A Branch and Price Algorithm for the Heterogeneous Fleet Multi-depot Multi-trip Vehicle Routing Problem with Time Windows

Munise Kübra Şahin
ORSTAT, Faculty of Economics and Business, KU Leuven, 3000 Leuven, Belgium. munisekubra.sahin@kuleuven.be

Hande Yaman
ORSTAT, Faculty of Economics and Business, KU Leuven, 3000 Leuven, Belgium. hande.yaman@kuleuven.be

The multi-trip vehicle routing problem (MTVRP) extends the well-known VRP by allowing vehicles to perform several trips in a workday. The motivation arises from the new challenges in city logistics that push companies to use smaller and cleaner vehicles such as cargo bikes. With the integration of small vehicles into the fleet, many companies start to operate with a heterogeneous fleet and use multi depots located in the city centers to reload the small vehicles. Inspired by these new challenges the companies face, we study the heterogeneous fleet multi-depot MTVRP with time windows under shared depot resources where small and large vehicles have different travel times in certain areas. We formulate this problem using workday variables and propose a branch and price algorithm that exhibits an enhanced performance by a new heuristic algorithm based on the reduction in the graph size. The proposed algorithm introduces a new way to compute the completion bounds using the iterative structure of the state-space augmenting algorithm and eliminates the need for solving a separate relaxation. We conduct experiments on modified small and medium-size instances from Solomon's benchmark set. The results of our computational experiments show that the proposed algorithm is very effective and can solve instances with up to 40 customers, three depots and two types of vehicles.

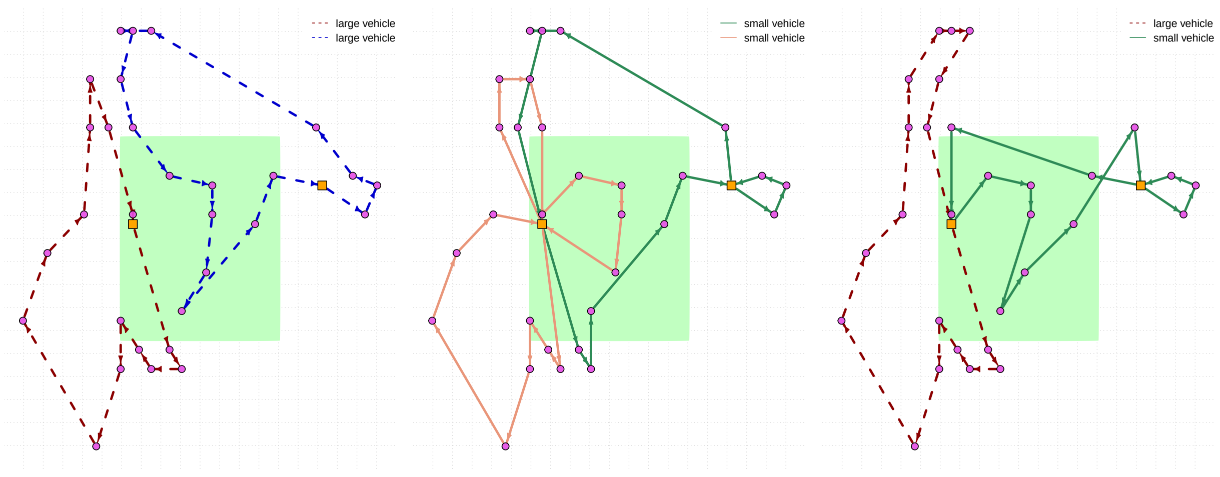
Key words: multi-trip vehicle routing; multi-depot vehicle routing; heterogeneous fleet; time windows; branch and price

1. Introduction

The United Nations estimates that two out of three people will live in urban areas by 2050. Worldwide, nearly one out of seven customers was an online shopper in 2019 (Vision Monday, 2021). This ratio is expected to increase substantially with the effect of the Covid-19 pandemic on the customers' buying behaviours. The upward trend in urban population density and the growth of e-commerce result in significant increase in urban freight demand causing more congestion and pollution and making freight distribution a significant challenge in city logistics. Many countries have implemented pull and push measures to cope with the negative effects of freight vehicles on the quality of life in urban areas. Push measures, such as defining restricted zones where large vehicles are prohibited during busy hours, discourage the use of large vehicles. Pull measures, such as improving bicycle infrastructure (Pucher, Dill, and Handy, 2010), promote cycling by improving the convenience of bicycle use. Five main principles have been proposed for bicycle infrastructure planning (Groot, 2007), among which directness is the most relevant one to our work. The directness principle aims to provide as direct a route as possible with fewer traffic lights and intersections with sidewalks to keep the travel time by bike shorter than by car. With this aim, some European cities such as Copenhagen and Antwerp have constructed bicycle bridges to provide short-cuts for bikers. These measures and incentives drove companies such as UPS, DHL and FedEx to deploy more sustainable vehicles such as cargo bikes for last-mile deliveries. Besides the new roads constructed to encourage sustainable mobility, these vehicles can also use the narrow streets common in European cities as short-cuts and spend less time searching for a parking space. However, cargo bikes do not provide an all-embracing solution to the delivery companies due to their limited speed on larger roads and their low load capacity compared to traditional vehicles. Hence most delivery companies using cargo bikes operate with a heterogeneous fleet to make use of the compatibility of different vehicle types to different routes. With the integration of small vehicles to the fleet, obliging vehicles to perform at most one route, as in the traditional setting of the VRP, leads to an oversized fleet and inefficient utilization of workdays (Cattaruzza et al., 2014). Relaxation of this assumption for a heterogeneous fleet and allowing vehicles to perform several trips per day, results in a heterogeneous fleet multi-trip VRP (HFMTVRP).

Multi-trip VRP (MTVRP)'s relevance to city logistics has captured the attention of researchers, especially in the last decades. However, to the best of our knowledge, all currently available exact methods are for problems with a single depot and a homogeneous fleet. A single depot setting requires a depot with a large capacity to satisfy the customers' demand, and the depots with large capacities are typically located outside the city. As small vehicles visit the depot several times per day and have lower speed limits, time spent driving from and to a depot increases travel time more compared to the case of larger vehicles and single trips. Avoiding these long intermediate trips to a depot is crucial for delivery companies as the last-mile costs constitute between 13% and 75% of total logistic costs (Gevaers et al., 2009), and last-mile deliveries are subject to tight deadlines to maintain the customers' satisfaction. Therefore, companies use several micro-depots located in urban centers to replenish the vehicles (Boysen, Fedtke, and Schwerdfeger, 2020). For example, DHL, one of the most experienced companies in using electric vehicles, has already implemented a pilot project in Utrecht and Frankfurt to reduce the effect of multi-trips on the travel cost. In this project, special containers designed for cargo bikes are loaded onto the customized trailers at the main depot. The company's vans bring these trailers to the city center, where each container is loaded onto a cargo bike for the last-mile deliveries. Cargo bikes use these trailers to reload during the day and return the empty containers at the end of the day.

Motivated by these exciting changes, we study a heterogeneous fleet multi-depot MTVRP with time windows. We plan delivery routes for companies that own multiple micro-depots and a heterogeneous fleet of small and large vehicles and that operate in a city where the directness of roads are different for bikes and cars in certain areas. The routes should be planned so that the demand of each customer is delivered within its time window by one vehicle. We consider different travel times for different types of vehicles to capture the effect of the physical city structures and the compatibility of vehicle types to these structures. Vehicles have capacities and their working time is limited by the duration of a workday. We assume that vehicles are unlimited in number and can be used by incurring a fixed cost for a workday. We define the capacities of the depots in terms of containers. This definition of depot capacities is inspired by real-life applications and makes the problem easier to model. To avoid long trips back to a depot, we allow vehicles to start and end their workdays at different depots. However, to be in line with the multi-depot literature, we impose that the number of vehicles that return to a depot at the end of the workday is the same as the number of those that started their workday at this depot. This assumption allows the same schedule to be used the next day if needed and ensures that the depots have enough capacity for the empty containers. With the increasing number of companies integrating small vehicles and micro-depots into their operations and the cities investing in bike-friendly infrastructures, we believe that this is a comprehensive and realistic setting for the MTVRP. Even though the resulting problem is very complex, it is worth the effort since the routes produced can be rather different from the routes of the problem with a homogeneous fleet. An example based on instance C202 with 30 customers and two depots derived from Solomon (1987) is given in Figure 1. This figure depicts the optimal solutions with a homogeneous fleet of large vehicles, of small vehicles and a heterogeneous fleet. The circles and rectangles represent the customers and depots, respectively. The green area is the urban center and small vehicles can use the short-cuts to visit the nodes in this area. The dashed lines represent the workdays of large vehicles, while the solid lines represent the workdays of small vehicles. The number of vehicles in the optimal solution is the same for each problem. The optimal value of the first problem is 378, whereas it reduces to 357 when small vehicles are used instead of large ones, and it further reduces to 345 when both vehicle types are used. Comparing the workdays of each vehicle type in the homogeneous and the heterogeneous fleet cases, we can also see how small and large vehicles shape their workdays according to each other in the heterogeneous fleet case.



(a) homogeneous fleet of large vehicles **(b) homogeneous fleet of small vehicles** **(c) heterogeneous fleet**

Figure 1 The optimal solution of C202 instance with 30 customers and two depots for different variants

In this study, we formulate the heterogeneous fleet multi-depot multi-trip VRP with time windows using workday (sequence of trips) based variables and propose a branch and price algorithm.

In addition to introducing a new problem and devising an exact solution approach, we make the following methodological contributions:

- We use the iterative nature of the state-space augmenting algorithm to compute completion bounds without solving a separate relaxation. We do this by switching between forward and backward extensions in consecutive iterations.
- Although accelerating the pricing problem with heuristic methods has been commonly used, we contribute to the literature by introducing a new pricing heuristic based on the reduction in the graph size using reduced costs.
- We show that the multi-trip feature might lead the labelling algorithm to produce columns that cannot be a part of an optimal solution, and the existing dominance rules do not detect these columns. We show how to detect such columns and provide computational evidence on the effectiveness of this approach.

The remainder of the paper is organized as follows. In Section 2, we review the literature. Section 3 provides a formal definition of the heterogeneous fleet multi-depot MTVRP with time windows and introduces our formulation with workday-based variables. We describe the proposed branch and price algorithm in Section 4. In Section 5, we present the results of our computational experiments. Finally, we conclude in Section 6.

2. Literature Review

Since there is a vast body of literature on the multi-depot and the heterogeneous fleet extensions of VRP, we refer the reader to the comprehensive surveys of Montoya-Torres et al. (2015); Koç et al.

(2016) and Costa, Contardo, and Desaulniers (2019). Among the studies covered in these surveys, the most relevant one to our work is Bettinelli, Ceselli, and Righini (2011), where the authors propose a branch-and-cut-and-price algorithm for the heterogeneous fleet multi-depot VRPTW. They assume that the number of vehicles of each type is limited and allow vehicles to be at any depot at the beginning of the workday. They define the capacities of the depots in terms of the number of available drivers who can use any type of vehicle. Therefore, the number of trips assigned to a depot is limited, as in our problem. However, in their problem setting, each vehicle type requires one unit of the depot's capacity. The authors assume that the travel times are the same for each vehicle type and impose that each vehicle must finish its workday at its starting depot. They examine the impacts of different pricing and cutting techniques for this problem. To the best of our knowledge, this is the only exact algorithm proposed up to now for the heterogeneous fleet multi-depot VRPTW.

The rest of this section focuses on the exact methods on MTVRPs with time windows (MTVRPTW). Different extensions of MTVRP that have been addressed in the literature and the proposed methods can be found in the extensive survey of Cattaruzza, Absi, and Feillet (2016).

