3.81 | 126 | 5 |
| 46_$v$2 | 2504 | 2469 | 2469 | 1.11 | 27.37 | 302 | 39 |
| 47_$v$1 | 3463 | 3385 | 3339 | 3.38 | 3710.35 | 89246 | 21169 |
| 47_$v$2 | 1992 | 1947 | 1946 | 1.22 | 68.96 | 556 | 37 |
| 48_$v$1 | 3331 | – | 3325 | 0 | 1.15 | 48 | 1 |
| 48_$v$2 | 2418 | 2386 | 2380 | 1.65 | 477.83 | 8384 | 1493 |
| 49_$v$1 | 3598 | 3538 | 3534 | 1.82 | 104.26 | 3084 | 595 |
| 49_$v$2 | 2605 | 2507 | 2492 | 0.55 | 13.62 | 207 | 9 |
| 50_$v$1 | 2779 | – | 2752 | 0 | 8.74 | 114 | 1 |
| 50_$v$2 | 2551 | 2449 | 2443 | 0.38 | 164.371 | 1038 | 119 |

**Table 8**
Results for the VRPHRDL instances with parameter configuration (10, 10, *F*) and the *MFC* branching rule.

| Instance | Initial solution | Root IP solution | Best solution | Best known solution | Integrality gap (%) | Solution time (s) | Iterations | Nodes |
|---|---|---|---|---|---|---|---|---|
| 1 | 777 | – | 773 | 773 | 0 | 0.62 | 23 | 1 |
| 2 | 1111 | – | 1065 | 1065 | 0 | 0.08 | 9 | 1 |
| 3 | 991 | – | 988 | 988 | 0 | 0.11 | 15 | 1 |
| 4 | 916 | – | 914 | 914 | 0 | 0.17 | 16 | 1 |
| 5 | 1710 | – | 1710 | 1710 | 0 | 0.04 | 10 | 1 |
| 6 | 1099 | 1099 | 1099 | 1099 | 2.04 | 2.84 | 147 | 23 |
| 7 | 1020 | 1010 | 996 | 996 | 0.67 | 11.02 | 290 | 36 |
| 8 | 1457 | – | 1346 | 1346 | 0 | 0.33 | 24 | 1 |
| 9 | 998 | – | 997 | 997 | 0 | 0.56 | 22 | 1 |
| 10 | 1167 | – | 1166 | 1166 | 0 | 0.18 | 26 | 1 |
| 11 | 1607 | – | 1587 | 1587 | 0 | 8.54 | 80 | 1 |
| 12 | 1834 | – | 1808 | 1808 | 0 | 4.70 | 53 | 1 |
| 13 | 1563 | – | 1563 | 1563 | 0 | 3.38 | 38 | 1 |
| 14 | 1058 | – | 1058 | 1058 | 0 | 3.26 | 45 | 1 |
| 15 | 1363 | 1355 | 1347 | 1347 | 2.06 | 155.93 | 1232 | 175 |
| 16 | 1574 | 1564 | 1517 | 1517 | 2.09 | TL | 139823 | 30731 |
| 17 | 1446 | 1445 | 1445 | 1445 | 0 | 2.14 | 58 | 1 |
| 18 | 1679 | 1627 | 1627 | 1627 | 0.95 | 26.67 | 271 | 35 |
| 19 | 1468 | – | 1461 | 1461 | 0 | 1.59 | 38 | 1 |
| 20 | 1730 | – | 1715 | 1715 | 0 | 2.09 | 49 | 1 |
| 21 | 2860 | – | 2580 | 2580 | 0 | 396.47 | 495 | 1 |
| 22 | 2213 | 2213 | 2213 | 2213 | 4.98 | TL | 7654 | 591 |
| 23 | 3393 | 3373 | 3363 | 3363 | 0.18 | 194.98 | 372 | 23 |
| 24 | 2587 | 2574 | 2574 | **2569** | 4.70 | TL | 8400 | 596 |
| 25 | 2429 | 2414 | 2400 | 2400 | 5.38 | TL | 8898 | 622 |
| 26 | 2881 | 2847 | 2846 | **2845** | 0.80 | TL | 19931 | 2414 |
| 27 | 2521 | – | 2518 | 2518 | 0 | 33.85 | 114 | 1 |
| 28 | 2830 | 2758 | 2758 | 2758 | 0.04 | 3,392.94 | 578 | 5 |
| 29 | 2917 | 2913 | 2913 | **2892** | 6.93 | TL | 22862 | 3460 |
| 30 | 2711 | – | 2691 | 2691 | 0 | 41.77 | 106 | 1 |
| 31 | 3991 | 3984 | 3984 | 3984 | – | TL | 1876 | 0 |
| 32 | 4012 | 3958 | 3958 | 3958 | – | TL | 1011 | 0 |
| 33 | 3828 | 3645 | 3645 | **3630** | – | TL | 2155 | 0 |
| 34 | 4065 | 3958 | 3958 | **3891** | – | TL | 1598 | 0 |
| 35 | 3290 | 3255 | 3255 | 3255 | – | TL | 1905 | 0 |
| 36 | 4607 | 4533 | 4533 | **4525** | – | TL | 1102 | 0 |
| 37 | 3585 | 3395 | 3395 | 3395 | – | TL | 2063 | 0 |
| 38 | 4131 | 3980 | 3980 | **3976** | – | TL | 1343 | 0 |
| 39 | 4686 | 4316 | 4316 | 4316 | – | TL | 1519 | 0 |
| 40 | 3980 | 3680 | 3680 | 3680 | – | TL | 1876 | 0 |

