Transportation Research Part B 116 (2018) 1–24

Contents lists available at ScienceDirect

Transportation Research Part B

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

The time-dependent pickup and delivery problem with time windows

Peng Sun[a,*], Lucas P. Veelenturf[b], Mike Hewitt[c], Tom Van Woensel[b]

[a] *Kühne Logistics University Hamburg, Germany*
[b] *School of Industrial Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands*
[c] *Quinlan School of Business Loyola University Chicago, Chicago, USA*

ARTICLE INFO

Article history:
Received 10 February 2018
Revised 23 June 2018
Accepted 2 July 2018

Keywords:
Pickup and delivery problem
Time windows
Time-dependent travel times
Branch-and-price

ABSTRACT

In this paper, we study a family of time-dependent pickup and delivery problems with time windows to optimize the service of a transportation provider under two dimensions of operational flexibility. In the first, we consider problems wherein the transportation service provider can choose the transportation requests it serves in order to maximize profit. In the second, we consider problems wherein they can take advantage of periods of light traffic by dictating to drivers when their routes should begin. We also consider problems wherein these flexibilities are not present. We propose an exact solution approach for solving problems from this family that is based upon branch and price, wherein columns are generated via a tailored labeling algorithm. We augment the framework with adaptations of various speed-up techniques from the literature, including limited-memory subset-row cuts and route enumeration. With an extensive computational study, we assess the effectiveness of the proposed framework and the impact of the adapted techniques.

1. Introduction

The advent of E-commerce has led to increased shipping volumes both when the customer is a business (i.e., B2B) and when the customer is an individual (B2C). In either case, the trend in customer expectations is for shorter, and more precisely specified (i.e., just-in-time logistics), delivery times. To handle these trends at low cost and environmental impact, transportation companies need tools for effectively optimizing how they should respond to and serve transportation requests. In this paper, we focus on transportation requests that indicate a pickup location for a shipment, a time window during which the shipment must be picked up, a delivery location for the shipment, and a time window during which the shipment must be delivered. In the literature (Cordeau et al., 2008; Parragh et al., 2008a; 2008b), optimizing the service of such requests with a fixed fleet of capacitated vehicles has been termed the pickup and delivery problem with time windows (PDPTW)

The objective of the classical PDPTW is to minimize the transportation costs incurred when executing routes designed under the assumption that all transportation requests must be served. This assumption is particularly relevant to supply chains based upon transportation that is either owned or outsourced to a dedicated carrier. A growing trend in supply chain management is to instead outsource transportation needs on a request basis through the use of “load markets” or “boards”.

[*] Corresponding author.
E-mail addresses: peng.sun@the-klu.org (P. Sun), l.p.veelenturf@tue.nl (L.P. Veelenturf), mhewitt3@luc.edu (M. Hewitt), t.v.woensel@tue.nl (T. Van Woensel).

https://doi.org/10.1016/j.trb.2018.07.002
0191-2615/© 2018 Elsevier Ltd. All rights reserved.

Some of the growing demand for transporting individual requests can be attributed to the capabilities and ease-of-use of the software platforms supporting such boards. From the transportation supply side, the easy (sometimes mobile) access that these platforms provide also enables transportation service providers to pick and choose the requests that yield the greatest profits. As a result, in this paper we study variants of the PDPTW wherein the transportation company can select a subset of the requests to serve based on the revenues it receives for serving those requests. In these variants, the objective is then to maximize profits.

The routing community has long studied problems (including PDPTWs) that assumed that travel times are constant, or, time-independent. However, both demographic trends towards larger populations in urban areas and improved technology for solving routing problems has led researchers (Ichoua et al., 2003; Jabali et al., 2009; Dabia et al., 2013; Van Woensel et al., 2008) to study those that consider time-dependent travel times (as reviewed in Gendreau et al., 2015). Such problems are particularly important as customer expectations for short delivery times reduce the flexibility transportation companies have for accommodating variations from “average” travel times. While some variation may be due to unforeseen events, some is due to traffic and congestion. As traffic patterns are somewhat predictable, embedding their impact on travel times in deterministic routing models is likely to yield plans that are more likely to meet customer expectations at low cost than those that ignore those impacts.

Thus, in all the variants of the PDPTW that we study in this paper, we consider time-dependent travel times, meaning that the time it takes to travel between two locations depends on the time when travel begins. Like Ichoua et al. (2003), we assume these travel times adhere to the “first in first out” (FIFO) property, meaning that the earlier the departure, the earlier the arrival. However, recognizing that travel times may vary throughout the day brings a new dimension to the PDPTW. Specifically, there can be value in optimizing the time at which a route begins. As providing a driver with a different start time each day may not be feasible in all labor/operational settings, we consider variants where the departure time is fixed as well as those where it can be optimized.

We present an optimization-based framework for variants of the PDPTW that can recognize time-dependent travel times as well as specify vehicle start times. To do so, and like Røpke and Cordeau (2009), the framework models vehicle departure times with continuous variables. As a result, the framework optimizes vehicle routes based upon precise representations of the timings of vehicle departures and arrivals. Relatedly, continuous time variables make it easy to model travel times that adhere to the FIFO property. An alternative approach, and one proposed by Mahmoudi and Zhou (2016), is to optimize decisions on a network that represents both the physical and temporal attributes of decisions (i.e., a space-time network). While such a network can facilitate the representation of time-dependent travel times, it also necessitates discretizing time, which can introduce approximation errors into an optimization model.

In total, we study four variants of the time-dependent pickup and delivery problem with time windows (TDPDPTW), that differ with respect to two attributes: (1) whether the variant requires that all transportation requests be served, and, (2) whether the variant requires that all routes begin at a fixed departure time. For the sake of clarity, let Ω be our base problem, i.e. the time-dependent pickup and delivery problem with time windows. We denote the four variants then as follows: Ω(*Fix, All*), Ω(*Fix, Prof*), Ω(*Flex, All*) and Ω(*Flex, Prof*), where *Fix* versus *Flex* refers to fixed versus flexible departure times, and *All* versus *Prof* refers to handling all versus only the profitable requests. We present a solution framework based on branch-and-price that can solve each of these variants to optimality. Our framework adapts and extends the work presented in Røpke and Cordeau (2009) and Røpke et al. (2009) for solving the PDPTW and employs some of the techniques presented in Sun et al. (2018) in an algorithm for solving single-vehicle variants of the problems we study.

As such, this paper makes the following contributions to the literature:

- We present a general mathematical model for variants of the time-dependent pickup and delivery problems with time windows that are inspired by changing trends in the transportation industry.
- We present an exact algorithm for solving the time-dependent pickup and delivery problem with time windows and the variants we consider. To the best of our knowledge, this is the first algorithm that is able to solve all of these variants.
- We illustrate how to adapt known speed-up techniques from the literature on other, related, routing problems to the problems we seek to solve. With an extended computational study demonstrate their effectiveness.

The remainder of this paper is organized as follows. Section 2 presents a brief review of the existing work related to this paper. Section 3 introduces the mathematical formulation for the variants of the time-dependent pickup and delivery problem with time windows we consider in this paper. In Section 4, we describe our exact framework. Finally, computational results are reported in Section 5, followed by conclusions in Section 6.

2. Literature review

There are several problems in the literature that share characteristics with the time-dependent pickup and delivery problem with time windows. These include the pickup and delivery problem with time windows (PDPTW), the vehicle routing problem with profits (VRPP) and the time-dependent vehicle routing problem with time windows (TDVRPTW). We preview how the problems we study intersect with those problems in Fig. 1 and will next discuss those intersections in detail.

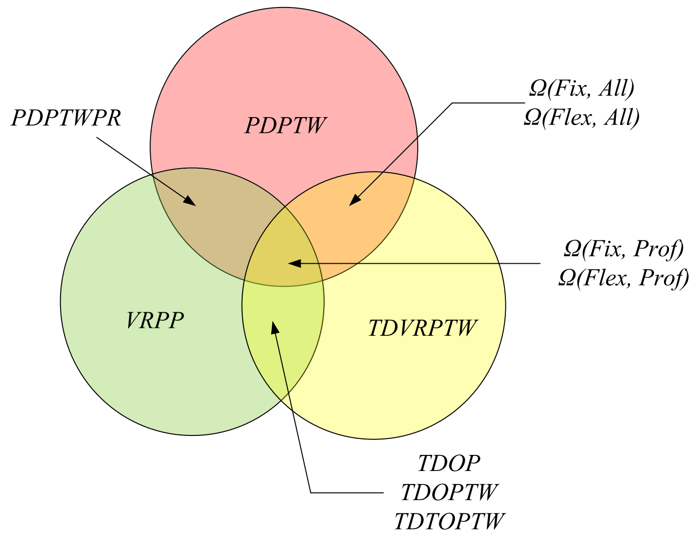

Fig. 1. Classification of related problems.

2.1. The pickup and delivery problem with time windows (PDPTW)

The PDPTW models a variety of operational planning problems in transportation and logistics. This results in a rich literature on PDPTW-related problems over the last few decades. In this problem, a set of homogeneous vehicles with limited capacity based at a depot is required to carry out a group of geographically scattered requests. Each request needs to be transported from a specified origin and delivered to a specified destination within the predefined time windows, which is the interval of time during which the service at a node must start. The interested readers are referred to Cordeau et al. (2008) and Parragh et al. (2008a,b) for recent surveys. Moreover, due to the difficulty of the problem, much of the existing literature concentrates on heuristics.

Less work has been done on exact algorithms for these types of problems. The pioneering work is done by Dumas et al. (1991) who proposed the first branch-cut-price algorithm for the PDPTW. This method was able to solve tightly constrained instances with up to 55 requests. Recently, the exact approach was also used in Sigurd and Pisinger (2004), Sol (1994), Røpke and Cordeau (2009) and Røpke et al. (2009). The method of Røpke and Cordeau (2009) is a branch-cut-and-price algorithm that uses different classes of valid inequalities to improve the lower bound and two different methods for solving the pricing problem. Baldacci et al. (2011) have designed two exact algorithms relying on column generation and variable fixing based on reduced cost. Recently, Azadian et al. (2017) proposed an exact method based on a decomposition approach for a variant of the PDPTW wherein requests are unpaired. In the problem they study, delivery costs are time-dependent, but travel times are time-independent.

Unlike the problems studied in this paper, the majority of related publications consider a constant travel time environment. Moreover, unlike some of the problems we study, the majority of the literature studies problems wherein all requests must be served.

2.2. The vehicle routing problem with profits (VRPP)

The vehicle routing problem with profits (VRPP) also arises in a wide range of transportation and logistics applications. In these problems, along with Ω(*Fix, Prof*) and Ω(*Flex, Prof*), there is a request selection decision that must be made. Instead of maximizing the difference between total revenues and total costs, which defines the profitable tour problem (PTP), the orienteering problem (OP) aims to maximize the total collected profits subject to a maximum tour length. Conversely, the prize-collecting traveling salesman problem (PCTSP) aims to minimize the total traveling cost with the constraint that the total collected profits cannot be less than a given amount. A recent survey (Feillet et al., 2005) of routing problems with a single vehicle and profits indicates that most of existing literature focus on the OP with its variants. As an example, Righini and Salani (2009) proposed an extended bidirectional dynamic programming algorithm to solve the orienteering problem with time windows (OPTW), which recognizes the presence of time window constraints. More recently, Duquei et al. (2015) adapted the pulse algorithm to the OPTW by adding new problem-specific strategies. The resulting algorithm (computationally) outperforms the algorithm proposed by Righini and Salani (2009). Extending the orienteering problem to teams yields the team orienteering problem (TOP). Similarly, the OPTW has been extended to the team orienteering problem with time windows (TOPTW). Exact and heuristic solution strategies have been developed for both of these problems. An exact algorithm for the TOP can be found in Boussier et al. (2007), whereas Labadie et al. (2012) developed a granular variable neighborhood search for the TOPTW. Recently, Archetti et al. (2013) proposed a branch-cut-and-price algorithm to solve a variant of the TOP known as the capacitated team orienteering problem (CTOP), which also models vehicle capacity. We refer the reader to the survey presented by Vansteenwegen et al. (2011) and Gunawan et al. (2016) for a detailed review of the OP and its variants. For more details on VRPP, we refer the readers to the technical report presented by Archetti et al. (2014). However, ultimately, much of the literature on these problems assumes a constant travel time environment.

2.3. Time-dependent vehicle routing problems

Less research has been done on problems that allow travel times between origins and destinations to vary based on the time of departure from the origin. One problem that has been studied is the time-dependent vehicle routing problem with time windows (TDVRPTW). The objective of this problem is to find a set of routes that visit all customers with minimum traveling duration. Malandraki and Daskin (1992) presented a mixed integer programming-based formulation of this problem, some construction heuristics, and a cutting plane-based solution method. Later, Ichoua et al. (2003) proposed a parallel tabu search to solve the problem. However, they also made the important observation that previous models of time-dependent travel times did not adher to the “first in first out” (FIFO) property. Namely, that if two identical vehicles traverse the same arc, the one that leaves first will arrive first. As a result, they proposed a speed model that is a step-wise function and satisfies the FIFO principle. This speed model was also used in Donati et al. (2008), which presented a multiple ant colony system (ACS) solution framework for the TDVRPTW. This ACS framework was then enhanced in Balseiro et al. (2011). Regarding exact approaches, Dabia et al. (2013) developed a branch and price algorithm for TDVRPTW that employs a tailored labeling algorithm to solve a pricing problem that can be formulated as a time-dependent shortest path problem with resource constraints (TDSPPRC). The proposed algorithm is able to solve instances with up to 100 customers. Readers interested in the TDVRPTW and its variants should consult the recent survey by Gendreau et al. (2015). The TDVRPTW differs from the problems we study in this paper as it does not consider the precedence relationships between locations (i.e., the pickup node of a request must be visited before the corresponding destination node).