The extension of MTVRPTW with limited trip duration has been studied by several authors. The presence of time windows, low vehicle capacity and limited trip duration allow visiting only a few customers in each trip. This makes the enumeration of all feasible trips possible for small and medium size instances. Exact methods proposed by Azi, Gendreau, and Potvin (2010), Macedo et al. (2011) and Hernandez et al. (2014) use this observation and solve the problem in two phases. In the first phase, they generate all feasible non-dominated trips and in the second phase, they convert the trips generated in the first phase into workdays. Algorithms differ in the methods used in the second phase. Both Azi, Gendreau, and Potvin (2010) and Macedo et al. (2011) consider the problem with profits where visiting all customers are not mandatory. Whereas Azi, Gendreau, and Potvin (2010) apply a branch and price algorithm in the second phase, Macedo et al. (2011) solve a minimum cost flow model. Although the latter has a drawback because of the need of time discretization, it outperforms the former algorithm. Different from these two studies, Hernandez et al. (2014) consider the problem where all customers have to be served. In the second phase, they propose a set covering formulation with side constraints where columns represent trips with a fixed schedule and apply a branch and price algorithm. The resulting pricing problem has a pseudo-polynomial complexity. Their computational results show that this algorithm performs significantly better than the one proposed in Azi, Gendreau, and Potvin (2010) and obtains comparable results with the one proposed in Macedo et al. (2011). Which algorithm performs better depends much on the type of instance.

Later, Hernandez et al. (2016) consider the MTVRPTW without the limit on the duration of trips, which prevents the enumeration of all feasible trips. They propose two different set covering formulations, first based on workdays and the second based on trips and apply a branch and price algorithm. The pricing problem corresponds to an elementary shortest path problem with resource constraints (ESPPRC) for both formulations. Their computational results show that both algorithms outperform the two-phase algorithms mentioned above, and the formulation with trips globally outperforms the one with workdays.

Table 1 A classification of the studies in the literature

	time windows	profit	multi depot	heterogeneous fleet	multi trip	limited trip duration	variable type	enumeration of all feasible columns
Azi, Gendreau, and Potvin (2010)	✓	✓	✗	✗	✓	✓	workday	✓
Bettinelli, Ceselli, and Righini (2011)	✓	✗	✓	✓	✗	✓	trip	✗
Macedo et al. (2011)	✓	✓	✗	✗	✓	✓	trip	✓
Hernandez et al. (2014)	✓	✗	✗	✗	✓	✓	trip	✓
Hernandez et al. (2016)	✓	✗	✗	✗	✓	✗	workday trip	✗
Paradiso et al. (2020)	✓	✗	✗	✗	✓	✓	structure	✗

In a recent study, Paradiso et al. (2020) propose a set partitioning formulation with side constraints using structure based variables and a column generation and cutting plane based exact algorithm. They show that with small modifications, the proposed solution framework can be used to solve the MTVRPTW with loading times, with limited trip duration, with release dates and the drone routing problem. Their computational results show that this algorithm can solve more instances and is on average seven times faster than the one proposed in Hernandez et al. (2016).

In Table 1, we provide the classification of the studies mentioned above. As can be seen from Table 1, neither the multi-depot nor the heterogeneous fleet extensions are considered along with the multi-trip feature. In this study, we aim to fill this gap in the literature.

3. Problem Definition and Formulation

We first introduce the notation and give a definition of our problem. The set of nodes is represented by $N = I \cup J$ where $I = \{1, \ldots, m\}$ is the set of depots and $J = \{m+1, \ldots, m+n\}$ is the set of customers. The set of different types of vehicles is denoted by V. For each type $v \in V$, there are unlimited number of vehicles with capacity Q^v and a fixed cost f^v at each depot. Each node i has a known demand represented by q_i, service time represented by s_i (demands and service times of the depots are zero) and must be visited within a specified time interval $[e_i, l_i]$. Vehicles can neither depart from nor arrive at a depot outside of its time window. If the vehicle arrives at customer j

earlier than e_j, it has to wait until time e_j. Arrivals later than l_j are not allowed. For each $j \in J$, the set of vehicles that can serve customer j is represented by $V_j = \{v \in V \mid q_j \leq Q^v\}$. Each vehicle is allowed to perform several trips and visit different depots in-between to reload. Vehicle trips can start and end at different depots; however, for each vehicle type, the number of vehicles leaving a depot at the beginning of the workday must be equal to that of those returning to it at the end of the workday. Each depot i has a given capacity of h_i, and vehicle type $v \in V$ consumes a_i^v units of its capacity.

The aim is to find the sets of trips both starting and ending at depots with minimum cost. The set of trips should also satisfy that each customer is visited exactly once, total demand of the customers in each trip does not exceed the vehicle capacity, and total capacity consumption of vehicles assigned to a depot does not exceed its capacity.

To replace multi trips of a vehicle with a single trip, we use replenishment arcs proposed in Boland, Clarke, and Nemhauser (2000); if the last visited node before going to depot $i \in I$ is node $j \in J$ and the next trip starts at node $k \in J$, arcs (j,i) and (i,k) can be replaced by arc (i,j,k). As small vehicles can use short-cuts that are not compatible with large vehicles, different vehicle types have access to different arcs between the same pairs of customers. The time spent by vehicle type $v \in V$ to visit customer $k \in J$ right after customer $j \in J$ using a normal arc is denoted by τ_{jk}^v and using a replenishment arc that traverses depot $i \in I$ by ρ_{ijk}^v. We assume that the travel times satisfy the triangle inequality. For each vehicle type $v \in V$, $A^v = \{(i,j) \mid i,j \in N, i \neq j \text{ such that } l_j - e_i - s_i \geq \tau_{ij}^v,\ q_i + q_j \leq Q^v\} \setminus \{(i,j) \mid i,j \in I\}$ represents the set of arcs and $R^v = \{(i,j,k) \mid i \in I, j,k \in J, j \neq k \text{ such that } l_k - e_j - s_j \geq \rho_{ijk}^v,\ q_j \leq Q^v,\ q_k \leq Q^v\}$ represents the set of replenishment arcs that can be used by this vehicle type. The cost of using arc $(i,j) \in A^v$ is c_{ij}^v, and the cost of using replenishment arc $(i,j,k) \in R^v$ is $r_{ijk}^v = c_{ji}^v + c_{ik}^v$ for vehicle type $v \in V$.

We define workday p as an ordered list of nodes that starts and ends at a depot and can start within a time interval $[e_p, l_p]$ such that the total travel time, which is the minimum time required to visit the given nodes in the given order, remains the same for each start time within this interval. We call a workday feasible if it satisfies the time window restrictions and vehicle capacity constraints. As different vehicle types have different capacities and can use different arcs, we define the set of all feasible workdays separately for each vehicle type v as P^v, the set of all feasible workdays that start at depot s and end at depot t as P_{st}^v with $P^v = \cup_{s,t \in I} P_{st}^v$, the set of customers visited in workday $p \in P^v$ as J_p, the set of arcs and replenishment arcs used to visit the nodes using the order of $p \in P^v$ as A_p^v and R_p^v, respectively, and the cost of workday $p \in P^v$ as $\Delta_p^v = \sum_{(i,j) \in A_p^v} c_{ij}^v + \sum_{(i,j,k) \in R_p^v} r_{ijk}^v$. The variable z_p^v takes value 1 if workday $p \in P^v$ is used and 0 otherwise. Using these variables, our problem can be formulated as follows:

$$\min \sum_{v \in V} \sum_{p \in P^v} (f^v + \Delta_p^v) z_p^v \tag{1}$$

$$\text{s.t} \sum_{v \in V_j} \sum_{p \in P^v : j \in J_p} z_p^v = 1 \qquad \forall j \in J, \tag{2}$$

$$\sum_{v \in V} \sum_{p \in P^v} a_i^v \left(\sum_{j:(i,j) \in A_p^v} z_p^v + \sum_{j,k:(i,j,k) \in R_p^v} z_p^v \right) \leq h_i \qquad \forall i \in I, \tag{3}$$

$$\sum_{j \in I} \sum_{p \in P_{ij}^v} z_p^v = \sum_{j \in I} \sum_{p \in P_{ji}^v} z_p^v \qquad \forall i \in I, v \in V \tag{4}$$

$$z_p^v \in \{0,1\} \qquad \forall v \in V, p \in P^v. \tag{5}$$

The objective function (1) minimizes the total routing cost. Constraints (2) impose that each customer is visited exactly once. Constraints (3) ensure that the total capacity consumption of vehicles assigned to a depot does not exceed its capacity. Constraints (4) are vehicle balance equations for depots, and constraints (5) are variable restrictions.

4. Branch and Price Algorithm

As formulation (1) - (5) contains an exponential number of variables, we propose a branch and price algorithm to solve it. At each node of the branch and bound tree, we first check whether the node is pruned by bound using the lower bound of its parent node. If not, we apply column generation to solve the LP relaxation. When column generation terminates, we check whether the node is infeasible or pruned by bound. If not, either the node is pruned by optimality if the current solution is integer or else branching is applied. In the following, we explain how we start, how we generate columns, the techniques we apply to accelerate the solution of the pricing problem as well as the used branching scheme.

4.1. Initialization

4.1.1. Preprocessing Since the travel times satisfy the triangle inequality, the time windows can be tightened by modifying the procedure described in Dash et al. (2012), as shown below. We apply this procedure to the graph of each vehicle type $v \in V$ separately. As the arc and replenishment arc sets are different for different types of vehicles, the resulting time windows might also be different. Therefore, for each node i, we define e_i^v and l_i^v as the earliest and latest arrival time at node i by vehicle type v, respectively. We set $e_i^v = e_i$ and $l_i^v = l_i$ for each $i \in N$ and $v \in V$.

Step 1. $e_j^v \leftarrow \max\Big\{e_j^v, \min\big\{ \min\limits_{k:(k,j) \in A^v} \{e_k^v + s_k + \tau_{kj}^v\}, \min\limits_{\substack{i,k:\\(i,k,j) \in R^v}} \{e_k^v + s_k + \rho_{ikj}^v\}\big\}\Big\} \quad \forall j \in J.$

Step 2. $e_j^v \leftarrow \max\Big\{e_j^v, \min\big\{l_j^v, \min\{ \min\limits_{\substack{k \in J:\\(j,k) \in A^v}} \{e_k^v - s_j - \tau_{jk}^v\}, \min\limits_{\substack{i,k:\\(i,j,k) \in R^v}} \{e_k^v - s_j - \rho_{ijk}^v\}\}\big\}\Big\} \; \forall j \in J.$

Step 3. $l_j^v \leftarrow \min\Big\{l_j^v, \max\big\{e_j^v, \max\{ \max\limits_{\substack{k \in J:\\(k,j) \in A^v}} \{l_k^v + s_k + \tau_{kj}^v\}, \max\limits_{\substack{i,k:\\(i,k,j) \in R^v}} \{l_k^v + s_k + \rho_{ikj}^v\}\}\big\}\Big\} \; \forall j \in J.$

Step 4. $l_j^v \leftarrow \min\Big\{l_j^v, \max\big\{ \max\limits_{k:(j,k) \in A^v} \{l_k^v - s_j - \tau_{jk}^v\}, \max\limits_{\substack{i,k:\\(i,j,k) \in R^v}} \{l_k^v - s_j - \rho_{ijk}^v\}\big\}\Big\} \; \forall j \in J.$

In *Step* 1, if the earliest possible arrival time at a customer is greater than its time window start, the start of the time window is set to the earliest possible arrival time. If arriving at a customer later than its time window start does not lead to waiting at its successors, *Step* 2 sets the start of the time window to the latest such arrival time. If the latest possible arrival time at a customer is less than its time window end, *Step* 3 decreases the end of the time window to the latest possible arrival time. *Step* 4 sets the time window end of a customer to the latest possible arrival time that satisfies all time window constraints of its successors. We apply these steps iteratively until no more changes are possible.

4.1.2. Initial Solution We use a greedy algorithm to produce a starting solution to the initial rmp at the root node. If the algorithm fails to produce a feasible solution, it returns a set of feasible and artificial columns formed in a way to satisfy the feasibility conditions of a column as much as possible.

We divide the initial solution algorithm into two parts: clustering and scheduling. First, we cluster the customers based on their coordinates. Then we schedule the visits of customers in the same cluster using a time metric to form a trip from each cluster. This two-step approach aims to take both distance and time into account and reduce the search space in the scheduling part by clustering the customers first.

In the first part, we use the k-means clustering algorithm (Lloyd, 1982). As we consider the depots with a limited capacity, we define the number of clusters as the maximum number of trips: $\sum_{i\in I} h_i$. In the second part, we build the tours using a time-oriented nearest-neighbour heuristic.