**Table 9**
Comparison of the VRP, the VRPRDL, and the VRPHRDL solutions.

| Instance | VRP | | VRPRDL | | | VRPHRDL | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Routes | Best solution | Home/total | Routes | Best solution | Home/total | Routes | Best solution | Savings wrt best VRP solution (%) |
| 1 | 3 | 864 | 0.47 | 4 | 901 | 0.60 | 3 | 773 | 10.53 |
| 2 | 4 | 1187 | 0.47 | 5 | 1286 | 0.73 | 4 | 1065 | 10.28 |
| 3 | 5 | 1305 | 0.60 | 4 | 991 | 0.67 | 3 | 988 | 24.29 |
| 4 | 3 | 974 | 0.53 | 5 | 1062 | 0.87 | 3 | 914 | 6.16 |
| 5 | 7 | 2171 | 0.53 | 6 | 1832 | 0.67 | 6 | 1710 | 21.23 |
| 6 | 4 | 1246 | 0.65 | 5 | 1294 | 0.75 | 4 | 1099 | 11.80 |
| 7 | 4 | 1156 | 0.55 | 4 | 1155 | 0.85 | 3 | 996 | 13.84 |
| 8 | 5 | 1636 | 0.60 | 6 | 1455 | 0.70 | 5 | 1346 | 17.73 |
| 9 | 4 | 1215 | 0.35 | 5 | 1260 | 0.60 | 4 | 997 | 17.94 |
| 10 | 5 | 1444 | 0.60 | 8 | 1684 | 0.85 | 4 | 1166 | 19.25 |
| 11 | 8 | 2491 | 0.50 | 7 | 1922 | 0.70 | 5 | 1587 | 36.29 |
| 12 | 6 | 2001 | 0.50 | 8 | 2324 | 0.67 | 6 | 1808 | 9.65 |
| 13 | 6 | 1774 | 0.60 | 6 | 1747 | 0.73 | 6 | 1563 | 11.89 |
| 14 | 5 | 1648 | 0.50 | 6 | 1273 | 0.80 | 4 | 1058 | 35.80 |
| 15 | 6 | 1935 | 0.47 | 6 | 1694 | 0.57 | 5 | 1347 | 30.39 |
| 16 | 6 | 1890 | 0.47 | 7 | 1938 | 0.80 | 5 | 1517* | 19.74 |
| 17 | 7 | 2199 | 0.33 | 8 | 1965 | 0.57 | 5 | 1445 | 34.29 |
| 18 | 6 | 1836 | 0.43 | 7 | 1827 | 0.73 | 5 | 1627 | 11.38 |
| 19 | 6 | 1889 | 0.70 | 7 | 2083 | 0.83 | 5 | 1461 | 22.66 |
| 20 | 6 | 1823 | 0.83 | 6 | 1822 | 0.83 | 6 | 1715 | 5.92 |
| 21 | 9 | 3046 | 0.47 | 13 | 3761 | 0.80 | 8 | 2580 | 15.30 |
| 22 | 7 | 2452* | 0.57 | 10 | 2828 | 0.78 | 7 | 2213* | 9.75 |
| 23 | 12 | 3949 | 0.48 | 16 | 4440 | 0.87 | 10 | 3363 | 14.84 |
| 24 | 9 | 2924 | 0.43 | 11 | 3378 | 0.80 | 8 | 2569* | 12.14 |
| 25 | 8 | 2648* | 0.47 | 11 | 3161 | 0.78 | 8 | 2400* | 9.37 |
| 26 | 11 | 3663 | 0.47 | 16 | 4536 | 0.73 | 9 | 2845* | 22.33 |
| 27 | 11 | 3438 | 0.60 | 10 | 2865 | 0.72 | 8 | 2518 | 26.76 |
| 28 | 10 | 3423 | 0.55 | 14 | 4173 | 0.83 | 8 | 2758 | 19.43 |