2.4. Closely related problems

We have (briefly) reviewed three different classes of problems: (1) those that model pickups and deliveries that must occur with time windows, (2) those that model request selection, and, (3) those that model time-dependent travel times. Research has been done on problems that model attributes of two of these problem classes. For example, Li et al. (2016) consider a PDPTW in a profitable context, which they refer to as the pickup and delivery problem with time windows, profits, and reserved requests (PDPTWPR). In this problem, there is one type of request that must be served (called reserved requests), and another which may be rejected or outsourced. They propose an adaptive large neighborhood search to solve this problem.

Similarly, researchers have studied the Time-dependent orienteering problem (TDOP) and its variants. Fomin and Lingas (2002) provide a $(2+\epsilon)-$ approximation algorithm for the TDOP which runs in polynomial time if the ratio between the minimum and maximum traveling time of any two sites is constant. Li (2011) designed a dynamic labeling algorithm for the TDOP, where discrete time units are used. Verbeeck et al. (2014) proposed an ant colony optimization-based heuristic for the TDOP. To the best of our knowledge, the only heuristic for the TDTOPTW is the iterated local search-based method proposed by Gavalas et al. (2015).

Recently, Sun et al. (2018) introduced the time-dependent capacitated profitable tour problem with time windows and precedence constraints. A tailored labeling algorithm is proposed that is able to solve instances with up to 75 requests to optimality. They also propose a dynamic programming-based heuristic for instances that cannot be solved by the exact method. However, unlike the problems considered in this paper, it only considers a single vehicle.

Also related to the work presented in this paper is that of Mahmoudi and Zhou (2016), which studies a time-dependent PDPTW seen in ride-sharing settings. They propose a solution approach that couples a Lagrangian relaxation scheme for producing dual bounds with a dynamic programming-based method for producing primal solutions. Their approach is based on a state-space-time network representation of the problem, which is based on a discretization of time.

As discussed previously, the framework presented in this paper and the method of Mahmoudi and Zhou (2016) model the timing of decisions differently. However, they also consider profits in different ways. The variants considered in this paper treat the profit earned from serving a customer request as a known and fixed value, and thus the framework we propose seeks to maximize the difference between the profits associated with serving customers and the costs incurred by doing so. The problem studied in Mahmoudi and Zhou (2016) focuses on minimizing the total cost associated with transporting all passengers. However, the Lagrangian component of their method relaxes the requirement that each passenger is transported. Associated with the relaxed constraints are dual variables that are iteratively updated, and converge to values that can be interpreted as marginal profits. Relatedly, the procedure for producing primal solutions allows for passengers to be assigned to virtual vehicles.

Finally, the framework in this paper and the method of Mahmoudi and Zhou (2016) differ in the known guarantees that can be made regarding their performance. The framework we present is based upon branch-and-price, an algorithmic strategy for solving integer programs that is known to be exact. Namely, that when run for sufficient time, it is guaranteed to

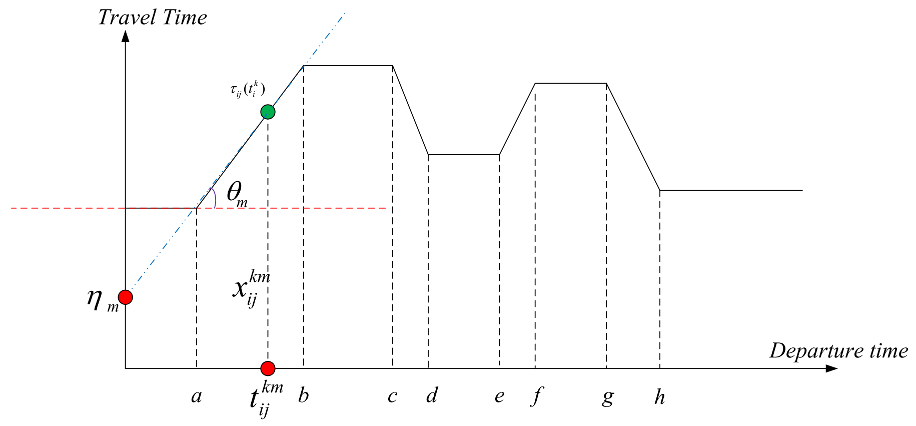

Fig. 2. Travel time function.

produce both a primal solution that is optimal and a certificate of its optimality. This guarantee is based on appropriately-designed subroutines for generating new variables and partitioning the solution space. We propose such subroutines and thus exact algorithms for the variants we consider. As Mahmoudi and Zhou (2016) do not include a proof that the method they propose is exact, such a guarantee is not yet known.

3. Problem description and mathematical formulation

In this section, we first present a mathematical programming formulation of Ω(*Flex, Prof*). From this formulation we show how to derive formulations of the other three variants by adapting its objective function and/or fixing certain variables. As a result, we refer to the Ω(*Flex, Prof*) as the *general* time-dependent pickup and delivery problem with time windows

The general time-dependent pickup and delivery problem with time windows considers a fleet of homogeneous vehicles that travel on routes constructed to serve customer pickup and delivery requests. To serve a request, which in turn generates profits, the vehicle must visit the pickup location during a given time window and then later visit the delivery location during another given time window. Visiting either a pickup or delivery location requires a service time. To model time-dependent travel times between a pair of locations, we divide the planning horizon into time zones for travel between those locations, wherein a different speed may be associated with each time zone. The objective of the problem is to maximize the difference between the profits earned by serving requests and costs associated with transportation. Transportation costs include costs associated with time spent traveling and a fixed cost incurred when a vehicle is used.

The formulation we present for the Ω(*Flex, Prof*) is inspired by the PDPTW formulation presented in Røpke and Cordeau (2009). We define a directed graph $G=(N,A)$, wherein $N=0,1,\dots,2n+1$ is the set of all nodes and A is the set of arcs. Nodes 0 and $2n+1$ represent the origin and destination depot of the vehicles. The sets $N_P=1,\dots,n,\subseteq N$ and $N_D=n+1,\dots,2n,\subseteq N$ represent the sets of pickup and delivery locations, respectively. There is a set of n requests $R_1,\dots,R_n$, wherein serving request i yields profit p_i. Associated with request R_i is a pickup node i, with service time s_i, and delivery node $n+i$, with service time s_{n+i}. A time window $[e_i, l_i]$ is associated with every node $i\in N_P\cup N_D$, wherein e_i and l_i represent the earliest and latest times, respectively, at which service may start at node i. If the vehicle arrives at i earlier than e_i, it waits; it cannot arrive later than l_i. The depot nodes also have time windows $[e_0, l_0]$, $[e_{2n+1}, l_{2n+1}]$ representing the earliest and latest times, respectively, at which the vehicle may leave from and return to the depot.

Each request has a size, q_i. We associate with the pickup node i the quantity q_i and the delivery node $n+i$ the quantity $q_{n+i}(=-q_i)$. We presume there is no inventory at the depot, and thus $q_0=q_{2n+1}=0$. Similarly, we assume $s_0=s_{2n+1}=0$. To serve requests, there is a fleet of K identical vehicles, each with capacity Q and a fixed cost, Z, that is paid when the vehicle is used.

We let $\tau_{ij}(t_i^k)$ denote the travel time from node i to node j when vehicle k departs from node i at time t_i^k. We define the set of arcs as $A=\{(i,j)\in N\times N : i\neq j \text{ and } e_i+s_i+\tau_{ij}(e_i+s_i)\leq l_j\}$. In other words, an arc from node i to node j is included only if they can be visited in succession while respecting their time windows. Similar to Ichoua et al. (2003), Jabali et al. (2009), Dabia et al. (2013) and Franceschetti et al. (2013), associated with each $(i, j)\in A$ is a speed profile that can be represented by a step-wise function based upon a division of the planning horizon into time zones. In this profile, speeds are constant within a time zone but may vary from one zone to the next. With such a speed profile, we can generate travel times that satisfy the FIFO principle.

Specifically, we use a travel time function that is piece-wise linear and continuous (see Fig. 2 for an example). This function is based on a set of breakpoints that denote when a change of speed (and hence travel time) occurs. Specifically, we let T_{ij} represent the set of time zones modeled by the travel time function $\tau_{ij}(\cdot)$. A time zone $T_{ij}^m\in T_{ij}$ can be defined by two consecutive travel time breakpoints, $T_{ij}^m=[w_m,w_{m+1}]$, wherein the travel time function is linear within those breakpoints.

Thus, for time zone T_{ij}^m, given the values w_m, w_{m+1}, $\tau_{ij}(w_m)$, and $\tau_{ij}(w_{m+1})$, we calculate the slope, θ_m, of the function and its intersection, η_m, with the y-axis. From this slope and intercept, we compute the travel time when departure occurs at time $t_i^k \in T_{ij}^m$ as follows:

$$\tau_{ij}(t_i^k) = \theta_m t_i^k + \eta_m. \qquad \forall t_i^k \in T_{ij}^m \tag{1}$$

To model this piecewise-linear function within an integer program, we define the binary variable $x_{ij}^{km}, i, j \in N, k \in K, m = 1, \ldots, \mid T_{ij} \mid$, to represent whether vehicle k travels from node i to node j by departing from i in time zone T_{ij}^m. Relatedly, we let the continuous variable $t_{ij}^{km}, i, j \in N, k \in K$, represent the time when vehicle k departs from i to j during time zone T_{ij}^m. If vehicle k does not depart from i to j during time zone T_{ij}^m (i.e., $x_{ij}^{km} = 0$)), then $t_{ij}^{km} = 0$. With these variables, we can express the travel time function $\tau_{ij}(t_i^k)$ of arc (i, j):

$$\tau_{ij}(t_i^k) = \sum_{m=1}^{|T_{ij}|} (\theta_m t_{ij}^{km} + \eta_m x_{ij}^{km}) \tag{2}$$

Regarding request selection, we let $y_i^k, i \in N$, be a binary variable that represents whether vehicle k visits node i. Relatedly, we define $Q_i^k, i \in N, k \in K$ as a nonnegative integer variable that equals the load of vehicle k when leaving node i. Finally, we presume travel costs are a linear function of travel times, and represent this relationship with the cost per unit parameter, c_t.

With these variables and parameters, we formulate Ω(*Flex, Prof*) as the following mixed integer program:

$$Obj_{\Omega(Flex,Prof)} = \max \sum_{k \in K} \left[\sum_{i \in N_P} p_i y_i^k - c_t (t_{2n+1}^k - t_0^k) - Z y_0^k \right] \tag{3}$$

subject to

$$\sum_{j \in N_P} \sum_{m=1}^{|T_{0j}|} x_{0j}^{km} = y_0^k \qquad \forall k \in K, \tag{4}$$

$$\sum_{i \in N_D} \sum_{m=1}^{|T_{i,2n+1}|} x_{i,2n+1}^{km} = y_{2n+1}^k \qquad \forall k \in K, \tag{5}$$

$$\sum_{k \in K} y_j^k \le 1 \qquad \forall j \in N_P \cup N_D, \tag{6}$$

$$\sum_{i \in N \setminus \{2n+1\}} \sum_{m=1}^{|T_{if}|} x_{if}^{km} - \sum_{j \in N \setminus \{0\}} \sum_{m=1}^{|T_{fj}|} x_{fj}^{km} = 0 \qquad \forall f, i, j \in N_P \cup N_D, \forall k \in K, \tag{7}$$

$$\sum_{i \in N \setminus \{2n+1\}} \sum_{m=1}^{|T_{ij}|} x_{ij}^{km} = y_j^k \qquad \forall j \in N_P \cup N_D, \tag{8}$$

$$\sum_{j \in N \setminus \{0\}} \sum_{m=1}^{|T_{ij}|} x_{ij}^{km} - \sum_{j \in N \setminus \{0\}} \sum_{m=1}^{|T_{n+i,j}|} x_{n+i,j}^{km} = 0 \qquad \forall i \in N_P, \forall k \in K, \tag{9}$$

$$t_i^k + \tau_{ij}(t_i^k) + s_j \le t_j^k + M(1 - x_{ij}^{km}) \qquad \forall i, j \in N_P \cup N_D, m = 1, \ldots, \mid T_{ij} \mid, \forall k \in K, \tag{10}$$

$$Q_i^k + q_j \le Q_j^k + M(1 - x_{ij}^{km}) \qquad \forall i, j \in N, m = 1, \ldots, \mid T_{ij} \mid, \forall k \in K, \tag{11}$$

$$t_i^k = \sum_{j \in N \setminus \{0\}} \sum_{m=1}^{|T_{ij}|} t_{ij}^{km} \qquad \forall i \in N \setminus \{2n+1\}, \forall k \in K, \tag{12}$$

$$t_{n+i}^k \ge t_i^k \qquad \forall i \in N_P, \forall k \in K, \tag{13}$$

$$w_m x_{ij}^{km} \leq t_{ij}^{km} \leq w_{m+1} x_{ij}^{km} \qquad \forall i, j \in N, m = 1, \ldots, \mid T_{ij} \mid -1, \forall k \in K, \tag{14}$$

$$(e_i + s_i) y_i^k \leq t_i^k \leq (l_i + s_i) y_i^k \qquad \forall i \in N, \forall k \in K, \tag{15}$$

$$\max\{0, q_i\} \leq Q_i^k \leq \min\{Q, Q + q_i\} \qquad \forall i \in N, \forall k \in K, \tag{16}$$

$$x_{ij}^{km}, y_i^k \in \{0, 1\} \qquad \forall i, j \in N, \forall m = 1, \ldots, \mid T_{ij} \mid, \forall k \in K. \tag{17}$$

The objective function (3) measures the difference between profits collected and costs incurred due to travel and vehicle use. Constraints (4) and (5) guarantee that if vehicle k is used, its route starts from and ends at the depot. Constraints (6) ensure that every request is served at most once. Constraints (7) are classical flow conservation constraints. Constraints (8) and (9) ensure both the pickup and delivery nodes are visited when a request is served, and by the same vehicle. Given a vehicle's route, constraints (10) ensure its departure times from locations respect travel and service times. Constraints (11) compute the quantity carried by the vehicle throughout its route. Constraints (12) calculate the departure time for vehicle k from location i. Constraints (13) ensure that the vehicle visits the pickup node for a request before its delivery node. Constraints (14) ensure that the departure time for a vehicle from a location is associated with the correct time zone. Constraints (15) ensure that a vehicle departs at a location after starting service within the time window. Constraints (16) ensure that the vehicle capacity is respected. The last set of constraints define decision variables and their domains.