4.2. The Pricing Problem

Let $\overline{P} \subseteq P = \cup_{v\in V} P^v$ be a subset of workdays for which the restricted master problem (rmp, the LP relaxation of formulation (1)-(5) when $z_p^v = 0$ for all $v \in V$ and $p \in P^v \setminus \overline{P}$) is feasible. We drop $z_p^v \leq 1$ for all for all $v \in V$ and $p \in P^v$ as these are implied by the set of constraints (2). The pricing problem decomposes for each vehicle type and checks whether there exists a workday performed by this vehicle type with a negative reduced cost. Let $N^v = \{i \in N | q_i \leq Q^v\}$ represent the set of nodes that can be visited by vehicle type $v \in V$. For each vehicle type $v \in V$, we introduce graph $G'^v = (N'^v, A'^v \cup R^v)$ where $N'^v = N^v \cup \{o, d\}$ is the set of nodes, $A'^v = \{(o,i)|i \in I\} \cup \{(i,d)|i \in I\} \cup A^v$ is the set of arcs and R^v is the set of replenishment arcs. Nodes o and d are dummy nodes that represent the origin and destination of workdays. If we associate dual variable vectors α, λ and θ to the set of the constraints (2), (3) and (4), respectively, the pricing problem, for each $v \in V$, seeks to find a workday $p \in P^v$ such that

$$\sum_{(i,j)\in A_p^v} w_{ij}^v + \sum_{(i,j,k)\in R_p^v} u_{ijk}^v < 0$$

where

$$w_{ij}^v = \begin{cases} f^v - \theta_j^v + a_i^v \lambda_j, & \text{if } i = o \\ \theta_i^v, & \text{if } j = d \\ c_{ij}^v, & \text{if } i \neq o,\ j \in I \\ c_{ij}^v - \alpha_j, & \text{otherwise} \end{cases} \qquad \forall (i,j) \in A'^v$$

$$u_{ijk}^v = r_{ijk}^v - \alpha_k + a_i^v \lambda_i \qquad \forall (i,j,k) \in R^v$$

For each vehicle type $v \in V$, the resulting pricing problem is an ESPPRC where w_{ij}^v and u_{ijk}^v are the costs of arcs $(i,j) \in A'^v$ and $(i,j,k) \in R^v$, respectively. To simplify the notation, we drop vehicle index v in the following sections.

We solve ESPPRC using the state-space augmenting algorithm proposed by Boland, Dethridge, and Dumitrescu (2006). This is a label setting method that starts without the elementarity constraints and iteratively includes them for a subset of nodes based on the information obtained at the previous iteration until the relaxed problem yields an elementary optimal solution. The labelling algorithm can be performed by extending the labels from the origin to the destination or vice versa. What follows next is a detailed explanation of the label setting algorithm for both extension methods and how changing the direction of the extension at different iterations can be used to obtain completion bounds without requiring to solve a separate relaxation.

4.2.1. Forward Labelling Each forward label represents a path from the origin to the node it is associated with and stores the end node, cost and resource consumption. Forward labels are iteratively extended from the origin to the destination. Extending a forward label associated with node i to j corresponds to adding arc (i,j) to the path from the origin to i. In order to decide the order in which labels are extended, we sort them breaking ties first by their duration, then cost, and then capacity consumption. We apply the dominance test to the labels associated with the same node and discard the dominated ones. A label is dominated only if there exists another, not more costly label for which all feasible extensions of the dominated one are also feasible. This iterative procedure continues until all non-dominated labels have been extended.

Feillet et al. (2004) indicate that keeping track of nodes that cannot be in any extension of the label because of the resource constraints enables to identify larger numbers of dominated labels. These nodes are called *unreachable* and the resources used to identify them must satisfy the triangle inequality. If an *unreachable* node is found, dominance rules are applied as if the label has visited that node which is similar to the idea of using the same node resource for the nodes that cannot be on the same path, as proposed by Kohl (1995). We use this observation in the label generation procedure by taking time windows as the resource constraint to identify the *unreachable* nodes.

Notice that as we allow visits to the depots in between trips, the consumption of capacity resource does not satisfy the triangle inequality, consequently cannot be used to identify such nodes.

Let $\bar{r}$ be a forward path starting at node o and ending at node $i_{\bar{r}}$ at time $t_{\bar{r}}$ such that every customer in the path is visited within its time window and capacity constraints are satisfied, and $\mathcal{S}$ be the set of *critical nodes* that are not allowed to be visited more than once. We represent every forward path with a label $L_{\bar{r}} = (\tilde{c}_{\bar{r}}, \tilde{q}_{\bar{r}}, i_{\bar{r}}, t_{\bar{r}}, S_{\bar{r}})$ where $\tilde{c}_{\bar{r}}$ is the reduced cost, $\tilde{q}_{\bar{r}}$ is the total demand of the nodes visited after the last visit to a depot and $S_{\bar{r}}$ is a binary *node-visit* resource vector of size $|\mathcal{S}|$ with $S^i_{\bar{r}} = 1$ if the i^{th} node in $\mathcal{S}$ has already been visited or identified as *unreachable*, 0 otherwise for $i \in \{1, \dots |\mathcal{S}|\}$. To extend *node-visit* resources, we define a vector b_j of size $|\mathcal{S}|$ for each node $j \in N$: $b^i_j = 1$ if node j is the i^{th} node in $\mathcal{S}$, 0 otherwise ($b_i = \vec{0}$, $\forall i \in I$).

To generate such paths, we initialize the dynamic programming algorithm with a label for each depot $i \in I$ defined as $\tilde{c}_{\bar{r}_i} = w_{oi}$, $\tilde{q}_{\bar{r}_i} = 0$, $i_{\bar{r}_i} = i$, $t_{\bar{r}_i} = e_i$, and $S_{\bar{r}_i} = \vec{0}$. We distinguish the extension rules first according to two main cases: if the label is associated with a customer or with a depot. If it is associated with a customer, we define different rules for the extensions via a normal arc and a replenishment arc. We do not allow vehicles to traverse a depot using a normal arc in-between their trips as we use replenishment arcs for this purpose. Therefore, we divide the extension rules of the labels associated with a depot into two: if it is the beginning of the workday (the first iteration), it must be extended to a customer; if not, to the destination node. For the sake of clarity, in the following, we explain the extension procedure at the first and subsequent iterations separately.

The First Iteration: Let $L_{\bar{r}'}$ represent the label generated by extending $L_{\bar{r}}$ with $i_{\bar{r}} = i \in I$ (initial labels) towards node $j \in J$ such that $(i,j) \in A$. It is constructed as follows:

- $\tilde{c}_{\bar{r}'} = \tilde{c}_{\bar{r}} + w_{ij}$;
- $\tilde{q}_{\bar{r}'} = \tilde{q}_{\bar{r}} + q_j$;
- $i_{\bar{r}'} = j$;
- $t_{\bar{r}'} = \max\{t_{\bar{r}} + s_i + \tau_{ij}, e_j\}$
- $S_{\bar{r}'} = S_{\bar{r}} + b_j$ and update the reachability of other *critical nodes*.

Next Iterations:

- If $j \in J$, $k \in N$ such that $(j,k) \in A$ and $S^\top_{\bar{r}} b_k = 0$. $L_{\bar{r}'}$ is constructed as follows:
  - $\tilde{c}_{\bar{r}'} = \tilde{c}_{\bar{r}} + w_{jk}$;
  - $\tilde{q}_{\bar{r}'} = \tilde{q}_{\bar{r}} + q_k$;
  - $i_{\bar{r}'} = k$;
  - $t_{\bar{r}'} = \max\{t_{\bar{r}} + s_j + \tau_{jk}, e_k\}$
  - $S_{\bar{r}'} = S_{\bar{r}} + b_k$ and update the reachability of other *critical nodes* if $k \in J$, $S_{\bar{r}'} = S_{\bar{r}}$ otherwise.
- If $j \in J$ and $k \in J$ such that $S^\top_{\bar{r}} b_k = 0$, different labels are constructed for each replenishment arc $(i,j,k) \in R$ as follows:

- $\tilde{c}_{\bar{r}'} = \tilde{c}_{\bar{r}} + u_{ijk}$;
- $\tilde{q}_{\bar{r}'} = q_k$;
- $i_{\bar{r}'} = k$;
- $t_{\bar{r}'} = \max\{t_{\bar{r}} + s_j + \rho_{ijk}, e_k\}$;
- $S_{\bar{r}'} = S_{\bar{r}} + b_k$ and update the reachability of other *critical nodes*.

Note that for the problem with uncapacitated depots, if the costs and travel times of the arcs are equal, generating a label only for the replenishment arc with the shortest travel time suffices. However, due to u_{ijk} variables, it is not possible to establish a direct dominance rule for the replenishment arcs between the same pair of customers when depots' capacities are considered.

- If $j \in I$, it is only allowed to be extended to the destination node as follows: $\tilde{c}_{\bar{r}'} = \tilde{c}_{\bar{r}} + w_{jd}$; $\tilde{q}_{\bar{r}'} = \tilde{q}_{\bar{r}}$; $i_{\bar{r}'} = d$; $t_{\bar{r}'} = t_{\bar{r}}$; $S_{\bar{r}'} = S_{\bar{r}}$.

Due to the time window and vehicle capacity constraints, a forward label $L_{\bar{r}}$ with $i_{\bar{r}} = j$ is feasible only if $t_{\bar{r}} \leq l_j$ and $\tilde{q}_{\bar{r}} \leq Q$.

To reduce the number of forward labels generated, we apply the following dominance rule: Let $L_{\bar{r}} = (\tilde{c}_{\bar{r}}, \tilde{q}_{\bar{r}}, i_{\bar{r}}, t_{\bar{r}}, S_{\bar{r}})$ and $L_{\bar{r}'} = (\tilde{c}_{\bar{r}'}, \tilde{q}_{\bar{r}'}, i_{\bar{r}'}, t_{\bar{r}'}, S_{\bar{r}'})$ be two labels. $L_{\bar{r}}$ dominates $L_{\bar{r}'}$ if the following conditions are satisfied:

$$i_{\bar{r}} = i_{\bar{r}'},\ \tilde{q}_{\bar{r}} \leq \tilde{q}_{\bar{r}'},\ t_{\bar{r}} \leq t_{\bar{r}'},\ S_{\bar{r}} \leq S_{\bar{r}'} \text{ and } \tilde{c}_{\bar{r}} \leq \tilde{c}_{\bar{r}'}.$$

The first four conditions ensure that every feasible extension of $L_{\bar{r}'}$ is also feasible for $L_{\bar{r}}$ and would not result in higher cost due to the last condition. If all conditions hold with equality, we arbitrarily choose one of the labels.

As the above problem is identical to ESPPRC when the elementarity constraints are relaxed for the nodes in $J \setminus \mathcal{S}$, if an optimal path found at the end of label setting iteration is elementary, it is also optimal for ESPPRC. Otherwise, the *node-visit* resources are redefined, and algorithm is restarted with the updated resources until an elementary optimal path is found. As when $\mathcal{S} = J$, this is just ESPPRC, it is guaranteed that the state-space augmenting algorithm detects negative cost paths if any exists.

In our implementation, we solve the pricing problem to optimality and add all generated columns with a negative reduced cost to the restricted master problem. If the algorithm fails to find an elementary optimal path at the end of an iteration, we define the *node-visit* resource for the node with the highest multiplicity in an optimal path.

4.2.2. Backward Labelling Labelling algorithm can also be performed in the reverse way by extending the labels from the destination to the origin. In this case, each backward label represents a path from the node it is associated with to the destination, and extending a backward label associated with node i to j corresponds to adding arc (j, i) to the path from i to the destination.

For backward labels, first we calculate the latest feasible arrival time to each depot $i \in I : T_i = \min\{l_i, \max_{j:(j,i)\in A}\{l_j + s_j + \tau_{ji}\}\}$. We initialize the algorithm with a backward label for each depot $i \in I$ defined as $\tilde{c}_{\hat{r}_i} = w_{id}, \tilde{q}_{\hat{r}_i} = 0, i_{\hat{r}_i} = i, t_{\hat{r}_i} = T_i$ and $S_{\hat{r}_i} = \vec{0}$. The backward labels are extended in the same way as in the forward labels except the time update. Let $L_{\hat{r}'}$ represent the label generated by extending $L_{\hat{r}}$ with $i_{\hat{r}} = j \in J$ or $j \in I$ (if it is an initial label) towards node $k \in N$ such that $(k,j) \in A$ and $S_{\hat{r}}^{\top} b_k = 0$, then $t_{\hat{r}'} = \min\{t_{\hat{r}} - s_k - \tau_{kj}, l_k\}$. If the label is extended to node $k \in J$ through $(i,k,j) \in R$, $t_{\hat{r}'} = \min\{t_{\hat{r}} - s_k - \rho_{ikj}, l_k\}$. A backward label $L_{\hat{r}}$ with $i_{\hat{r}} = j$ is feasible only if $t_{\hat{r}} \geq e_j$ and $q_{\hat{r}} \leq Q$.