| 29 | 12 | 4010* | 0.48 | 14 | 3964 | 0.75 | 9 | 2892* | 27.88 |
| 30 | 12 | 3924 | 0.38 | 14 | 4107 | 0.77 | 8 | 2691 | 31.42 |
| 31 | 16 | 4919 | 0.53 | 18 | 4935 | 0.82 | 14 | 3984* | 19.01 |
| 32 | 15 | 4593* | 0.46 | 19 | 5278* | 0.80 | 13 | 3958* | 13.83 |
| 33 | 14 | 4296* | 0.53 | 18 | 5083* | 0.73 | 13 | 3630* | 15.50 |
| 34 | 14 | 4613* | 0.52 | 17 | 5218 | 0.83 | 13 | 3891* | 15.65 |
| 35 | 11 | 3725* | 0.48 | 20 | 5519* | 0.86 | 11 | 3255* | 12.62 |
| 36 | 18 | 5745* | 0.51 | 22 | 6498 | 0.83 | 15 | 4525* | 21.24 |
| 37 | 14 | 4804* | 0.48 | 17 | 4845* | 0.74 | 11 | 3395* | 29.33 |
| 38 | 14 | 4395* | 0.55 | 21 | 5608* | 0.79 | 14 | 3976* | 9.53 |
| 39 | 19 | 6028* | 0.45 | 24 | 5849* | 0.84 | 15 | 4316* | 28.40 |
| 40 | 13 | 3871* | 0.46 | 19 | 5048* | 0.85 | 13 | 3680* | 4.93 |
| **Average** | **8.88** | **2828.75** | **0.51** | **11.10** | **3065.23** | **0.76** | **7.65** | **2290.53** | **18.26** |

where allowing deliveries only to the trunk of a customer's car may not lead to cost savings, but may, in fact, lead to an increase in cost, compared to being able to only deliver at the customer's home. This is the reason that companies are more likely to deploy a hybrid model in which deliveries can either be made to the customer's home or to the customer's car.

To assess the benefits of trunk delivery, we compute, for the same instance, a VRP solution, in which deliveries can only be made at the customer's home location, a VRPRDL solution, in which deliveries can only be made to the trunk of the customer's car, and a VRPHRDL solution, in which deliveries can be made either to the customer's home or to the trunk of the customer's car. The results can be found in Table 9, where the VRP solution is also computed with our branch-and-price algorithm. We report the cost, the fraction of deliveries made at a customer's home, and the number of delivery routes. We use a star to indicate that the best solution is not necessarily optimal, i.e., the algorithm reached the time limit.