3.1. *Formulation for* Ω(Fix, All)

This is the most restrictive variant, as it requires all customer requests to be served and each vehicle route to depart from the depot at time 0. We model these restrictions by adding constraints to the model presented above, which in turn, simplifies the objective function. The resulting integer program is as follows:

$$Obj_{\Omega(Fix,All)} = \max \sum_{i \in N_P} p_i - \sum_{k \in K} (c_t t_{2n+1}^k + Z y_0^k) \tag{18}$$

subject to (4), (5), (7)–(17), with the following extra constraints:

$$\sum_{k \in K} y_j^k = 1 \qquad \forall j \in N_P \cup N_D \tag{19}$$

$$t_0^k = 0 \qquad \forall k \in K \tag{20}$$

Constraints (19) ensure that each request is served, while constraints (20) ensure that each route departs from the depot at time 0. Due to the fixed variables the objective function reduces to (18).

3.2. *Formulation for* Ω(Flex, All)

This variant requires all customer requests to be served, but allows for flexibility in when a route departs from the depot. As a result, we formulate the problem as follows:

$$Obj_{\Omega(Flex,All)} = \max \sum_{i \in N_P} p_i - \sum_{k \in K} [c_t (t_{2n+1}^k - t_0^k) + Z y_0^k] \tag{21}$$

subject to (4), (5), (7)–(17) and (19).

3.3. *Formulation for* Ω(Fix, Prof)

This variant does not require that each customer request be served, but does require that each route departs from the depot at time 0. The resulting integer program is as follows:

$$Obj_{\Omega(Fix,Prof)} = \max \sum_{k \in K} \left(\sum_{i \in N_P} p_i y_i^k - c_t t_{2n+1}^k - Z y_0^k \right) \tag{22}$$

subject to (4)–(17) and (20).

3.4. Relation between objective function values of the variants

As these four optimization problems are related, and can each be solved for the same instance, we can determine relationships between their optimal objective function values. For example, as $\Omega(Flex, Prof)$ is the least restricted variant, for a given instance, we have that $Obj_{\Omega(Flex,\ Prof)}$ will be the largest optimal objective function value. Conversely, as $\Omega(Fix, All)$ is the most restricted variant, we have that $Obj_{\Omega(Fix,\ All)}$ will be the smallest optimal objective function value for a given instance. Similarly, one can observe that $Obj_{\Omega(Fix,\ All)} \leq Obj_{\Omega(Fix,\ Prof)}$ and $Obj_{\Omega(Fix,\ All)} \leq Obj_{\Omega(Flex,\ All)}$. While statements cannot be made regarding the relationship between $Obj_{\Omega(Fix,\ Prof)}$ and $Obj_{\Omega(Flex,\ All)}$ for general instances, we will analyze the relationship between these two quantities in our computational study.

4. Solution approach

It is well known that compact formulations of routing problems, such as the one presented above, have weaker linear relaxations than extended formulations, wherein each variable models an entire vehicle route. The stronger bounds that result from using an extended formulation can greatly speed up the solution of the integer program, but come at the expense of a large number of variables. In particular, such extended formulations often involve variable sets that are either too large to enumerate in a reasonable run-time, or to fit in run-time memory, or both. However, using column generation, our algorithm begins with a model that includes a small portion of the variable set and then generates more variables (columns) when there is evidence they may appear in an optimal solution. We refer interested readers to Desaulniers et al. (2006) for a detailed description of column generation. Therefore, this algorithm can retain the benefits of using an extended formulation without the challenges presented by the ensuing variable set.

As a result, we propose a solution framework that is based on branch-and-price (Barnhart et al., 1998), a branch-and-bound-based technique for solving integer programs that relies on generating variables dynamically through column generation. We employ several techniques to improve the performance of the algorithm, including limited-memory subset-row cuts and route enumeration, which we will detail later. We next present the general structure of our framework.

- Step 1. Choose an unprocessed node from the branch-and-bound tree. If the difference between the upper bound associated with this node and the current best lower bound is below the predefined tolerance, fathom the node and return to Step 1. Otherwise, continue.
- Step 2. Solve the linear relaxation of the restricted master problem (RMP), (see Section 4.1).
- Step 3. Use the heuristic pricing procedure (see Section 4.3.3) to search for columns with positive reduced cost. If such columns are found, add them to RMP and go back to Step 2. Otherwise, continue.
- Step 4. Use the exact pricing procedure (see Sections 4.3.1 and 4.3.2) to search for columns with positive reduced cost. If new columns are found, add them to the RMP and go to Step 2. Otherwise, the optimal value of RMP is an upper bound on the optimal objective function value of the problem. If this upper bound is below the current best lower bound, fathom the node and return to Step 1. Otherwise, continue.
- Step 5. Generate $lm - SRCs$ cuts (see Section 4.2). If any violated cuts are found, add them to the RMP and go to Step 2. Otherwise, continue.
- Step 6. If the solution of the linear relaxation of the RMP is fractional, perform route enumeration (see Section 4.5) to generate extra columns. Update the lower bound by solving the RMP as an integer program.
- Step 7. Solve the linear relaxation of the RMP and branch (see Section 4.4) if the solution is fractional, adding the resulting child nodes to the set of unprocessed branch nodes. Mark the current node as processed and go back to Step 1.

We note that executing the exact pricing procedure (Step 4) when the heuristic procedure does not yield any columns with positive reduced cost ensures that the approach will solve the linear relaxation of the master problem at a node of the branch-and-bound tree. Much of this framework can be used, unchanged, to solve any of the four variants we consider. However, in the following sections, we will highlight the steps that need to be adapted when solving a specific variant.

4.1. Set packing formulation

In this section, we describe how we reformulate the integer program presented in Section 3 to a set packing problem. To do so, we let Ψ represent the set of feasible routes satisfying Constraints (7)–(17). We associate with each route $r \in \Psi$ the value v_r, which represents the profits generated by that route minus the costs it incurs:

$$v_r = \sum_{i \in r} p_i - c_t(\delta^r_{2n+1}(t^r_0) - t^r_0) - Z \tag{23}$$

Here, t^r_0 is the optimal departure time from the depot (0) on route r and $\delta^r_{2n+1}(t^r_0)$ is the resulting arrival time at the depot. For the variants $\Omega(Fix, All)$ and $\Omega(Fix, Prof)$, we set $t^r_0 = 0$.

For the two variants with flexible departure time, $\Omega(Flex, All)$ and $\Omega(Flex, Prof)$, we must solve for t^r_0. To do so, we let u_i denote the node at position i in route r (u_{i-1} is then the predecessor node of u_i on that route). We define $\delta^r_{u_i}(t)$ to be a "ready time" function, as it will indicate when a vehicle following route r can depart from u_i, assuming it departs from the depot at time t. Because travel times adhere to the FIFO property, this ready time function is nondecreasing in t, and can be calculated recursively for each node in the route as follows:

$$\delta^r_{u_i}(t) = \begin{cases} t & \text{if } i = 0, \\ \max\{e_{u_i} + s_{u_i}, \delta^r_{u_{i-1}}(t) + \tau_{u_{i-1},u_i}(\delta^r_{u_{i-1}}(t)) + s_{u_i}\} & \text{otherwise.} \end{cases} \tag{24}$$

The duration of the route, given that it departs from node 0 at time t can be calculated as $\delta^r_{2n+1}(t) - t$. As a result, the departure time from the depot that leads to the shortest route duration can be calculated as follows:

$$t^r_0 = argmin_{t \in T}\{\delta^r_{2n+1}(t) - t\} \tag{25}$$

We also associate with route $r \in \Psi$ the parameter a_{ir}, $i \in N_P$, which denotes whether route r visits node i. Finally, we let the binary variable w_r indicate whether route r is chosen. The resulting set packing formulation is as follows:

$$v(MP) = \max \sum_{r \in \Psi} v_r w_r \tag{26}$$

subject to

$$\sum_{r \in \Psi} a_{ir} w_r \leq 1 \qquad \forall i \in N_P \tag{27}$$

$$\sum_{r \in \Psi} w_r \leq K \tag{28}$$

$$w_r \in \{0, 1\} \qquad \forall r \in \Psi \tag{29}$$

Constraints (27) ensures that each request is visited at most once. For the Ω(*Fix, All*) and Ω(*Flex, All*) variants, this constraint is changed to an equality constraint. Constraint (28) limits the number of vehicles used to K.

As noted, because there may be a large number of routes (i.e., $|\Psi|$ is large) for many instances, we dynamically generate routes, r, to add to Ψ with column generation (Desaulniers et al., 2006). To do so, we repeatedly solve a restricted master problem, $RMP(\Psi')$, based on a subset, $\Psi' \subset \Psi$, of feasible routes. The subset Ψ' is initialized with a set of routes wherein each route visits a single pair of requests. At each iteration of the algorithm, the linear relaxation of $RMP(\Psi')$ is solved and the corresponding dual variables are recorded. We let π_i, $i \in N_P$, denote the nonnegative dual variable associated with constraint (27) and π_0 the nonnegative dual variable associated with constraint (28). The algorithm then solves a pricing problem to determine whether there are variables, $w_r, r \in \Psi/\Psi'$, that are not yet considered in $RMP(\Psi')$ and have positive reduced cost. In this context, the reduced cost of a variable r is calculated as:

$$\bar{v}_r = v_r - \sum_{i \in N_P \cup \{0\}} a_{ir}\pi_i = \left(\sum_{i \in r} p_i - c_t(\delta^r_{2n+1}(t^r_0) - t^r_0) - Z_r \right) - \sum_{i \in N_P \cup \{0\}} a_{ir}\pi_i \tag{30}$$

We note that whenever we have an upper bound, $\bar{\bar{v}}_r$ on the route with the largest reduced cost, we can derive an upper bound on the optimal objective function value of the original problem (Lübbecke, 2010). Specifically, we let $v(RMP)_{LPR}$ denote the optimal value of the linear relaxation of the RMP. $v(MP)_{LPR}$ is defined similarly. Given that at most k routes can be chosen, we have:

$$v(RMP)^{dual}_{LPR} = v(RMP)_{LPR} + k * \bar{\bar{v}} \geq v(MP)_{LPR} \tag{31}$$

This bound, $v(RMP)^{dual}_{LPR}$, provides a stopping criteria for column generation other than the absence of columns with positive reduced cost. Speficially, if $(v(RMP)^{dual}_{LPR} - v(RMP)_{LPR})/v(RMP)_{LPR}$ is within some tolerance (i.e., 1% in our experiments), we can terminate the column generation procedure.

4.2. Valid inequalities

Extended formulations of a problem can be further strengthened with the addition of valid inequalities, or, *cuts*. Jepsen et al. (2008) applied the Chvatal-Gomory rounding procedure to Constraints (27) to develop the following family of valid inequalities for VRPTWs.

Specifically, given a base set $C \subseteq N_P$ and a multiplier γ, $0 < \gamma < 1$, we define the following $(C, \gamma)-$Subset Row Cut (*SRC*) as follows:

$$\sum_{r \in \Psi} \lfloor \gamma \sum_{i \in C} a_{ir} \rfloor w_r \leq \lfloor \gamma |C| \rfloor \tag{32}$$

While these cuts have been shown (Jepsen et al., 2008) to strengthen the linear relaxation, as they are defined on the route variables they can make the pricing subproblem harder to solve with dynamic programming. Specifically, each cut adds a dimension to the labels that must be stored. Recently, Pecin et al. (2014) proposed limited-memory *SRCs*, that, while weaker than the *SRCs*, have a reduced impact on the pricing subproblem. As such, while the use of these limited-memory cuts, in comparison to those of Jepsen et al. (2008), may lead to more time spent generating cuts, less time solving the pricing problem, and an overall reduction in computation time.

The definition of the limited memory (C, M, γ)–Subset Row Cut ($lm-SRC$) requires an additional set M, $C \subset M \subset N_P$. The resulting cut is:

$$\sum_{r\in\Psi} \alpha(C, M, \gamma, r) w_r \leq \lfloor \gamma |C| \rfloor \tag{33}$$

where α is a function of C, M, γ, r computed by the Algorithm 1.

Algorithm 1: Computing $\alpha(C, M, \gamma, r)$ of w_r in a $lm-SRC$.

```
input : A feasible route $r$
output: A coefficient $\alpha$
1  $\alpha \leftarrow 0, state \leftarrow 0$
2  for (each node $i$ in $N_P$) do
3      if ($i \notin M$) then
4          $state \leftarrow 0$
5      else
6          if ($i \in C$) then
7              $state \leftarrow state + \gamma$
8              if ($state > 1$) then
9                  $\alpha \leftarrow \alpha + 1$
10                 $state \leftarrow state + 1$
```

When $M = N_P$, the function α will return $\lfloor \gamma \Sigma_{i\in C} a_{ir} \rfloor$ and the $lm-SRC$ will be identical to a SRC. On the other hand, when M is not equal to N_P, the $lm-SRC$ may be a weakening of its corresponding SRC. This happens because the variable *state* in the function is set to 0 once the route r leaves M.