Dominance rule for the forwards labels is applied to the backwards labels with the change in the time condition as follows : $L_{\hat{r}}$ dominates $L_{\hat{r}'}$ if $i_{\hat{r}} = i_{\hat{r}'}$, $\tilde{q}_{\hat{r}} \leq \tilde{q}_{\hat{r}'}$, $t_{\hat{r}} \geq t_{\hat{r}'}$, $S_{\hat{r}} \leq S_{\hat{r}'}$ and $\tilde{c}_{\hat{r}} \leq \tilde{c}_{\hat{r}'}$.

4.2.3. Completion Bounds We complement the labelling algorithm with the completion bounds as in Baldacci, Mingozzi, and Roberti (2011) to identify a larger number of dominated labels. A completion bound corresponds to a lower bound on the completion cost of a path into a workday. These bounds are used to prove that some paths cannot be completed into a workday with a negative cost, hence can be eliminated. Since, if no elementary path with a negative cost is found, the state-space augmenting algorithm iteratively increases the cardinality of the set of nodes for which the *node-visit* resources are defined, every iteration provides a lower bound for the next iteration. Contardo and Martinelli (2014) use this structure of the algorithm to solve the multi-depot VRP. They use forward extension and compute the completion bounds of the labels at a given iteration from the labels generated at the previous iteration. They turn the partial paths obtained at the previous iteration backwards to complete the ones at the current iteration. However, this approach cannot be directly extended to the problems with time windows as reversing a path might yield infeasibility in this case. In the presence of time windows, a path represented by a forward label (path from the source to a node) can be completed by a path represented by a backward label (path from a node to the destination). Therefore, we use the forward labelling at one iteration and the backward labelling at the next so that the lower bound obtained at the previous iteration can be used as a completion bound at the next iteration. Pecin et al. (2017) also use the backward labelling to compute the completion bounds for the forward labels and vice versa. However, as they do not solve the pricing problem iteratively (as in the state-space augmenting algorithm), they solve a different relaxation only to compute the completion bounds. Our implementation eliminates the need to solve a different problem, as in Pecin et al. (2017), and can be used for a broader range of problems than the method of Contardo and Martinelli (2014).

We use the completion bounds only when no elementary negative cost path is found at the first iteration (otherwise the algorithm terminates). As stated before, the completion bounds for the forward labels are obtained by the backward labels generated at the previous iteration. A forward label $L_{\bar{r}}$ with $i_{\bar{r}} = i$ is eliminated if $\tilde{c}_{\bar{r}} + \tilde{c}_{\hat{r}} \geq 0$ where $L_{\hat{r}}$ is the backward label with the minimum reduced cost obtained at the previous iteration such that $i_{\hat{r}} = i$, $\tilde{q}_{\hat{r}} + \tilde{q}_{\bar{r}} - q_i \leq Q$ and $t_{\bar{r}} \leq t_{\hat{r}}$. Equivalently, a backward label $L_{\hat{r}}$ with $i_{\hat{r}} = i$ is eliminated if $\tilde{c}_{\hat{r}} + \tilde{c}_{\bar{r}} \geq 0$ where $L_{\bar{r}}$ is the forward label with the minimum reduced cost obtained at the previous iteration such that $i_{\bar{r}} = i$, $\tilde{q}_{\hat{r}} + \tilde{q}_{\bar{r}} - q_i \leq Q$ and $t_{\bar{r}} \leq t_{\hat{r}}$. One could obtain better completion bounds by taking the *node-visit* resources into account (by using the label that satisfies the above conditions and violates the consumption of the *node-visit* resources for the smallest number of nodes). However, our priori experiments showed that the time spent to find such a label increases the overall computation time.

4.2.4. Heuristic Pricing

Even with the implementation of the state-space augmenting algorithm, solving the pricing problem to optimality can still be quite time-consuming. Therefore, we use heuristic pricing algorithms to detect negative reduced cost columns and only resort to the exact method when heuristic algorithms fail. First, we construct a reduced graph based on the observation that the arcs with smaller reduced costs are more likely to be selected in the pricing algorithm. To do so, we select a pre-determined number of nodes whose corresponding dual variables take the highest values at each iteration. Consequently, the incoming arcs of these nodes are presumably the ones with smaller reduced costs. Then we construct a new graph by discarding the incoming arcs of these nodes and the arcs adjacent to the depots and setting the remaining arcs' costs to their reduced cost values. We solve the pricing problem over a reduced graph that contains the incoming arcs of the selected nodes, the arcs adjacent to the depots and the arcs in the minimum spanning forest of the new graph.

As identifying negative reduced cost columns, if any exists, gets difficult towards the end of column generation iterations, we select a smaller number of nodes at the earlier iterations, hence a graph of smaller size, and increase this number at the later iterations. This choice is also motivated by the following reasoning. Since the arcs in the reduced graph are selected based on the dual information, choosing a good subset of arcs highly depends on the quality of the dual estimation. Therefore, if there is an improvement in the objective function value of the restricted master problem, we increase the number of selected nodes to make better use of the dual information. However, we put an upper limit on the number of selected nodes to prevent the size of the reduced graph from becoming too large. Upon reaching this limit, we keep the number of selected nodes equal to it at the subsequent iterations.

Algorithm 1 Pricing Algorithm ($iter$)

```
1:  $truncated \leftarrow true$
2:  if $iter = 1$
3:      $n^{iter} \leftarrow n^0$                        ▷ $n^0$ : the initial value of the number of selected nodes
4:  else if $z_{LP}^{*iter} < z_{LP}^{*iter-1}$ and $n^{iter-1} < \bar{n}$    ▷ $\bar{n}$ : the upper bound on the number of selected nodes
5:      $n^{iter} \leftarrow n^{iter-1} + 1$
6:  else
7:      $n^{iter} \leftarrow n^{iter-1}$
8:  Select $n^{iter}$ nodes whose corresponding dual variables take the highest values and put them into set $\mathcal{S}$
9:  $A_s \leftarrow \cup_{i \in I} \delta(i) \bigcup \cup_{j \in \mathcal{S}} \delta^-(j)$    ▷ $\delta(i)$ : the set of adjacent arcs, $\delta^-(i)$ : the set of incoming arcs of node $i$
10: Let $A_f$ be the set of arcs of a minimum spanning forest in $graph(N, A \setminus A_s)$ where the costs of the arcs
    are set to their reduced cost values
11: $A_s \leftarrow A_s \cup A_f$
12: Apply the state-space augmenting algorithm on $graph(N, A_s)$
13: if no negative reduced cost column is found
14:     Apply the state-space augmenting algorithm on $graph(N, A)$
15:     if no negative reduced cost column is found
16:         $truncated \leftarrow false$
17:         Apply the state-space augmenting algorithm on $graph(N, A)$
```

Using the structure of this reduced graph, we define the set $\mathcal{S}$ of critical nodes as the nodes with the highest dual values (the ones whose incoming arcs are added to the reduced graph). We start the state-space augmenting algorithm with this set instead of starting it without any *node-visit* resource to reduce the number of label setting iterations. For further reduction in the solution time of the pricing problem, whenever a non-elementary negative cost path is found, we convert it into an elementary path by keeping only the first appearance of the nodes visited more than once. As the triangular inequality guarantees that skipping some nodes cannot increase the travel time, obtained path satisfies the time window constraints. If the reduced cost of this path is still negative and the used arcs comply with the branching history of the node, we add it to the list of returned paths.

To further accelerate the heuristic pricing, we store only a small number of labels at each node. If the number of labels at any node reaches this limit and a label with a smaller cost than at least one of the existing ones is found, we drop the label with the highest cost and store the new one. Although dropping some of the non-dominated labels might increase the number of pricing iterations, as it decreases the number of applied dominance tests, which is the trickiest part of the pricing procedure, it usually reduces the overall computation time. If the algorithm fails to find a negative cost path using the reduced graph, first we search the original graph again, storing the limited number of labels and run the exact pricing only if no negative cost path is generated during this search. We provide an outline of this procedure in Algorithm 1.

4.2.5. Column Elimination When vehicles are allowed to perform several trips, the pricing algorithm might generate columns whose two consecutive trips can be merged (the total demand of the customers visited in these two trips is smaller than or equal to the vehicle capacity). If such a column exists, and if the required arc to merge these two trips complies with the branching history of the node, a column where these trips are performed as one trip is also feasible and has a smaller or equal cost due to the triangle inequality. Therefore, a column whose two consecutive trips can be merged either cannot appear in an optimal solution or has an alternative. However, if the vehicle completes its workday right after such trips and this column has a negative reduced cost, the pricing algorithm adds this non-promising column to the rmp. To prevent this, before adding a column with a negative reduced cost to the rmp, we check whether two consecutive trips of this column can be merged. If so, we store it in a different list. If no elementary path with a negative reduced cost that is not contained in this list is found at the end of a pricing iteration, we check if this list contains any columns. Note that the pricing algorithm generates the non-promising column but eliminates the stronger one when the labels corresponding to these two columns have equal cost and time resource consumption. In this case, the label that visits the depot in-between dominates the one that does not since it has a smaller capacity consumption, and the rest of the domination conditions (cost, time and *node-visit* resource consumptions) hold with equality. If the list is not empty, we merge the trips that satisfy the above definition in the column with the minimum reduced cost and add only this column. If the required arc to merge such trips cannot be used, we add this column without any change.

4.3. Branching Scheme

If the optimal solution of the master problem is fractional, we apply hierarchical branching in which we prioritize the aggregated arc variables and branch on the individual arcs when all aggregated arc variables take integer values. We first branch on the normal arc variables and then on the replenishment arc variables at both aggregated and individual arc branching decisions.

First, if there exists an arc $(i,j) \in A = \cup_{v\in V} A^v$ with fractional x_{ij} value defined as $\sum_{\substack{v\in V:\\(i,j)\in A^v}} x_{ij}^v$ where $x_{ij}^v = \sum_{\substack{p\in P^v:\\(i,j)\in A_p^v}} z_p^v$, we branch in a way that (i,j) must be used in one branch and cannot be used in the other. In the first child node, if $i \in I$, we only inherit the columns that use arc (i,j) or do not visit node j from the parent node. To ensure that the new columns generated at each iteration comply with the branching decision, we modify the graph of each vehicle type by eliminating all incoming arcs of node j except for arc (i,j). Analogously, if $j \in I$, only the columns that use arc (i,j) or do not visit node i are passed to the child node and all outgoing arcs of node i except for arc (i,j) are eliminated from the graph of each vehicle type. If arc (i,j) is not adjacent to a depot, only the columns that use arc (i,j) or visit neither node i nor j are inherited from the parent node and all outgoing arcs of node i and all incoming arcs of node j except for arc (i,j) are eliminated from the graph of each vehicle type. This ensures that $x_{ij}=1$ in every feasible solution of the rmp of the resulting child. In the second child node, the columns that use arc (i,j) are not inherited from the parent node and arc (i,j) is eliminated from the graph of each vehicle type to ensure that no new column that uses this arc is generated. If all x_{ij} variables are integer, we check whether there exists a replenishment arc $(i,j,k)\in R=\cup_{v\in V}R^v$ with fractional y_{ijk} value defined as $\sum_{\substack{v\in V:\\(i,j,k)\in R^v}} y^v_{ijk}$ where $y^v_{ijk}=\sum_{\substack{p\in P^v:\\(i,j,k)\in R^v_p}} z^v_p$. If yes, we branch in the same fashion. In one branch, we only inherit the columns that use arc (i,j,k) or visit neither node j nor k and eliminate all outgoing arcs of node j and all incoming arcs of node k except for (i,j,k). In the other branch, we do not inherit any columns that use arc (i,j,k) and eliminate this arc from the graph of each vehicle type.

