Transportation Research Part B 128 (2019) 207–235

Contents lists available at ScienceDirect

Transportation Research Part B

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

An iterative re-optimization framework for the dynamic vehicle routing problem with roaming delivery locations

Gizem Ozbaygin[a,b,*], Martin Savelsbergh[b]

[a] Industrial Engineering Program, Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul 34956, Turkey
[b] H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332, USA

ARTICLE INFO

Article history:
Received 28 August 2018
Revised 7 August 2019
Accepted 8 August 2019
Available online 14 August 2019

Keywords:
Iterative re-optimization
Branch-and-price
Dynamic vehicle routing
Roaming delivery locations

ABSTRACT

Branch-and-price has established itself as an effective solution methodology for a wide variety of planning problems. We investigate its potential as a solution methodology for solving *operational* problems. Specifically, we explore its potential in the context of a dynamic variant of the vehicle routing problem with roaming delivery locations, in which customer itineraries may change during the execution of a planned delivery schedule, which, in turn, may cause the planned delivery schedule to become suboptimal or even infeasible. We propose an iterative solution framework in which an active delivery schedule is re-optimized whenever a customer itinerary update is revealed. We use a branch-and-price algorithm for solving the planning problem (to obtain an initial delivery schedule) as well as the re-optimization problems (to obtain modified delivery schedules). As the re-optimization problems are solved during the execution of the (active) delivery schedule, it is critical to produce solutions quickly. This is accomplished by re-using, suitably modified, columns generated during preceding branch-and-price solves. The results of our computational experiments demonstrate the viability of using branch-and-price for solving operational problems and the benefit (necessity) of re-using information from previous solves.

1. Introduction

Branch-and-price (Barnhart et al., 1998) is one of the most effective optimization approaches for solving certain classes of integer programming problems. Up to now, it has been used almost exclusively to solve planning problems, in which ample time is available to produce optimal or high-quality solutions. The primary goal of this study is to demonstrate that branch-and-price algorithms, when designed and implemented appropriately, can also be used in dynamic decision making environments, in which little time is available to produce optimal or high-quality solutions. To do so, we consider a dynamic variant of the vehicle routing problem with roaming delivery locations and develop a branch-and-price based iterative re-optimization approach to solve it.

Motivated by the increasing interest in trunk delivery, after e-commerce giant Amazon partnered with Audi and DHL to offer this innovative delivery option to its customers (Popken, 2015; Geuss, 2015; Audi, 2015), the vehicle routing problem with roaming delivery locations (VRPRDL) seeks to find a least cost set of delivery routes for a fleet of capacitated vehicles,

* Corresponding author at: Industrial Engineering Program, Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul 34956, Turkey.
E-mail address: ozbaygin@sabanciuniv.edu (G. Ozbaygin).

https://doi.org/10.1016/j.trb.2019.08.004

delivering customer orders to the trunk of their cars at times when the car is parked at a location in the customer's (known) travel itinerary. The VRPRDL was introduced by Reyes et al. (2017), who propose construction and improvement heuristics for its solution. Ozbaygin et al. (2017) develop a branch-and-price algorithm for the VRPRDL.

In the VRPRDL, it is implicitly assumed that the itinerary of a customer is known in advance and that it will remain unchanged during the execution of the planned delivery schedule. Here, instead, we consider a dynamic variant of the VRPRDL, in which customer itineraries may change during the execution of the delivery schedule, which, in turn, may cause the delivery schedule to become suboptimal or even infeasible. Suppose, for example, that a customer plans to go home right after a meeting scheduled to start at 3 p.m. and end at 4 p.m., but that the meeting gets cancelled, and therefore the customer leaves work earlier than expected at 3 p.m. (i.e., the customer's itinerary used to plan the delivery schedule changes). If the delivery of this customer's order is scheduled to take place at his work location after 3 p.m., then the planned delivery schedule will become infeasible. In some cases, it may be possible to modify the delivery routes to recover feasibility, e.g., by re-routing vehicles and by changing the delivery locations of customer orders, but in others it may not (and delivery of certain orders may have to be postponed or by a separate express delivery service). Note that we assume that a single product has to be delivered to all customers, which implies that a vehicle can make deliveries to customers that were previously assigned to other vehicles' routes as long as there is enough time to do so and there is sufficient product remaining. This assumption may not hold in many delivery settings, but always holds in collection settings, where vehicles collect from the trunk of a car rather than deliver to the trunk of a car, e.g., to handle returns.

The literature on dynamic vehicle routing dates back to 40 years ago, when a single vehicle dial-a-ride problem with dynamically arriving trip requests was presented in Wilson and Colvin (1977). Psaraftis (1980) considers an "immediate request" version of the same problem in which the vehicle's route is re-planned immediately upon the arrival of a new trip request ensuring that the customer making the request is served as early as possible. With advances in positioning and navigation systems, increases in computing power and data processing capabilities, and the emergence of smart devices, the interest of researchers and practitioners in dynamic routing and scheduling has skyrocketed.

Most studies on dynamic routing and scheduling focus on the arrival of orders during the execution of planned vehicle routes as the source of dynamism, see for example Attanasio et al. (2004), Campbell and Savelsbergh (2005), Mitrović-Minić and Laporte (2004), Chen and Xu (2006), Ichoua et al. (2006), Mes et al. (2007) and Goel and Gruhn (2008). Although fewer in number, there are studies considering two other dynamic aspects of real-life vehicle routing problems: travel times (Fleischmann et al., 2004; Taniguchi and Shimamoto, 2004; Chen et al., 2006; Barceló et al., 2007; Tagmouti et al., 2011) and vehicle breakdowns (Li et al., 2009a; 2009b; Mu et al., 2011). For a comprehensive review and a detailed taxonomy of dynamic vehicle routing problems, we refer the interested reader to Pillac et al. (2013), Psaraftis et al. (2016), and Bektas et al. (2014).

The dynamic variant of VRPRDL that we consider, can be classified as a dynamic and deterministic vehicle routing problem. It is dynamic, because part of the input, i.e., the customer itineraries, may change during the planning horizon, and deterministic, because there is no stochastic information about the changes that may occur to the input during the planning horizon. We propose an iterative framework using the branch-and-price algorithm developed for the VRPRDL to re-optimize the vehicle routes every time the itinerary of a customer is updated. This is in line with the existing solution methods on dynamic and deterministic vehicle routing problems, which are mostly based on periodic re-optimization, either at pre-determined decision epochs or when a certain number of changes to the input data have occurred. (Our solution approach falls into the latter category.)

Re-optimizing the previously planned vehicle routes requires solving a static problem with the latest available data. In order to put the updated vehicle routes into effect as soon as possible, it is critical that the re-optimization is done efficiently. Therefore, most approaches presented in the literature rely on heuristics, e.g., Gendreau et al. (1999, 2006), Attanasio et al. (2004), Montemanni et al. (2005), Campbell and Savelsbergh (2005), Chen and Xu (2006), Taniguchi and Shimamoto (2004), Azi et al. (2012) and Li et al. (2009a,b). However, by making effective use of information collected during the solution of preceding (re-)optimization problems, it may be possible to obtain optimal or near-optimal solutions in a short amount of time using an exact approach. How to do so in the context of a dynamic variant of the VRPRDL is the focus of our study. More specifically, we explore how to generate "relevant" columns efficiently using previously generated columns (i.e., generated during preceding executions of the branch-and-price algorithm) when solving a re-optimization problem, as opposed to solving a pricing problem to generate columns from scratch.

To the best of our knowledge, the only other study employing a column generation based approach for solving a dynamic vehicle routing problem is Chen and Xu (2006). The problem considered and the methodology used are different from ours in the following respects:

- The focus is on dynamically arriving pickup orders, while, in our case, the source of dynamism is changes to the time windows in a customer itinerary. All pickup orders have a hard time window and rejection of arriving orders is not allowed. As a consequence, it has to be assumed that the order placement time is far enough in advance that it is possible to dispatch a new vehicle from the depot, and that there is an unlimited number of available vehicles. In the dynamic variant of the VRPRDL, rejection of service is allowed, because we do not make any assumptions about when an itinerary update is revealed.
- The vehicle routes are updated at pre-defined decision epochs, whereas we re-optimize each time new information is revealed.

- Only heuristics are used to generate columns (using existing columns to guide the search) and a set partitioning model involving the (heuristically) generated columns is solved (using commercial solver CPLEX). We solve the re-optimization problem exactly, solving pricing problems exactly, if needed, at any node of the search tree (e.g., we employ a full-fledged branch-and-price algorithm).

The major contributions of this study are: (1) introducing a dynamic variant of the VRPRDL, (2) demonstrating that branch-and-price algorithms can be used effectively in dynamic problem solving environments, and (3) providing insights in how to best re-use information from preceding branch-and-price solves.

The rest of the paper is organized as follows. In Section 2, we formally define the dynamic variant of the VRPRDL, we specify the types of customer itinerary changes considered, and give the set covering formulation of the re-optimization problem solved in each iteration and the associated pricing problems. In Section 3, we present the details of our solution approach, i.e., how existing columns are modified to be re-used. Section 4 gives the set up of our computational experiments, presents the results of the computational experiments, and discusses and interprets these results. Finally, we provide some concluding remarks in Section 5.

2. Problem definition and formulations

In this section, we provide a formal description of the dynamic variant of the vehicle routing problem with roaming delivery locations (D-VRPRDL), in which for each customer, an itinerary, specifying when and where the customer's order can be delivered, is available at the planning stage, but where the itinerary may change during execution of the delivery routes. As a consequence, the delivery routes may have to be revised to accommodate itinerary changes. For the sake of consistency, we use the notation introduced in Ozbaygin et al. (2017).

Let $G = (N, A)$ with $N = \{0, 1, \dots, n\}$ be a complete directed graph in which node 0 corresponds to the depot and the remaining nodes correspond to the locations of interest. Each arc $(i, j) \in A$ has an associated travel time t_{ij} and cost w_{ij} both satisfying the triangle inequality. The set of customers that require a delivery during the planning horizon $[0, T]$ is represented by C. The delivery for a customer $c \in C$ is characterized by a demand quantity d_c and a geographic profile which specifies where and when a delivery can be made. Let $N_c \subseteq N$ denote the set of locations that customer c will visit throughout the planning horizon. By duplicating locations, we may assume $N_c \cap N_{c'} = \emptyset$ for different customers $c, c' \in C$. Note that we can express the set of nodes as $N = N_0 \cup \{i \in N_c \mid c \in C\}$, where $N_0 = \{0\}$. The locations $i \in N_c$ have non-overlapping time windows $[e_i, l_i]$ during which the delivery to customer $c \in C$ can take place and correspond to the customer's original vehicle itinerary during the planning horizon. We use $c(i)$ to denote the customer associated with location i and we let $c(0) = 0$. A sufficiently large number of homogeneous vehicles, each with capacity Q, is available to make deliveries; vehicles start and end their delivery routes at the depot.

The goal is to find a set of delivery routes visiting each customer at one of the locations in the customer's itinerary, during the time that the customer is at that location, and such that the demand delivered on a route is no more than Q, the duration of a route does not exceed T, and the total transportation cost is minimized. As deviations from the original customer itineraries can render the planned delivery schedule infeasible or suboptimal, the vehicle routes are re-optimized after each update in an iterative fashion.

A feasible set of delivery routes shows the planned delivery location for every customer, i.e., the location where the customer's order will be delivered. Furthermore, it provides information on the earliest time that a vehicle can arrive at each customer location after serving the previous customers assigned to its route (assuming that it was dispatched from the depot at time 0) as well as the latest time that the vehicle should depart from each of these locations to serve the subsequent customers in its route within their respective time windows and return to the depot by time T. For planned delivery location i, we use ea_i and ld_i to represent the earliest arrival and the latest departure times of the assigned vehicle to and from i, respectively. We say that a given route is feasible, if for every node i in the route, the node's time window $[e_i, l_i]$ intersects with the delivery vehicle's time window $[ea_i, ld_i]$, that is, if $e_i \le ld_i$ and $l_i \ge ea_i$.

For simplicity, we assume that itinerary updates are revealed one at a time, i.e., the delivery planner does not receive information about the itinerary change of two or more customers simultaneously. For a customer $c \in C$, we restrict ourselves to updates caused by an early or a late departure from a location $i \in N_c$. Since the travel times are deterministic in our setting, the resulting deviation is absorbed by the subsequent locations in the customer's itinerary. We assume that every customer will always visit all locations in his original itinerary. Therefore, if a customer departs earlier or later than expected from a location, the time windows associated with (some of the) subsequent locations in his itinerary will get wider or narrower. When absorbing changes at subsequent locations, we do not allow a time window to expand or shrink by more than 75% of its original width (More details on this will be provided later in Section 4.1.) We note here that because the travel times are fixed, when a customer departs earlier (later) from a location, he will naturally arrive earlier (later) at the next location in his itinerary. For the convenience, we will refer to the updates as *early departure* and *late arrival* throughout the remainder of the paper. If we denote an updated time window at location i by $[e'_i, l'_i]$, then $l'_i < l_i$ indicates an early departure from i, and $e'_i > e_i$ indicates a late arrival at i.

Another assumption we make is that the delivery vehicles can be diverted from their planned routes to serve customers that are not originally assigned to them although we do not allow en route diversions. Suppose that an itinerary update is revealed at time t. A vehicle that is on its way to a customer location i at time t can be re-routed only after arriving at i, whereas if the vehicle is at location i at time t and $e_i > t$, i.e., the vehicle is waiting at location i, it can be re-routed immediately. Redirecting a vehicle to serve a customer while it is in transit to another customer may provide additional flexibility and may yield additional savings compared to adopting a "no en route diversion" strategy, but it requires responding to changes in the input data almost instantly. As pointed out in Bektas et al. (2014), it may be possible to identify solutions with higher quality by investing more time in re-optimization instead. Hence, en route diversion has received limited attention in the literature (a few examples include Regan et al., 1998; Ichoua et al., 2000; Attanasio et al., 2007; Klundert and Wormer, 2010; Respen et al., 2014; Ferrucci and Bock, 2015).

Depending on the time of a change as well as the magnitude of the change, it may, or may not, be possible to deliver the remaining orders – without violating their time window restrictions – using the vehicles that are executing the active delivery routes. Therefore, we assume that additional vehicles can be dispatched from the depot if needed. However, even allowing the use of additional vehicles does not guarantee that a feasible delivery schedule exists, especially when a change is revealed towards the end of the planning horizon. Hence, postponing service (to the next day) is allowed in our setting.

To solve the D-VRPRDL, we develop an iterative re-optimization scheme that solves a series of VRPRDL instances using the branch-and-price algorithm of Ozbaygin et al. (2017). The first VRPRDL instance covers the entire planning horizon [0, T] and uses the original customer itineraries. All subsequent instances can be viewed as instances of an extended version of the VRPRDL, in which some of the vehicles have to start from pre-specified locations at pre-specified times. In the following subsection, we describe the problem that needs to be solved at decision point t, denoted by SP hereafter, and provide a set covering based formulation.