The function $\alpha(C, M, \gamma, r)$ also impacts how the $lm-SRCs$ should be taken into account in the pricing problem discussed later. Specifically, for each $lm-SRC$, each label needs to store its state in the corresponding partial paths. However, the coefficient does not need to be stored in the label. Instead, whenever a label extension causes an increment of the coefficient of an $lm-SRCs$ according to function α, the value of its dual variable is immediately added to the reduced cost of the generated label. Moreover, the number of possible states of a $lm-SRC$ depends on its γ. In this paper, we only consider $lm-SRCs$ defined for subsets C with cardinality 3. In this case, $\gamma = 1/2$ and the state can be only 0 or 1/2. Therefore, it can be represented by a single bit.

The $lm-SRCs$ have potential advantage over the classical $SRCs$ due to their reduced impact on the pricing problem when $|M| < < |N_P|$. In the following, we introduce the strategy for separating $lm-SRCs$ such that small memory sets can be obtained. First, a violated $SRC(C, \gamma)$ need to be identified. Then, a minimal set M is determined such that $lm-SRC(C, M, \gamma)$ has the same violation. For example, take a given fractional solution with paths that visit $C = \{1, 2, 3\}$ at least twice. In this example they are: $r_1 = (0-1-4-2-5-\ldots-0)$ with $\lambda_1 = 0.5$, $r_2 = (0-3-6-2-5-\ldots-0)$ with $\lambda_2 = 0.5$ and $r_3 = (0-3-7-1-4-\ldots-0)$ with $\lambda_3 = 0.5$. The $SRC(C, 1/2)$ $\lambda_1 + \lambda_2 + \lambda_3 \leq 1$ has a violation of 0.5. A minimal set M that yields a $lm-SRC(C, M, 1/2)$ with the same violation can be $M = C \cup \{4\} \cup \{6\} \cup \{7\}$. Moreover, like Cherkesly et al. (2015) and Pecin et al. (2014), we also limit the usage of $lm-SRCs$ to prevent running out of memory when solving the pricing problem.

4.3. The pricing problem

In this section, we present the algorithm used to solve the pricing problem, which aims to find a route, r that minimizes $\bar{v}_r$ of function (30) while taking into consideration the constraints that model feasibility. The pricing problem can be seen as a variant of the elementary shortest path problem, which is known to be NP-hard. We solve this problem with the tailored labeling algorithm proposed by Sun et al. (2018).

In this algorithm, a label L is used to record the information associated with a path that starts at the depot and ends at node $v(L)$. The label includes information regarding the requests already visited, the cumulative reduced cost, and the ready time function. Starting from the initial label L_0 associated with the depot, the algorithm propagates labels toward the destination depot with a feasible extension process. Dominance tests are also employed to eliminate unpromising labels.

We next briefly present the tailored labeling algorithm introduced by Sun et al. (2018), which is used to solve the pricing problem in our branch-and-price framework. Then, we explain how to incorporate the dual information associated with generated $lm-SRCs$ into this tailored labeling algorithm. We finish with a discussion of a pricing heuristic we use to accelerate the branch-cut-and-price algorithm.

4.3.1. Pricing without lm-SRCs

In this section, we introduce the forward tailored labeling algorithm based on Sun et al. (2018). The interested reader is referred to Sun et al. (2018) for the backward version and related bidirectional search techniques. For each forward label L_f, we use the same notation as introduced in Sun et al. (2018), which is explained as the following:

$p(L_f)$	*	The partial path of label L_f.
$v(L_f)$	*	The last node of the partial path $p(L_f)$.
$L^{-1}(L_f)$	*	The parent label from which L_f originates by extending it with $v(L_f)$.
$O(L_f)$	*	The set of onboard requests in the partial path $p(L_f)$
$U(L_f)$	*	The set of unreachable requests in the partial path $p(L_f)$.
$q(L_f)$	*	The load of the vehicle after visiting the last node $v(L_f)$.
$\delta_{L_f}(t)$	*	The ready time function at $v(L_f)$
$r(L_f)$	*	The overall profits collected by serving the requests visited on the partial path $p(L_f)$.
$\pi(L_f)$	*	The sum of the dual values associated with the pickup nodes visited on the partial path $p(L_f)$ and dual value π_0 associated with constraint (28).

Only the items marked with a * are stored in the label. A request is said to be on-board if it has been started, but not completed, i.e., the request has been picked up but not delivered to its delivery mode. A request R_i is said to be unreachable if i has already been visited, or if traveling either directly or indirectly from $v(L_f)$ to i violates the time window at pickup node i. Therefore, $O(L_f)\subseteq U(L_f)$. The ready time function $\delta_{L_f}(t)$ is piece-wise linear and represents the ready time at $v(L_f)$ if the vehicle departed at the origin depot at t and reached $v(L_f)$ through partial path $p(L_f)$. For $\Omega(\mathit{Flex, All})$ and $\Omega(\mathit{Flex, Prof})$, we need to store this function for each generated label, while for $\Omega(\mathit{Fix, All})$ and $\Omega(\mathit{Fix, Prof})$ we only need to store $\delta_{L_f}(0)$, which is the earliest ready time at $v(L_f)$ as the departure time at the origin depot is imposed to be 0. The partial path can be deduced from iteratively checking the last node visited in the parent label of which the label was an extension.

Given a label L'_f, its extension along an arc $(v(L'_f), j)$ is feasible only if it satisfies the following conditions:

$$\delta_{L'_f}(0) + \tau_{v(L'_f),j}(\delta_{L'_f}(0)) + s_j \le \min\{t_m, l_j + s_j\} \quad \wedge \quad j \in N \setminus \{0\} \tag{34}$$

$$q(L_f) + q_j \le Q \quad \wedge \quad j \in N \setminus \{0\} \tag{35}$$

Condition (34) stipulates that only if node j can be reached within its time window and the extension to node j takes place before t_m is exceeded, the extension to node j is allowed to be performed. Condition (35) ensures that the label can only be extended to node j if the remaining capacity is enough to deal with the load of node j. Besides, L'_f and j need satisfy one of the following three conditions:

$$j \notin U(L'_f) \quad \wedge \quad j \in N_P \tag{36}$$

$$j - n \in O(L'_f) \quad \wedge \quad j \in N_D \tag{37}$$

$$O(L'_f) = \emptyset \quad \wedge \quad j = 2n+1 \tag{38}$$

Condition (36) states that if node j is a pickup node, it should have never been visited before on the route. Condition (37) shows that if j is a delivery node then its corresponding pickup node $j-n$ should already been visited. Condition (38) states that if j is the end depot, then all visited requests should have been completed. In the presence of those conditions, we only keep elementary paths that satisfy precedence constraint (10). If the extension along the arc (L'_f, j) is feasible, then a new label L_f is created. The information in label L_f is updated as follows:

$$L^{-1}(L_f) = L'_f \tag{39}$$

$$v(L_f) = j \tag{40}$$

$$\delta_{L_f}(t) = \max\{e_j + s_j, \delta_{L'_f}(t) + \tau_{L'_f,j}(\delta_{L'_f}(t)) + s_j\} \tag{41}$$

$$q(L_f) = q(L'_f) + q_j \tag{42}$$

$$r(L_f) = \begin{cases} r(L'_f) + r_j & \text{if } j \in N_P, \\ r(L'_f) & \text{otherwise.} \end{cases} \tag{43}$$

$$\pi(L_f) = \begin{cases} \pi(L_f^{'}) + \pi_j & \text{if } j \in N_P, \\ \pi(L_f^{'}) & \text{otherwise.} \end{cases} \tag{44}$$

$$O(L_f) = \begin{cases} O(L_f^{'}) \cup \{j\} & \text{if } j \in N_P, \\ O(L_f^{'}) \setminus \{j-n\} & \text{if } j \in N_D. \end{cases} \tag{45}$$

$$U(L_f) = \begin{cases} U(L_f^{'}) \cup \{j\} & \text{if } j \in N_P, \\ U(L_f^{'}) & \text{otherwise.} \end{cases} \tag{46}$$

Eqs. (39)–(43) set the parent label, the last visited node, the ready time function, the load, and the collected profits of the new label, respectively. Eq. (45) updates the set of onboard requests $O(L_f)$ and Eq. (46) updates the set of unreachable requests $U(L_f)$. Similar to the idea of Feillet et al. (2004) and Sun et al. (2018), the unreachable request set $U(L_f)$ is further enriched by adding requests of which the pickup node is unreachable from $v(L_f)$.

Before the dominance criterion is introduced some definitions need to be provided. The first one is the definition of interval I as explained in Sun et al. (2018).

$$I = (-\infty, \max(dom(L_f^1)) - \max(dom(L_f^2))) \tag{47}$$

Based on I, we define a real number $\phi(L_f^1, L_f^2)$,

$$\phi(L_f^1, L_f^2) = \max\{x \in I : \delta_{L_f^1}(\max\{0, t+x\}) \leq \delta_{L_f^2}(t), \forall t \in dom(L_f^2)\} \tag{48}$$

This function describes for the partial path represented by label L_f^1, how much the departure time can be postponed (when $\phi(L_f^1, L_f^2)$ is positive) or arranged in advance (if $\phi(L_f^1, L_f^2)$ is negative), compared to the departure time of partial path $p(L_f^2)$, such that $v(L_f^1)$ is reached at the same time or earlier via partial path $p(L_f^2)$ than via partial path $p(L_f^2)$.

A label L_f^1 dominates a label L_f^2 if every feasible extension of $p(L_f^2)$ yields a complete route with reduced cost smaller than the feasible route generated by using the same extension into $p(L_f^1)$. The following six conditions in Proposition 4.1, together, are sufficient to ensure such dominance:

Proposition 4.1. *Label L_f^2 is dominated by label L_f^1 if*

1. $v(L_f^1) = v(L_f^2)$
2. $U(L_f^1) \subseteq U(L_f^2)$
3. $O(L_f^1) = O(L_f^2)$
4. $\delta_{L_f^1}(0) \leq \delta_{L_f^2}(0)$
5. $r(L_f^1) - \pi(L_f^1) \geq r(L_f^2) - \pi(L_f^2) - \phi(L_f^1, L_f^2)$
6. $q(L_f^1) \leq q(L_f^2)$

Proof of Proposition 4.1. To prove this proposition, we consider two labels L_f^1 and L_f^2, each of which satisfies the six conditions of Proposition 4.1. We will prove two claims regarding these labels, which will in turn prove this proposition:

1. Any feasible extension L of $p(L_f^2)$ that yields a complete route, $p(L_f^2 \bigoplus L)$, is also a feasible extension of $p(L_f^1)$, and yields a complete route $p(L_f^1 \bigoplus L)$, and,
2. For any feasible extension L that led to $p(L_f^1 \bigoplus L)$ and $p(L_f^2 \bigoplus L)$, the reduced cost $\bar{v}(L_f^1 \bigoplus L)$ is not smaller than the reduced cost $\bar{v}(L_f^2 \bigoplus L)$.

Regarding claim (1), we must show that $p(L_f^1 \bigoplus L)$ does not carry more load than the capacity of the vehicle, is elementary, and does not violate any time windows. Regarding vehicle capacity, we note that due to condition 6, the load associated with path $p(L_f^1 \bigoplus L)$ will not exceed vehicle capacity as it did not on path $p(L_f^2 \bigoplus L)$. We can conclude that $p(L_f^1 \bigoplus L)$ is elementary because conditions 2 and 3 ensure that all nodes visited in $p(L_f^1)$ are either nodes visited in $p(L_f^2)$ or nodes which could not be reached by any extension of label L_f^2. Thus, as $p(L_f^2 \bigoplus L)$ is elementary, so is $p(L_f^1 \bigoplus L)$. Turning to time windows, we observe that condition 4, coupled with the presumption that travel times adhere to the FIFO property implies that $p(L_f^1 \bigoplus L)$ will not violate any time windows. For example by departing at 0, the vehicle arrives at $v(L_f^1)$ at time $\delta_{L_f^1}(0)$ which is by condition 4 smaller than or equal to $\delta_{L_f^2}(0)$. Therefore, a departure at 0 over path $p(L_f^1)$ does not violate any time windows.