If the branching on the aggregated arc variables does not suffice for the integrality, we check whether there exists an arc $(i,j)\in A^v$ with fractional x^v_{ij} value for vehicle type v. If so, we set x^v_{ij} to 1 in the first child node. We filter the columns of vehicle type v and modify its arc set as defined above in the fixed arc decision. If $i\in I$ $(j\in I)$, we do not inherit any columns of other vehicle types that visit node j (i). If arc (i,j) is not adjacent to a depot, we filter the columns of other vehicle types that visit either node i or j. We also eliminate this arc from their arc sets. In the other child node, we set x^v_{ij} to 0. We filter the columns of vehicle type v and modify its arc set as defined above in the forbidden arc decision.

When all x^v_{ij} variables are integer, if there exists a replenishment arc $(i,j,k)\in R^v$ with fractional y^v_{ijk} value for vehicle type v, we branch on this arc. We apply the fixed and forbidden replenishment arc decisions for vehicle type v. In the branch where y^v_{ijk} is set to 1, we filter the columns of other vehicle types that visits either node j or k and eliminate this arc from their replenishment arc sets.

We select an arc with value closest to 0.5 and break the ties by selecting the most frequently used one.

4.4. Artificial Columns after Branching

After a branching decision is performed, the columns passed from the parent node are filtered as explained above. The columns that remain after filtering may not be sufficient to provide a feasible solution to the initial rmp of the corresponding child node, which surfaces the need for artificial columns. To provide the initial rmp at each node with a starting solution, we use the initial columns generated to start the initial rmp at the root node. In the rest of this section, we use initial columns to refer to the initial columns at the root node. Although the same set of artificial columns can be used at each node as the execution of the branching decision does not require modifying the master problem, we modify the set of initial columns considering the branching history of the node to reduce the *head-in* effect. At each node of the tree, we inherit all initial columns of the parent node. These columns can be artificial or non-artificial. Since the problem in the child node is a restriction of the problem in the parent node, we pass the artificial columns as they are. For each non-artificial initial column, we check whether it is still feasible after the current branching decision is performed. If it is, we inherit it without any change. Otherwise, we redefine the column as artificial and change its cost with a big value M. As each node has all initial columns, either as artificial or not, this procedure ensures that the initial rmp at each node has a starting solution.

5. Computational Experiments

In this section we report the results of our computational experiments where we investigate the computational effectiveness of our algorithm for both the homogeneous and heterogeneous fleet instances and the effect of using a heterogeneous fleet on the transportation costs. All experiments are carried out on a 64-bit machine with Intel Core i7 processor at 1.90 GHz and 16 GB of RAM using Java and CPLEX 12.8.

5.1. Computational Performance for the Homogeneous Fleet Multi-depot MTVRPTW

First, we conduct the computational experiments on the problem with the homogeneous fleet of small vehicles to investigate the effect of the proposed techniques on the computation time. The solution method described above can be used to solve the homogeneous fleet problem without any modifications since the heterogeneous fleet multi-depot MTVRPTW where V consists of only one vehicle type corresponds to the problem with a homogeneous fleet.

In our experiments, we use 216 instances derived from the type 2 instances proposed by Solomon (1987). These instances are commonly used in the MTVRP literature as the short planning horizon and tight time windows of type 1 instances do not allow vehicles to perform more than one trip. These instances consist of three groups; C, R and RC where customers are clustered, randomly located and half clustered half randomly located, respectively. The coordinates of the customers are the same for all instances in the same group but the time windows differ. We consider instances with 25, 30, 35 and 40 customers selecting the first customers from the Solomon instances and with 2 and 3 depots. As the depots in our content represent the micro-depots that can be placed in central areas such as parking lots or shopping malls, we divide the customers into as many clusters as the number of depots based on their coordinates and locate each depot in the center of a cluster. As the locations of customers are the same for the same group of instances so are the

Table 2 Comparison of exact and heuristic pricing for instances with 25 customers and two depots

	exact pricing					**heuristic pricing**				
inst.	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**
C201	0	7.31	13	339	13	0	7.79	31	605	28.1
C202	0	86.14	1	23	-	0	8.32	1	94	-
C203	NA	3600	0	4	NA	0	472.54	1	88	-
C204	NA	3600	0	3	NA	NA	3600	0	66	NA
C205	0	47.11	1	42	-	0	5.99	1	89	-
C206	0	124.42	1	31	-	0	14.94	1	143	-
C207	0	1126.55	1	22	-	0	27.11	1	140	-
C208	4.44	3600	88	1671	4.44	0	1206.88	541	10605	5.6
avg	0.74	1523.94	13.13	266.88	8.72	0	667.95	72.13	1478.75	16.85
solved					5					7
R201	0	21.34	15	81	9.73	0	4.72	9	112	2.92
R202	0	575.62	39	130	5.05	0	28.58	51	275	4.49
R203	NA	3600	0	3	NA	0	255.02	133	1298	0.84
R204	NA	3600	0	3	NA	0	856.68	99	714	7.33
R205	0	95.64	1	26	-	0	7.25	1	76	-
R206	8.19	3600	1	16	8.19	0	422.59	169	1745	4.03
R207	NA	3600	0	3	NA	0	115.42	1	81	-
R208	NA	3600	0	2	NA	0	3310.77	1	108	-
R209	0	1421.96	23	194	6.81	0	61.13	25	306	6.61
R210	0	853.65	3	35	0.43	0	34.56	3	104	7.06
R211	NA	3600	0	3	NA	11.14	3600	79	1288	11.14
avg	1.37	2233.47	7.45	45.09	6.04	1.01	790.61	51.91	555.18	5.55
solved					5					10
RC201	0	7.27	1	27	-	0	1.23	1	57	-
RC202	0	409.06	1	19	-	0	8.13	1	46	-
RC203	NA	3600	0	3	NA	0	15.2	1	58	-
RC204	NA	3600	0	2	NA	0	472.82	1	73	-
RC205	0	99.99	7	68	9.09	0	13.11	11	162	4.88
RC206	0	591.28	81	622	6.12	0	95.05	127	1647	12.59
RC207	0	474.49	1	19	-	0	11.98	1	73	-
RC208	NA	3600	0	2	NA	0	2160.36	1	93	-
avg	0	1547.76	11.38	95.25	7.61	0	347.24	18	276.13	2.18
solved					5					8

locations of the depots (for the instances with the same number of customers) whereas they differ among different groups. We assume that depots have the same capacity and set it to the nearest integer to $\frac{2}{3}\left\lceil\frac{\sum_{j\in J} q_j}{Q}\right\rceil$ for the instances with two depots and to $\frac{1}{2}\left\lceil\frac{\sum_{j\in J} q_j}{Q}\right\rceil$ for the instances with three depots. We set the time windows of the depots to the depot's time window provided in the instance. We consider vehicles of capacity 100 and fixed cost 35. We set the cost and travel times of an arc to the Euclidean distance between its endpoints rounded to the nearest integer. If we detect an arc violating the triangle inequality, we set its cost and time to the shortest distance between the corresponding nodes.

Based on some initial experiments, we choose the parameters for the heuristic pricing as follows: we start the pricing algorithm at each node by defining the *node-visit* resource for one-third of the nodes whose corresponding dual variables take the highest values (the incoming arcs of these nodes are used in the reduced graph). At each subsequent pricing iteration where the objective function value of the rmp improves, we increase the number of nodes for which the *node-visit* resource is defined by one until it is defined for half of the nodes. We store at most ten labels in truncated-search mode.

In our first experiment, we analyze the impact of the straightforward version of the proposed heuristic pricing algorithm on the computation time. We compare the exact pricing algorithm with the heuristic pricing algorithm where the labels are extended only forward, completion bounds are not used, and the columns that cannot be in an optimal solution are not eliminated. We present our results for instances with two depots and 25 customers in Table 2. We set the time limit to one hour. We report the gap between the best obtained upper and lower bounds for the instances that are not solved to optimality within the time limit, the solution time, the number of nodes processed and the number of pricing iterations performed. When the column generation terminates at the root node, we solve the integer problem that contains all generated columns to compute a better upper bound earlier in the tree. We report the optimality gap of this upper bound in the RootIP gap column. If an optimal solution is not obtained, we use the best obtained lower bound to calculate this gap. If the algorithm reaches the time limit before solving the root node to optimality, as a valid lower bound is not obtained, we use NA in the gap and RootIP gap columns. We use dash in the RootIP gap column to indicate the instances for which column generation returns an integer solution at the root node as in this case the algorithm terminates without solving the integer problem at the root node.

The results clearly demonstrate the benefit of using the heuristic pricing for each instance group. It reduces the computation time of C instances by more than a factor of two, of R instances by almost a factor of three and of RC instances by more than a factor of four. Overall, the algorithm with the heuristic pricing can solve 25 instances out of 27 in an average time of seven minutes, whereas it fails to solve 12 instances within the time limit when only exact pricing is used.

In our next experiment, we look at the effect of different extension methods, the completion bounds and the column elimination explained in Section 4.2.5. Since we report the results when the labels are extended forward from the source in Table 2, we only report the results when the labels are extended backward from the destination, when the completion bounds are used (forward extension at one iteration, backward at the next) and when the completion bounds are supplemented by the column elimination in Table 3.

The first observation is that the performances of forward and backward extensions depend highly on the instance group even though same resources and similar dominance rules are used. Whereas using the backward extension takes a shorter time to solve R instances and provides a better bound for R211 that cannot be solved optimally by both extension methods, it is outperformed for

Table 3 Comparison of different variants of label extension, completion bounds and column elimination for instances with 25 customers and two depots

	backward extension					completion bounds					+ column elimination				
inst.	gap (%)	time	node	pricing iter	RootIP gap(%)	gap (%)	time	node	pricing iter	RootIP gap(%)	gap (%)	time	node	pricing iter	RootIP gap(%)
C201	0	8.43	17	411	4.66	0	8.66	31	605	28.1	0	6.96	17	542	15.33
C202	0	15.99	1	126	-	0	7.17	1	69	-	0	7.71	1	73	-
C203	0	159.63	1	155	-	0	172.17	1	108	-	0	63.59	1	84	-
C204	NA	3600	0	72	NA	NA	3600	0	64	NA	NA	3600	0	72	NA
C205	0	12.13	1	161	-	0	7.33	1	114	-	0	4.42	1	83	-
C206	0	23.26	1	193	-	0	9.73	1	103	-	0	9.30	1	100	-
C207	0	34.56	1	146	-	0	15.41	1	141	-	0	15.63	1	132	-
C208	0	1781.83	577	11561	0.93	0	767.56	263	6976	0	0	689.25	245	6354	0
avg	0	704.48	74.88	1603.13	2.80	0	573.50	37.38	1022.5	14.05	0	549.61	33.38	930	7.67
solved					7					7					7
R201	0	5.93	7	99	0.53	0	2.81	5	72	1.91	0	3.24	15	117	0.53
R202	0	31.14	45	241	1.21	0	14.84	29	184	1.41	0	13.39	37	160	6.32
R203	0	670.34	231	2340	6	0	215.06	255	1878	5.81	0	142.94	117	1179	0.63
R204	0	894.96	39	329	2.57	0	295.50	121	903	11.65	0	150.45	45	428	0.48
R205	0	7.70	1	72	-	0	4.35	1	64	-	0	2.88	1	54	-
R206	0	657.04	239	1900	2.38	0	327.19	193	2258	5.44	0	261.04	179	2134	0
R207	0	260.83	1	87	-	0	32.12	1	73	-	0	14.11	1	73	-
R208	0	1549.40	1	168	-	0	671.83	1	87	-	0	343.25	1	109	-
R209	0	148.21	63	584	2.88	0	50.21	35	347	3.95	0	45.49	17	341	0
R210	0	35.49	3	89	2.95	0	17.61	3	86	1.91	0	15.62	3	93	2.12
R211	8.74	3600	66	932	8.74	0	1532.79	255	2527	7.83	0	1350.55	199	2826	5.07
avg	0.79	714.64	63.27	621.91	3.41	0	287.66	81.73	770.82	4.99	0	213	55.91	683.09	1.89
solved					10					11					11
RC201	0	1.07	1	47	-	0	0.89	1	38	-	0	0.72	1	43	-
RC202	0	7.80	1	58	-	0	4.28	1	50	-	0	5.46	1	60	-
RC203	0	19.11	1	54	-	0	12.93	1	52	-	0	11.24	1	64	-
RC204	NA	3600	0	61	NA	0	553.07	1	47	-	0	66.03	1	55	-
RC205	0	6.12	3	71	0	0	7.54	9	113	0.8	0	6.17	9	123	0.54
RC206	0	132.19	187	2265	0	0	70.53	211	1615	3.16	0	51.74	131	1303	1.34
RC207	0	11.28	1	59	-	0	6.01	1	64	-	0	4.08	1	45	-
RC208	0	1834.55	1	78	-	0	507.94	1	71	-	0	352.83	1	94	-
avg	0	701.52	24.38	336.63	0	0	145.4	28.25	256.25	1.98	0	62.28	18.25	223.38	0.94
solved					7					8					8

the RC group. The performance difference between the two methods is even more significant for distinct instances. While using the backward extension halves the solution time of R208 compared to the forward extension, it fails to solve RC204 which is solved in eight minutes using the forward extension. We believe that this might be caused by unevenly spread time windows and demands. We also observe that using the completion bounds globally outperforms both methods and solves all instances except C204 within one hour time limit. It reduces the average computation time for all instance groups, and its superiority is more evident for R and RC groups. It is more than two times faster for these groups than the best performed mono-directional method. Besides significantly reducing the computation time, as our implementation uses extensions in both ways, it abolishes the priori computations to decide the direction of the extension. Our final observation is that when the completion bounds are supplemented with the column elimination, better solution times are obtained for all instance groups, and 26 of the 27 instances are solved to optimality in less than three minutes on average.