2.1. Static problem formulation

Let O_t be the set of vehicle origin nodes for the problem defined over the horizon $[t, T]$ with $t > 0$. This set contains the depot node, and a node for each vehicle executing a delivery route at time t, indicating the location where the vehicle will be available for re-routing. Specifically, if a vehicle is en route at time t, then the corresponding origin node is the vehicle's current destination; otherwise, it is the current location of the vehicle. Every origin node $o \in O_t$ has an associated time st_o and a quantity qr_o specifying the earliest time at which the vehicle at origin o can depart from o and the remaining delivery capacity of the vehicle when it departs from o at time st_o, respectively. We take $st_o = t$ for the vehicles that are at the depot or at a customer location. For a vehicle that is en route, we take st_o to be the time that the vehicle reaches its destination. We compute qr_o assuming the vehicle departs from the depot fully loaded and considering the deliveries made up to time t. We use $C_t \subseteq C$ to represent the set of customers that require a delivery during $[t, T]$ ($C_0 = C$), and $N_c^t \subseteq N_c$ is the set of locations in the itinerary of customer $c \in C_t$ for which the time windows did not close earlier than t ($N_c^0 = N_c$). We define $N^t = \bigcup_{c \in C_t} N_c^t$.

We denote by R_o^t the set of all feasible delivery routes (i.e., respecting capacity and time window constraints) originating from $o \in O_t$ with an earliest start time of st_o and a delivery capacity of qr_o units. We define $R^t = \bigcup_{o \in O_t} R_o^t$. Let w_r be the cost of route $r \in R^t$, and let a_{ir} for every $i \in N^t$ and $r \in R^t$ indicate whether location i is visited on route r ($a_{ir} = 1$) or not ($a_{ir} = 0$). Then, SP can be formulated as follows:

$$\min \sum_{r \in R^t} w_r z_r \tag{1}$$

$$\sum_{r \in R^t} \sum_{i \in N_c^t} a_{ir} z_r \geq 1 \qquad \forall c \in C_t, \tag{2}$$

$$\sum_{r \in R_o^t} z_r = 1 \qquad o \in O_t \setminus \{0\}, \tag{3}$$

$$z_r \in \mathbb{Z}_+ \qquad r \in R^t, \tag{4}$$

where z_r is the number of times route r is used. This formulation is similar to that of the static VRPRDL given in Ozbaygin et al. (2017). Here, we have an additional set of constraints (3) specifying the location where each vehicle dispatched before t should originate from in the revised routing plan. Note that we do not restrict the number of vehicles originating from node 0, since we assume that a sufficiently large number of vehicles is available at the depot and can be dispatched whenever necessary.

The formulation above has exponentially many variables as the number of routes is exponential in the size of N^t. Therefore, we use the branch-and-price algorithm described in Ozbaygin et al. (2017) to solve it. However, because that algorithm was developed for the VRPRDL, in which all vehicles depart from the depot, minor modifications are needed to handle different starting locations for vehicles. These modifications are discussed in Section 3.1. Next, we derive the pricing problem associated with a given starting location.

2.2. Pricing problems

Consider the LP relaxation of (1)–(4), which will be referred to as the master problem from now on. Let $\bar{R}^t \subset R^t$ be such that there exists a feasible solution to the master problem when $z_r = 0$ for all $r \in R^t \setminus \bar{R}^t$. A formulation involving only routes in $\bar{R}^t$ is called a restricted master problem (RMP). The dual of the RMP is:

$$\max \sum_{c\in C_t}\lambda_c + \sum_{o\in O_t\setminus\{0\}} \mu_o \tag{5}$$

$$\sum_{c\in C_t}\sum_{i\in N_c^t} a_{ir}\lambda_c + \mu_o \le w_r \qquad o\in O_t, r\in \bar{R}_o^t, \tag{6}$$

$$\lambda_c \ge 0 \qquad c\in C_t, \tag{7}$$

$$\mu_o \in \mathbb{R} \qquad o\in O_t\setminus\{0\}, \tag{8}$$

where $\lambda = (\lambda_1, \ldots, \lambda_{|C_t|})$ and $\mu = (\mu_1, \ldots, \mu_{|O_t|-1})$ are dual variable vectors associated with the constraints (2) and (3), and $\mu_0 = 0$. Denote the optimal solution to this dual problem by (λ^*, μ^*). In the pricing problem for an origin, the goal is to identify a column with negative reduced cost with respect to the original master problem. In other words, for $o\in O_t$, we aim to find a column for which the associated dual constraint is violated. Such a column is a route r for which the following condition is satisfied:

$$w_r - \mu_o^* - \sum_{c\in C_t}\sum_{i\in N_c^t} a_{ir}\lambda_c^* < 0.$$

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

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

for every $o\in O_t$. Thus, the pricing problem for a particular origin o is an elementary shortest path problem with time window and capacity constraints (ESPPTWCC), where the cost of arc (i, j) is set to

$$\bar{w}_{ij} = \begin{cases} w_{ij} - \mu_o^*, & \text{if } i = o \\ w_{ij} - \lambda_{c(i)}^*, & \text{if } i \in N^t \end{cases}$$

for $j\in N^t\cup\{0\}$. In the ESPPTWCC, the goal is to find an elementary path of shortest length starting at origin node $o\in O_t$ and ending at the depot while respecting capacity and time window constraints. It is important to note here that the nodes in the set $O_t\setminus\{0\}$ are copies of the actual customer locations. More specifically, if a vehicle is at/en route to customer location i, then we add i' to O_t, where i' is a copy of node i. This is necessary to distinguish between the cases where the vehicle actually makes a delivery at location i and where the vehicle visits location i but leaves without delivering the order of customer $c(i)$. We model these situations by adding arcs from node i' to every other node in the problem graph – except those that are in $O_t\setminus\{0\}$. If the vehicle's re-optimized route uses arc (i', i), then we say that the delivery of customer $c(i)$ is made by this vehicle at location i.

3. An iterative re-optimization framework

To solve D-VRPRDL, we propose an iterative approach which dynamically handles deviations from the original customer itineraries. It starts by solving the VRPRDL with the initially provided itineraries. The delivery schedule obtained is executed as long as the customer itineraries do not change. However, once an itinerary change is revealed, the active vehicle routes may become infeasible or suboptimal, and thus, they are re-optimized taking into account the remaining deliveries and the current positions of the vehicles. The revised delivery plan is then executed until another itinerary change is revealed, after which another re-optimization problem is solved. Briefly, our iterative solution scheme produces an updated set of delivery routes whenever there is a change in one of the customer itineraries. As mentioned earlier, each re-optimization problem is a VRPRDL with an additional set of constraints, and solved through the branch-and-price algorithm developed for the VRPRDL. Further details on the solution approach are presented in the following subsections.

3.1. Constructing and preprocessing the graph of SP

When there is a change in the itinerary of a customer, the problem graph (used in the pricing problem) has to be updated to reflect the change. Some nodes and arcs in the (current) graph can no longer be a part of a feasible route and some nodes and arcs that were eliminated during the last preprocessing step may now, again, be part of a feasible route. For simplicity, instead of modifying the existing problem graph, we construct a new problem graph. We start with a complete graph on node set $N^t\cup\{0\}$, where we duplicate the depot node, i.e., have 0 and0′ to differentiate between the depot as origin and as destination. Therefore, the arc set is $\{(i, j)\colon i\in N^t\cup\{0\}, j\in (N^t\setminus\{i\})\cup\{0'\}\}\setminus\{(0, 0')\}$. (All arcs going out of the depot are connected to 0, while all arcs coming into the depot are connected to0′.) Next, we add the nodes in $O^t\setminus\{0\}$ along with the arcs $\{(o, j)\colon o\in O_t\setminus\{0\}, j\in N^t\cup\{0'\}\}$. Finally, the following preprocessing steps are applied to obtain the (new) problem graph for *SP*:

1. For every $i \in N^t$, if $st_o + t_{o,i} > l_i$ or $\max\{st_o + t_{o,i}, e_i\} + t_{i,0'} > T$ or $qr_o < d_{c(i)}$, then remove the arc (o, i) from the graph.
2. For every $i \in N^t$, if i has no incoming arcs, then remove i from the graph.
3. For every pair of nodes $i, j \in N^t$, let e^i_j denote the earliest possible arrival at j after visiting i, i.e., $e^i_j = \max\{\min_{o\in O_t}\{st_o + t_{o,i}\}, e_i\} + t_{i,j}$. If $e^i_j > l_j$ or $e^i_j + t_{j,0'} > T$, then remove arc (i, j) from the graph.

3.2. Solving the pricing problems

As mentioned earlier, the dual of the restricted master problem associated with *SP* yields pricing problems for each origin in O_t. The pricing problem corresponding to a particular origin o is an ESPPTWCC defined over the subgraph induced by the node set $N^t \cup \{o, 0'\}$, where o and $0'$ are the source and the sink nodes, respectively. The pricing subroutine in the branch-and-price algorithm presented in Ozbaygin et al. (2017) was designed to solve a single pricing problem starting from a single node (the depot). Rather than modifying the pricing subroutine, we extend the problem graph of *SP* and solve all the pricing problems at once. To achieve this, we add an artificial source node s^* and artificial arcs (s^*, o) for all $o \in O_t$ to the problem graph. We take the cost of each artificial arc to be zero. The time and capacity resource consumptions for arc (s^*, o) are set to st_o and $Q - qr_o$, respectively. We designate s^* as the source node and $0'$ as the sink node, and solve the ESPPTWCC on the extended graph. Observe that in this case, any path should use exactly one of the artificial arcs and contain exactly one of the origins since there are no arcs connecting any two origin nodes. Moreover, the time consumed and the quantity delivered on a feasible path, say p, using arc (s^*, o), i.e., a feasible path originating from o, cannot exceed $T - st_o$ and qr_o, respectively. Hence, the path obtained by joining together $p\backslash\{(s^*, o)\}$ and the path from the depot to node o corresponds to a feasible vehicle route over the planning horizon $[0, T]$.

3.3. Initial set of columns for SP

As the LP relaxation of the master problem is solved by means of column generation, we need to provide an initial set of columns that guarantees the LP feasibility of the restricted master problem. The initial set of columns can have a significant impact on the performance of the branch-and-price algorithm, especially for large instances. To initialize the first restricted master problem, we use the feasible solution found by the heuristic of Reyes et al. (2017). To initialize subsequent restricted master problems, we derive a set of columns from the active delivery schedule, i.e., the set of vehicle routes obtained when solving the previous problem. Obviously, if the active delivery schedule remains feasible after a change in a customer itinerary, then we can take the as-yet unexecuted parts of the routes to define columns to initialize the next restricted master problem. Otherwise, one of the routes will have become infeasible, and we attempt to restore feasibility by modifying this infeasible route.

Suppose that c is the critical customer, i.e., the customer whose itinerary change is revealed at time t, i is the planned delivery location for c, and $[e'_i, l'_i]$ is the updated time window of this customer at location i. If the customer's time window $[e'_i, l'_i]$ and the time window of the delivery vehicle performing the drop-off, $[ea_i, ld_i]$, overlap, then the planned delivery route remains feasible. In this case, we omit the executed part of each route r for which $z_r = 1$ in the previous problem, and use the resulting set of columns to initialize the restricted master problem associated with *SP*. Otherwise, we try to recover a feasible starting solution by applying the procedure in Algorithm 1, which is described below.

Algorithm 1: *restoreFeasibility*($\bar{r}_f$, i, t).

Data: Time t at which the latest update occurs, the route segment $\bar{r}_f$ with origin $o \in O_t$, and the node i in $\bar{r}_f$ whose time window update is revealed at time t
Result: A set of routes to be used when initializing the restricted master problem for *SP*
Define a GTSPTW over the subgraph of the problem graph of *SP* induced by $\bar{N} = \bigcup_{j\in\Psi} N^t_{c(j)} \cup \{o, 0'\}$ where Ψ is the set of nodes in $\bar{r}_f$ corresponding to customer locations
if *GTSPTW is feasible* **then**
    Find the optimal GTSPTW tour r^*
    Return r^*
else
    if $l'_i < l_i$ **then**
        Return *earlyDepartureRecovery*($\bar{r}_f$, i, t)
    else if $e'_i > e_i$ **then**
        Return *lateArrivalRecovery*($\bar{r}_f$, i, t)

Let $\bar{r}$ with $z_{\bar{r}} = 1$ be the route containing node i, $\bar{r}_f$ be the as-yet unexecuted part of $\bar{r}$, Ψ be the set of nodes in $\bar{r}_f$ corresponding to customer locations, and $o \in O_t$ be the origin of the vehicle performing $\bar{r}$. We know that this route has become infeasible as a result of a change in the customer itinerary of c, but it may be possible to identify a feasible route by changing the customer sequence or the delivery locations for some customers in $\bar{r}_f$. To this end, we take the subgraph $\bar{G}$ of the problem graph induced by the node set $\bar{N} = \bigcup_{j\in\Psi} N^t_{c(j)} \cup \{o, 0'\}$, and attempt to solve, using CPLEX, a generalized traveling salesman problem with time windows (GTSPTW) to obtain an alternative route, starting from o and ending at the

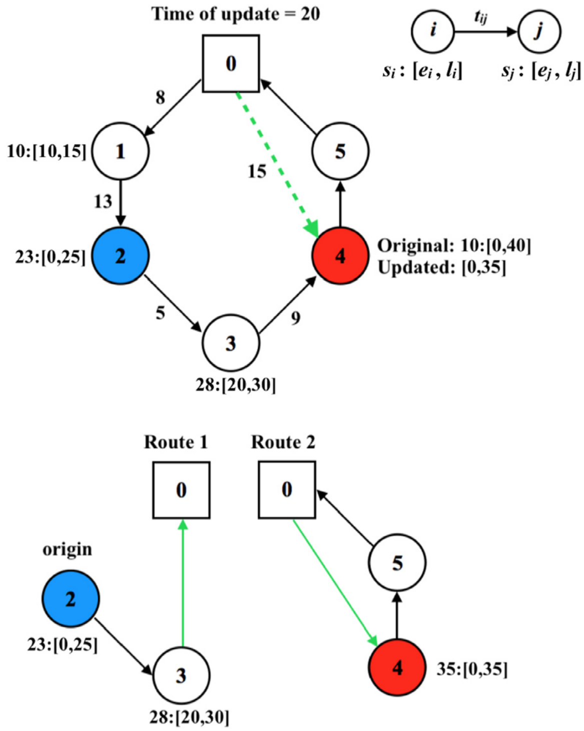

Fig. 1. Two feasible routes obtained by route splitting where s_i is the service start time at node i.

depot, in which all customers in the set $\bigcup_{j \in \Psi} c(j)$ are served. However, the instance of GTSPTW defined over the graph $\bar{G}$ may be infeasible. In that case, we invoke the route splitting subroutine presented in Section 3.3.1, which tries to split $\bar{r}_f$ in such a way that it produces two feasible routes. If no splitting opportunities exist, the node elimination subroutine presented in Section 3.3.2 is invoked.