We observe that having the flexibility to deliver either to a customer's home or to the trunk of the customer's car has huge advantages. The average percentage of cost-savings is over 18% (with a minimum cost savings of 4.93% and a maximum cost savings of 36.29%). This is, in a large part, because fewer delivery routes are required; on average 8.88 for the VRP solutions and on average 7.65 for the VRPHRDL solutions. Interestingly, the average number of delivery routes in the VRPRDL solutions is 11.10. Even though there are more delivery locations to choose from, the time during which deliveries can be made is less, because deliveries cannot be made while the car is driving. Also interesting to note is that the fraction of deliveries made at a customer's home location in VRPHRDL solutions is quite high, on average 0.76.

The difference in VRP, VRPRDL, and VRPHRDL solutions can be seen quite well in Fig. 1, where we show the optimal VRP, VRPRDL, and VRPHRDL solutions for Instance 28. The empty rectangles represent the home locations while the filled

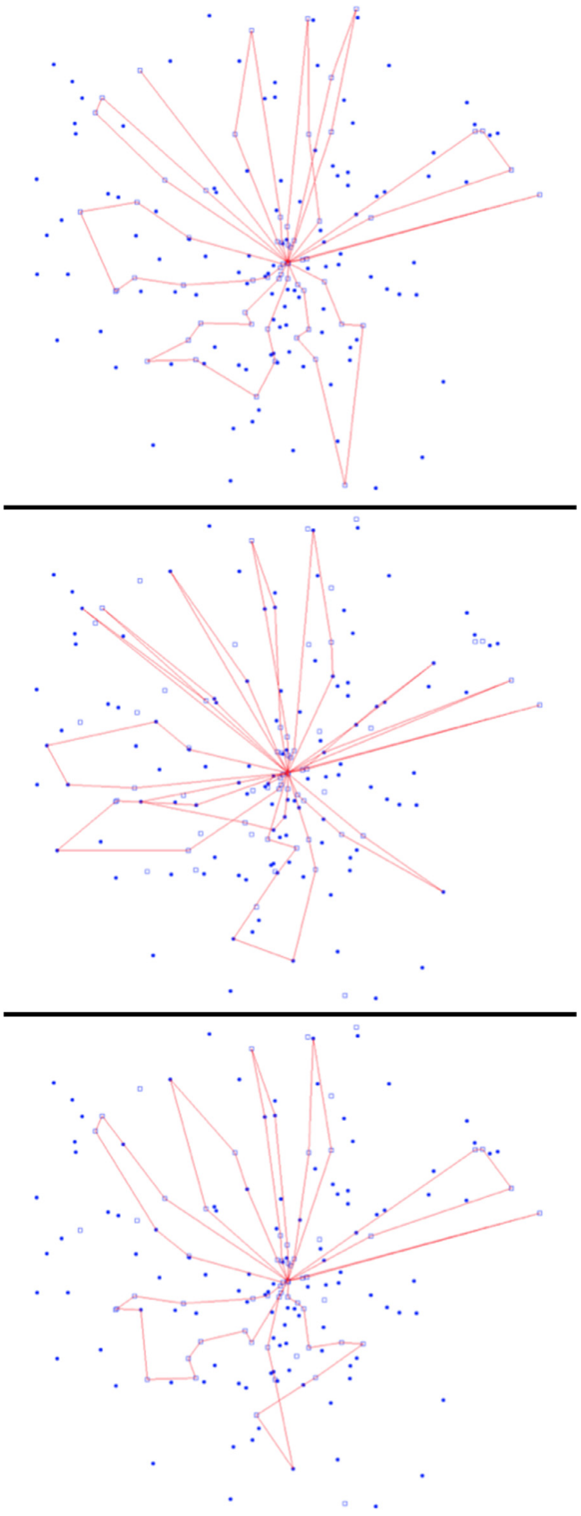

**Fig. 1.** The optimal VRP, VRPRDL and VRPHRDL solutions for Instance 28 from top to bottom, respectively .

circles represent the other potential delivery locations. The optimal VRP solution has 10 delivery routes and a cost of 3423, whereas the number of delivery routes in the optimal VRPRDL solution is 14 with cost 4173 (55% of the deliveries are made at home locations), and the number of delivery routes in the optimal VRPHRDL solutions is only 8 with cost 2758 (83% of the deliveries are made at home locations).