Next, we investigate the impact of the reduced graph on the computation time using the final version of the heuristic pricing algorithm where both completion bounds and column elimination are used. We present the results of the heuristic pricing algorithm where the original graph is searched storing a limited number of labels (lines 9-13 of Algorithm 1 are skipped) for the same dataset in Table 4.

Table 4 Results of the truncated heuristic pricing with completion bounds and column elimination, without the reduced graph for instances with 25 customers and two depots

inst.	gap (%)	time	node	pricing iter	RootIP gap(%)	inst.	gap (%)	time	node	pricing iter	RootIP gap(%)	inst.	gap (%)	time	node	pricing iter	RootIP gap(%)
C201	0	3.63	9	181	1.97	R201	0	5.01	7	55	0.70	RC201	0	1.34	1	20	-
C202	0	16.07	1	40	-	R202	0	25.91	25	122	1.41	RC202	0	9.09	1	30	-
C203	0	60.48	1	42	-	R203	0	180.99	107	734	7.48	RC203	0	107.18	1	28	-
C204	NA	3600	0	18	NA	R204	0	341.03	35	315	1.18	RC204	0	292.56	1	34	-
C205	0	8.56	1	53	-	R205	0	7.75	1	24	-	RC205	0	18.91	13	112	0
C206	0	12.66	1	49	-	R206	0	337.04	173	1357	10.5	RC206	0	75.59	147	803	1.08
C207	0	28.48	1	69	-	R207	0	41.19	1	42	-	RC207	0	13.75	1	34	-
C208	0	1403.86	747	8251	10.36	R208	0	554.14	1	43	-	RC208	0	1444.51	1	30	-
						R209	0	56.64	17	165	6.01						
						R210	0	34.13	3	47	8.53						
						R211	0	1917.57	211	2583	1.44						
avg	0	641.72	95.12	1087.88	6.16		0	318.31	52.82	498.82	4.66		0	245.37	20.75	136.38	0.54
solved					7						11						8

The instances solved to optimality at the root node show that the total number of pricing iterations significantly decreases when the truncated search is performed directly on the original graph. However, the increase in the time per pricing problem deteriorates the overall performance of the algorithm. These results show that first searching the reduced graph is quite effective for all instance groups.

In our last experiment to test the performance of the proposed techniques, we measure the impact of these techniques on the computation time when a good initial upper bound is known. As the computation times of the instances solved to optimality at the root node, 14 of 27 instances, are not affected by the quality of the upper bound, we exclude these instances. We also exclude C204 since an integer solution could not be found for this instance. We use the optimal value as an initial upper bound for the 12 remaining instances. The average computation times of these instances are 1755.95, 523.16, 595.03, 236.59 and 180.31 for the exact pricing, heuristic pricing where the labels are extended only forward, heuristic pricing where the labels are extended only backward, when the completion bounds are used and when the post-processing is applied, respectively. The average computation times of these instances reported in Tables 2 and 3 (without a good initial upper bound) are 1797.60, 548.84, 664.31, 275.86 and 228.07 for the same order of techniques as above. These results show that starting the algorithm with a good upper bound shortens the computation time, and the efficacy of the proposed techniques remains valid.

As these experiments demonstrate the benefit of improvement ideas, using a reduced graph, eliminating the columns that cannot be in an optimal solution and using the forward extension at one iteration and the backward at the next complemented by the completion bounds, we use this version for larger instances. We set the time limit to two hours for the rest of the experiments. First, to see the effect of the number of customers on the computation time, we report our results for 30, 35 and 40 customers and two depots in Table 5.

Table 5 Results for instances with 30, 35 and 40 customers and two depots

	30 customers					**35 customers**					**40 customers**				
inst.	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**
C201	0	69.53	197	3677	10.65	0	7.66	1	183	-	0	570.35	155	10886	11.27
C202	0	62.36	45	834	8.46	0	21.32	1	121	-	0	130.15	9	355	0.2
C203	0	5217.38	147	1453	13.43	0.25	7200	28	768	10.34	0.33	7200	52	1988	7.13
C204	NA	7200	0	69	NA	NA	7200	0	125	NA	NA	7200	0	123	NA
C205	0	134.64	83	1707	13.71	0	17.87	1	137	-	0	101.41	3	377	0.4
C206	0	515.08	179	4252	0.55	0	22.29	1	118	-	0	501.37	41	1233	13.71
C207	0	2549.74	885	15842	0.27	0	52.38	1	137	-	0	3856.77	131	4308	1.23
C208	0	230.37	51	1581	0	0	39.81	1	119	-	0	2171.95	85	4300	3.39
avg	0	1997.39	198.38	3676.88	6.72	0.04	1820.17	4.25	213.50	10.34	0.05	2716.50	59.50	2946.25	5.33
R201	0	30.52	67	406	1.22	0	60.78	49	395	4.26	0	81.86	49	528	0
R202	0	33.58	27	220	0.71	0	142.38	23	278	9.16	0	318.57	53	628	2.16
R203	0	893.35	303	2981	10.98	0	301.59	15	177	15.95	0	2896.61	247	3062	6.78
R204	3.29	7200	14	304	3.29	NA	7200	0	96	NA	NA	7200	0	85	NA
R205	0	133.99	103	947	5.75	0	83.57	9	196	4.84	0	373.7	101	735	2.22
R206	0	135.5	27	366	7.65	0	388.2	19	357	9.15	5.11	7200	427	6662	5.11
R207	0	1536.58	5	173	2.75	9.87	7200	2	144	9.87	16.33	7200	18	383	16.33
R208	NA	7200	0	104	NA	NA	7200	0	111	NA	NA	7200	0	131	NA
R209	0	307.6	101	1023	5.34	0	1529.17	173	2512	7.46	0	2935.17	255	2444	8.64
R210	0	84.34	7	158	1.73	0	235.62	19	250	10.49	0	6321.22	847	5949	9.16
R211	14.81	7200	144	1786	14.81	12.07	7200	31	367	12.07	11.4	7200	11	248	11.4
avg	1.81	2250.50	72.55	769.82	5.42	2.44	2867.39	30.91	443.91	9.25	3.65	4447.92	182.55	1895.91	6.87
RC201	0	1.39	1	44	-	0	126.02	193	1547	0.54	0	5684.35	13023	73310	0.56
RC202	0	5.72	1	54	-	0.37	7200	3812	35359	4.24	7.99	7200	2533	23133	7.99
RC203	0	42.55	1	69	-	10.3	7200	28	294	10.3	12.86	7200	48	518	12.86
RC204	0	373.69	1	75	-	NA	7200	0	57	NA	NA	7200	0	61	NA
RC205	0	10.53	19	94	0.53	0	6481.84	8903	41654	0.55	8.22	7200	4157	24416	8.22
RC206	0	94.76	127	1198	0.89	2.94	7200	4340	30187	2.94	8.32	7200	3527	21978	8.32
RC207	0	13.05	1	59	-	2.25	7200	1140	16295	3.12	12.22	7200	684	8464	12.22
RC208	0	706.95	1	65	-	NA	7200	0	90	NA	NA	7200	0	50	NA
avg	0	156.08	19.00	207.25	0.71	2.64	6225.98	2302.00	15685.38	3.62	8.27	7010.54	2996.50	18991.25	8.36

We see that the difficulty of C204 instance remains valid with a larger number of customers. However, the algorithm is able to solve the rest of C instances with 30 customers and except C203 with 35 and 40 customers for which a high-quality solution is obtained. However, its performance significantly deteriorates for RC instances. While all instances with 30 customers in RC group are solved in less than three minutes on average, only two instances with 35 customers and one instance with 40 customers could be solved within two hours. Based on the results of instances with 30 and 35 customers, we also observe that the difficulty of an instance is not only affected by its size but also by the locations and the surplus capacity of depots. The results show that instances with 35

Table 6 Results for the instances with 25 and 30 customers and three depots

	25 customers					**30 customers**				
inst.	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**
C201	0	34.06	83	2021	17.52	0	30.87	7	890	0.32
C202	0	16.11	3	125	15.22	0	185.70	45	2161	0
C203	0	31.45	1	119	-	0	327.74	39	1649	0.33
C204	0	6103.67	17	303	5.05	13.55	7200	2	139	13.55
C205	0	5.74	1	84	-	0	14.19	1	109	-
C206	0	12.89	1	120	-	0	23.47	1	150	-
C207	0	33.14	3	117	0	0	98.47	5	239	4.15
C208	0	17.25	1	118	-	0	59.93	5	285	4.76
avg	0	781.79	13.75	375.88	9.45	1.69	992.55	13.13	702.75	3.85
R201	0	4.24	13	127	3.98	0	25.94	41	269	0.33
R202	0	16.68	15	152	5.28	0	100.43	53	353	2.19
R203	0	38.2	25	154	1.97	0	151.35	25	273	8.07
R204	0	60.23	7	149	0	0	4528.37	43	369	3.08
R205	0	50.46	51	631	2.99	0	132.98	73	521	6.22
R206	0	64.37	31	392	8.04	0	199.56	25	306	12.83
R207	0	18.98	1	97	-	0	1512.22	17	350	7.38
R208	0	1276.72	7	275	11.58	NA	7200	0	208	NA
R209	0	300.57	211	2479	9.45	0	171.77	27	485	0.2
R210	0	18.08	3	101	9.28	0	188.38	27	342	11.49
R211	0	1642.87	211	3535	10.91	0	3483.27	165	1982	11.61
avg	0	317.40	52.27	735.64	6.35	0	1608.57	45.09	496.18	6.34
RC201	0	0.8	1	37	-	0	21.35	33	256	6.08
RC202	0	2.03	1	35	-	0	2194.15	1389	12616	7.36
RC203	0	6.55	1	50	-	0.87	7200	1670	16867	8.96
RC204	0	8.08	1	53	-	NA	7200	0	30	NA
RC205	0	2.44	1	46	-	0	827.28	731	6337	1.03
RC206	0	1.93	1	40	-	2.20	7200	3487	33433	7.75
RC207	0	3.14	1	44	-	1.28	7200	1927	17860	6.25
RC208	0	125.89	3	136	0	NA	7200	0	62	NA
avg	0	18.86	1.25	55.13	0	0.73	4880.35	1154.63	10932.63	6.24

customers in C group except C203 are easier to solve than those with 30 customers as the optimal solution of the master problem at the root node is integer for these instances.

Next, we analyze the impact of the number of depots on the computation time and report our results for instances with 25 and 30 customers and three depots in Table 6 and with 35 and 40 customers in Table 7. The results show that the algorithm can solve all instances with 25 customers in an average computation time of six minutes, all but two instances with 30 customers in C and R groups in 11 minutes on average. However, only three instances with 30 customers in RC group are solved. It solves 19 of 27 instances in an average computation time of 28 minutes with 35 customers, 17 of 27 instances in 21 minutes on average with 40 customers, and provides high quality solutions for two instances in both cases. Another observation is that RC group exhibits a different behaviour than of those with two depots; instances with 35 and 40 customers are easier to solve compared to the ones with 30 customers.