It is important to note here that in recovering a feasible solution, we do not alter the planned delivery routes that remain feasible after the update at time t. Instead, we try to achieve feasibility by modifying the unexecuted part of $\bar{r}$ and, if needed, dispatching additional vehicles from the depot. However, our attempts may fail as it is not always possible to restore feasibility without changing the routes of other vehicles (e.g. a vehicle dispatched from the depot at time t may not be able to reach any $j \in N_c^t$ in time, while one of the vehicles already out for delivery can). In such cases, we define an artificial out-and-back route from the depot for customer c with a very high cost, and remove i from $\bar{r}_f$. If this artificial column appears in the final solution to *SP*, then we can conclude that customer c cannot be served unless another customer itinerary change is revealed which makes a delivery to customer c possible again. Details of our recovery heuristic are given in Algorithm 1, where *pred*(j) and *succ*(j) for $j \in \bar{r}_f$ correspond to the immediate predecessor and the immediate successor of j. Except for solving a GTSPTW, all recovery subroutines proposed as a part of our solution approach (here and later) are based on deleting locations from a given route.

3.3.1. Route splitting

Suppose that we have $l_i' < l_i$, that is, customer c has left location i earlier than expected. Since travel times satisfy the triangle inequality, we know that the vehicle assigned to $\bar{r}$ can return to the depot by time T after visiting the location that immediately precedes i in $\bar{r}_f$, for otherwise $\bar{r}$ would not be feasible before the itinerary update (recall that $\bar{r}$ is the route that becomes infeasible because of a change in the time window at node i). Furthermore, since $\bar{r}$ has become infeasible, we also know that $l_i' < ea_i$, and thus $l_i' < ld_i$. This implies that a vehicle which departs from location i at time l_i' can serve the subsequent customers in $\bar{r}_f$ within their respective time windows and return to the depot on time. Hence, if a new vehicle dispatched from the depot at time t can reach location i by time l_i', it means that we can split $\bar{r}_f$ into two feasible routes, one executing the part of the original route up to and including the predecessor of i, and one executing the part of the original route starting from i. An example of route splitting is illustrated in Fig. 1. Of course, there may exist alternative splitting opportunities. If that is the case, we select the one that incurs the least cost among all alternatives. Similar reasoning applies when $e_i' > e_i$, i.e., when customer c arrives at location i later than expected. Details of the route splitting procedure are provided in Algorithms 2 and 3 for early departure and late arrival cases, respectively.

Algorithm 2: *earlyDepartureRecovery*($\bar{r}_f$, i, t).

```
Let F = ∅
  ▷ Check whether recovery is possible by route splitting; if yes, find the node for which splitting the route incurs the least cost
if t + t_{0i} ≤ l'_i then
    Let splitNode = i, minCostOfSplit = w_{pred(i),0'} + w_{0,i} − w_{pred(i),i}, u_1 = pred(i), u_2 = i, and LAT = l'_i
    while u_1 ≠ o do
        if t + t_{0,u_1} ≤ l_{u_1} and max{e_{u_1}, t + t_{0,u_1}} + t_{u_1,u_2} ≤ LAT then
            if w_{pred(u_1),0'} + w_{0,u_1} − w_{pred(u_1),u_1} < minCostOfSplit then
                splitNode ← u_1, minCostOfSplit ← w_{pred(u_1),0'} + w_{0,u_1} − w_{pred(u_1),u_1}
            LAT ← LAT − t_{u_1,u_2}, u_2 ← u_1, u_1 ← pred(u_1)
        else
            Break the loop
    F ← F ∪ {r_1, r_2} where r_1 = (o, ..., pred(splitNode), 0') and r_2 = (0, splitNode, ..., 0')
  ▷ Else, attempt to recover by creating an out-and-back route from the depot or by node elimination
else
    Let u_0 = findNodeForOutAndBackRoute(c(i))
  ▷ If the critical customer can be served in an out-and-back route from the depot, or if node elimination is not applicable, remove
    the critical customer from its current route and create an out-and-back route from the depot for that customer
    if u_0 ≠ −1 or pred(i) = o or st_o + t_{oi} > l'_i then
        Define r_1 = r̄_f \ {i} and r_2 = (0, j, 0'), where j = u_0 if u_0 ≠ −1; else, j = i and r_2 is an artificial column
        F ← F ∪ {r_1, r_2}
    ▷ Else, execute the node elimination procedure
    else
        Let u_1 = pred(i), u_2 = i, LAT = l'_i, and H_0, H_1 = ∅
        while u_1 ≠ o do
            u_0 ← findNodeForOutAndBackRoute(c(u_1))
            if u_0 ≠ −1 then
                H_0 ← H_0 ∪ {u_0} and H_1 ← H_1 ∪ {u_1}
                if pred(u_1) = o or max{ea_{pred(u_1)}, e_{pred(u_1)}} + t_{pred(u_1),u_2} ≤ LAT then
                    for j ∈ H_0 do
                        F ← F ∪ {(0, j, 0')}
                    F ← F ∪ {r̄_f \ H_1}
                    Break the loop
                else
                    u_1 ← pred(u_1)
            else if pred(u_1) ≠ o and max{st_o + t_{o,u_1}, e_{u_1}} + t_{u_1,u_2} ≤ LAT then
                LAT ← LAT − t_{u_1,u_2}, u_2 ← u_1, u_1 ← pred(u_1)
            else
                Define r_1 = r̄_f \ {i} and r_2 = (0, i, 0'), where r_2 is an artificial column
                F ← F ∪ {r_1, r_2}
                Break the loop
Return F
```

3.3.2. Node elimination

If splitting fails, our final attempt to restore feasibility is to use a node elimination subroutine in which we create an out-and-back route from the depot for one or more customers and remove their delivery locations from $\bar{r}_f$. More specifically, we first check whether there exists a $j \in N_c^t$ for which $0 - j - 0'$ is feasible (Algorithm 4). If we can find such a location j in the itinerary of customer c, then the recovery procedure returns the out-and-back route $0 - j - 0'$ and the route obtained by removing node i from $\bar{r}_f$. Otherwise, starting with i's immediate predecessor (in the early departure case, i.e., $l'_i < l_i$), we iterate over the nodes of $\bar{r}_f$ that were to be visited prior to i and apply the following steps. If no feasible out-and-back route exists for the customer associated with the current node, then we move to the previous node and continue. Else, we remove the current node from $\bar{r}_f$, and check whether the resulting route is feasible. We stop as soon as feasibility is achieved, in which case the recovery procedure returns the shortened route and a set of out-and-back routes (for the nodes removed from $\bar{r}_f$).

When node elimination fails to produce a feasible set of routes, the recovery procedure returns an artificial out-and-back route for customer c and the route obtained by removing node i from the original $\bar{r}_f$ (the one containing all nodes in $\Psi \backslash \{i\}$). Fig. 2 depicts an example of node elimination. Details are given in Algorithm 2. The recovery in case of late arrival of customer to location i ($e'_i > e_i$) is handled in a similar fashion (see Algorithm 3). The only difference is that we iterate over the nodes of $\bar{r}_f$ that were planned to be visited after location i starting with the immediate successor of i if no feasible out-and-back route for customer c can be found.

Algorithm 3: *lateArrivalRecovery*$(\bar{r}_f,\ i,\ t)$.

```
Let F = ∅
    ▷ If route splitting or node elimination is not possible, remove the critical customer from its current route, and create an out-and-back route from the depot for that customer
if succ(i) = 0′ or e′_i + t_{i0′} > T then
    Let u_0 = findNodeForOutAndBackRoute(c(i))
    Define r_1 = r̄_f \ {i} and r_2 = (0, j, 0′), where j = u_0 if u_0 ≠ −1; else, j = i and r_2 is an artificial column
    F ← F ∪ {r_1, r_2}
▷ Else, execute the recovery procedures
else
    Let splitNode = −1, minCostOfSplit = +∞, u_1 = i, u_2 = succ(i), and EDT = e′_i
        ▷ Look for a (least cost) route splitting opportunity
    while u_2 ≠ 0′ do
        if t + t_{0,u_2} ≤ min{l_{u_2}, ld_{u_2}} and w_{u_1,0′} + w_{0,u_2} − w_{u_1,u_2} < minCostOfSplit then
            splitNode ← u_2, minCostOfSplit ← w_{u_1,0′} + w_{0,u_2} − w_{u_1,u_2}
        if min{l_{u_2}, T − t_{u_2,0′}} − t_{u_1,u_2} ≥ EDT then
            EDT ← EDT + t_{u_1,u_2}, u_1 ← u_2, u_2 ← succ(u_2)
        else
            Break the loop
    if splitNode ≠ −1 then
        F ← F ∪ {r_1, r_2} where r_1 = (o, ..., pred(splitNode), 0′) and r_2 = (0, splitNode, ..., 0′)
    ▷ If recovery by route splitting is not possible, attempt to recover by creating an out-and-back route from the depot or by node elimination
    else
        Let u_0 = findNodeForOutAndBackRoute(c(i))
            ▷ If the critical customer can be served in an out-and-back route from the depot, remove the critical customer from its current route, and create an out-and-back route from the depot for that customer
        if u_0 ≠ −1 then
            F ← F ∪ {r̄_f \ {i}, (0, u_0, 0′)}
            ▷ Else, execute the node elimination procedure
        else
            u_1 ← i, u_2 ← succ(i), and EDT ← e′_i
            Let H_0, H_1 = ∅
            while u_2 ≠ 0′ do
                u_0 ← findNodeForOutAndBackRoute(c(u_2))
                if u_0 ≠ −1 then
                    H_0 ← H_0 ∪ {u_0} and H_1 ← H_1 ∪ {u_2}
                    if succ(u_2) = 0′ or EDT + t_{u_1,succ(u_2)} ≤ min{l_{succ(u_2)}, ld_{succ(u_2)}} then
                        for j ∈ H_0 do
                            F ← F ∪ {(0, j, 0)}
                        F ← F ∪ {r̄_f \ {i}}
                        Break the loop
                    else
                        u_2 ← succ(u_2)
                else if succ(u_2) ≠ 0′ and EDT + t_{u_1,u_2} ≤ min{l_{u_2}, T − t_{u_2,0′}} then
                    EDT ← EDT + t_{u_1,u_2}, u_1 ← u_2, u_2 ← succ(u_2)
                else
                    Define r_1 = r̄_f \ {i} and r_2 = (0, i, 0′), where r_2 is an artificial column
                    F ← F ∪ {r_1, r_2}
                    Break the loop
Return F
```

Algorithm 4: *findNodeForOutAndBackRoute*$(c,\ t)$.

```
Let node = −1 and minCost = +∞
for j ∈ N_c^t do
    if t + t_{0j} ≤ l_j and max{t + t_{0j}, e_j} + t_{j0′} ≤ T and w_{0j} < minCost then
        node ← j
        minCost ← w_{0,j}
Return node
```

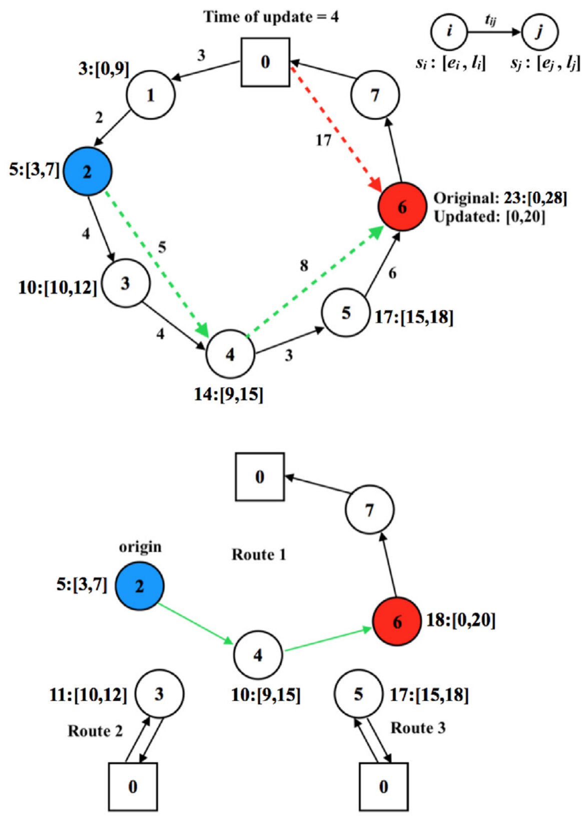

Fig. 2. Feasible routes obtained by removing nodes 3 and 5 from the route where s_i is the service start time at node i.

3.4. Generating additional initial columns

Usually, sufficient time is available to solve the initial problem during the planning stage, and, thus, it is possible to find optimal or high-quality solutions. On the other hand, only a short amount of time is available to solve the re-optimization problems during the execution stage. To speed up the solution of a re-optimization problem, we propose to re-use (parts of) columns that have been generated during preceding solves.

Let Ω contain the columns used to initialize the restricted master problem associated with the preceding (re-)optimization problem and the columns generated during the solution of the preceding (re-)optimization problem. We exclude the columns in the final solution to the preceding (re-)optimization problem as we have already described how they are processed to create an initial solution to *SP*. Suppose that for each route $r \in \Omega$, we retain the part of r for which at least one vehicle can perform a delivery at the first node, i.e., taking into account the earliest departure time of the vehicle from its origin (st_o) and its remaining delivery capacity (qr_o). This is equivalent to identifying the first node in r that has not been eliminated from the problem graph of *SP* during the preprocessing phase and taking the part of r from that node to the end. Note that it may not be possible to make deliveries to all the locations in the retained part of the route, r_f, for several reasons. First, it may contain nodes and/or arcs that are not in the current problem graph. Second, it may include the node whose time window has changed in the latest update, which may render it infeasible. Finally, being able to reach the first node in time and having enough remaining delivery capacity to make a delivery does not guarantee that the vehicle will also be able to make deliveries at subsequent locations and return to the depot in time.

The simplest way to obtain a set of additional initial columns for *SP* is to generate, as described above, a route segment r_f for every $r \in \Omega$, check whether that route segment can be executed (in its entirety) by at least one vehicle, and, if so, keep it and otherwise ignore it. However, in that way, we may be discarding (smaller) route segments that can yield useful columns. Hence, for each route segment r_f extracted from Ω, we apply the following five-step procedure to convert it to a feasible route.

1. Identify the nodes in r_f that cannot be visited in any feasible solution to *SP* and remove them from r_f.
2. Identify the arcs in r_f that cannot be traversed in any feasible solution of *SP* and remove them from r_f, thereby, breaking it into smaller route segments. In particular, the number of route segments obtained will be equal to the number of arcs

removed plus one. Let $r_f^1, \ldots, r_f^\phi$ represent the smaller route segments resulting from the removal of the arcs of r_f that cannot be used in any feasible solution of *SP*. Assume without loss of generality that $r_f^1, \ldots, r_f^\phi$ are ordered such that for $1 \le k < l \le \phi$, i.e., the nodes of r_f^k come before the nodes of r_f^l in the original route segment r_f. This implies that only r_f^ϕ will contain the node $0'$, and, therefore, we append $0'$ to the end of r_f^k for all $k < \phi$.

3. Assign an origin to r_f^k for $k = 1, \ldots, \phi$. First, we determine candidate origins, i.e., the origins which have a vehicle that can reach the first node of r_f^k before its time window closes and that has enough remaining delivery capacity to make the delivery at the first node of r_f^k. Among the candidate origins, if any, we select one which reaches the first node of r_f^k the earliest. The selected origin, say o_k, is added to the beginning of r_f^k.
4. Ensure time feasibility of route r_f^k for every $k = 1, \ldots, \phi$, by removing nodes from r_f^k if necessary.
5. Ensure capacity feasibility of route r_f^k for every $k = 1, \ldots, \phi$ by removing nodes from r_f^k if necessary.