Regarding claim (2), consider an extension, L, of $p(L_f^2)$ that yields a complete route (i.e., $v(L_f^2 \bigoplus L) = 2n+1$. Letting $L_f^{1*} = L_f^1 \bigoplus L$ and $L_f^{2*} = L_f^2 \bigoplus L$, we will show that $\bar{v}(L_f^{1*}) \geq \bar{v}(L_f^{2*})$. To do so, we let $t_0^2 = argmin_{t \in dom(L_f^{2*})}\{\delta_{L_f^{2*}}(t) - t\}$ represent the optimal departure time from the depot for path $p(L_f^{2*})$ and $r(L)$ the sum of the profits associated with the nodes visited along the path $p(L)$ associated with the extension, L. Given this, we have that the reduced cost of the path is:

$$\begin{aligned}\overline{v}(L_f^{2*}) &= r(L_f^{2*}) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2) - \pi(L_f^{2*}) - Z \\ &= r(L_f^2) + r(L) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2) - (\pi(L_f^2) + \pi(L)) - Z \end{aligned} \tag{49}$$

Next, consider the path $p(L_f^{1*})$, which departs from the depot at time $t_0^1 = \max\{0, t_0^2 + \phi(L_f^1, L_f^2)\}$. We note that the time t_0^1 is a feasible departure time for label L_f^{1*} because a departure time of 0 is always possible (as the extension of L_f^1 by L is feasible) and the definition of $\phi(L_f^1, L_f^2)$ in Eq. (48) implies that $t_0^2 + \phi(L_f^1, L_f^2)$ belongs to $dom(L_f^1)$ when it is nonnegative. This departure time t_0^1 ensures that we reach node $v(L_f^1)$ at time $\delta_{L_f^2}(t_0^2)$ or earlier, meaning we have the following inequality:

$$\delta_{L_f^1}(t_0^1) \leq \delta_{L_f^2}(t_0^2). \tag{50}$$

Moreover, this value of t_0^1 enables us to show that $\overline{v}(L_f^{2*})$ is a lower bound on $\overline{v}(L_f^{1*})$:

$$\overline{v}(L_f^{1*}) \geq r(L_f^{1*}) - (\delta_{L_f^{1*}}(t_0^1) - t_0^1) - \pi(L_f^{1*}) - Z \tag{51}$$

$$= r(L_f^1) + r(L) - (\delta_{L_f^{1*}}(t_0^1) - \max\{0, t_0^2 + \phi(L_f^1, L_f^2)\}) - (\pi(L_f^1) + \pi(L)) - Z \tag{52}$$

$$\geq (r(L_f^1) - \pi(L_f^1)) + (r(L) - \pi(L)) + (\delta_{L_f^{2*}}(t_0^2) - \max\{0, t_0^2 + \phi(L_f^1, L_f^2)\}) - Z \tag{53}$$

$$\geq (r(L_f^1) - \pi(L_f^1)) + (r(L) - \pi(L)) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2 - \phi(L_f^1, L_f^2)) - Z \tag{54}$$

$$\geq (r(L_f^2) - \pi(L_f^2) - \phi(L_f^1, L_f^2)) + (r(L) - \pi(L)) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2 - \phi(L_f^1, L_f^2)) - Z \tag{55}$$

$$= r(L_f^2) + r(L) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2) - (\pi(L_f^2) + \pi(L)) - Z \tag{56}$$

$$= r(L_f^{2*}) - (\delta_{L_f^{2*}}(t_0^2) - t_0^2) - \pi(L_f^{2*}) - Z = \overline{v}(L_f^{2*}) \tag{57}$$

We note that inequality (53) is derived in part from inequality (50). Inequality (54) is derived in part from the fact that $\forall x \in R : -\max\{0, x\} \leq -x$. Finally, inequality (55) relies upon condition 5 of Proposition 4.1. □

4.3.2. Pricing algorithm with lm-SRCs

When pricing in the presence of lm-SRCs, each label must have an additional dimension for each $lm - SRC$ with non-zero dual value in the current restricted master problem solution (hereafter called an active $lm - SRC$). To discuss this in more detail, as well as the corresponding dominance criteria, we define the following notation:

$E(L_f)$	*	The vector of states associated with the $lm - SRCs$.
σ_h	*	The dual variable associated with $lm - SRC$ h.

For an active $lm - SRC$ h, $E(L_f)[h]$ will first be assigned the value $E(L^{-1}(L_f))[h]$. However, if $v(L_f) \notin M(h)$, it is set to zero. Then, if $v(L_f) \notin C$, it is increased by γ. Finally, $E(L_f)[h]$ is decremented by 1 if it becomes larger than or equal to 1.

In addition, the calculation of $\pi(L_f)$ also sums over the duals associated with active $lm - SRC$s. We denote this quantity as $\sum_{h \in DEC_{L_f}} \sigma_h$, where DEC_{L_f} is subset of the active $lm - SRCs$ whose states were decremented by one when calculating L_f.

Given these definitions, the following six conditions, together, are sufficient to guarantee dominance:

Proposition 4.2. *Label L_f^2 is dominated by label L_f^1 if*

1. $v(L_f^1) = v(L_f^2)$
2. $U(L_f^1) \subseteq U(L_f^2)$
3. $O(L_f^1) = O(L_f^2)$
4. $\delta_{L_f^1}(0) \leq \delta_{L_f^2}(0)$
5. $r(L_f^1) - \pi(L_f^1) \geq r(L_f^2) - \pi(L_f^2) - \phi(L_f^1, L_f^2) - \sum_{h \in H : E(L_f^1)[h] > E(L_f^2)[h]} \sigma_h$
6. $q(L_f^1) \leq q(L_f^2)$

The conditions of Proposition 4.2 are analogous to those of Proposition 4.1. Specifically, only condition 5 differs, in that the expression on the right-hand-side reflects the duals associated with the currently active *lm* – *SRC*s. As such, the proof of Proposition 4.2 is nearly identical to that of Proposition 4.1 and we omit it for the sake of brevity.

We note that the traditional *SRCs* have a high potential for disrupting the label dominance by making Condition 5 much less likely to be satisfied. The limited-memory mechanism mitigates that difficulty. As the state of given cut h is reset to 0 whenever a route visit a pickup node which is not included in $M(h)$, its dual variable σ_h is involved in fewer label comparisons.

4.3.3. Heuristic pricing

To accelerate the performance of the branch-and-price algorithm, a heuristic is used to quickly search for routes with positive reduced cost. The heuristic speeds up the search by using dominance criteria that are "too strong" and may lead to the pruning of optimal solutions to the pricing problem. Specifically, the heuristic replaces $O(L_f^1) = O(L_f^2)$ with $O(L_f^1) \subseteq O(L_f^2)$, which is less restrictive and may lead to the pruning of more partial paths. However, these criteria are invalid for the problems we study as they assume the triangle inequality holds, something that is not necessarily true in the problems we study. In summary, the dominance conditions considered by the heuristic are as follows:

Heuristic pricing rule. Label L_f^2 is dominated by label L_f^1 if

1. $v(L_f^1) = v(L_f^2)$
2. $U(L_f^1) \subseteq U(L_f^2)$
3. $O(L_f^1) \subseteq O(L_f^2)$
4. $r(L_f^1) - \pi(L_f^1) \geq r(L_f^2) - \pi(L_f^2)$
5. $q(L_f^1) \leq q(L_f^2)$

4.4. Branching rules

While column generation is a scheme for solving linear programs, by embedding it within a branch-and-bound scheme it can be used to solve an integer program. Such a scheme requires defining how to sub-divide the feasible region associated with a node in a branch-and-bound tree when a solution to the linear relaxation of the RMP associated with that node is not integral (and hence not a solution to MP). Such a sub-division yields "child nodes" of the current node. We next present, in the order in which they are applied, the branching rules used by the algorithm for creating these child nodes. We note that all rules create two such child nodes.

- The first branching rule concerns the value of $m = \sum_{r\in\Psi} w_r$, the number of vehicles used in the solution. If this value is fractional, we impose $\Sigma_{r\in\Psi} w_r \leq \lfloor m \rfloor$ and $\Sigma_{r\in\Psi} w_r \geq \lceil m \rceil$ on the other branch. In the pricing problem, a dual variable associated with this new constraint in the master is attached to the depot.
- The second branching rule concerns whether a request is served. If $\Sigma_{r\in\Psi} a_{ir} w_r$ is fractional for some $i \in N_P$, we impose on one branch $\sum_{r\in\Psi} a_{ir} w_r = 0$ and $\sum_{r\in\Psi} a_{ir} w_r = 1$ on the other branch. For branch $\sum_{r\in\Psi} a_{ir} w_r = 0$, the pickup node i will be set as unreachable in the pricing problem. For branch $\sum_{r\in\Psi} a_{ir} w_r = 1$, a dual variable associated with this new constraint in the master will be attached to the pickup node i.
- The third branching rule considers an expression of the solution to the linear relaxation of RMP in terms of arc variables x_{ij}. Specifically, given the solution w^* to the linear relaxation of RMP, it calculates $x^*_{ij} = \sum_{r\in\Psi' :(i,j)\in r} w^*_r$. It then looks for pairs (i, j), $i, j \in N$ such that $x^*_{ij} + x^*_{ji}$ is close to 0.5. The algorithm then imposes two branches: (1) $x^*_{ij} + x^*_{ji} \leq \lfloor x^*_{ij} + x^*_{ji} \rfloor$, and, (2) $x^*_{ij} + x^*_{ji} \geq \lceil x^*_{ij} + x^*_{ji} \rceil$. For the node associated with $x^*_{ij} + x^*_{ji} \leq \lfloor x^*_{ij} + x^*_{ji} \rfloor$, the arc (i, j) will be removed from the graph constructed when solving the pricing problem. For the other branch, all arcs that emanate from i other than (i, j) are removed from the graph constructed when solving the pricing problem. In addition, $\sum_{r\in\Psi} a_{ir} w_r = 1$ is imposed on this branch. Thus, in the pricing problem, a dual variable will be attached to the pickup node i.

The proposed branching scheme is the same for both the Ω(*Fix, Prof*) and Ω(*Flex, Prof*), while, for Ω(*Fix, All*) and Ω(*Flex, All*), only the first rule and the last rule are applicable, since all the requests are required to be visited exactly once.

We note that branching constraints that restrict the number of vehicles used or whether a request is served can yield an infeasible *RMP*. To ensure the RMP remains feasible, we add to the formulation a variable κ with a large (in absolute value) and negative objective function coefficient. This variable is then included in the constraints representing a request that must be served and the number of vehicles used. The resulting constraints are $\sum_{r\in\Psi} a_{ir} w_r + \kappa = 1$, $\sum_{r\in\Psi} w_r - \kappa \leq \lfloor m \rfloor$, and $\sum_{r\in\Psi} w_r + \kappa \geq \lceil m \rceil$.

Finally, we note that the second and third rules may yield multiple candidates for branching. Namely, there may be multiple requests that are partially served (the second rule) and/or there may be multiple arcs that are fractionally chosen (the third rule). Given a rule that yields multiple candidates, we choose one by employing strong branching (Røpke, 2012).

Table 1
Characteristics of the instances used in computational study.

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

4.5. Route enumeration

For the algorithm to terminate with a provably (near-) optimal solution to the MP, it must find such a solution and produce a dual bound that guarantees its quality. The cuts presented above are used to reduce the time needed to produce such a dual bound. To reduce the time required to find a high-quality primal solution, we employ a route enumeration technique (Baldacci et al., 2008). This technique generates routes, outside of the context of solving the pricing problem, that can then be added to the RMP. With these extra routes, the algorithm solves the RMP (an integer program) to produce a potentially improving primal solution to the MP.

We employ the restricted dynamic programming heuristic proposed above to generate routes. However, as we do not want the algorithm spending a lot of time generating routes, this heuristic limits the number of partial paths that are extended. Specifically, at each stage of its execution, the heuristic only considers the best $B/2$ partial paths that expand to a pickup node and the best $B/2$ partial paths that expand to a delivery node for further expansion (B is an algorithm parameter that we set to 8000 in our experiments). To select these partial paths, the heuristic uses a mechanism similar to that presented in Sun et al. (2018). First, we order the expanded partial paths by the earliest arrival time. Then, if two expanded partial paths have the same earliest arrival time, we order them by their objective function value. The heuristic pricing rule is also applied to prune unpromising labels. After the heuristic has finished, it returns a set of potential routes, along with the reduced cost associated with each route.

However, we do not add all generated routes to the RMP. Instead, the algorithm filters the set of generated routes based on a prediction of whether a route will appear in an improving solution. To do so, we let z_{ub}^{bn} denote the upper bound associated with the node in the branch-and-bound tree that is currently being processed. Similarly, we let z_{lb}^{cbest} denote the objective function value of the best primal solution to the MP found. With these values, we calculate the relative gap $\varrho = (z_{ub}^{bn} - z_{lb}^{cbest})/z_{ub}^{bn} \times 100$. We observe that a route can only appear in an improving solution if its reduced cost is smaller than the gap ϱ.

5. Computational results

In this section, we present the results of a computational study on the effectiveness of the proposed algorithmic framework at solving the problems we are interested in. All algorithms are coded in Java, using Eclipse SDK version 4.2.0, and our experiments are performed on a single thread of a server with four CPU's (2.4 GHz/6 cores) and 64GB RAM. The LP relaxations are solved with Gurobi 5.6.3. A time limit of 10 h and a maximum memory allowance of 16GB RAM has been imposed on each run.

For our numerical study, we generated a set of instances by adopting the instances presented in Sun et al. (2018), which in turn were derived from those proposed by Røpke and Cordeau (2009) for the PDPTW. Specifically, our instances are based on the same node coordinates, time windows, vehicle capacity, and load quantities as those in Røpke and Cordeau (2009). The instances are divided into four groups, with each group containing nine instances. Table 1 presents the vehicle capacity, Q, and time window width, W, for each group. The profits were adapted in order to balance the fixed cost associated with using a vehicle (which was not included in Sun et al., 2018) and the profit associated with serving a request. Specifically, we set the fixed cost equal to 100, while randomly assigning to each request a profit in the interval [50,150].

Regarding the time-dependent travel times, we use the same values as in Sun et al. (2018). There, road congestion is modeled by a so-called speed model which consists of different speed profiles. It is used to determine the travel time between two nodes on a specific departure time. As denoted in Table 2, five types of speed profiles are considered: slow speed (SS), normal speed with morning peak (NSMP), normal speed with evening peak (NSEP), fast speed with two peaks (FSTP) and high speed (HS). We refer the reader to Sun et al. (2018) for detailed information regarding the explanation of the different speed profiles and how they are assigned to the arcs. Furthermore, without loss of generality, it is assumed that the breakpoints are the same for all speed profiles as congestion tends to happen around the same time regardless of the speed profiles' type. Each speed profile has four non-overlapping time periods with constant speed, reflecting two congested periods and two periods with normal traffic conditions. As a result, in terms of our model, the number of time zones m equals 7 for the instances we solve.