We repeated the last experiment for the instances of the second set and found similar results (see Table 10 in the Appendix B).

## 6. Concluding remarks

Trunk delivery is one of the innovative ideas being explored to reduce delivery cost in the business-to-consumer retail market sector. Our computational study shows that a hybrid model in which a delivery can be made either at a customer's home or to the trunk of the customer's car has huge potential. On the instances in our test suite, the cost savings were in the order of 20%.

We have assumed deterministic travel times and complete knowledge of the itinerary of a customer (more specifically the itinerary of the customer's car). These are strong assumptions, but they are, in our opinion, not completely unreasonable. There are well-funded and highly-regarded start-up companies, e.g., Roadie (www.roadie.com), with a business proposition that is entirely based on the assumption that in the future, there will be reliable and predictable information on the travel patterns of people (based on 24 h monitoring of the location of their smart phone and predictive analytics). Starting to explore new business models assuming this information is available is important, and we are among the first to do so.

However, we recognize that an important next step is to investigate operational strategies and related optimization models that can dynamically handle deviations from the planned customer itineraries. This is precisely what we are currently pursuing.

## Acknowledgements

We would like to thank Luis Damian Reyes Rodriguez for generating new test instances and providing associated heuristic solutions. The research of the first author is supported by the Scientific and Technological Research Council of Turkey, grant number BIDEB-2214-A.

## Appendix A

Let $L_i$ denote the set of all labels associated with the efficient (i.e., non-dominated) partial paths ending at node $i$, and let the source $s$ and the sink $t$ correspond to the depot (Algorithm 2).

**Algorithm 2:** labelSetting(S).

```
Data: Source node $s$, sink node $t$, resource limits $R_i^r$ for $i \in N$ and $r \in \{1,2\}$, resource consumptions $h_{ij}^r$ for $(i,j) \in A$ and
      $r \in \{1,2\}$, opening of time window $e_i$ for every $i \in N$
Result: The set $\mathcal{P}^*$ of all paths corresponding to non-dominated labels on the sink node $t$
Initialization
Let $L_0 = \{(0,0,0)\}$, $L_i = \emptyset$ for all $i \in N \setminus \{s\}$, and $L = \bigcup_{i \in N} L_i$.
Label selection and treatment
while $L \neq \emptyset$ do
    Select $i \in N$ and $\mathbf{U^i} \in L_i$ so that $\mathbf{U^i}$ is the lexicographically minimal label in $L$
    $L \leftarrow L \setminus \{\mathbf{U^i}\}$
    for $(i,j) \in A$ do
        if canExtendLabel($i, \mathbf{U^i}, \mathbf{j}$) then
            $\mathbf{U^j}$ = extendLabel($i, \mathbf{U^i}, \mathbf{j}$)
            performDominanceCheck($\mathbf{U^j}, \mathbf{j}$)
Return $\mathcal{P}^*$
```

**Subroutine 3:** *canExtendLabel(i, $\mathbf{U^i}$, j)*

```
Returns false if the extension of label $\mathbf{U^i}$ along $(i, j)$ is not resource-feasible. Note that the feasibility of extension along $(i, t)$ is established before creating $\mathbf{U^i}$
if $j \neq t$ then
    if $c(j) \in S$ then
        Let $k$ be the order of customer $c(j)$ in $S$
        if $U^i_{k+2} = 1$ then
            Return $false$
    if $max\{U^i_1 + h^1_{ij}, e_j\} > R^1_j$ or $max\{U^i_1 + h^1_{ij}, e_j\} + h^1_{jt} > R^1_t$ then
        Return $false$
    else if $U^i_2 + h^2_{ij} > R^2_j$ then
        Return $false$
    else
        Return $true$
else
    Return $true$
```