Further details of the last two steps are provided below and in Algorithms 5–7.

3.4.1. Fixing time window violations

Given a route r with origin o, we consider two cases when fixing time window violations. We assume that for the nodes j in r, we have computed values ea_j, the earliest time the vehicle can reach j, and ld_j, the latest time the vehicle can depart from j, even though at this time $ea_j > ld_j$ for one or more j in r. **Case (i):** r contains an $i \in N_c^t$, i.e., a location in the itinerary of the critical customer, and the arc (o, i) is in the problem graph of *SP*. If (o, i) is not in the problem graph, we remove i from r (the resulting route falls under Case (ii)).

First, as in Algorithm 1, we solve an instance of the GTSPTW to try and recover feasibility. If successful, we have found a time feasible route. If not, let $\Gamma_p = (o, \ldots, pred(i))$ and $\Gamma_s = (succ(i), \ldots, 0')$ be the parts of r that precede and succeed i, respectively. We try to restore time feasibility by removing nodes from Γ_p and Γ_s. We iterate over the nodes in Γ_p starting with $succ(o)$. For a given node j, we calculate the earliest possible arrival time at j of a vehicle leaving the origin o at time st_o and serving the customers that precede $c(j)$ in r. If the earliest possible arrival time at j exceeds l_j, or if the earliest possible arrival time at i after serving j exceeds $\min\{l_i, T - t_{i,0'}\}$, we remove j from r. After processing all nodes in Γ_p, we can compute the earliest possible arrival time at_i at node i. Next, we find the first node $j \in \Gamma_s$ for which $\max\{at_i, e_i\} + t_{ij} \le \min\{l_j, ld_j\}$, and remove nodes $succ(i), \ldots, pred(j)$ in Γ_s. The existence of such a node j is guaranteed by the assumption that the arc (o, i) is in the problem graph, which implies that at least node $0'$ satisfies the above condition. The route defined by Γ_p, i, and Γ_s is time feasible. **Case (ii):** r does not contain an $i \in N_c^t$.

Starting with $succ(o)$, we iterate over the nodes in r until we find a node j for which $ea_j > \min\{l_j, ld_j\}$. Then, we let $dt_{pred(j)} = \max\{e_{pred(j)}, ea_{pred(j)}\}$ and determine the first node k after j such that $dt_{pred(j)} + t_{pred(j),k} \le \min\{l_k, ld_k\}$. Removing the nodes $i, \ldots, pred(k)$ from r produces a time feasible route.

Algorithm 5: *fixTWAndCapacityViolations*(r, c).

```
▷ Check whether customer c is in route r
Let criticalNode = −1
for j ∈ r do
    if c(j) = c then
        criticalNode ← j
        Break the loop
▷ If customer c is in route r, but not reachable from origin o, remove it from r
if criticalNode ≠ −1 and (o, criticalNode) ∉ A^k then
    r ← r \ {criticalNode}
    criticalNode ← −1
▷ Fix time window violations (if any) by iterating over the locations in r
if criticalNode ≠ −1 then
    r ← fixTWViolationsWithCriticalCustomer(r, criticalNode)
else
    r ← fixTWViolationsWithoutCriticalCustomer(r)
▷ Calculate the total demand of the customers in r
Let totalDemand = 0, u_1 = succ(o)
for j ∈ r do
    totalDemand ← totalDemand + d_c(j)
▷ If the total demand exceeds the available capacity, iteratively remove the last customer in r until capacity-feasibility is achieved
while totalDemand > qr_o and pred(0′) ≠ o do
    totalDemand ← totalDemand − d_c(pred(0′))
    r ← r \ {pred(0′)}
Return r
```

Algorithm 6: *fixTWViolationsWithCriticalCustomer*(r, *criticalNode*).

```
if r is infeasible wrt time window restrictions then
    Let i = criticalNode
    Define a GTSPTW over the subgraph of the problem graph of SP induced by N̄ = ⋃_{j∈Ψ} N^t_{c(j)} ∪ {o, 0′} where Ψ is the set of nodes in r̄
    corresponding to customer locations
    if GTSPTW is feasible then
        Find the optimal GTSPTW tour r*
        r ← r*
    else
        Let u_1 = o, u_2 = succ(o), and edt = st_o
        ▷ Iterate over the nodes in r that precede the critical node and delete the ones violating the time window-feasibility of
        the route
        while u_2 ≠ criticalNode do
            Let eat = edt + t_{u_1,u_2}
            if eat ≤ l_{u_2} and max{eat, e_{u_2}} + t_{u_2,i} ≤ min{l_i, T − t_{i,0′}} then
                edt ← max{eat, e_{u_2}}, u_1 ← u_2, u_2 ← succ(u_2)
            else
                r ← r \ {u_2}, u_2 ← succ(u_2)
        edt ← max{e_i, edt + t_{u_1,i}} and u_2 ← succ(u_2)
        ▷ Iterate over the nodes in r that succeed the critical node and delete the ones violating the time window-feasibility of
        the route
        while u_2 ≠ 0′ do
            if edt + t_{i,u_2} > min{l_{u_2}, ld_{u_2}} then
                r ← r \ {u_2}, u_2 ← succ(u_2)
            else
                Break the loop
Return r
```

Algorithm 7: *fixTWViolationsWithoutCriticalCustomer*(r).

```
if r is infeasible wrt time window restrictions then
    Let u_1 = o, u_2 = succ(o), and edt = st_o
    ▷ Iterate over the nodes in r and delete a sequence of nodes to ensure time window feasibility
    outer_loop
    while u_2 ≠ 0′ do
        Let eat = edt + t_{u_1,u_2}
        ▷ If the sequence (o, ..., u_2, 0′) is time window feasible, go to the next node in r, and perform another iteration of the
        outer_loop
        if eat ≤ min{l_{u_2}, T − t_{u_2,0′}} then
            edt ← max{eat, e_{u_2}}, u_1 ← u_2, u_2 ← succ(u_2)
        ▷ Else, remove u_2 from r, and set the next node in r as u_2
        else
            r ← r \ {u_2}, u_2 ← succ(u_2)
            ▷ Iterate over the nodes in the subsequence (u_2, ..., pred(0′)) of r and remove all nodes from r until r is reduced to a
            time window feasible sequence (o, ..., u_1, u_2, ..., 0′)
            while u_2 ≠ 0′ do
                if edt + t_{u_1,u_2} > min{l_{u_2}, ld_{u_2}} then
                    r ← r \ {u_2}, u_2 ← succ(u_2)
                else
                    Break outer_loop
Return r
```

3.4.2. Fixing capacity constraint violations

Given a time feasible route r with origin o, we check whether the remaining delivery capacity of the vehicle is sufficient to serve the demand of all customers in r. If not, then we repeatedly remove the last customer from the route until it is capacity feasible.

3.4.3. Storing & processing columns

For efficiency, we store the columns in Ω in a tree structure to facilitate processing columns with a common initial sequence of nodes. Only after the third step of the procedure described in Section 3.4, we extract the columns from the tree as a set, and iterate over that set to fix time window and capacity violations. An example of a "column tree" is depicted in Fig. 3 in which nodes represent customer locations and every path from s^* to a leaf node $0'$ corresponds to a column (a vehicle route).

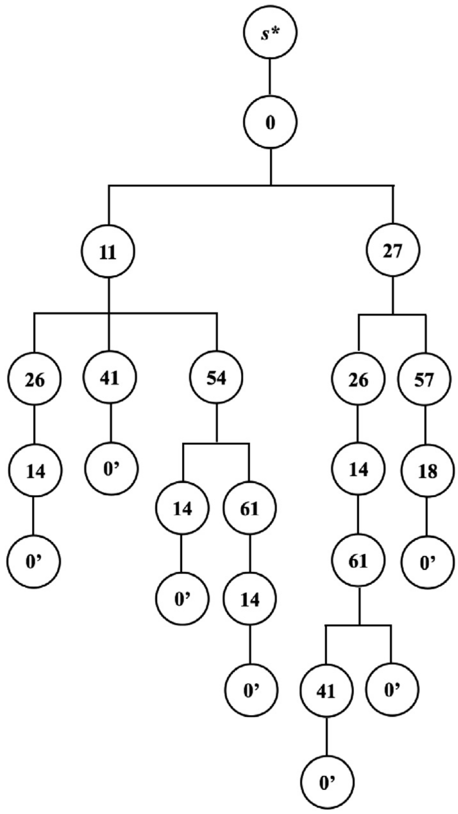

Fig. 3. An example tree of columns.

4. Computational study

Our computational study explores whether a branch-and-price algorithm can be used effectively to solve the dynamic variant of the VRPRDL. The results demonstrate that, generally, the delivery schedule can be re-optimized efficiently after a change in a customer itinerary is revealed, especially when information from preceding solves is exploited. Not surprisingly, the few exceptions arise when a change in a customer itinerary occurs almost immediately after execution of the delivery schedule commences, i.e., very early on in the planning horizon. In such cases, solving the re-optimization problem is almost equivalent to (and sometimes even harder than) solving the original planning problem, and, the information transferred from the solve of the original planning problem may not be sufficient to drastically reduce computation times. Fortunately, however, even if the transferred information may not help to solve the re-optimization problem to optimality in a short amount of time, it does help to find high-quality solutions quickly. Details regarding the test instances, algorithm implementation, experimental setup, and the results of the experiments along with their analyses are provided in the following sections.

4.1. Test instances & update generation scheme

We perform our experiments using the small and medium size instances in the first set of test problems used in Ozbaygin et al. (2017). To generate changes in the customer itineraries, we use the following procedure. Let K be the maximum number of itinerary changes that will be revealed during a given time horizon $[T_1, T_2]$. We divide $[T_1, T_2]$ into K intervals of equal length and attempt to generate one itinerary change in each interval using the following procedure. For $k = 1, \ldots, K$:

1. Randomly pick a point t in time interval $[(k-1)(T_2 - T_1)/K, k(T_2 - T_1)/K]$ at which the delivery planner will be notified about the next itinerary change.
2. Randomly select a route until finding one for which the assigned vehicle still has at least one delivery to make. Stop if no such route can be found. Else, go to Step 3 with the selected route r.
3. Randomly select a node in r until finding one for which multiple locations exist in the associated customer's itinerary. Stop if no such node can be found. Else, go to Step 4 with the selected node i.
4. Compute the time window $[ea_i, ld_i]$ for the vehicle assigned to route r at location i, i.e., the earliest and the latest times that the vehicle can arrive at and has to depart from location i.

5. Decide on the type of itinerary update for customer $c(i)$. If $l_i - ea_i \leq ld_i - e_i$, we choose early departure; otherwise, we choose late arrival.
6. If the type of update is early departure, starting with i, we shrink the time windows of the nodes $j \in N_{c(i)}$: $l_j \leq l_i$, i.e., the nodes corresponding to the locations visited in the customer's itinerary preceding the arrival at i, one at a time, until $l'_i < ea_i$ (i.e., customer $c(i)$ leaves location i before the earliest possible arrival time of the vehicle that is supposed to serve the customer at location i), or shrinking has been applied to all nodes (in which case, the vehicle route involving customer $c(i)$, and, thus, the latest delivery schedule will remain feasible), whichever happens first. When we shrink $[e_j, l_j]$, the time windows associated with the nodes succeeding j in $N_{c(i)}$ are shifted backwards to preserve the travel times, except for the last node, i.e., the customer's home location, which absorbs the change (i.e., its time window expands). In the late arrival case, we shrink time windows associated with the nodes $j \in N_{c(i)}$: $e_j \geq e_i$ in a similar fashion. After shrinking $[e_j, l_j]$, the time windows of the nodes in $N_{c(i)}$ that precede j are shifted forward, except for the first, which absorbs the change.

Note that when we shrink the time window for a node j, we decrease its length by $\lfloor 0.75(l_j - e_j) \rfloor$. The shrinking coefficient is chosen as 0.75 so that the time windows do not become unreasonably narrow, and, more importantly, the resulting itinerary updates are more likely to render the planned delivery schedule infeasible, and, therefore, more useful from a computational point of view, especially in observing the effectiveness of the proposed recovery procedures and the benefits of employing recovered information in re-optimization. Note also that the above scheme may not always produce an update. For example, all vehicles may have performed their routes by time t, or the remaining part of the selected route may not contain any customer whose itinerary involves multiple locations. This is why K represents the *maximum* number of updates.

4.2. Implementation and experimental setup

The iterative re-optimization scheme is implemented in Java using the branch-and-price framework of *jORLib* (Kinable et al., 2016) and *JGraphT* (Michail et al., 2019). After setting up an optimization problem, we invoke the branch-and-price algorithm described in Ozbaygin et al. (2017) with pricing parameter configuration (5, 5, F), i.e., at most five columns are returned to the restricted master at every pricing iteration, five non-dominated labels are stored at each node, and multiple iterations are allowed in truncated label setting), and branching rule *MFC* (select the arc that appears most frequently in a given fractional solution, and, in case of ties, select the one with value closest to 0.5).

Restricted master problems are solved with CPLEX 12.7 accessed through Concert Technology. All experiments are performed on a single thread of a 64-bit VM with Intel Xeon E5-2640 v4 processor at 2.40 GHz. We focus on the single update case (i.e., $K = 1$ in the update generation scheme), because we observed during preliminary experiments that when multiple updates occur during the planning horizon, the associated re-optimization problems tend to become easier to solve. This is not surprising because the problem size (and thus, the solution space) tends to get smaller with time (due to the fact that fewer deliveries and fewer customer locations remain). Hence, focusing on the first re-optimization problem provides the most valuable insights into the potential of using branch-and-price in a dynamic environment.

4.3. Results

First, we conduct an experiment to analyze the impact of (1) the scheme used to initialize the restricted master problem of the re-optimization problem, i.e., the set of columns found during the solution of the planning problem that is re-used, and (2) the time that a change in a customer itinerary occurs (is revealed to the decision maker).

For a given re-optimization problem, we experiment with the following five initialization strategies for the associated restricted master problem:

(i) *FS*: the columns derived from the planned vehicle routes,
(ii) *RF: FS* plus the columns derived from those that were generated at the root node of the branch-and-price tree (eliminating infeasible ones),
(iii) *RR: FS* plus the columns derived from those that were generated at the root node of the branch-and-price tree (recovering infeasible ones),
(iv) *AF: FS* plus the columns derived from those that were generated throughout the entire branch-and-price tree (eliminating infeasible ones),
(v) *AR: FS* plus the columns derived from those that were generated throughout the entire branch-and-price tree (recovering infeasible ones).

In addition, to evaluate the value of re-using columns from the planning problem solve, we also solve the re-optimization problems with a base initialization scheme, *BS*, in which no columns generated earlier are re-used. In *BS*, we define a column for each customer that has not received a delivery before the update realization time (t) corresponding to an out-and-back route (possibly artificial) from the depot, and we define a column for each origin $o \in O_t \setminus \{0\}$ corresponding to a direct trip back to the depot.