Table 7 **Results for the instances with 35 and 40 customers and three depots**

	35 customers					**40 customers**				
inst.	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**
C201	0	124.32	15	2319	10	0	135.81	5	1229	1.07
C202	0	1012.24	111	6908	2.44	0	968.7	25	1769	0
C203	0.56	7200	72	3832	0.84	0.23	7200	7	695	1.12
C204	NA	7200	0	123	NA	NA	7200	0	134	NA
C205	0	57.38	1	185	-	0	1053.15	41	2605	6.95
C206	0	65.52	1	169	-	0	373.44	7	477	0
C207	0	4095.37	235	8993	10.83	9.44	7200	62	3937	9.44
C208	0	869	67	2349	10.58	0.56	7200	195	8326	0.78
avg	0.08	2577.98	62.75	3109.75	6.94	1.46	3916.39	42.75	2396.50	2.77
R201	0	187.64	145	1089	0.14	0	158.43	81	770	0.51
R202	0	557.23	161	989	7.39	0	449.08	57	453	6.85
R203	0	1841.4	101	1043	11.92	0	598.18	31	334	0.47
R204	NA	7.200	0	103	NA	NA	7200	0	95	NA
R205	0	2626.64	533	6041	10.88	0	177.75	13	159	8.41
R206	0	2400.06	329	2203	14.55	0	4394.54	119	1953	12.5
R207	0	2179.29	9	200	15.72	2.71	7200	23	514	2.71
R208	NA	7200	0	126	NA	NA	7200	0	96	NA
R209	0	5517.8	595	6995	15.67	0	1668.31	47	836	7.49
R210	0	1961.96	121	1536	15.32	0	2856.72	153	1262	6.99
R211	12.23	7200	65	921	12.23	16.11	7200	11	211	16.11
avg	1.36	3533.82	187.18	1931.45	11.54	2.09	3554.82	48.64	607.55	6.89
RC201	0	14.75	17	149	7.48	0	92.33	39	414	0.55
RC202	0.56	7200	3415	40959	4.5	0	150.54	11	170	0
RC203	0	7165.86	705	10312	10.67	0	709.99	11	264	4.42
RC204	17.73	7200	3	256	17.73	NA	7200	0	85	NA
RC205	0	413.79	223	2090	5.97	0	2474.54	571	5021	1.01
RC206	0	517.34	167	1600	2.51	0	2020.21	465	3723	1.90
RC207	0	39.62	1	90	-	0	2923.89	113	1470	5.52
RC208	2.65	7200	8	198	11.91	9.59	7200	3	151	9.59
avg	2.62	3718.92	567.38	6956.75	8.68	1.37	2846.44	151.63	1412.25	3.28

5.2. Computational Performance for the Heterogeneous Fleet Multi-depot MTVRPTW

We classify the types of vehicles based on three characteristics: cost, capacity and network. We use the same parameters as in the previous section for the small vehicles to analyze the effect of the integration of large vehicles. We consider the large vehicles of capacity 300 and fixed cost of 50.

Table 8 **The number of nodes in the center for each instance group**

group	25-2	30-2	35-2	40-2
C	7	9	12	11
R	3	7	6	9
RC	9	10	10	9

As stated before, small vehicles have the advantage of using some streets, mostly in the city centers, that are not compatible with or allowed (due to traffic rules) to large vehicles. To reflect the permeability of cities, we specify a central area and set the travel times of the arcs adjacent to the nodes in this area to higher values for large vehicles. We put the grid's borders ten units away from the closest customer. We define one-third of the grid in the center as the central area, as shown in Figure 1, for C and R groups. Since almost all customers are located outside the geometric center in RC instances, we move the central area to the left border of the grid for this group. We first multiply the travel times of all incoming and outgoing arcs of the nodes in this area with a random number between 1 and 2, then round them up. We provide the number of nodes in the central areas in Table 8. Based on these numbers, the instances in C, R and RC groups have 28-34%, 12-23% and 22- 36% of the customers located in the central area, respectively.

Table 9 **Results for the instances with 25 and 30 customers, two depots and two types of vehicles**

	25 customers					**30 customers**				
inst.	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**
C201	0	3.38	1	125	-	0	57.82	73	1557	17.53
C202	0	492.45	501	6824	10.32	0	129.54	67	774	14.18
C203	0	1941.33	7	161	11.64	15.3	7200	3	105	15.3
C204	NA	7200	0	34	NA	NA	7200	0	39	NA
C205	0	4.64	1	59	-	0	900.93	1059	8434	6.2
C206	0	13.51	3	91	9.89	0	2011.62	1173	14125	8.16
C207	0	452.37	43	551	3.04	0.99	7200	284	4848	8.49
C208	0	1591.84	537	9745	9.38	0	93.08	7	188	0
avg	0	1462.44	136.62	2198.75	8.85	2.33	3099.12	333.25	3758.75	9.98
R201	0	4.1	3	55	0.53	0	27.68	45	270	2.12
R202	0	31.76	37	239	7.56	0	43.33	15	130	4.14
R203	0	519.7	219	2043	7.84	0	2351.83	295	2979	6.82
R204	0	523.99	57	465	0.95	NA	7200	0	36	NA
R205	0	14.44	9	92	1.92	0	168.75	101	920	0.89
R206	0	1012.33	559	5465	8.13	0	294.69	49	476	10.98
R207	0	244.05	3	74	9.33	0.23	7200	4	154	7.28
R208	NA	7200	0	33	NA	NA	7200	0	32	NA
R209	0	77	21	284	10.79	0	388.20	89	750	13.61
R210	0	40.35	3	93	6.49	0	184.28	7	154	4.12
R211	0	5872.07	387	3757	9.05	9.73	7200	43	538	9.73
avg	0	1412.71	118	1145.45	6.26	1.11	2932.61	58.91	585.36	6.63
RC201	0	2.42	1	38	-	0	3.57	1	40	-
RC202	0	7.95	1	34	-	0	16.47	3	46	0
RC203	0	20.03	1	36	-	0	66.34	1	50	-
RC204	1.96	7200	21	206	7.36	0	905.86	1	61	-
RC205	0	53.43	63	513	0.81	0	11.95	1	35	-
RC206	0	132.4	129	1178	0.54	0	255.35	157	1477	0
RC207	0	10.2	1	41	-	0	25.58	1	52	-
RC208	NA	7200	0	41	NA	2.43	7200	3	93	2.43
avg	0.28	1828.31	27.12	260.88	2.90	0.30	1060.64	21	231.75	0.81

Table 10 Results for the instances with 35 and 40 customers, two depots and two types of vehicles

	35 customers					40 customers				
inst.	gap (%)	time	node	pricing iter	RootIP gap(%)	gap (%)	time	node	pricing iter	RootIP gap(%)
C201	0	32.16	17	537	0	0	27.83	1	268	-
C202	0	719.44	77	2358	1.26	0	340.18	19	515	6.77
C203	NA	7200	0	76	NA	8.43	7200	2	160	8.43
C204	NA	7200	0	37	NA	NA	7200	0	72	NA
C205	0	522.7	153	2513	0.75	0	288.7	9	426	0
C206	0	1112.53	253	3987	1.75	0	1105.93	41	1264	7.96
C207	0	1034.85	13	517	0	0	530.38	1	148	-
C208	0	242.41	11	390	0	0	823.72	5	376	0
avg	0	2258.01	65.5	1301.88	0.63	1.2	2189.59	9.75	403.62	4.63
R201	0	73.97	53	288	4	0	124.06	83	589	0.36
R202	0	416.46	43	467	1.42	0	605.69	119	678	6.46
R203	0	6358.43	55	437	10.33	8.69	7200	167	1875	8.69
R204	NA	7200	0	46	NA	NA	7200	0	67	NA
R205	0	150.57	27	268	12.88	0	1048.29	163	1479	7.24
R206	0	2564.48	81	985	9.29	0.57	7200	277	3725	9.14
R207	13.41	7200	3	98	13.41	18.89	7200	6	152	18.89
R208	NA	7200	0	50	NA	NA	7200	0	48	NA
R209	0	6617.33	547	5864	6.25	12.68	7200	246	3487	12.68
R210	0	902	51	502	10.49	5.99	7200	271	3403	5.99
R211	12.35	7200	4	90	12.35	22.25	7200	2	116	22.25
avg	2.86	4171.20	78.55	826.82	8.94	7.67	5398	121.27	1419.91	10.19
RC201	0	3721.52	7547	35982	0	0	18.94	3	54	0
RC202	0.37	7200	2038	14819	3.55	0	44.59	1	47	-
RC203	1.21	7200	24	286	1.21	0	76.61	1	56	-
RC204	NA	7200	0	51	NA	NA	7200	0	114	NA
RC205	0.68	7200	1601	10425	3.01	0	1849.90	295	1713	0.80
RC206	0	628.97	135	919	11.61	0	49.78	1	74	-
RC207	3.82	7200	666	6556	3.82	0	125.10	1	77	-
RC208	NA	7200	0	62	NA	NA	7200	0	72	NA
avg	1.01	5943.81	1501.38	8637.50	3.87	0	2070.62	37.75	275.88	0.40

Note that even though all customers are accessible by each vehicle type, vehicle dependent travel times combined with the time windows might result in graphs with different densities for different types of vehicles.

For the heterogeneous fleet variant, we use the same values of the algorithmic parameters as before and test its performance on the instances with two depots. We present our results for instances with 25 and 30 customers in Table 9, and with 35 and 40 customers in Table 10.

Our first observation is that the performance of the proposed algorithm does not show a significant degradation for instances with two depots when a heterogeneous fleet is considered even though fewer instances can be solved at the root node compared to the homogeneous fleet case. For the homogeneous fleet case, 77 of 108 instances with two depots are solved in 11.31 minutes on average, and for the heterogeneous fleet case, 72 of 108 instances are solved in 12.31 minutes on average. We also observe that while the difficulties of C and R instances exhibit a similar pattern as in the homogeneous fleet case, instances in the RC group with 40 customers become easier to solve. Whereas only one instance with 40 customers is solved to optimality when a homogeneous fleet of small vehicles are considered, six of eight instances are solved when large vehicles are also incorporated into the fleet.

In our next experiment, we look at the instances with three depots and two types of vehicles. We report our results for instances with 25 and 30 customers and three depots in Table 11 and with 35 and 40 customers in Table 12.

Table 11 Results for the instances with 25 and 30 customers, three depots and two types of vehicles

	25 customers					**30 customers**				
inst.	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**
C201	0	389.85	541	12269	30.77	0	92.49	17	1165	0
C202	0	40.65	13	170	14.95	0	257.38	29	1112	4.44
C203	0	831.28	1	103	-	0	1400.26	17	437	1.96
C204	NA	7200	0	53	NA	NA	7200	0	51	NA
C205	0	33.05	13	150	12.15	0	26.33	1	87	-
C206	0	49.56	17	187	22.31	0	57.58	1	136	-
C207	0	951.83	55	1012	0	0	328.91	3	217	3.85
C208	0	44.12	1	82	-	0	282	13	464	3.85
avg	0	1192.54	80.13	1753.25	16.04	0	1205.62	10.13	458.63	2.82
R201	0	13.86	13	163	1.3	0	55.84	33	282	0
R202	0	41.82	11	152	0	0	237.66	87	523	5.13
R203	0	156.95	29	196	13.85	0	371.04	37	347	12.19
R204	0	418.21	7	164	0	9.53	7200	29	499	9.53
R205	0	114.67	73	553	12.16	0	359.15	61	818	2.4
R206	0	224.72	81	566	9.81	0	341.79	21	225	9.61
R207	0	587.79	1	81	-	0	4.303.58	21	244	8.32
R208	10.41	7200	2	147	10.41	NA	7200	0	51	NA
R209	0	602.74	165	2074	8.69	0	429.19	47	499	7.22
R210	0	54.12	5	123	8.33	0	678.93	77	726	2.57
R211	0.33	7200	230	3656	16.25	9.12	7200	40	630	9.12
avg	0.98	1510.44	56.09	715.91	8.08	1.87	2407.36	41.18	440.36	6.61
RC201	0	1.31	1	33	-	0	329.72	357	2695	0.2
RC202	0	6.53	1	34	-	5.39	7200	1052	11643	14.39
RC203	0	9.5	1	29	-	NA	7200	0	21	NA
RC204	0	6.7	1	33	-	NA	7200	0	41	NA
RC205	0	9.25	1	43	-	2.35	7200	3912	30962	5.13
RC206	0	3.78	1	31	-	13.74	7200	2038	19389	13.74
RC207	0	6.42	1	46	-	14	7200	244	2345	14
RC208	0	564.15	7	289	0	NA	7200	0	49	NA
avg	0	75.96	1.75	67.25	0	7.10	6341.22	950.38	8393.13	9.49

We see that the algorithm performs well for instances with 25 customers and three depots and can solve 24 of 27 instances in less than four minutes on average. However, instances with a larger number of customers become significantly more difficult than in the homogeneous fleet case. While 83 of 108 instances with three depots are solved to optimality in 15.47 minutes on average for the homogeneous fleet case, the number of solved instances decreases to 68 with an average computation time of 15.49 minutes for the heterogeneous fleet case.