Our computational experiments focus on understanding the following issues. First, we compare the performance of a commercial MIP solver (Gurobi) when solving the compact formulation of each variant with that of the pure branch and

Table 2
Speed profiles.

Congestion description Time periods	Morning peak 7 am–9 am	Normal 9 am–5 pm	Evening peak 5 pm–7 pm	Normal 7 pm–9 pm
1.SS	0.5	0.81	0.5	0.81
2.NSMP	0.67	1.33	0.88	1.33
3.NSEP	0.88	1.33	0.67	1.33
4.FSTP	0.85	1.5	0.85	1.5
5.HS	1.0	2.0	1.0	2.0

Table 3
% Instances solved by MIP solver and BP.

	$\Omega(Fix, All)$		$\Omega(Fix, Prof)$		$\Omega(Flex, All)$		$\Omega(Flex, Prof)$	
# Requests	Gurobi	Exact	Gurobi	Exact	Gurobi	Exact	Gurobi	Exact
10	100.00%	100.00%	100.00%	100.00%	75.00%	100.00%	50.00%	100.00%
15	50.00%	100.00%	25.00%	100.00%	0.00%	100.00%	0.00%	100.00%
20	0.00%	100.00%	0.00%	100.00%	0.00%	100.00%	0.00%	100.00%
25	0.00%	50.00%	0.00%	100.00%	0.00%	100.00%	0.00%	100.00%
Average	37.50%	87.50%	31.25%	100.00%	18.75%	100.00%	12.50%	100.00%

Table 4
Solve time (s) for MIP solver and the exact framework.

	$\Omega(Fix, All)$		$\Omega(Fix, Prof)$		$\Omega(Flex, All)$		$\Omega(Flex, Prof)$	
# Requests	Gurobi	Exact	Gurobi	Exact	Gurobi	Exact	Gurobi	Exact
10	1704.1	0.6	643.6	0.4	16,961.3	0.9	18,245.2	1.0
15	20,890.3	4.4	27,595.1	1.9	36,000.0	7.0	36,000.0	7.1
20	36,000.0	210.3	36,000.0	7.4	36,000.0	371.7	36,000.0	21.2
25	36,000.0	21,237.2	36,000.0	27.4	36,000.0	1584.9	36,000.0	126.2
Average	23,648.6	5363.1	25,059.7	9.3	31,240.3	491.1	31,561.3	38.9

price framework presented in Section 4 solving the corresponding extended formulation. Second, we perform an in-depth analysis of the performance of the exact framework in order to determine the appropriate combination of speed-up techniques (i.e., $lm - SRCs$ and Route Enumeration) for each variant. Third, as each variant represents a different degree of operational flexibility, we analyze solutions to the different variants to quantify the value of that flexibility. We note that to test the correctness of our algorithm and implementation, we executed it on instances solved by the method presented in Røpke and Cordeau (2009). We observed that the framework we propose was able to generate the same results, however, as expected, in larger computation times. These larger computation times can be attributed to the fact that algorithm presented in Røpke and Cordeau (2009) includes techniques that presume the triangle inequality is valid. Our algorithm is designed for problems with time-dependent travel times, wherein the triangle inequality does not necessarily hold, and thus does not employ those techniques. Detailed, instance-level results can be found in the Appendix A in Table 15.

We note that while the following sections contain summary statistics derived from the computational results, detailed results can be found in Appendix A.

5.1. Performance comparison of MIP solver and exact framework

We first, for each variant, compare the performance of a MIP solver, Gurobi 5.6.3, when solving the compact formulation with the proposed exact framework solving the corresponding extended formulation. We note that Gurobi was executed for ten hours and with all parameters set to their default values. In this analysis, we only consider four instances from each group (AA, BB, CC, and DD); those with 10,15, 20 or 25 requests, as Gurobi already could not solve instances with 20 or 25 requests.

We first present in Table 3 the percentage of instances solved by the MIP solver (Gurobi) and the exact framework (Exact), for each variant by the number of requests. We see that while the MIP solver is unable to solve many of the instances, including none with 20 or 25 requests, the proposed exact framework is able to solve almost all instances within 10 h.

Continuing the comparison, in Table 4 we present averages of the time to termination for both Gurobi and the exact method. We see that in addition to solving all of the instances, the exact framework was able to do so in relatively little time. Gurobi required on average the largest computation time for the most flexible variant, $\Omega(Flex, Prof)$ (i.e., the variant

Table 5
Performance of exact framework with and without enhancements.

	Ω(*Fix, All*)				Ω(*Fix, Prof*)			
Options	*#Better*	*#Equal*	*#Worse*	Overall	*#Better*	*#Equal*	*#Worse*	Overall
BP + 1	1	7	5	−4	0	1	4	−4
BP + 2	4	5	4	0	0	0	5	−5
BP + 1 + 2	5	5	3	2	0	0	5	−5
	Ω(*Flex, All*)				Ω(*Flex, Prof*)			
Options	*#Better*	*#Equal*	*#Worse*	Overall	*#Better*	*#Equal*	*#Worse*	Overall
BP + 1	2	3	8	−6	1	4	6	−5
BP + 2	6	3	4	2	2	4	5	−3
BP + 1 + 2	9	3	1	8	4	3	4	0

1: Route enumeration. 2: Limited-memory subset-row cuts.

Table 6
Performance of exact framework on larger instances.

	Ω(*Fix, All*)			Ω(*Fix, Prof*)			Ω(*Flex, All*)			Ω(*Flex, Prof*)		
# Requests	% Solved	Time	Gap	% Solved	Time	Gap	% Solved	Time	Gap	% Solved	Time	Gap
10	100%	0.7	0.0%	100%	0.4	0.4%	100%	1.0	0.0%	100%	1.0	0.0%
15	100%	2.8	0.0%	100%	1.9	0.0%	100%	5.9	0.0%	100%	7.1	0.0%
20	100%	1015.3	0.0%	100%	7.4	0.0%	100%	317.4	0.0%	100%	21.2	0.0%
25	75%	11,809.4	1.9%	100%	27.4	0.0%	100%	2138.0	0.0%	100%	126.2	0.0%
30	50%	18,276.2	8.4%	100%	475.0	0.0%	50%	24,340.3	4.7%	100%	4405.3	0.1%
35	50%	18,584.4	20.4%	50%	18,104.0	1.1%	25%	27,006.9	3.1%	75%	19,506.2	0.1%
40	50%	18,417.9	12.5%	50%	18,191.0	1.0%	25%	27,996.8	0.1%	50%	18,239.1	0.6%
45	25%	27,876.3	14.5%	50%	18,084.6	5.0%	25%	27,898.3	1.9%	50%	18,357.0	0.8%
Average	68.8%	11,997.88	7.2%	81.3%	6861.5	0.9%	65.6%	13,713.1	1.2%	84.4%	7582.9	0.2%

Table 7
Impact of cuts and branching.

	Ω(*Fix, All*)			Ω(*Fix, Prof*)		Ω(*Flex, All*)			Ω(*Flex, Prof*)	
Instances	# Cuts	# Branches	Bound gap	# Branches	Bound gap	# Cuts	# Branches	Bound gap	# Branches	Bound gap
Unsolved	0.5	131.75	−3.49%	283.00	−2.70%	10.45	3.55	−0.71%	27.00	−1.27%
Solved	35.68	111.41	−2.87%	4.81	−0.34%	9.05	7.67	−0.90%	161.30	−1.29%
Average	24.69	116.83	−3.03%	49.67	−0.66%	9.53	6.25	−0.76%	140.31	−1.29%

with the largest solution space), while our framework required on average the largest computation time for the least flexible variant, Ω(*Fix, All*) (i.e., the variant with the smallest solution space). Therefore, we did a more in depth analysis of the performance of our framework in the next section. Full details of the results generated with the MIP solver can be found in Appendix A, Tables 8 and 9.

5.2. Analyzing the performance of the exact framework

Having established that the exact framework is more effective at solving instances of each variant, we next study its performance in greater detail. For each variant, we assess whether Limited-memory Subset-row cut and Route enumeration techniques improve the performance of the exact framework. To do so, we report in Table 5 how often the exact framework, when augmented with one or both of these techniques, performs better than when it does not use either. We note that in these results, we only consider instances that have 30 or fewer requests. We also do not consider instances that can be solved at the root node before any of the techniques would be employed.

How we compare the performance of two methods (the exact framework with techniques and without) on an instance depends on whether either method could solve the instance to optimality in the given time limit. If either could solve the instance, then we record the one that could do so in less time as performing “better” on that instance, and the one that took more time as performing “worse.” Note that if one method solves and the other does not, then the one that does is recorded as performing “better” and the one that does not is recorded as performing “worse.” If neither could solve the instance, then we compare the methods in terms of the quality of the objective function values of the feasible solutions they produce. For either metric, there are also instances wherein the two methods perform equally well. We summarize these relative performance measures in the column “Overall”, which we calculate as “# Better” - “# Worse.” We see that

	Fix, Profit	Flex, All	Flex, Profit
10	41.96%	114.88%	196.59%
15	19.50%	26.96%	39.60%
20	41.07%	39.36%	50.77%
25	24.03%	20.73%	27.15%
30	13.62%	14.88%	18.90%
35	5.27%	6.83%	9.86%
40	6.50%	4.60%	9.75%
45	4.96%	2.69%	5.59%

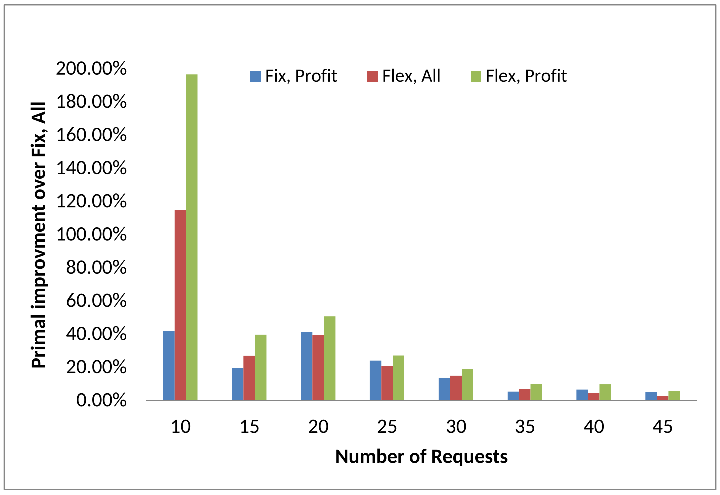

Fig. 3. Value of flexibility.

for both $\Omega(-, All)$ variants, using both techniques ($BP + 1 + 2$) leads to the best performance. However, the original branch and price algorithm (BP) performs much better than $BP + 1 + 2$ on Ω(*Fix, Prof*) and comparably on Ω(*Flex, Prof*). Therefore, in the remainder of this section, we will use the Branch-Cut-and-Price framework with route enumeration ($BP + 1 + 2$) for the $\Omega(-, All)$ variants and our regular Branch-and-Price framework (BP) for the $\Omega(-, Prof)$ variants.

Next, we study the performance of the selected exact frameworks on larger instances, with up to 45 requests. To that effect, we present in Table 6 three statistics measuring the performance of the algorithm when solving instances of each variant and by number of requests: (1) The percentage of instances solved (% Solved), (2) The average time to termination, in seconds, (Time), and, (3) The optimality gap reported at termination (Gap). Detailed, instance-level results can be found in the Appendix A in Tables 10–13.

We see that for both $\Omega(-, Prof)$ variants, all instances up to 30 requests can be solved to optimality, while the same is not true for the $\Omega(-, All)$ instances (one instance (DD30) for Ω(*Fix, All*) and two instances (AA30 and DD30) for Ω(*Flex, All*)). We also note that 30 requests is the point at which the exact framework begins to struggle to solve instances, with a dramatic increase in the solution time and optimality gap reported at termination. We conclude from this table that the exact framework is more effectiving at solving the $\Omega(-, Prof)$ variants, as it tends to solve more instances, take less time, and report a smaller optimality gap at termination.

To better understand when the exact framework is able to solve an instance, we present in Table 7 statistics related to the impact of valid inequalities and branching, averaged over the instances that are and are not solved. Specifically, we report the number of $lm - SRCs$ generated (# Cuts), the number of branches created (# Branches), and the improvement in the bound between when the root node linear programming relaxation is solved (but before cuts are added) and when the algorithm terminates (Bound gap). Specifically, for this last statistic, we let z_{LPR} represent the optimal value of the linear programming relaxation and z_{Final} represent the final bound produced. With these statistics, we calculate the Bound gap as $(z_{Final} - z_{LPR})/z_{Final}$.

From these results, we conclude that the $\Omega(Fix, -)$ instances are hard to solve when few $lm - SRCs$ are generated, as doing so leads to a large number of branches, with branching having a limited effect on the dual bound. On the other hand, we observe that for the $\Omega(Flex, -)$ instances, branching is more effective, as fewer branches lead to a bigger decrease in the dual bound.