**Subroutine 4:** *extendLabel(i, $\mathbf{U^i}$, j)*

```
Extends the label $\mathbf{U^i}$ through arc $(i, j)$ by updating the cost component and resource consumptions, and then returns the new label
$U^j_0 \leftarrow U^i_0 + (w_{ij} - \lambda_{c(i)})$
$U^j_1 \leftarrow max\{U^i_1 + h^1_{ij}, e_j\}$
$U^j_2 \leftarrow U^i_2 + h^2_{ij}$
if $j \neq t$ then
    if $c(j) \in S$ then
        Let $k$ be the order of customer $c(j)$ in $S$
        $U^j_{k+2} \leftarrow 1$
    For strong dominance, consume the visitation resource for each customer in S that is not on the partial path extended to j, and yet is unreachable from node j
    for $c \in S$ do
        Let $l$ be the order of customer $c$ in S
        if $U^j_{l+2} = 0$ and isReachable(j, $\mathbf{U^j}$, $\mathbf{c}$) = false then
            $U^j_{l+2} \leftarrow 1$
Return $\mathbf{U^j}$
```

**Subroutine 5:** *isReachable(i, $\mathbf{U^i}$, c)*

```
Returns true if any of the locations corresponding to customer c is reachable from node i by extending the partial path associated with the label $U^i$
for $j \in N_c$ do
    if canExtendLabel($i$, $\mathbf{U^i}$, $j$) then
        Return $true$
Return $false$
```

**Subroutine 6:** *performDominanceCheck(*$\mathbf{U^i}$*, i)*

```
If the label $\mathbf{U^i}$ is dominated by an existing label, then this method returns false; otherwise it adds $\mathbf{U^i}$ to $L_i$ and $L$, removes the labels dominated by $\mathbf{U^i}$ from $L_i$ and $L$ (if any such label is found), and returns true
for $\mathbf{U} \in L_i$ do
    if $\mathbf{U^i} \neq \mathbf{U}$ then
        if $\mathbf{U}$ dominates $\mathbf{U^i}$ then
            Return false
        else if $\mathbf{U_i}$ dominates $\mathbf{U}$ then
            $L_i \leftarrow L_i \setminus \{\mathbf{U}\}$
            $L \leftarrow L \setminus \{\mathbf{U}\}$
        else
            Continue
    else
        Break the loop
$L_i \leftarrow L_i \cup \{\mathbf{U^i}\}$
if $i \neq t$ then
    $L \leftarrow L \cup \{\mathbf{U^i}\}$
Return true
```

## Appendix B

**Table 10**
Comparison of the VRP, the VRPRDL, and the VRPHRDL solutions for instances in the second set.

| Instance | VRP | | VRPRDL | | | VRPHRDL | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Routes | Best solution | Home/total | Routes | Best solution | Home/total | Routes | Best solution | Savings wrt best VRP solution (%) |
| 41_$v$1 | 8 | 2745 | 0.55 | 10 | 3203 | 0.80 | 8 | 2662 | 3.02 |
| 41_$v$2 | 8 | 2738 | 0.53 | 7 | 2133 | 0.65 | 6 | 1998 | 27.03 |
| 42_$v$1 | 9 | 2805 | 0.58 | 9 | 2799 | 0.80 | 8 | 2610 | 6.95 |
| 42_$v$2 | 9 | 2806 | 0.58 | 7 | 1946 | 0.58 | 6 | 1946* | 30.65 |
| 43_$v$1 | 8 | 2536* | 0.45 | 8 | 2607* | 0.70 | 7 | 2260 | 10.88 |
| 43_$v$2 | 8 | 2536* | 0.43 | 8 | 1966 | 0.50 | 6 | 1830 | 27.84 |
| 44_$v$1 | 7 | 2318 | 0.65 | 7 | 2261 | 0.75 | 7 | 2147 | 7.38 |
| 44_$v$2 | 7 | 2315 | 0.60 | 6 | 1610 | 0.70 | 5 | 1478 | 36.16 |
| 45_$v$1 | 10 | 3191 | 0.80 | 10 | 3217 | 0.90 | 10 | 3172 | 0.60 |
| 45_$v$2 | 10 | 3192 | 0.70 | 8 | 2478 | 0.73 | 8 | 2466 | 22.74 |
| 46_$v$1 | 8 | 2718 | 0.53 | 9 | 2805 | 0.75 | 8 | 2616 | 3.75 |
| 46_$v$2 | 8 | 2717 | 0.50 | 8 | 2469 | 0.58 | 8 | 2388 | 12.11 |
| 47_$v$1 | 9 | 3069 | 0.53 | 10 | 3339 | 0.70 | 9 | 3011* | 1.89 |
| 47_$v$2 | 9 | 3060 | 0.40 | 7 | 1946 | 0.50 | 6 | 1848 | 39.61 |
| 48_$v$1 | 10 | 3358 | 0.50 | 10 | 3325 | 0.65 | 10 | 3278 | 2.38 |
| 48_$v$2 | 10 | 3359 | 0.40 | 8 | 2380 | 0.60 | 7 | 2264 | 32.60 |
| 49_$v$1 | 11 | 3615 | 0.55 | 11 | 3534 | 0.65 | 11 | 3514* | 2.79 |
| 49_$v$2 | 11 | 3615 | 0.45 | 8 | 2492 | 0.55 | 8 | 2457 | 32.03 |
| 50_$v$1 | 9 | 2928 | 0.50 | 10 | 2752 | 0.55 | 10 | 2727 | 6.86 |
| 50_$v$2 | 9 | 2930 | 0.55 | 8 | 2443 | 0.73 | 7 | 2302 | 21.43 |
| **Average** | **8.90** | **2927.55** | **0.54** | **8.45** | **2585.25** | **0.67** | **7.75** | **2448.70** | **16.44** |