Furthermore, we force the change of a customer itinerary to occur in one of three "update realization" intervals (0, T/4), [T/4, T/2), and [T/2, 3T/4), i.e., the first, second, and third quarter of the day.

Table 1
Instance groups & statistics from planning problem solve.

Instance group	Instances in group	Customers per instance	Pricing iterations	Columns generated	Nodes in B&P tree	Solution time (s)	Routes in solution
1–5	5	15	16.0	66.2	1.0	0.11	4.8
6–10	5	20	140.8	455.8	23.0	1.56	6.0
11–20	10	30	346.1	1010.1	63.6	6.35	6.8
21–30	10	60	521.7	1830.2	68.1	67.69	12.9

For the sake of brevity when presenting the results, we group the test instances based on the number of customers they involve. Table 1 provides the number of instances in each group as well as the number of customers per instance. Moreover, the last five columns of the table show the values of the typical branch-and-price related statistics for the planning problem (i.e., pricing iterations, columns generated, nodes in the branch-and-price tree, and solution time) and the number of routes in the optimal solution, all averaged over the instances belonging to the group. The results for the individual instances can be found in Table 4 of Appendix A.

Given an instance and an update realization interval, we invoke the update generation scheme, and, if it returns an itinerary change, we solve the resulting re-optimization problem. For each instance group and update realization interval, and each initialization scheme, Table 2 reports the average number of initial columns, pricing iterations, columns generated, nodes evaluated, and the average solution time. The results for the individual instances can be found in Tables 5–8 of Appendix B. Note that when recording re-optimization solution times, we account for the time spent on constructing the initial set of columns (by processing some or all of the columns, depending on the initialization strategy, from the planning problem solve).

Most importantly, we observe that although the delivery routes can be re-optimized quickly after a customer itinerary change (with all initialization schemes), it is evident that re-using columns from the planning problem solve, even using the simplest strategy, *FS*, in which only the optimal columns are re-used, yields notably better results compared to the base initialization scheme, *BS*. Moreover, we see that utilizing additional columns (i.e., using any of the strategies *RF, RR, AF*, and *AR*) reduces the number of pricing iterations, the number of columns generated, and the solution time, even though it does not always reduce the number of nodes evaluated. The results also demonstrate an important trade-off. Strategies *AF* and *AR*, which consider all columns generated during the planning problem solve, clearly result in the fewest number of pricing iterations, columns generated, and nodes evaluated, but, because of the additional initial processing time (to create the initial restricted master problem) and the larger linear and integer programs that have to be solved (due to the fact that there are more columns in the restricted master problem) they do not (necessarily) result in the fastest solution times.

The impact of the time of the itinerary change is also clearly observable. As expected, the later the change in a customer itinerary occurs, the easier it is to solve the re-optimization problem. The impact is also seen in the number of initial columns, pricing iterations, and columns generated. There is a huge difference between strategies *FS* and *AR* when the itinerary change occurs in the first quarter of the planning horizon, but the difference is almost negligible when the itinerary change occurs in the third quarter of the planning horizon. This is mainly due to the fact that the re-optimization problems, especially when the itinerary change is realized relatively late in the planning horizon, have a smaller solution space than their associated initial planning problems. The size of the graph corresponding to each planning and re-optimization problem, i.e., the number of nodes and the number of arcs, are provided in Table 4 of Appendix A and in Table 5 of Appendix B, respectively (column headings $|N|$ and $|A|$).

The results in Table 2 are insufficient to assess the value of the recovery procedures as the differences between *RF* and *RR* and between *AF* and *AR* are minimal. This is, in part, due to the fact that most of the re-optimization instances in this experiment are solved to optimality at the root node of the branch-and-price tree, or by evaluating only a small number of nodes (see the detailed results in Tables 6–8 of Appendix B). Hence, to better assess the value of the different initialization schemes and to gain further insights in the potential of using branch-and-price in dynamic environments, we conduct a second, more detailed computational experiment using Instance 29 for which re-optimization is, relatively speaking, more difficult.

However, before presenting the results of these computational experiments, we provide, in Table 3, more information about the itinerary changes produced by the update generation scheme, the impact of these changes on the feasibility of the planned delivery schedules as well as on the feasibility of the resulting re-optimization problem, and the effectiveness of the recovery procedures. Furthermore, for every update realization interval, we report the minimum, average, and maximum number of routes in a re-optimized solution that do not belong to the initial set of routes (over all feasible re-optimization problems for that interval) for the different initialization schemes (*FS, RF, RR, AF*, and *AR*).

We see that for 86 out of 90 instances, the update generation scheme produced an itinerary change, i.e., a re-optimization problem to be solved. For 78 out of these 86 re-optimization problems, the itinerary change caused the (planned) delivery schedule to become infeasible, and for 31, the re-optimization problem itself is infeasible. The recovery procedures were able to restore feasibility for the remaining 47 re-optimization problems.

Table 2
Results of re-optimization for each instance group .

	Init. strategy\Instance group	Update realization interval											
		(0, T/4)				[T/4, T/2)				[T/2, 3T/4)			
		1–5	6–10	11–20	21–30	1–5	6–10	11–20	21–30	1–5	6–10	11–20	21–30
Number of initial columns	BS	18.80	22.75	33.50	66.70	14.60	20.00	26.20	52.00	6.80	11.20	15.00	29.88
	FS	5.20	6.00	7.20	13.50	5.00	6.00	7.10	13.80	3.20	4.80	5.60	11.13
	RF	45.20	62.00	148.90	379.70	17.60	28.75	49.40	121.10	6.40	10.60	20.30	41.88
	RA	50.60	74.75	165.60	445.10	21.20	33.75	61.60	149.20	7.40	11.60	22.40	47.88
	AF	45.20	113.50	347.60	599.60	17.60	38.00	112.70	164.00	6.40	11.80	32.60	46.00
	AA	50.60	152.25	391.40	691.20	21.20	48.25	133.80	204.60	7.40	12.80	35.20	53.25
Number of pricing iterations	BS	25.60	34.75	98.20	268.30	11.80	18.25	29.80	64.70	3.80	5.80	10.10	19.25
	FS	7.60	10.25	53.40	202.90	5.40	8.00	19.00	32.40	2.20	3.20	4.20	7.38
	RF	5.60	6.75	28.30	135.60	4.20	6.75	11.30	18.80	3.40	3.00	3.60	5.38
	RA	5.80	6.50	27.50	135.30	4.40	6.25	10.90	16.90	2.60	3.00	3.80	5.13
	AF	5.60	6.25	16.80	93.20	4.20	5.50	9.20	15.60	3.40	2.80	2.50	5.38
	AA	5.80	4.75	20.70	92.40	4.40	5.25	9.40	15.00	2.60	2.80	2.70	5.00
Number of columns generated	BS	114.20	157.75	423.10	1206.40	44.80	75.75	131.50	304.00	9.40	20.40	34.80	80.38
	FS	27.40	32.50	210.10	778.00	13.20	21.50	64.20	122.20	3.40	6.80	10.10	24.50
	RF	18.00	24.75	94.10	461.90	10.00	18.50	36.60	69.80	4.60	5.60	7.40	14.63
	RA	17.00	21.75	97.90	451.40	10.60	17.75	31.40	58.20	3.20	6.00	8.10	14.13
	AF	18.00	19.25	49.70	315.30	10.00	15.50	28.30	56.10	4.60	5.20	4.40	14.00
	AA	17.00	14.00	58.70	301.10	10.60	15.00	26.00	47.10	3.20	5.40	4.90	13.13
Nodes evaluated	BS	1.00	1.00	6.00	8.20	1.00	1.00	1.20	1.40	1.00	1.00	1.00	1.00
	FS	1.00	1.00	4.60	17.60	1.00	1.00	2.00	1.60	1.00	1.00	1.00	1.00
	RF	1.00	1.00	4.00	16.80	1.00	1.00	2.00	1.40	1.00	1.00	1.00	1.00
	RA	1.00	1.00	3.00	17.80	1.00	1.00	2.20	1.40	1.00	1.00	1.00	1.00
	AF	1.00	1.00	3.00	12.20	1.00	1.00	2.00	1.20	1.00	1.00	1.00	1.00
	AA	1.00	1.00	4.00	12.40	1.00	1.00	2.20	1.60	1.00	1.00	1.00	1.00
Solution time (s)	BS	0.11	0.12	1.14	14.95	0.03	0.04	0.08	0.35	0.01	0.01	0.01	0.04
	FS	0.08	0.08	0.69	9.90	0.03	0.03	0.07	0.21	0.02	0.02	0.01	0.03
	RF	0.05	0.08	0.50	7.71	0.01	0.02	0.05	0.15	0.01	0.01	0.01	0.02
	RA	0.06	0.06	0.55	7.97	0.01	0.03	0.06	0.15	0.01	0.01	0.01	0.02
	AF	0.11	0.18	0.37	8.77	0.03	0.04	0.06	0.14	0.02	0.02	0.02	0.02
	AA	0.05	0.15	0.53	8.20	0.02	0.03	0.06	0.16	0.01	0.01	0.01	0.02

Table 3
Feasibility status, success of recovery, and route changes in re-optimization.

			Update realization interval		
			(0, $T/4$)	[$T/4$, $T/2$)	[$T/2$, $3T/4$)
Number of instances	no update		1	1	2
	infeasible after update		26	26	26
	infeasible re-optimization problem		4	9	18
	feasibility restored	GTSPTW	12	8	5
		Split route	2	2	0
		Node elimination	0	0	0
		Out-and-back route	8	7	3
Number of route changes	*FS*	min	0	0	0
		avg	1.63	1.14	0.3
		max	5	4	2
	RF	min	0	0	0
		avg	1.04	0.76	0.2
		max	4	3	1
	RR	min	0	0	0
		avg	0.96	0.76	0.2
		max	4	3	1
	AF	min	0	0	0
		avg	0.93	0.71	0.2
		max	4	3	1
	AR	min	0	0	0
		avg	0.85	0.71	0.2
		max	4	3	1

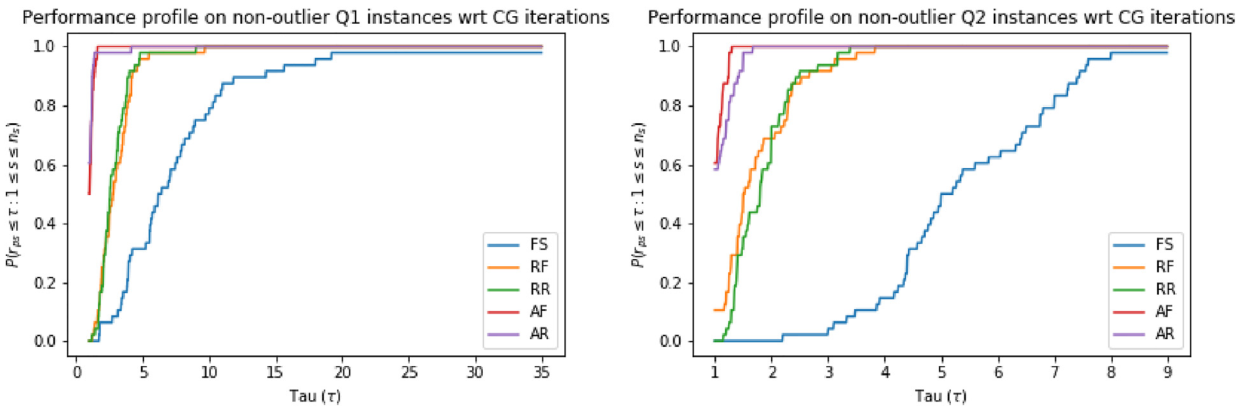

Fig. 4. Performance profiles comparing the number of column generation iterations for the five different initialization schemes for re-optimization problems corresponding to itinerary changes occurring in the first and the second quarter (for non-outlier instances).

Looking in more detail at the success rates of the recovery techniques, we observe that solving a GTSPTW succeeds in 25 cases (the majority), and that we have to resort to introducing an out-and-back route to serve the customer with the changed itinerary in 18 cases. Examining the number of route changes, which records the number of routes in the optimal solution to the re-optimization problem that were *not* among the routes in the initial restricted master problem, and, thus, had to be generated during the execution of the branch-and-price algorithm, is largest with *FS* and smallest with *AR* (on average). Moreover, for all initialization strategies, we observe that when an itinerary change occurs later in the planning horizon, few, if any, routes have to be generated during the execution of the branch-and-price algorithm because the re-optimization problems have smaller size, and in most cases, the routes in the optimal solution are already present in the initial restricted master problem.

Next, we analyze the results of our second set of computational experiments with Instance 29, in which we generated a total of 100 re-optimization problems, 50 in which the customer itinerary changes occur in the first quarter of the planning horizon and 50 in which the customer itinerary changes occur in the second quarter. Detailed results can be found in Tables 9 and 10 in Appendix C. (Recall that Instance 29 was the one for which the re-optimization problem in our first set of computational experiments was the most difficult.)

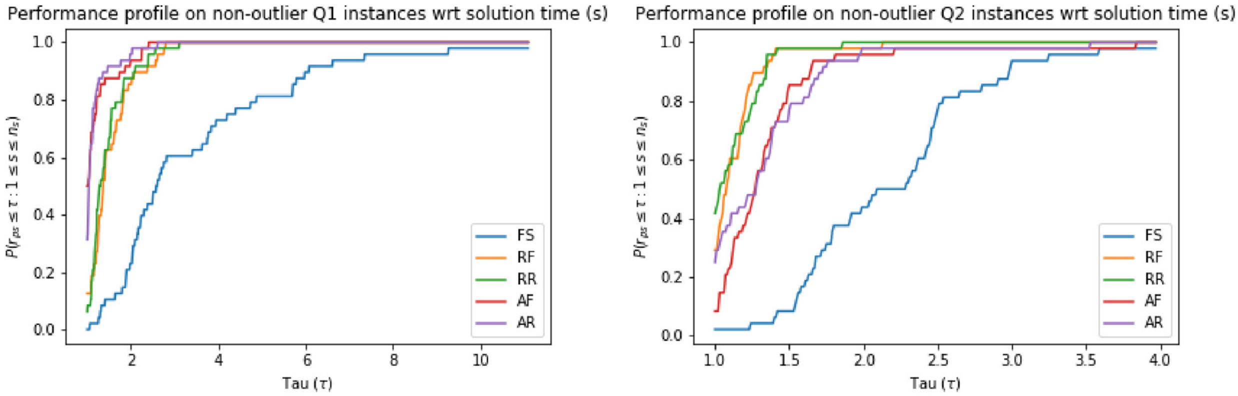

Fig. 5. Performance profiles comparing the solution time for the five different initialization schemes for re-optimization problems corresponding to itinerary changes occurring in the first and the second quarter (for non-outlier instances).

In Figs. 4 and 5, we present performance profiles Dolan and More (2002) comparing the five different initialization schemes for re-optimization problems corresponding to itinerary changes occurring in the first and in the second quarter, restricting ourselves to non-outliers.