5.3. Value of flexibility

Each variant studied in this paper considers a different degree of flexibility in operations, wherein the Ω(*Flex, Prof*) variant is the most flexible and the Ω(*Fix, All*) is the least. As noted in Section 3, one can also derive relationships between objective function values of optimal solutions. Namely, that one must have $Obj_{\Omega(Fix,\ All)} \leq Obj_{\Omega(Fix,\ Prof)} \leq Obj_{\Omega(Flex,\ Prof)}$, and, $Obj_{\Omega(Fix,\ All)} \leq Obj_{\Omega(Flex,\ All)} \leq Obj_{\Omega(Flex,\ Prof)}$. We next quantify the relative differences in these objective function values. Specifically, for instances wherein Ω(*Fix, All*) could be solved to optimality, we calculate the percentage improvement in objective function value for a given variant with respect to Ω(*Fix, All*) with the expression $(Obj_{other} - Obj_{\Omega(Fix,All)})/(Obj_{\Omega(Fix,All)})$. Then, in Fig. 3, we illustrate the averages of these differences for each variant and by number of requests (see Table 14 in Appendix A for the full details).

We see in Fig. 3 that the flexibility afforded by the Ω(*Flex, Prof*) variant leads to a significant increase in profits. This suggests that there is significant value for a carrier in moving away from publishing a fixed schedule to which they must adhere. We also see that no relationship can be derived between Ω(*Fix, Prof*) and Ω(*Flex, All*), as sometimes one yields a higher objective function value and sometimes the other. We also note that there seems to be a correlation between the number of requests and these differences. The larger the number of requests, the smaller the gains associated with adding flexibility.

6. Conclusions and future research opportunities

In this paper, we introduced the time-dependent pickup and delivery problem with time windows, as well as three variants of the problem that model different levels of operational flexibility. We presented an exact branch-and-price-based algorithmic framework that can solve each of these variants. This framework incorporates a tailored labeling algorithm for solving the pricing problem, as well as speed-up techniques from the literature. The results of an extensive computational study indicate that the framework can solve to optimality instances with up to 45 pickup and delivery requests. We also assessed the impact of the speed-up techniques and the relative computational complexity of the four variants. Adding cuts to the framework only works out for the situation where all customers need to be served. For the more flexible variants, where it is allowed to skip some customers, the pure branch-and-price framework performs best. Surprisingly, these variants in which it is allowed to skip customers can be solved much faster than their respective counter parts in which all customers need to be served.

Regarding future research, we see multiple opportunities for adapting the framework we propose to solve related problems. For example, the framework could be adapted to solve problems wherein vehicles are heterogeneous. Similarly, it could be adapted to solve problems wherein not all vehicles are based at the same depot. We believe that both adaptations can be performed by changing the way the framework prices out new columns. Finally, we are exploring adapting the framework to a variant wherein travel times are both time dependent and stochastic. However, such an adaptation requires more algorithmic development.

Acknowledgments

The authors gratefully acknowledge funds provided by China Scholarship Council (CSC) under Grant No.201206160092 and the German Research Foundation (DFG, project GO 1841/5-1).

Appendix A. Detailed tables

Table 8
Results of solving the Ω(*Fix, All*) and Ω(*Fix, Prof*) using Gurobi.

	Ω(*Fix, All*)				Ω(*Fix, Prof*)			
Instances	LB	UB	Time	Gap (%)	LB	UB	Time	Gap (%)
AA10	110.92	110.92	194.5	0	110.92	110.92	203.1	0
BB10	156.61	156.61	18.0	0	156.61	156.61	108.9	0
CC10	157.36	157.36	527.2	0	224.62	224.62	950.4	0
DD10	497.70	497.70	6076.7	0	206.39	206.39	1311.9	0
AA15	315.31	1076.49	36,000.0	241	421.03	1018.00	36,000.0	142
BB15	369.74	369.74	6042.0	0	516.61	516.61	2380.3	0
CC15	563.36	1118.00	360,00.0	98.5	563.36	1157.14	36,000.0	105
DD15	497.70	497.70	5519.1	0	541.46	1156.00	36,000.0	113
AA20	331.09	1458.00	36,000.0	340	659.21	1458.00	36,000.0	121
BB20	580.75	884.61	36,000.0	52.3	789.17	879.15	36,000.0	11.4
CC20	658.96	1640.00	36,000.0	149	810.96	1640.00	36,000.0	102
DD20	905.48	1640.00	36,000.0	81.1	871.48	1640.00	36,000.0	88.2
AA25	776.93	1922.00	36,000.0	147	1023.01	1922.00	36,000.0	87.9
BB25	885.49	1959.02	36,000.0	121	1169.59	1940.17	36,000.0	65.9
CC25	–	2122.00	36,000.0	–	1210.71	2122.00	36,000.0	75.3
DD25	–	2122.00	36,000.0	–	1165.66	2122.00	36,000.0	82
Average			23,648.6	87.9			25,059.7	62.1

Table 9
Results of solving the Ω(*Flex, All*) and Ω(*Flex, Prof*) using Gurobi.

	Ω(*Flex, All*)				Ω(*Flex, Prof*)			
Instances	LB	UB	Time	Gap (%)	LB	UB	Time	Gap (%)
AA10	256.92	256.92	390.9	0	262.66	262.66	938.8	0
BB10	211.66	211.66	51.8	0	222.60	222.60	42.0	0
CC10	372.37	522.43	36,000.0	40.3	372.37	574.00	36,000.0	54.1
DD10	197.04	197.04	31,402.6	0	271.96	378.92	36,000.0	39.3
AA15	517.09	1021.84	36,000.0	97.6	520.81	1018.00	36,000.0	95.5
BB15	420.42	834.57	36,000.0	98.5	554.24	918.00	36,000.0	65.6
CC15	593.08	1118.00	36,000.0	88.5	684.31	1157.30	36,000.0	69.1
DD15	467.80	1112.44	36,000.0	138	606.48	1156.00	36,000.0	90.6
AA20	668.68	1458.00	36,000.0	118	706.36	1458.00	36,000.0	106
BB20	703.46	1350.95	36,000.0	92	843.94	1445.59	36,000.0	71.3
CC20	856.99	1640.00	36,000.0	91.4	964.99	1640.00	36,000.0	69.9
DD20	–	1640.00	36,000.0	–	870.12	1640.00	36,000.0	88.5
AA25	992.54	1922.00	36,000.0	93.6	1051.75	1922.00	36,000.0	82.7
BB25	918.17	2022.00	36,000.0	120	1184.67	2023.66	36,000.0	70.8
CC25	738.25	2122.00	36,000.0	187	1269.14	2122.00	36,000.0	67.2
DD25	992.48	2122.00	36,000.0	114	1129.80	2122.00	36,000.0	87.8
Average			31,240.3	85.26			31,561.3	66.15

Table 10
Detailed computational results for the Ω(*Fix, All*).

Instance	LB	RootUB	BestUB	Time	Gap (%)	Branches	Cuts	Served
AA10	110.92	110.92	110.92	0.6	0.00	1	0	10
BB10	156.61	156.61	156.61	0.4	0.00	1	0	10
CC10	157.36	157.36	157.36	0.7	0.00	1	0	10
DD10	91.70	91.70	91.70	1.0	0.00	1	0	10
AA15	315.31	315.31	315.31	2.3	0.00	1	5	15
BB15	369.73	369.73	369.73	1.7	0.00	1	5	15
CC15	563.36	563.36	563.36	4.7	0.00	1	0	15
DD15	497.70	497.70	497.70	2.5	0.00	1	0	15
AA20	331.09	331.09	331.09	390.9	0.00	1	55	20
BB20	580.74	580.74	580.74	201.8	0.00	2	75	20
CC20	687.57	862.89	688.14	3457.6	0.08	126	30	20
DD20	905.48	905.48	905.48	10.8	0.00	1	0	20
AA25	776.93	776.93	776.93	7.6	0.00	1	90	25
BB25	1018.25	1023.33	1018.25	557.7	0.00	3	55	25
CC25	1079.42	1233.22	1080.46	10,672.4	0.10	1101	15	25
DD25	992.71	1138.53	1075.18	36,000.0	7.67	561	5	25
AA30	966.29	1096.22	966.29	1021.3	0.00	19	35	30
BB30	1393.12	1393.12	1393.12	83.7	0.00	1	120	30
CC30	1450.51	1622.67	1455.38	36,000.0	0.33	39	0	30
DD30	1005.95	1599.88	1503.48	36,000.0	33.09	42	0	30
AA35	1419.48	1510.69	1419.48	2304.1	0.00	1139	50	35
BB35	1843.79	1843.79	1843.79	33.6	0.00	1	10	35
CC35	1635.02	2008.12	1941.69	36,000.0	15.79	51	0	35
DD35	673.79	1975.92	1975.92	36,000.0	65.90	1	0	35
AA40	1685.98	1687.32	1687.32	1034.5	0.08	1	15	40
BB40	2000.09	2020.92	2001.84	637.0	0.09	4	225	40
CC40	1446.21	2305.85	2305.85	36,000.0	37.28	1	0	40
DD40	–	2276.72	–	36,000.0	–	–	0	–
AA45	1927.92	2088.80	2074.39	36,000.0	7.06	358	0	45
BB45	2391.44	2448.23	2393.71	3505.3	0.09	43	0	45
CC45	1656.90	2606.69	2606.69	36,000.0	36.44	1	0	45
DD45	–	2564.44	–	36,000.0	–	–	0	–

* LB: the lower bound, RootUB: the root upper bound, BestUB: the best upper bound within the time limit, Time: the processing time in seconds, Gap%: gap, Branches: number of branches considered in the algorithm and Served: the number of served requests in the lower bound solution.

Table 11
Detailed computational results for the Ω(*Fix, Prof*).

Instance	LB	RootUB	BestUB	Time	Gap (%)	Branches	Served
AA10	110.92	110.92	110.92	0.2	0.00	1	10
BB10	156.61	156.61	156.61	0.2	0.00	1	10
CC10	224.62	224.62	224.62	0.4	0.00	1	7
DD10	206.39	206.39	206.39	0.6	0.00	1	7
AA15	421.03	421.03	421.03	1.0	0.00	1	14
BB15	516.61	516.61	516.61	1.2	0.00	1	13
CC15	563.36	563.36	563.36	3.7	0.00	1	15
DD15	621.36	621.36	621.36	1.8	0.00	1	14
AA20	659.21	659.21	659.21	3.9	0.00	1	16
BB20	789.17	789.17	789.17	2.6	0.00	1	17
CC20	888.96	888.96	888.96	12.5	0.00	1	19
DD20	905.48	905.48	905.48	10.5	0.00	1	20
AA25	1023.01	1023.01	1023.01	4.2	0.00	2	22
BB25	1169.59	1169.59	1169.59	4.3	0.00	1	19
CC25	1296.93	1296.93	1296.93	26.9	0.00	1	23
DD25	1284.99	1284.99	1284.99	74.2	0.00	1	22
AA30	1196.72	1204.55	1196.72	263.2	0.00	2	23
BB30	1471.75	1471.75	1471.75	23.1	0.00	1	22
CC30	1615.59	1650.49	1615.59	409.9	0.00	1	26
DD30	1591.88	1613.11	1591.88	1203.9	0.00	1	27
AA35	1569.12	1587.34	1569.12	120.2	0.00	27	33
BB35	1843.79	1891.99	1843.79	295.7	0.00	36	35
CC35	1903.91	2019.04	1945.01	36,000.0	2.11	187	34
DD35	1880.32	1988.88	1920.44	36,000.0	2.09	11	27
AA40	1811.67	1811.67	1811.67	525.5	0.00	22	37
BB40	2110.83	2133.10	2110.83	238.5	0.00	13	37
CC40	2120.81	2258.60	2183.52	36,000.0	2.87	1215	36
DD40	–	2279.46	–	36,000.0	–	–	–
AA45	2193.08	2195.61	2193.08	265.3	0.00	4	41
BB45	2509.95	2509.95	2509.95	73.0	0.00	1	42
CC45	2228.02	2618.63	2618.63	36,000.0	14.92	1	38
DD45	–	2567.93	–	36,000.0	–	1	–

Table 12
Detailed computational results for the Ω(*Flex, All*).

Instance	LB	RootUB	BestUB	Time	Gap (%)	Branches	Cuts	Served
AA10	256.92	256.92	256.92	0.5	0.00	1	0	10
BB10	211.66	211.66	211.66	0.5	0.00	1	0	10
CC10	372.37	372.37	372.37	1.0	0.00	1	0	10
DD10	197.04	197.04	197.04	1.8	0.00	1	0	10
AA15	517.09	517.09	517.09	3.1	0.00	1	5	15
BB15	420.41	420.42	420.42	2.6	0.00	1	5	15
CC15	684.31	684.31	684.31	7.9	0.00	1	0	15
DD15	540.76	540.76	540.76	10.2	0.00	1	0	15
AA20	668.68	668.68	668.68	6.7	0.00	1	10	20
BB20	705.68	705.68	705.68	44.9	0.00	1	60	20
CC20	900.45	961.90	901.24	1204.4	0.07	73	0	20
DD20	932.60	932.60	932.60	13.6	0.00	1	0	20
AA25	1032.34	1032.34	1032.34	12.7	0.00	1	10	25
BB25	1071.91	1085.12	1071.91	571.1	0.00	3	65	25
CC25	1298.40	1344.70	1299.63	5409.3	0.03	19	0	25
DD25	1235.69	1296.89	1236.73	2559.1	0.07	36	0	25
AA30	1175.56	1178.74	1178.74	36,000.0	0.27	1	75	30
BB30	1483.97	1483.97	1483.97	41.2	0.00	1	25	30
CC30	1689.39	1711.53	1689.78	25,103.9	0.00	4	0	30
DD30	1361.70	1669.71	1669.71	36,000.0	18.45	1	0	30
AA35	1571.87	1610.55	1610.55	36,000.0	2.40	1	40	35
BB35	1897.79	1897.79	1897.79	27.5	0.00	1	5	35
CC35	1925.26	2048.17	1945.01	36,000.0	1.02	1	0	35
DD35	1893.00	2082.59	2082.59	36,000.0	9.10	1	0	35
AA40	1791.66	1810.23	1810.23	36,000.0	0.10	1	0	40
BB40	2058.63	2072.87	2059.81	3987.1	0.06	9	5	40
CC40	–	2347.11	2347.11	36,000.0	–	1	0	–
DD40	2585.33	2589.86	2588.20	36,000.0	0.11	1	0	45
AA45	2101.25	2190.45	2182.90	36,000.0	3.74	29	0	45
BB45	2455.72	2474.87	2457.44	3593.0	0.07	3	0	45
CC45	–	2664.00	2664.00	36,000.0	–	1	0	–
DD45	–	2698.68	2698.68	36,000.0	–	1	0	–

Table 13
Detailed computational results for the Ω(*Flex, Prof*).