## References

Afsar, H.M., Prins, C., Santos, A.C., 2014. Exact and heuristic algorithms for solving the generalized vehicle routing problem with flexible fleet size. Int. Trans. Oper. Res. 21 (1), 153–175.

Agra, A., Christiansen, M., Figueiredo, R., Hvattum, L.M., Poss, M., Requejo, C., 2013. The robust vehicle routing problem with time windows. Comput. Oper. Res. 40 (3), 856–866.

Audi, 2015. Audi, DHL and Amazon Deliver Convenience. https://www.audiusa.com/newsroom/news/press-releases/2015/04/audi-dhl-and-amazon-deliver-convenience. Accessed: 2016-07-24.

Baldacci, R., Bartolini, E., Laporte, G., 2010. Some applications of the generalized vehicle routing problem. J. Oper. Res. Soc. 61 (7), 1072–1077.

Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W., Vance, P.H., 1998. Branch-and-price: column generation for solving huge integer programs. Oper. Res. 46 (3), 316–329.

Behrmann, E., Weiss, R., 2016. Soon Your Smart Car Will Also Be An Amazon Locker. http://www.bloomberg.com/news/articles/2016-07-25/smart-city-cars-to-become-delivery-stations-in-dhl-german-test. Accessed: 2016-08-07.
Bektas, T., Erdogan, G., Røpke, S., 2011. Formulations and branch-and-cut algorithms for the generalized vehicle routing problem. Transp. Sci. 45 (3), 299–316.
Biesinger, B., Hu, B., Raidl, G., 2016. An integer l-shaped method for the generalized vehicle routing problem with stochastic demands. Electron. Notes Discrete Math. 52, 245–252.
Boland, N., Dethridge, J., Dumitrescu, I., 2006. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. Oper. Res. Lett. 34 (1), 58–68.
Christiansen, C.H., Lysgaard, J., 2007. A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. Oper. Res. Lett. 35 (6), 773–781.
Dabia, S., Ropke, S., Van Woensel, T., De Kok, T., 2013. Branch and price for the time-dependent vehicle routing problem with time windows. Transp. Sci. 47 (3), 380–396.
Dell'Amico, M., Righini, G., Salani, M., 2006. A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. Transp. Sci. 40 (2), 235–247.
Desrochers, M., Desrosiers, J., Solomon, M., 1992. A new optimization algorithm for the vehicle routing problem with time windows. Oper Res 40 (2), 342–354.
Desrochers, M., Lenstra, J.K., Savelsbergh, M.W., Soumis, F., 1988. Vehicle routing with time windows: optimization and approximation. Veh. Rout. Methods Stud. 16, 65–84.
Farber, M., 2016. Consumers Are Now Doing Most of their Shopping Online. http://fortune.com/2016/06/08/online-shopping-increases/. Accessed: 2016-07-24.
Feillet, D., 2010. A tutorial on column generation and branch-and-price for vehicle routing problems. 4OR 8 (4), 407–424.
Feillet, D., Dejax, P., Gendreau, M., Gueguen, C., 2004. An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks 44 (3), 216–229.
Garcia, A., 2015. Amazon Prime Day Shattered Global Sales Records. http://money.cnn.com/2015/07/15/news/amazon-walmart-sales/. Accessed: 2015-09-22.