Table 12 Results for the instances with 35 and 40 customers, three depots and two types of vehicles

	35 customers					**40 customers**				
inst.	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**	**gap (%)**	**time**	**node**	**pricing iter**	**RootIP gap(%)**
C201	0	42.64	3	430	0	0	149.03	5	856	0
C202	0	982.97	71	3203	0.28	0	3294.84	33	2918	10.89
C203	2.98	7200	2	256	2.98	1.27	7200	1	192	1.27
C204	NA	7200	0	42	NA	NA	7200	0	47	NA
C205	0	83.67	1	179	-	0	1966.06	57	2875	2.57
C206	0	118.55	1	194	-	0	894.22	17	697	0
C207	0	7198.83	217	6451	0.28	4.14	7200	28	1145	4.14
C208	0	1425.04	35	1733	0	1.25	7200	109	3884	1.25
avg	0.43	3031.46	41.25	1561	0.71	0.95	4388.02	31.25	1576.75	2.87
R201	0	276.89	123	784	1.55	0	298.95	147	993	3.43
R202	0	1438.23	199	1361	11.14	0	640.67	47	387	3.48
R203	14.62	7200	44	641	14.62	0	1132.041	33	293	0.77
R204	NA	7200	0	44	NA	NA	7200	0	46	NA
R205	0	4117.71	525	5271	6.71	0	191.84	3	132	0
R206	0	6225.14	355	2456	11.82	0	4962.13	171	1535	9.84
R207	13.20	7200	1	80	13.20	8.30	7200	17	351	8.30
R208	NA	7200	0	42	NA	NA	7200	0	57	NA
R209	16.33	7200	250	4070	16.33	0	3404.67	51	846	16.08
R210	0	3486.72	169	1245	11.74	0	4527.08	125	1440	10.13
R211	16.27	7200	7	150	16.27	20.41	7200	7	221	20.41
avg	6.71	5340.43	152.09	1467.64	11.49	3.19	3996.13	54.64	572.82	8.05
RC201	0	31.38	5	111	5.63	0	451.60	117	839	5.57
RC202	1.50	7200	1107	11287	6.25	2.31	7200	428	3323	2.31
RC203	4.30	7200	241	2666	5.55	3.62	7200	127	1355	3.77
RC204	NA	7200	0	30	NA	NA	7200	0	68	NA
RC205	0	1050	145	1401	0.36	0.67	7200	575	4212	2.75
RC206	0	957.81	123	1242	0.18	0	905.62	63	576	0
RC207	0	124.66	1	66	-	1.92	7200	180	1575	7.26
RC208	5.75	7200	1	75	5.75	NA	7200	0	66	NA
avg	1.65	3870.48	202.88	2109.75	3.95	1.42	5569.65	186.25	1501.75	3.61

5.3. The Impact of the Heterogeneous Fleet

To assess the effect of operating a heterogeneous fleet on the delivery cost, we compare the problems with a homogeneous fleet of large vehicles and a heterogeneous fleet of small and large vehicles. Due to depots' capacities and customers' time windows, using a fleet of only large vehicles yields infeasibility for C instances with 35 and 40 customers, and RC instances with 30 and 35 customers. We increase the capacity of each depot by one for these instances. We consider only the instances solved to optimality for both variants. We report the average values of the total number of vehicles, the number of small vehicles (for the heterogeneous fleet), the optimal value and the saving of these instances in Table 13.

Table 13 Comparison of the homogeneous and the heterogeneous fleet

instance group	# of customers	total demand	# of solved inst.	homogeneous fleet		heterogeneous fleet			saving %
				vehicles	cost	vehicles	small vehicles	cost	
C	25	460	5	1.60	357.20	1.86	1	314.20	12.04
	30	520	5	2	404.80	2	1	355.40	12.20
	35	640	6	2	414.67	2	1	382.67	7.72
	40	730	6	2	493.33	2	1	473.50	4.02
R	25	332	10	1.80	496.40	2	2	458.30	7.68
	30	411	7	2	611.14	2.43	2.43	546.29	10.61
	35	494	7	2	640.71	2.43	2.14	606.14	5.40
	40	563	2	2	881.50	3.50	3.50	765.50	13.16
RC	25	540	6	2	424.00	2.50	2.33	353.83	16.55
	30	620	8	3	518.88	2.88	1.50	494.50	4.70
	35	710	6	3.5	640.33	3.33	1.33	621.83	2.89
	40	830	6	3.67	708.83	3.83	0.67	693.83	2.12

The results show that using a heterogeneous fleet provides 7.75% cost saving on average and has a more pronounced impact on the cost for smaller instances. On the other hand, the average number of vehicles is almost the same for both cases. Whereas 2.32 vehicles on average are used for the homogeneous fleet case, it is 2.50 for the heterogeneous fleet case where more than 64% of the fleet consists of small vehicles. The results on the number of used vehicles demonstrate the benefit of allowing vehicles to perform multiple trips per day.

We also observe that the vehicle selection strategy depends on the instance group and varies even among the instances from the same group. Whereas one large and one small vehicle are selected on average for C instances, the optimal solution almost always favours small vehicles for R instances. Although the total demand of the R instances with 25 customers being slightly higher than the capacity of the large vehicle might cause an under-utilization of vehicles and impact the vehicle selection, the results of larger instances support that more factors need to be taken into account. Note that when more than one vehicle of the same type is used, these vehicles have the flexibility to choose their returning depots as long as the vehicle balance constraints of depots are satisfied. On the other hand, when only one vehicle is used, it has to go back to its starting depot to satisfy the vehicle balance constraints. Since, on average, at least two small vehicles are used in R instances, we investigate whether this provides an advantage to small vehicles and dominates the solutions where one large vehicle is used. To this end, we solved the problem where each vehicle has to start and end its workday at the same depot and found out that the selection of the vehicle type remains the same. Another observation is that while the ratio of the number of small vehicles to the total fleet size remains almost the same for different sized instances in C and R groups, it decreases proportionally to the number of customers in RC group that contains the customers with higher demands.

6. Conclusion

In this study, we presented an effective branch and price algorithm for the heterogeneous fleet multi-depot MTVRPTWs under shared depot resources in which we considered the effect of bike-friendly infrastructures on the travel times. The results of conducted experiments demonstrate the efficacy of our solution approach and suggest that, besides the size of the instance, the tightness of the capacities at depots as well as the customers' spatial distribution influence the difficulty of the problem.

One of the limitations of our study is that the proposed method cannot be used when the depot's capacities are defined in terms of the amount of products. In this case, the dual variables corresponding to the depot capacity constraints cannot be decomposed over arcs as it depends both on the visited customer and the depot used before visiting this customer. Developing a technique for this case is an important future research direction.

Our study considers the effect of physical city structures such as the density of narrow streets and infrastructures like bike bridges on the travel times of small and large vehicles. Another promising direction of future work is to incorporate access time restrictions during which large vehicles cannot enter certain areas. Without such restrictions, the streets that different vehicle types can use differ based on the physical city structures, whereas under these restrictions, the time of the day is also a determinative factor on the vehicle selection. With the integration of the access time restrictions, another promising direction of future work is to allow large vehicles to be used as mobile depots. In this extension, small vehicles deliver the products to the customers in the restricted zones, and large vehicles can be used to replenish the load of small vehicles in-between their trips. Since the vehicles have to meet at a location for loading, the workdays of different vehicle types become inter-dependent, which raises the need for synchronization constraints. We believe that these are interesting and challenging extensions of our current work.

Acknowledgments

We would like to thank Prof. Gizem Ozbaygin, Sabanci University, for her help with the implementation.

References

Azi N, Gendreau M, Potvin JY, 2010 *An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. European Journal of Operational Research* 202(3):756–763.

Baldacci R, Mingozzi A, Roberti R, 2011 *New route relaxation and pricing strategies for the vehicle routing problem. Operations research* 59(5):1269–1283.

Bettinelli A, Ceselli A, Righini G, 2011 *A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. Transportation Research Part C: Emerging Technologies* 19(5):723–740.

Boland N, Dethridge J, Dumitrescu I, 2006 *Accelerated label setting algorithms for the elementary resource constrained shortest path problem. Operations Research Letters* 34(1):58–68.

Boland NL, Clarke LW, Nemhauser GL, 2000 *The asymmetric traveling salesman problem with replenishment arcs. European Journal of Operational Research* 123(2):408–427.

Boysen N, Fedtke S, Schwerdfeger S, 2020 *Last-mile delivery concepts: a survey from an operational research perspective. OR Spectrum* 1–58.

Cattaruzza D, Absi N, Feillet D, 2016 *Vehicle routing problems with multiple trips. 4OR* 14(3):223–259.

Cattaruzza D, Absi N, Feillet D, Vigo D, 2014 *An iterated local search for the multi-commodity multi-trip vehicle routing problem with time windows. Computers & Operations Research* 51:257–267.

Contardo C, Martinelli R, 2014 *A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. Discrete Optimization* 12:129–146.

Costa L, Contardo C, Desaulniers G, 2019 *Exact branch-price-and-cut algorithms for vehicle routing. Transportation Science* 53(4):946–985.

Dash S, Günlük O, Lodi A, Tramontani A, 2012 *A time bucket formulation for the traveling salesman problem with time windows. INFORMS Journal on Computing* 24(1):132–147.

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: An International Journal* 44(3):216–229.

Gevaers R, Van de Voorde E, Vanelslander T, et al., 2009 *Characteristics of innovations in last-mile logistics-using best practices, case studies and making the link with green and sustainable logistics. Association for European Transport and contributors* 1–21.

Groot Rd, 2007 *Design manual for bicycle traffic.* Number 25.

Hernandez F, Feillet D, Giroudeau R, Naud O, 2014 *A new exact algorithm to solve the multi-trip vehicle routing problem with time windows and limited duration. 4or* 12(3):235–259.

Hernandez F, Feillet D, Giroudeau R, Naud O, 2016 *Branch-and-price algorithms for the solution of the multi-trip vehicle routing problem with time windows. European Journal of Operational Research* 249(2):551–559.

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

Kohl N, 1995 *Exact methods for time constrained routing and related scheduling problems.* Ph.D. thesis, Technical University of Denmark.

Lloyd S, 1982 *Least squares quantization in pcm. IEEE transactions on information theory* 28(2):129–137.

Macedo R, Alves C, de Carvalho JV, Clautiaux F, Hanafi S, 2011 *Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. European Journal of Operational Research* 214(3):536–545.

Montoya-Torres JR, Franco JL, Isaza SN, Jiménez HF, Herazo-Padilla N, 2015 *A literature review on the vehicle routing problem with multiple depots. Computers & Industrial Engineering* 79:115–129.

Paradiso R, Roberti R, Laganá D, Dullaert W, 2020 *An exact solution framework for multitrip vehicle-routing problems with time windows. Operations Research* 68(1):180–198.

Pecin D, Pessoa A, Poggi M, Uchoa E, 2017 *Improved branch-cut-and-price for capacitated vehicle routing. Mathematical Programming Computation* 9(1):61–100.

Pucher J, Dill J, Handy S, 2010 *Infrastructure, programs, and policies to increase bicycling: an international review. Preventive medicine* 50:S106–S125.

Solomon MM, 1987 *Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research* 35(2):254–265.

Vision Monday, 2021 *E-commerce share of total global retail sales from 2015 to 2024.* Accessed August 9, 2021, https://www.statista.com/statistics/534123/e-commerce-share-of-retail-sales-worldwide/.