We observe, as before, that the number of column generation iterations is smallest for the initialization schemes that consider all columns generated during the planning problem solve (Fig. 4). Interestingly, that is true also for the solution time, but only for the more difficult re-optimization problems, i.e., the re-optimization problem arising when the itinerary change occurs in the first quarter of the planning horizon (left plot in Fig. 5). When the itinerary change occurs in the second quarter of the planning horizon, and the re-optimization problems are easier, the extra time invested in processing columns and solving larger restricted master problems is not worth it (right plot in Fig. 5). In that case, re-using the columns generated at the root node during the solution of the planning problem leads to the best performance. It can also be seen that trying to recover feasibility (*AR*) is slightly better than only using feasible routes (*AF*) when solving more difficult instances, although not by much.

For a few re-optimization problems, i.e., the outliers pointed out in Appendix C, the search trees get rather large. In those cases, the initialization schemes that produce larger restricted master problems, i.e., *AF* and *AR*, suffer, as each solve takes longer. In a dynamic environment, however, it may not be necessary (or even best) to solve the re-optimization problem to optimality. Therefore, we focus next on how quickly a first feasible is found and the quality of the first feasible solution found when using different initialization schemes for the restricted master problem. Note that we solve the initial restricted master problem as an integer program before running branch-and-price for re-optimization (except when using *FS*, as the initial columns form the only, and thus optimal, feasible solution). In this way, the branch-and-price algorithm starts not only with a set of initial columns, but also with a (hopefully) strong upper bound. As a consequence, the first feasible solution refers to the solution obtained by solving the initial restricted master problem as an integer program provided that there exists a feasible solution to this problem. Otherwise, it corresponds to the first integer solution found by the branch-and-price algorithm. (Note that if a re-optimization instance does not have a feasible solution, this is detected at the root node of the search tree.)

The results can be found in Table 11 in Appendix C, where we present, for all re-optimization instances that have a feasible solution (a total of 80 instances), for each of the different initialization schemes, the number of columns in the initial restricted master problem, the time required to find a first feasible solution, and the quality of the first feasible solution, where we define quality as the ratio of the value of this solution to the value of the optimal solution. In Fig. 6, we provide a performance profile of the quality of the first feasible solution.

We see that re-using columns from the planning problem solve and spending time on recovering feasibility of these columns is beneficial, i.e., *AR* is better than *AF, RR* is better than *RF*, and all are better than *FS*. More specifically, the first feasible solution produced with initialization scheme *AR* is 60.21% better than the quality of the first feasible solution produced with initialization scheme *FS* (on average). Furthermore, the quality of the first feasible solution produced with initialization scheme *AR* is remarkably high: the optimality gap is 1.01% on average, and only 0.84% on average for the outlier instances.

If the feasibility of the solution from the planning problem can be restored, then the first feasible solution is found by solving the integer version of the initial restricted master problem. Otherwise, it will be found during the search. Because initialization schemes *AF* and *AR* produce the largest initial restricted master problems, in terms of number of columns, these take the longest to find a first feasible solution. However, the time is surprisingly small: 0.63 s on average for *AF* and 0.46 s on average for *AR* (another indication that restoring feasibility is beneficial); the maximum time is 10.10 s for *AF* and 7.48 s for *AR*.

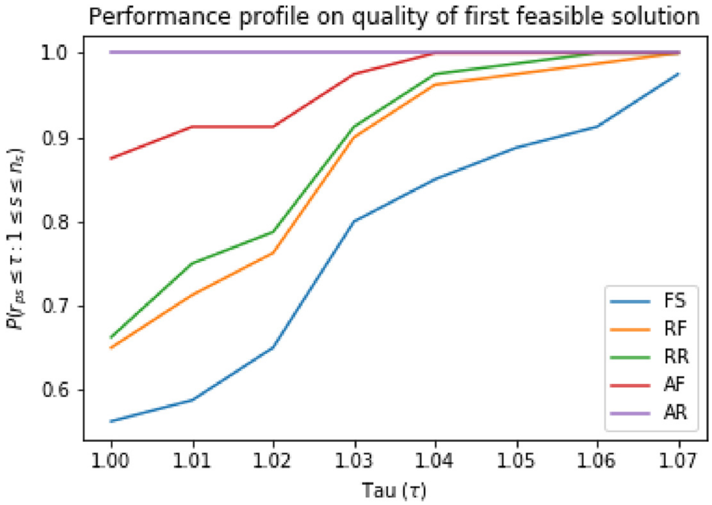

Fig. 6. Performance profile comparing the quality of the first feasible solution found for the five different initialization schemes for re-optimization problems.

For the sake of completeness, we report in Table 12 in Appendix C, for each re-optimization problem, and all initialization strategies, whether feasibility can be restored (for the instances where the customer itinerary change caused the planned delivery schedule to become infeasible), the type of recovery strategy (where N/A indicates that the delivery schedule remains feasible), the number of route changes (where – indicates that the re-optimization problem has no feasible solution).

5. Concluding remarks

We have demonstrated that it is possible to use a branch-and-price algorithm in a dynamic decision making environment. Specifically, in the context of a, newly introduced, dynamic variant of the VRPRDL. We present an iterative approach, which starts by solving the planning problem (an instance of the VRPRDL) based on (initial) customer itineraries, and re-optimizes delivery routes whenever a customer itinerary change is revealed (an instance of a slightly extended version of the VRPRDL). To ensure computational efficiency when solving re-optimization problems, we propose and employ methods that re-use information collected during the solution of previous optimization problems.

Although this is just one example in which a branch-and-price algorithm can be used effectively in a dynamic, operational setting, it is an encouraging example, and we hope it will stimulate others to pursue similar research efforts, and that, in the future, branch-and-price will be considered as one of the tools available for near real-time optimization.

Acknowledgement

Part of this research has been conducted when the first author was a visiting Ph.D. student at Georgia Institute of Technology. We gratefully acknowledge the financial support provided by the Scientific and Technological Research Council of Turkey, grant BIDEB-2214-A, that funded her visit.

Appendix A. Planning problem solve

Table 4
Detailed results for the planning problem solve .

Instance	$\|N\|$	$\|A\|$	Pricing iterations	Columns generated	Nodes in B&P tree	Solution time (s)	Routes in solution
1	42	610	28	117	1	0.18	4
2	35	406	11	47	1	0.09	5
3	40	548	16	71	1	0.18	4
4	30	340	13	56	1	0.06	5
5	33	282	12	40	1	0.03	6
6	43	625	197	592	47	1.98	5
7	37	559	329	1091	41	4.48	4
8	47	671	31	141	1	0.12	6
9	45	609	135	403	25	1.17	5
10	38	504	12	52	1	0.06	8
11	64	1452	110	431	11	1.96	7
12	72	1721	1452	3627	355	19.40	8
13	74	1766	54	236	1	1.66	6
14	64	1633	64	295	1	1.49	6
15	75	1695	40	189	1	1.10	6
16	76	1972	60	278	1	1.84	7
17	69	1447	70	315	1	1.20	8
18	52	865	49	214	1	0.49	7
19	58	1112	1478	4130	263	30.89	7
20	58	1472	84	386	1	3.45	6
21	140	6514	114	560	1	9.85	13
22	122	5171	216	1037	1	26.54	10
23	132	5161	222	1021	8	13.01	16
24	126	5011	133	623	1	25.12	11
25	140	6535	346	1629	3	76.97	11
26	118	4209	86	393	1	4.86	16
27	118	4745	190	871	1	23.47	10
28	132	6316	873	3470	59	177.37	14
29	115	4100	2937	8241	605	314.82	14
30	118	3834	100	457	1	4.91	14

Appendix B. Detailed results of the first experiment

Table 5
Detailed results with the base initialization scheme *BS*.

Instance	(0, T/4)						[T/4, T/2)						[T/2, 3T/4)					
	|N|	|A|	Pricing iterations	Columns generated	Nodes in B&P tree	Solution time (s)	|N|	|A|	Pricing iterations	Columns generated	Nodes in B&P tree	Solution time (s)	|N|	|A|	Pricing iterations	Columns generated	nodes in B&P tree	solution time (s)
1	34	354	32	142	1	0.15	26	149	15	59	1	0.05	13	46	4	8	1	0.03
2	32	293	19	83	1	0.07	19	73	7	22	1	0.01	6	8	2	1	1	0.01
3	38	477	36	161	1	0.20	21	135	21	86	1	0.05	11	31	4	13	1	0.01
4	33	370	28	126	1	0.08	21	103	8	29	1	0.02	13	38	4	11	1	0.01
5	40	372	13	59	1	0.04	29	136	8	28	1	0.02	9	24	5	14	1	0.01
6	34	287	24	111	1	0.11	26	136	15	63	1	0.06	16	43	4	12	1	0.01
7	33	300	36	163	1	0.10	18	63	10	30	1	0.01	8	13	2	2	1	0.00
–8	–	–	–	–	–	–	–	–	–	–	–	–	20	83	7	30	1	0.02
9	50	598	39	184	1	0.13	37	268	18	77	1	0.03	14	50	6	17	1	0.01
10	39	465	40	173	1	0.15	36	305	30	133	1	0.06	18	82	10	41	1	0.01
11	60	1102	79	382	1	0.73	45	420	27	121	1	0.06	21	102	13	53	1	0.01
12	81	2152	341	1216	47	5.67	51	555	37	175	1	0.11	28	141	14	57	1	0.02
13	69	1332	82	389	1	0.79	48	440	57	251	3	0.29	25	112	16	58	1	0.02
14	45	566	36	171	1	0.10	26	171	13	56	1	0.02	10	20	2	2	1	0.00
15	76	1635	97	470	1	1.35	43	263	21	92	1	0.04	21	63	7	19	1	0.01
16	65	1321	84	396	1	1.12	50	519	38	180	1	0.13	25	151	14	50	1	0.02
17	58	734	68	294	5	0.42	32	165	22	88	1	0.03	15	55	9	27	1	0.01
18	46	567	55	257	1	0.18	34	202	23	101	1	0.03	15	46	7	20	1	0.01
19	55	766	72	330	1	0.52	38	279	34	140	1	0.06	20	86	8	26	1	0.01
20	54	966	68	326	1	0.55	36	297	26	111	1	0.05	20	76	11	36	1	0.01
21	101	2520	133	641	1	3.72	59	579	31	150	1	0.11	–	–	–	–	–	–
22	120	4733	440	2069	1	44.12	77	1418	109	514	1	0.81	45	394	33	139	1	0.08
23	111	2613	135	650	1	2.96	75	715	37	180	1	0.12	29	126	7	23	1	0.02
24	124	3897	260	1176	3	16.28	74	853	56	264	1	0.29	31	191	19	81	1	0.03
25	118	3423	163	803	1	5.32	64	771	64	308	1	0.27	35	209	17	72	1	0.03
26	120	2836	186	902	1	2.83	71	652	39	185	1	0.10	–	–	–	–	–	–
27	109	2968	251	1199	1	13.16	67	888	80	375	1	0.50	31	191	24	96	1	0.04
28	126	4482	225	1102	1	21.60	67	797	48	226	1	0.22	34	258	19	78	1	0.04
29	115	3053	559	1887	71	24.78	78	970	101	454	5	0.63	37	175	13	60	1	0.02
30	126	4229	331	1635	1	14.76	89	1196	82	384	1	0.47	45	249	22	94	1	0.03

Table 6
Detailed results for update realization interval (0, $T/4$) with *FS, RF, RR, AF, AR*.

Instance	Initial columns					Pricing iterations					Columns generated					Nodes evaluated					Time (s)				
	FS	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*
1	5	73	82	73	82	6	3	4	3	4	15	6	5	6	5	1	1	1	1	1	0.09	0.06	0.07	0.20	0.07
2	5	32	34	32	34	6	5	4	5	4	17	8	3	8	3	1	1	1	1	1	0.07	0.03	0.03	0.07	0.03
3	4	38	48	38	48	8	8	10	8	10	34	31	37	31	37	1	1	1	1	1	0.12	0.11	0.12	0.22	0.11
4	5	43	43	43	43	10	6	6	6	6	40	25	25	25	25	1	1	1	1	1	0.09	0.04	0.03	0.04	0.03
5	7	40	46	40	46	8	6	5	6	5	31	20	15	20	15	1	1	1	1	1	0.05	0.03	0.04	0.04	0.04
6	5	55	59	140	180	5	5	7	5	2	17	20	21	13	4	1	1	1	1	1	0.04	0.06	0.03	0.09	0.06
7	5	54	76	175	265	16	6	5	4	3	44	18	16	3	2	1	1	1	1	1	0.16	0.13	0.12	0.50	0.44
8	7	94	113	94	113	11	11	8	11	8	45	48	34	48	34	1	1	1	1	1	0.05	0.05	0.06	0.07	0.07
10	7	45	51	45	51	9	5	6	5	6	24	13	16	13	16	1	1	1	1	1	0.05	0.08	0.05	0.07	0.05
11	8	194	201	305	315	31	9	9	7	6	140	25	31	19	17	1	1	1	1	1	0.25	0.18	0.14	0.10	0.11
12	8	230	245	1165	1240	245	146	146	46	83	834	434	491	56	123	35	29	19	19	29	4.24	3.24	3.19	1.73	2.99
13	6	121	139	121	139	50	16	14	16	14	229	65	56	65	56	1	1	1	1	1	0.52	0.29	0.29	0.30	0.24
14	6	94	101	94	101	11	10	10	10	10	40	39	31	39	31	1	1	1	1	1	0.05	0.07	0.08	0.08	0.06
15	7	138	142	138	142	48	37	40	37	40	228	146	168	146	168	1	1	1	1	1	0.77	0.50	1.02	0.58	0.93
16	7	177	189	177	189	19	16	13	16	13	86	55	52	55	52	1	1	1	1	1	0.21	0.24	0.24	0.25	0.21
17	8	66	75	118	146	41	26	17	17	19	170	102	59	58	64	3	3	3	3	3	0.23	0.22	0.19	0.34	0.34
18	7	126	139	126	139	32	11	10	11	10	144	33	33	33	33	1	1	1	1	1	0.11	0.11	0.13	0.12	0.13
19	8	178	236	1067	1314	17	5	5	1	1	61	16	15	0	0	1	1	1	1	1	0.15	0.09	0.09	0.10	0.12
20	7	165	189	165	189	40	7	11	7	11	169	26	43	26	43	1	1	1	1	1	0.35	0.11	0.17	0.16	0.17
21	13	180	205	180	205	65	27	33	27	33	314	120	153	120	153	1	1	1	1	1	1.75	0.80	1.03	0.93	1.02
22	10	790	884	790	884	180	71	105	71	105	850	326	470	326	470	1	1	1	1	1	13.19	5.88	12.06	6.31	10.84
23	16	195	201	421	435	46	33	36	21	19	211	158	164	94	71	1	1	1	1	1	1.44	0.70	1.11	0.87	0.63
24	12	319	414	319	414	232	127	105	127	105	1054	549	462	549	462	3	3	3	3	3	11.49	8.21	5.93	8.81	6.29
25	12	401	554	483	654	77	24	22	23	25	367	105	99	104	79	1	1	1	1	1	3.16	0.98	0.71	0.80	0.89
26	17	200	224	200	224	71	34	34	34	34	335	134	157	134	157	1	1	1	1	1	1.17	0.62	0.73	0.59	0.70
27	11	515	587	515	587	123	71	72	71	72	484	332	297	332	297	1	1	1	1	1	5.17	4.04	3.28	3.63	3.48
28	14	475	597	1159	1483	87	21	18	9	9	427	77	76	30	17	1	1	1	1	1	7.33	11.58	8.48	28.78	24.38
29	15	281	322	1488	1563	1066	897	879	498	473	3363	2600	2424	1246	1093	165	157	167	111	113	50.91	41.89	43.79	34.49	31.17
30	15	441	463	441	463	82	51	49	51	49	375	218	212	218	212	1	1	1	1	1	3.42	2.37	2.63	2.50	2.62

Table 7
Detailed results for update realization interval [T/4, T/2) with *FS, RF, RR, AF, AR*.