Geuss, M., 2015. Amazon, Audi and DHL Want to Turn A Car Trunk into A Delivery Locker. http://arstechnica.com/business/2015/04/amazon-audi-and-dhl-want-to-turn-a-car-trunk-into-a-delivery-locker. Accessed: 2016-08-07.
Ghiani, G., Improta, G., 2000. An efficient transformation of the generalized vehicle routing problem. Eur. J. Oper. Res. 122 (1), 11–17.
Gustafson, K., 2016. Amazon Just had its Biggest Sales Day Ever. http://www.cnbc.com/2016/07/13/amazon-prime-day-is-biggest-day-for-online-retailer-ever.html. Accessed: 2016-07-24.
Koç, Ç., Bektaş, T., Jabali, O., Laporte, G., 2015. A hybrid evolutionary algorithm for heterogeneous fleet vehicle routing problems with time windows. Comput. Oper. Res. 64, 11–27.
Kovacs, A.A., Golden, B.L., Hartl, R.F., Parragh, S.N., 2014. The generalized consistent vehicle routing problem. Transp. Sci. 49 (4), 796–816.
Louati, A., et al., 2016. Modeling municipal solid waste collection: a generalized vehicle routing model with multiple transfer stations, gather sites and inhomogeneous vehicles in time windows. Waste Manage. 52, 34–49.
Moccia, L., Cordeau, J.-F., Laporte, G., 2012. An incremental tabu search heuristic for the generalized vehicle routing problem with time windows. J. Oper. Res. Soc. 63 (2), 232–244.
Popken, B., 2015. Amazon Tests Delivery to your car trunk. http://www.nbcnews.com/business/autos/amazon-testing-delivery-your-car-trunk-n346886. Accessed: 2016-08-07.
Quttineh, N.-H., Larsson, T., Van den Bergh, J., Beliën, J., 2015. A time-indexed generalized vehicle routing model and stabilized column generation for military aircraft mission planning. In: Optimization, Control, and Applications in the Information Age. Springer, pp. 299–314.
Reyes, D., Savelsbergh, M., Toriello, A., 2016. Vehicle routing with roaming delivery locations. Optim. Online 01–5281.
Salani, M., Vacca, I., 2011. Branch and price for the vehicle routing problem with discrete split deliveries and time windows. Eur. J. Oper. Res. 213 (3), 470–477.
Savelsbergh, M., 1985. Local search in routing problems with time windows. Ann. Oper. Res. 4 (1), 285–305.
Schneider, M., Stenger, A., Goeke, D., 2014. The electric vehicle-routing problem with time windows and recharging stations. Transp. Sci. 48 (4), 500–520.
Solomon, M.B., 2016. Amazon's First-Quarter Shipping Costs Hit $3.27 billion, 42-percent Jump from 2015. http://www.dcvelocity.com/articles/20160428-amazons-first-quarter-shipping-costs-hit-327-billion-42-percent-jump-from-2015. Accessed: 2016-08-08.
Solomon, M.M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper. Res. 35 (2), 254–265.
Taş, D., Gendreau, M., Dellaert, N., Van Woensel, T., De Kok, A., 2014. Vehicle routing with soft time windows and stochastic travel times: a column generation and branch-and-price solution approach. Eur. J. Oper. Res. 236 (3), 789–799.
Volvo, 2015. Volvo in-Car Delivery. https://incardelivery.volvocars.com
Volvo, 2016. Volvo's Solution for the Package Theft Epidemic: Your Car's Trunk. http://fortune.com/2016/05/10/volvo-urb-it-delivery.