Instance	LB	RootUB	BestUB	Time	Gap (%)	Branches	Served
AA10	262.66	262.66	262.66	0.3	0.00	1	9
BB10	222.60	222.60	222.60	0.3	0.00	1	8
CC10	372.37	372.37	372.37	1.5	0.00	1	10
DD10	271.96	271.96	271.96	1.8	0.00	1	6
AA15	520.81	547.43	520.81	3.4	0.00	2	14
BB15	554.24	554.24	554.24	1.0	0.00	1	12
CC15	684.31	684.31	684.31	14.5	0.00	1	15
DD15	606.48	606.48	606.48	9.6	0.00	1	12
AA20	706.36	743.31	706.36	47.0	0.09	57	18
BB20	843.94	848.79	843.94	3.4	0.00	3	15
CC20	964.99	964.99	964.99	24.6	0.00	1	18
DD20	942.38	942.38	942.38	9.6	0.00	1	17
AA25	1051.88	1074.07	1051.88	65.8	0.08	19	22
BB25	1184.67	1203.89	1184.67	14.3	0.00	2	19
CC25	1355.72	1362.03	1355.72	214.6	0.00	3	22
DD25	1303.20	1324.21	1303.20	210.0	0.00	2	21
AA30	1284.11	1331.86	1284.11	4651.8	0.00	1905	28
BB30	1506.89	1569.83	1506.89	267.3	0.09	59	27
CC30	1689.39	1722.56	1689.44	3544.1	0.02	8	30
DD30	1627.76	1686.20	1629.26	9158.2	0.10	36	24
AA35	1656.81	1684.16	1658.46	15,804.2	0.10	2175	33
BB35	1897.79	1925.87	1897.79	138.5	0.00	13	35
CC35	2028.82	2050.13	2030.72	26,082.1	0.09	36	34
DD35	1960.66	2099.28	2021.95	36,000.0	3.03	131	31
AA40	1887.74	1892.81	1889.57	433.4	0.10	12	37
BB40	2150.70	2154.69	2150.70	523.1	0.00	9	37
CC40	2317.31	2356.45	2356.45	36,000.0	1.66	1	37
DD40	–	2370.77	–	36,000.0	–	1	–
AA45	2259.00	2268.92	2259.00	1170.0	0.01	4	41
BB45	2525.08	2525.08	2525.08	257.9	0.00	1	42
CC45	–	2664.79	–	36000.0	–	1	–
DD45	2425.56	2703.40	2703.40	36,000.0	10.28	1	40

Table 14
The objective improvement of other 3 variants compare to Ω(*Fix, All*).

Instance	Ω(*Fix, All*)(Value)	Ω(*Fix, Prof*)(%)	Ω(*Flex, All*)(%)	Ω(*Flex, Prof*)(%)
AA10	110.919	0.00	131.63	136.81
BB10	156.608	0.00	35.15	42.14
CC10	157.355	42.75	136.64	136.64
DD10	91.695	125.08	114.88	196.59
AA15	315.306	33.53	64.00	65.17
BB15	369.735	39.72	13.71	49.90
CC15	563.355	0.00	21.47	21.47
DD15	497.695	24.85	8.65	21.86
AA20	331.091	99.10	101.96	113.34
BB20	580.745	35.89	21.51	45.32
CC20	687.57	29.29	30.96	40.35
DD20	905.483	0.00	3.00	4.08
AA25	776.928	31.67	32.87	35.39
BB25	1018.25	14.86	5.27	16.34
CC25	1079.42	20.15	20.29	25.60
DD25	992.712	29.44	24.48	31.28
AA30	966.289	23.85	21.66	32.07
BB30	1393.12	5.64	6.52	8.17
CC30	1450.51	11.38	16.47	16.47
DD30	1005.95	58.25	35.36	61.81
AA35	1419.48	10.54	10.74	16.72
BB35	1843.79	0.00	2.93	2.99
CC35	1635.02	16.45	17.75	24.09
DD35	673.79	179.07	180.95	190.99
AA40	1685.98	7.45	6.27	11.97
BB40	2000.09	5.54	2.93	7.53
CC40	1446.21	46.65	–	60.23
DD40	1927.92	–	34.10	–
AA45	1928.64	13.71	8.95	17.13
BB45	2391.44	4.96	2.69	5.59
CC45	1656.90	34.47	–	–
DD45	–	–	–	–

Table 15
Performance of the exact framework on instances of Røpke and Cordeau (2009). *Gap* represents the % difference of our optimal (or best found, in case the time limit of 36,000 s is exceeded) solution compared to the optimal solution of Røpke and Cordeau (2009).

	Røpke and Cordeau (2009)	Our framework	
Instance	Time (s)	Gap (%)	Time (s)
AA30	7	0	593
BB30	6	0	342
CC30	11	0	415
DD30	25	0	7178
AA35	16	0	1058
BB35	18	0	2559
CC35	59	0	2208
DD35	765	0	17,243
AA40	13	0	11,723
BB40	19	0	2617
CC40	255	0	17,414
DD40	161	*	36,000
AA45	16	0	11,060
BB45	46	0	35,077
CC45	176	32	36,000
DD45	525	*	36,000
Average	132		13,593

* = no feasible solution found within 36,000 s.

References

Archetti, C., Bianchessi, N., Speranza, M., 2013. Optimal solutions for routing problems with profits. Discrete Appl. Math. 161 (2), 547–557.
Archetti, C., Speranza, M., Vigo, D., 2014. Vehicle routing problems with profits. In: Toth, P., Vigo, D. (Eds.), Vehicle Routing: Problems, Methods, and Applications, Second Edition, pp. 273–298.
Azadian, F., Murat, A., Chinnam, R.B., 2017. An unpaired pickup and delivery problem with time dependent assignment costs: application in air cargo transportation. Eur. J. Oper. Res. 263 (1), 188–202.
Baldacci, R., Bartolini, E., Mingozzi, A., 2011. An exact algorithm for the pickup and delivery problem with time windows. Oper. Res. 59 (2), 414–426.
Baldacci, R., Christofides, N., Mingozzi, A., 2008. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. Math. Program. 115 (2), 351–385.
Balseiro, S.R., Loiseau, I., Ramont, J., 2011. An ant colony algorithm hybridized with insertion heuristics for the time dependent vehicle routing problem with time windows. Comput. Oper. Res. 38, 954–966.
Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H., 1998. Branch-and-price: column generation for solving huge integer programs. Oper. Res. 46 (3), 316–329.
Boussier, S., Feillet, D., Gendreau, M., 2007. An exact algorithm for team orienteering problems. 4 OR 5, 211–230.
Cherkesly, M., Desaulnier, G., Laporte, G., 2015. Branch-price-and-cut for the pickup and delivery problem with time windows and last-in-first-out loading. Transp. Sci. 49 (4), 752–766.
Cordeau, J.F., Laporte, G., Ropke, S., 2008. Recent models and algorithms for one-to-one pickup and delivery problems. In: Golden, B., Raghavan, S., Wasil, E. (Eds.), The Vehicle Routing Problem: Latest Advances and New Challenges. Springer, New York, pp. 327–357.
Dabia, S., Ropke, S., Van Woensel, T., de Kok, T., 2013. Branch and cut and price for the time dependent vehicle routing problem with time windows. Transp. Sci. 47 (3), 380–396.
Desaulniers, G., Desrosiers, J., Solomon, M.M., 2006. Column Generation, Vol.5. Springer Science & Business Media.
Donati, A.V., Montemanni, R., Casagrande, N., Rizzoli, A.E., Gambardella, L.M., 2008. Time dependent vehicle routing problem with a multiant colony system. Eur. J. Oper. Res. 185 (3), 1174–1191.
Dumas, Y., Desrosiers, J., Soumis, F.Y., 1991. The pickup and delivery problem with time windows. Eur. J. Oper. Res. 54 (1), 7–22.
Duquei, D., Lozano, L., Medaglia, A., 2015. Solving the orienteering problem with time windows via the pulse framework. Comput. Oper. Res. 54, 168–176.
Feillet, D., Dejax, P., Gendreau, M., 2004. An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks 44 (3), 216–229.
Feillet, D., Dejax, P., Gendreau, M., 2005. Traveling salesman problems with profits. Transp. Sci. 39, 188–205.
Fomin, F., Lingas, A., 2002. Approximation algorithms for time-dependent orienteering. Inf. Process. Lett. 83, 57–62.
Franceschetti, A., Honhon, D., Van Woensel, T., Bektaş, T., Laporte, G., 2013. The time-dependent pollution routing problem. Transp. Res. Part B 56, 265–293.
Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantizou, G., Vathis., N., 2015. Heuristics for the time dependent team orienteering problem: application to tourist route planning. Comput. Oper. Res. 62, 36–50.
Gendreau, M., Ghiani, G., Guerriero, E., 2015. Time-dependent routing problems: a review. Comput. Oper. Res. 64, 189–197.
Gunawan, A., Lau, H.C., Vansteenwegen, P., 2016. Orienteering problem: a survey of recent variants, solution approaches and applications. Eur. J. Oper. Res. 255 (2), 315–332.
Ichoua, S., Gendreau, M., Potvin, J.Y., 2003. Vehicle dispatching with time-dependent travel times. Eur. J. Oper. Res. 62, 379–396.
Jabali, O., Van Woensel, T., de Kok, A., Lecluyse, C., Peremans, H., 2009. Time-dependent vehicle routing subject to time delay perturbations. IIE Trans. 41 (12), 1049–1066. doi:10.1080/07408170902976194. https://doi.org/10.1080/07408170902976194
Jepsen, M., Petersen, B., Spoorendonk, S., Pisinger, D., 2008. Subset-row inequalities applied to the vehicle routing problem with time windows. Oper. Res. 56 (2), 497–511.
Labadie, N., Mansini, R., Melechovskỳ, J., Calvo, R.W., 2012. The team orienteering problem with time windows: an LP-based granular variable neighborhood search. Eur. J. Oper. Res. 220 (1), 15–27.
Li, J., 2011. Model and algorithm for time-dependent team orienteering problem. In: Lin, S., Huang, X. (Eds.), Advanced Research on Computer Education. Simulation and Modeling, Communications in Computer and Information Science, 175, pp. 1–7.
Li, Y., Chen, H., Prins, C., 2016. Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests. Eur. J. Oper. Res. 252 (1), 27–38.
Lübbecke, M.E., 2010. Column Generation. John Wiley and Sons, Inc. doi:10.1002/9780470400531.eorms0158.

Mahmoudi, M., Zhou, X., 2016. Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: a dynamic programming approach based on statespacetime network representations. Transp. Res. Part B 89, 19–42.
Malandraki, C., Daskin, M., 1992. Time dependent vehicle routing problems: formulations, properties, and heuristic algorithms. Transp. Sci. 26 (3), 185–200.
Parragh, S.N., Doerner, K.F., Hartl, R.F., 2008a. A survey on pickup and delivery problems (part I: transportation between customers and depot). J. für Betriebswirtschaft 58 (1), 21–51.
Parragh, S.N., Doerner, K.F., Hartl, R.F., 2008b. A survey on pickup and delivery problems (part II: transportation between pickup and delivery locations). J. für Betriebswirtschaft 58 (2), 81–117.
Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., 2014. Improved branch-cut-and-price for capacitated vehicle routing. In: Integer Programming and Combinatorial Optimization. Springer, pp. 393–403.
Righini, G., Salani, M., 2009. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Comput. Oper. Res. 36 (4), 1191–1203.
Røpke, S., 2012. Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems. Present. Column Gener. 2012.
Røpke, S., Cordeau, J.-F., 2009. Branch and cut and price for the pickup and delivery problem with time windows. Transp. Sci. 43 (3), 267–286.
Røpke, S., Cordeau, J.-F., Laporte, G., 2009. Models and branch-and-cut algorithms for pickup and delivery problem with time windows. Networks 49 (4), 258–272.
Sigurd, M., Pisinger, D., 2004. Scheduling transportation of live animals to avoid the spread of diseases. Transp. Sci. 38, 197–209.
Sol, M., 1994. Column Generation Techniques for Pickup and Delivery Problems. Technische Universiteit Eindhoven Ph.D. thesis.
Sun, P., Veelenturf, L.P., Dabia, S., Van Woensel, T., 2018. The time-dependent capacitated profitable tour problem with time windows and precedence constraints. Eur. J. Oper. Res. 264 (3), 1058–1073.
Van Woensel, T., Kerbache, L., Peremans, H., Vandaele, N., 2008. Vehicle routing with dynamic travel times: a queueing approach. Eur. J. Oper. Res. 186 (3), 990–1007.
Vansteenwegen, P., Souffriau, W., Van Oudheusden, D., 2011. The orienteering problem: a survey. Eur. J. Oper. Res. 209 (1), 1–10.
Verbeeck, C., Aghezzaf, E.H., Vansteenwegen, P., 2014. A fast solution method for the time-dependent orienteering problem with time windows. Eur. J. Oper. Res. 236, 419–432.