Instance	Initial columns					Pricing iterations					Columns generated					Nodes evaluated					Time (s)				
	FS	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*
1	5	30	38	30	38	4	3	3	3	3	6	7	7	7	7	1	1	1	1	1	0.04	0.01	0.01	0.04	0.01
2	4	11	14	11	14	3	2	1	2	1	3	1	0	1	0	1	1	1	1	1	0.02	0.01	0.01	0.02	0.01
3	4	11	11	11	11	6	6	6	6	6	16	20	20	20	20	1	1	1	1	1	0.03	0.01	0.01	0.02	0.02
4	5	15	16	15	16	3	3	3	3	3	8	4	8	4	8	1	1	1	1	1	0.02	0.01	0.01	0.01	0.02
5	7	21	27	21	27	11	7	9	7	9	33	18	18	18	18	1	1	1	1	1	0.04	0.02	0.03	0.04	0.02
6	5	33	36	62	77	4	4	4	1	1	11	10	10	0	0	1	1	1	1	1	0.03	0.01	0.02	0.08	0.03
7	5	21	26	29	43	7	5	4	3	3	17	5	4	3	3	1	1	1	1	1	0.02	0.02	0.01	0.01	0.03
8	7	35	39	35	39	12	8	7	8	7	35	28	26	28	26	1	1	1	1	1	0.04	0.02	0.02	0.02	0.02
10	7	26	34	26	34	9	10	10	10	10	23	31	31	31	31	1	1	1	1	1	0.03	0.03	0.05	0.03	0.04
11	7	58	73	93	113	10	7	3	3	3	32	26	8	8	6	1	1	1	1	1	0.04	0.03	0.02	0.06	0.04
12	8	69	85	298	368	22	17	14	5	4	94	73	62	11	13	1	1	1	1	1	0.11	0.09	0.08	0.08	0.10
13	7	52	61	52	61	84	43	44	43	44	273	117	104	117	104	11	11	13	11	13	0.36	0.22	0.23	0.22	0.25
14	6	28	34	28	34	8	7	6	7	6	27	30	21	30	21	1	1	1	1	1	0.02	0.02	0.02	0.02	0.02
15	7	34	46	34	46	10	11	11	11	11	35	38	35	38	35	1	1	1	1	1	0.02	0.04	0.03	0.03	0.03
16	7	57	70	57	70	8	8	9	8	9	25	34	36	34	36	1	1	1	1	1	0.04	0.05	0.08	0.06	0.08
17	7	31	33	50	54	10	5	5	4	4	27	14	14	9	9	1	1	1	1	1	0.02	0.01	0.01	0.01	0.01
18	7	41	50	41	50	11	5	3	5	3	39	16	4	16	4	1	1	1	1	1	0.03	0.02	0.03	0.03	0.02
19	8	73	93	423	471	12	7	7	3	3	23	8	8	10	10	1	1	1	1	1	0.04	0.03	0.04	0.06	0.05
20	7	51	71	51	71	15	3	7	3	7	67	10	22	10	22	1	1	1	1	1	0.04	0.03	0.04	0.02	0.05
21	14	60	75	60	75	12	9	8	9	8	50	32	28	32	28	1	1	1	1	1	0.07	0.06	0.11	0.09	0.08
22	11	209	274	209	274	58	30	24	30	24	248	126	84	126	84	1	1	1	1	1	0.51	0.36	0.28	0.34	0.29
23	17	73	81	112	122	14	12	8	6	4	58	43	33	17	13	1	1	1	1	1	0.07	0.07	0.04	0.04	0.04
24	12	108	138	108	138	22	26	25	26	25	98	109	105	109	105	1	1	1	1	1	0.14	0.18	0.17	0.15	0.17
25	12	121	166	142	196	25	10	11	15	9	50	36	35	51	27	1	1	1	1	1	0.11	0.07	0.07	0.10	0.06
26	17	67	74	67	74	15	16	10	16	10	67	58	42	58	42	1	1	1	1	1	0.06	0.07	0.05	0.09	0.06
27	11	192	224	192	224	45	22	24	22	24	118	63	48	63	48	1	1	1	1	1	0.23	0.14	0.14	0.15	0.15
28	15	101	129	157	206	17	14	11	4	4	72	61	39	15	12	1	1	1	1	1	0.09	0.13	0.10	0.05	0.09
29	15	128	157	441	563	92	38	39	17	33	351	126	131	46	75	7	5	5	3	7	0.63	0.33	0.38	0.26	0.53
30	14	152	174	152	174	24	11	9	11	9	110	44	37	44	37	1	1	1	1	1	0.18	0.13	0.12	0.12	0.12

Table 8
Detailed results for update realization interval $[T/2, 3T/4)$ with *FS, RF, RR, AF, AR* .

Instance	Initial columns					Pricing iterations					Columns generated					Nodes evaluated					Time (s)				
	FS	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*
1	5	9	11	9	11	2	4	3	4	3	1	6	2	6	2	1	1	1	1	1	0.03	0.01	0.01	0.04	0.01
2	1	2	2	2	2	1	2	2	2	2	0	1	1	1	1	1	1	1	1	1	0.02	0.01	0.01	0.01	0.01
3	4	5	5	5	5	3	3	3	3	3	8	8	8	8	8	1	1	1	1	1	0.02	0.01	0.00	0.01	0.01
4	4	9	9	9	9	2	2	2	2	2	4	3	3	3	3	1	1	1	1	1	0.01	0.01	0.01	0.01	0.01
5	2	7	10	7	10	3	6	3	6	3	4	5	2	5	2	1	1	1	1	1	0.01	0.01	0.01	0.01	0.01
6	6	11	11	13	13	2	2	2	1	1	1	2	2	0	0	1	1	1	1	1	0.01	0.01	0.01	0.02	0.01
7	3	3	3	3	3	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0.01	0.01	0.01	0.01	0.01
8	6	14	14	14	14	4	4	4	4	4	14	13	13	13	13	1	1	1	1	1	0.03	0.01	0.01	0.03	0.01
9	3	10	11	14	15	4	3	3	3	3	7	5	6	5	5	1	1	1	1	1	0.02	0.02	0.01	0.02	0.01
10	6	15	19	15	19	5	5	5	5	5	12	8	9	8	9	1	1	1	1	1	0.02	0.01	0.01	0.01	0.01
11	5	22	26	30	34	5	4	3	3	3	18	12	10	7	6	1	1	1	1	1	0.02	0.01	0.02	0.02	0.01
12	7	26	29	57	66	7	7	7	3	2	26	25	25	4	1	1	1	1	1	1	0.02	0.02	0.02	0.05	0.02
13	7	28	28	28	28	5	3	3	3	3	16	6	6	6	6	1	1	1	1	1	0.01	0.04	0.03	0.04	0.01
14	3	6	6	6	6	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0.01	0.01	0.01	0.01	0.01
15	7	12	13	12	13	3	2	3	2	3	4	1	6	1	6	1	1	1	1	1	0.02	0.01	0.01	0.01	0.01
16	6	22	26	22	26	4	6	8	6	8	8	19	24	19	24	1	1	1	1	1	0.02	0.02	0.02	0.02	0.02
17	4	15	15	21	21	3	2	2	2	2	10	1	1	3	3	1	1	1	1	1	0.01	0.01	0.01	0.02	0.01
18	6	15	15	15	15	3	1	1	1	1	9	0	0	0	0	1	1	1	1	1	0.01	0.01	0.01	0.01	0.01
19	6	43	47	121	124	7	7	7	1	1	6	6	6	0	0	1	1	1	1	1	0.01	0.01	0.01	0.02	0.02
20	5	14	19	14	19	4	3	3	3	3	4	4	3	4	3	1	1	1	1	1	0.01	0.01	0.02	0.01	0.01
22	11	62	78	62	78	9	6	5	6	5	34	16	13	16	13	1	1	1	1	1	0.04	0.04	0.04	0.05	0.05
23	13	18	18	21	21	2	3	3	3	3	5	9	9	10	10	1	1	1	1	1	0.01	0.01	0.02	0.02	0.01
24	10	33	35	33	35	9	4	4	4	4	40	9	11	9	11	1	1	1	1	1	0.04	0.02	0.02	0.01	0.02
25	9	41	51	46	57	7	6	6	7	5	29	20	21	23	20	1	1	1	1	1	0.03	0.03	0.02	0.02	0.02
27	9	55	62	55	62	12	9	9	9	9	16	17	17	17	17	1	1	1	1	1	0.02	0.03	0.04	0.03	0.02
28	11	43	48	58	72	7	6	5	5	5	27	22	17	13	10	1	1	1	1	1	0.03	0.03	0.02	0.03	0.04
29	13	33	37	43	47	6	5	5	5	5	19	13	14	13	13	1	1	1	1	1	0.02	0.02	0.02	0.02	0.02
30	13	50	54	50	54	7	4	4	4	4	26	11	11	11	11	1	1	1	1	1	0.02	0.02	0.02	0.02	0.02

Appendix C. Detailed results for the second experiment

In Tables 9 and 10, we report the number of pricing iterations, the number of columns generated, the number of nodes in the search tree, and the solution time (in seconds) for the different initialization schemes. In both tables, the time of the itinerary change is given in the second column (with heading t_u). Moreover, in the last two rows, we report averages. First, by taking the average over all rows, and second by taking the average over all rows except rows associated with *outliers*, namely, the instances 29-13, 29-19, 29-89, and 29-95. These are instances for which either *AF* or *AR* has a solution time of more than 100 seconds when the itinerary change occurs in the first quarter and instances for which either *AF* or *AR* has a solution time of more than 25 seconds when the itinerary change occurs in the second quarter. As the values for these outliers would impact and skew the averages significantly, we provide both averages.

Table 9
Detailed results for the second experiment for update realization interval (0, T/4).

Instance		Pricing iterations					Columns generated					Nodes evaluated					Time (s)				
29		2937					8241					605					313.18				
name	t_u	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*
29-1	90	171	87	81	31	33	756	302	287	70	64	9	9	9	9	9	5.24	3.46	3.02	2.80	2.83
29-2	99	148	87	68	33	21	653	314	231	49	26	9	9	9	9	9	5.03	3.35	2.94	2.59	2.46
29-3	75	167	76	67	22	25	716	268	215	28	36	9	9	9	9	9	6.78	3.19	3.70	2.85	2.64
29-4	71	125	32	30	11	8	604	144	130	36	34	1	1	1	1	1	3.81	1.08	0.87	1.22	0.87
29-5	93	185	75	72	34	30	771	237	226	61	44	11	13	13	13	13	6.34	3.65	3.68	3.36	3.41
29-6	101	285	196	186	125	105	1071	617	557	210	169	31	33	31	37	31	11.50	9.46	8.97	10.68	8.16
29-7	73	159	86	78	23	25	697	298	291	54	57	9	9	9	7	9	5.76	3.67	3.67	2.05	2.30
29-8	112	155	33	60	28	19	663	139	196	52	22	11	3	9	9	9	4.04	1.12	2.29	2.21	1.64
29-9	86	157	62	66	17	15	728	249	246	38	29	5	5	5	5	5	5.52	2.98	2.96	1.62	1.75
29-10	97	177	75	76	31	30	701	251	240	53	43	13	13	13	13	13	5.81	3.55	3.93	3.24	3.38
29-11	66	185	76	69	18	18	808	274	231	18	19	9	9	9	9	9	8.24	3.97	3.66	3.28	3.39
29-12	60	199	126	95	39	38	832	450	345	63	71	13	13	13	13	13	8.99	6.42	5.66	4.56	4.70
29-13	88	375	627	554	2269	1336	1342	1813	1533	4783	2703	37	105	91	679	413	11.45	21.69	20.63	109.22	70.87
29-14	99	80	33	18	10	8	390	160	74	39	33	1	1	1	1	1	2.39	1.17	0.51	1.01	0.42
29-15	77	96	19	24	8	5	439	79	112	26	14	1	1	1	1	1	3.27	0.58	0.69	0.46	0.45
29-16	53	203	44	79	40	33	898	191	289	108	82	9	3	9	9	9	9.78	2.01	5.13	4.50	4.07
29-17	62	88	31	18	8	11	430	135	78	32	43	1	1	1	1	1	3.25	1.27	0.54	0.56	0.64
29-18	84	143	67	56	18	20	608	242	185	21	25	7	9	9	9	9	5.18	2.67	2.70	1.98	2.10
29-19	79	881	523	486	1912	1892	2824	1509	1393	4312	3729	115	85	83	531	595	30.10	19.03	19.57	93.85	108.03
29-20	55	167	59	72	29	26	739	216	272	85	60	7	7	7	7	7	7.08	3.29	3.35	2.62	2.92
29-21	75	185	62	60	17	18	812	181	193	16	15	9	9	9	9	9	6.33	2.67	3.08	4.98	4.70
29-22	68	105	29	27	3	3	504	126	111	5	5	1	1	1	1	1	4.37	0.79	0.94	0.39	0.40
29-23	82	162	67	70	19	24	724	238	245	22	46	9	9	9	9	9	4.83	2.65	2.49	2.10	2.49
29-24	57	201	51	65	18	17	850	188	204	18	19	9	5	9	9	9	11.29	2.86	3.87	6.41	5.66
29-25	95	82	28	19	11	13	394	127	88	46	51	1	1	1	1	1	2.42	0.90	0.60	0.51	0.57
29-26	71	72	19	10	4	4	341	79	38	8	8	1	1	1	1	1	2.99	0.89	0.38	0.36	0.32
29-27	84	136	64	56	14	17	624	260	217	29	36	5	5	5	5	5	5.46	2.99	2.61	1.54	1.45
29-28	97	71	34	18	10	12	350	148	74	39	48	1	1	1	1	1	1.80	1.24	0.54	0.48	0.54
29-29	5	884	992	601	545	511	3391	3212	2049	1440	1260	71	125	71	107	101	90.36	110.47	68.62	80.58	76.99
29-30	14	346	235	751	223	195	1446	918	2623	599	538	23	17	89	33	31	31.91	25.51	79.14	44.35	34.62
29-31	36	186	70	69	37	33	849	280	302	112	102	5	5	5	5	5	13.49	5.17	4.48	3.36	3.22
29-32	32	231	120	122	59	69	1009	481	474	183	185	11	11	11	11	11	13.34	8.50	9.48	6.23	6.39
29-33	7	351	146	130	92	92	1488	555	482	248	253	19	13	13	13	15	29.98	18.56	14.84	14.72	15.69
29-34	16	279	116	170	83	97	1189	452	605	216	257	11	11	15	13	15	25.47	11.52	17.18	13.23	13.00
29-35	38	189	91	80	53	48	829	341	322	141	121	9	9	9	9	9	11.60	7.14	5.85	6.46	5.36
29-36	34	194	98	92	55	59	881	378	353	166	171	9	9	9	9	11	11.74	6.74	7.21	6.20	7.25
29-37	10	287	151	185	96	84	1224	543	670	260	217	15	13	13	13	13	25.80	18.10	17.76	13.27	12.73
29-38	29	116	40	45	16	13	557	178	210	53	59	1	1	1	1	1	7.79	1.96	2.16	1.43	1.18
29-39	36	232	146	121	60	59	994	570	452	160	189	11	13	11	11	11	15.55	10.32	8.71	6.14	5.66
29-40	12	304	163	160	103	79	1339	570	573	273	223	13	17	15	19	15	26.67	17.96	16.20	15.49	12.80
29-41	49	223	118	103	53	53	990	464	394	151	158	11	11	9	11	11	10.82	6.13	5.28	4.34	4.57
29-42	25	385	347	368	214	251	1497	1034	1114	549	559	31	45	47	43	47	26.85	31.81	32.03	25.47	27.30
29-43	51	182	91	97	43	33	804	347	330	112	85	9	9	9	11	9	9.73	5.04	5.29	4.37	4.46
29-44	90	247	173	191	61	72	972	576	578	132	117	21	27	37	19	29	7.96	8.64	9.54	6.14	7.61
29-45	101	150	95	82	38	27	642	370	300	87	62	7	9	9	9	7	5.63	3.51	3.25	2.74	2.12
29-46	5	328	202	238	105	140	1392	734	820	295	386	15	21	21	15	25	27.41	22.75	27.68	16.79	21.24
29-47	16	217	93	97	47	38	1042	427	437	175	142	3	3	3	3	3	19.40	7.85	7.52	5.48	5.06
29-48	45	214	82	71	15	62	941	320	291	52	206	9	7	7	3	9	11.35	5.11	4.70	1.96	5.11
29-49	56	88	27	26	10	12	416	124	113	37	48	1	1	1	1	1	3.47	1.12	0.76	0.58	0.60
29-50	18	165	63	50	21	23	804	294	228	85	92	1	1	1	1	1	12.42	3.94	4.58	2.18	2.25
Average 1		219.16	128.54	128.58	136.62	117.12	919.30	448.06	440.42	316.94	259.22	13.00	14.96	15.48	35.36	31.64	12.55	9.01	9.30	11.06	10.37
Average 2		202.13	109.94	112.27	55.21	54.75	870.81	397.52	397.81	140.67	136.02	10.38	11.63	12.50	11.63	11.96	12.21	8.54	8.85	7.29	7.07

Table 10
Detailed results for the second experiment for update realization interval [$T/4$, $T/2$).

Instance		Pricing iterations					Columns generated					Nodes evaluated					Time (s)				
29		2937					8241					605					313.18				
name	t_u	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*
29-51	270	77	37	45	16	19	295	87	118	29	30	7	7	7	7	7	0.45	0.34	0.36	0.46	0.32
29-52	279	28	5	9	5	5	125	12	36	11	10	1	1	1	1	1	0.13	0.05	0.10	0.07	0.05
29-53	255	27	4	5	5	5	120	10	10	10	10	1	1	1	1	1	0.13	0.06	0.06	0.10	0.10
29-54	251	48	13	11	6	10	223	52	48	18	34	1	1	1	1	1	0.21	0.10	0.08	0.14	0.14
29-55	273	29	5	6	4	6	119	10	25	8	7	1	1	1	1	1	0.11	0.06	0.05	0.07	0.07
29-56	281	48	14	17	11	12	199	32	57	23	21	3	3	3	3	3	0.20	0.15	0.11	0.19	0.16
29-57	253	81	36	31	22	21	320	81	70	32	29	7	9	9	9	9	0.47	0.28	0.30	0.38	0.36
29-58	292	22	7	10	5	6	101	16	31	17	18	1	1	1	1	1	0.08	0.05	0.05	0.04	0.05
29-59	266	35	10	16	7	7	166	39	56	22	25	1	1	1	1	1	0.16	0.09	0.11	0.12	0.11
29-60	277	28	5	8	4	6	133	10	31	8	7	1	1	1	1	1	0.13	0.05	0.06	0.06	0.07
29-61	246	100	39	30	20	19	384	93	71	26	24	7	9	9	9	9	0.57	0.35	0.30	0.33	0.30
29-62	240	36	4	7	5	6	146	11	22	12	15	1	1	1	1	1	0.21	0.06	0.08	0.09	0.09
29-63	268	22	5	8	5	5	100	16	30	15	8	1	1	1	1	1	0.09	0.09	0.11	0.15	0.15
29-64	279	21	9	8	8	7	100	37	31	25	17	1	1	1	1	1	0.09	0.05	0.05	0.07	0.08
29-65	257	27	5	8	4	5	122	10	31	10	9	1	1	1	1	1	0.14	0.07	0.09	0.09	0.09
29-66	233	107	49	47	24	22	429	135	122	36	37	9	9	9	9	9	0.77	0.52	0.50	0.53	0.50
29-67	242	32	7	8	6	8	150	30	29	21	28	1	1	1	1	1	0.16	0.09	0.07	0.09	0.11
29-68	264	81	39	31	17	19	307	97	63	32	29	7	7	7	7	7	0.37	0.27	0.22	0.32	0.22
29-69	259	22	6	7	5	5	97	19	29	12	10	1	1	1	1	1	0.13	0.08	0.11	0.17	0.16
29-70	235	90	60	66	46	41	374	193	153	115	107	7	7	7	7	7	0.61	0.54	0.50	0.51	0.50
29-71	255	82	32	29	22	21	310	78	71	31	31	9	9	9	9	9	0.48	0.31	0.34	0.43	0.61
29-72	248	31	9	9	7	10	138	23	33	19	21	1	1	1	1	1	0.19	0.09	0.08	0.11	0.13
29-73	262	37	6	9	5	5	172	24	28	10	10	1	1	1	1	1	0.15	0.07	0.08	0.06	0.06
29-74	237	133	70	58	46	43	461	151	120	57	55	19	19	19	19	19	1.01	0.76	0.64	0.78	0.91
29-75	275	26	9	8	6	8	122	29	32	18	19	1	1	1	1	1	0.09	0.06	0.05	0.07	0.09
29-76	251	28	4	8	5	5	130	11	29	10	14	1	1	1	1	1	0.15	0.05	0.07	0.08	0.07
29-77	264	32	10	16	7	7	144	39	56	22	25	1	1	1	1	1	0.13	0.10	0.10	0.10	0.11
29-78	277	25	9	8	6	6	116	29	32	18	17	1	1	1	1	1	0.10	0.06	0.05	0.07	0.08
29-79	185	250	134	118	59	62	985	426	382	111	111	23	21	19	17	25	3.16	2.25	2.05	2.57	2.15
29-80	194	135	111	56	30	29	552	349	189	80	75	7	15	7	7	7	2.09	1.89	0.89	3.41	3.13
29-81	216	45	13	17	7	8	203	57	70	15	25	1	1	1	1	1	0.41	0.14	0.17	0.23	0.24
29-82	212	35	12	9	7	7	161	50	38	22	25	1	1	1	1	1	0.35	0.16	0.13	0.16	0.13
29-83	187	158	63	59	25	27	665	214	195	48	57	9	9	9	9	9	2.38	0.98	0.80	0.94	1.03
29-84	196	133	52	49	24	22	531	152	123	52	50	9	9	9	9	9	1.75	0.76	0.99	1.00	0.88
29-85	218	89	63	57	18	18	338	195	183	26	26	7	9	9	9	9	0.74	0.59	0.57	0.45	0.42
29-86	214	35	9	9	6	6	160	26	26	12	12	1	1	1	1	1	0.43	0.11	0.11	0.12	0.11
29-87	190	130	55	61	22	18	552	181	192	37	29	9	9	9	9	7	1.62	0.80	0.84	0.82	0.66
29-88	209	43	13	11	9	8	176	55	50	20	18	1	1	1	1	1	0.44	0.18	0.15	0.18	0.18
29-89	203	410	515	62	1334	1322	1303	1485	188	2771	2720	57	91	9	365	385	5.38	7.59	1.29	27.54	25.96
29-90	216	153	81	83	52	46	537	200	206	79	52	19	19	19	19	19	1.53	0.99	1.04	0.93	0.93
29-91	192	134	56	57	20	18	568	182	176	30	24	9	9	9	9	7	1.71	0.96	0.82	0.76	0.69
29-92	229	32	8	7	5	7	146	29	25	13	16	1	1	1	1	1	0.22	0.09	0.09	0.09	0.10
29-93	205	255	112	104	61	49	964	312	292	117	64	25	21	21	23	23	2.83	1.51	1.48	1.53	1.51
29-94	231	34	7	6	5	5	138	21	22	11	15	1	1	1	1	1	0.19	0.08	0.08	0.09	0.09
29-95	194	521	325	83	1349	1249	1713	926	239	2857	2601	67	51	13	369	357	7.24	4.86	1.68	25.62	25.48
29-96	270	80	40	38	22	17	323	112	89	52	37	7	7	7	7	7	0.38	0.24	0.24	0.30	0.25
29-97	281	73	31	29	22	21	289	78	79	39	36	7	7	7	7	7	0.36	0.19	0.18	0.23	0.25
29-98	185	136	48	45	18	23	563	139	147	23	25	9	9	9	9	9	1.78	0.70	0.95	0.76	0.76
29-99	196	39	11	12	6	6	172	48	44	17	16	1	1	1	1	1	0.46	0.18	0.19	0.18	0.16
29-100	225	38	7	9	5	6	173	23	33	12	16	1	1	1	1	1	0.26	0.09	0.08	0.08	0.10
Average 1		86.26	45.48	30.08	68.7	66.26	336.3	132.68	89.04	142.22	134.34	7.32	7.84	5.24	19.48	19.72	0.86	0.60	0.39	1.47	1.42
Average 2		70.46	29.88	28.31	15.67	15.46	287.48	87.98	83.85	30.90	29.08	5.04	5.21	5.00	5.00	5.08	0.64	0.37	0.34	0.43	0.40

Table 11
Comparison of different initialization strategies w.r.t. the first feasible solution found.

Instance	Initial columns					Time to find first feasible solution(s)					Optimality gap of first feasible solution(%)				
	FS	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*	*FS*	*RF*	*RR*	*AF*	*AR*
29-1	15	254	303	1238	1369	0.14	0.12	0.08	0.28	0.17	4.20	2.60	2.60	2.60	2.60
29-2	14	256	297	1398	1476	0.06	0.07	0.06	0.19	0.14	0.00	0.00	0.00	0.00	0.00
29-3	14	267	308	1434	1509	0.06	0.12	0.12	0.37	0.27	0.49	0.00	0.00	0.00	0.00
29-4	15	271	319	1337	1441	0.06	0.10	0.09	0.53	0.47	8.31	7.01	7.01	2.23	2.23
29-5	15	266	300	1400	1473	0.04	0.08	0.06	0.32	0.21	1.11	1.11	1.11	0.00	0.00
29-6	14	258	301	1388	1475	0.06	0.09	0.10	0.20	0.16	0.00	0.00	0.00	0.00	0.00
29-7	15	275	313	1442	1514	0.06	0.07	0.07	0.21	0.15	3.18	3.18	3.18	3.18	3.18
29-8	15	246	284	1361	1426	0.04	0.06	0.06	0.28	0.18	2.29	0.73	0.73	0.00	0.00
29-9	15	245	296	1311	1392	0.07	0.08	0.07	0.21	0.21	3.89	3.89	3.89	1.68	1.46
29-10	15	266	300	1399	1473	0.05	0.06	0.07	0.21	0.22	1.11	1.11	1.11	0.00	0.00
29-11	14	300	337	1541	1616	0.05	0.07	0.08	0.15	0.14	0.00	0.00	0.00	0.00	0.00
29-12	16	296	340	1531	1613	0.06	0.06	0.07	0.18	0.20	6.84	5.06	5.06	0.00	0.00
29-13	15	245	296	1272	1389	0.05	0.32	0.39	3.90	1.58	3.41	3.41	3.41	3.41	0.37
29-14	15	264	299	1404	1469	0.04	0.05	0.05	0.14	0.16	8.21	8.21	8.21	8.21	5.62
29-15	16	274	313	1442	1516	0.04	0.09	0.07	0.16	0.16	5.48	5.48	5.48	4.39	4.39
29-16	14	317	362	1596	1684	0.07	0.09	0.09	0.28	0.21	0.00	0.00	0.00	0.00	0.00
29-17	15	301	338	1533	1609	0.04	0.11	0.06	0.17	0.17	7.71	7.71	7.71	5.38	5.28
29-18	14	274	310	1439	1508	0.03	0.06	0.06	0.12	0.13	0.00	0.00	0.00	0.00	0.00
29-19	15	257	305	1302	1421	0.04	0.27	0.87	5.30	1.70	3.41	3.41	3.41	3.41	0.37
29-20	14	278	332	1404	1546	0.05	0.09	0.07	0.22	0.23	0.85	0.85	0.85	0.85	0.85
29-21	15	277	314	1381	1510	0.07	0.10	0.10	2.92	2.66	6.09	0.11	0.11	0.00	0.00
29-23	15	270	306	1430	1497	0.04	0.06	0.08	0.15	0.14	0.00	0.00	0.00	0.00	0.00
29-24	15	302	339	1483	1612	0.08	0.11	0.10	3.49	3.02	5.72	0.11	0.11	0.00	0.00
29-25	15	264	299	1405	1470	0.03	0.05	0.07	0.14	0.16	8.21	8.21	8.21	8.21	5.62
29-27	15	248	299	1319	1401	0.05	0.07	0.08	0.21	0.21	3.89	3.89	3.89	1.68	1.46
29-28	15	264	299	1404	1470	0.04	0.06	0.06	0.13	0.14	8.21	8.21	8.21	8.21	5.62
29-29	14	452	484	2058	2187	0.20	0.42	0.91	5.09	2.98	0.15	0.15	0.15	0.15	0.15
29-30	15	445	477	1957	2128	0.16	0.33	0.36	10.10	7.48	3.03	3.03	3.03	0.00	0.00
29-31	15	333	391	1648	1759	0.07	0.13	0.17	0.37	0.23	3.34	3.34	3.34	1.25	1.25
29-32	14	347	398	1769	1875	0.06	0.12	0.09	0.18	0.17	0.00	0.00	0.00	0.00	0.00
29-33	14	471	487	2255	2286	0.07	0.12	0.13	0.30	0.26	0.00	0.00	0.00	0.00	0.00
29-34	14	452	488	2206	2275	0.08	0.13	0.09	0.28	0.25	0.00	0.00	0.00	0.00	0.00
29-35	14	362	402	1769	1858	0.06	0.11	0.09	0.30	0.21	0.00	0.00	0.00	0.00	0.00
29-36	14	352	400	1792	1876	0.07	0.16	0.30	0.38	0.46	0.42	0.00	0.00	0.00	0.00
29-37	14	463	484	2229	2270	0.06	0.10	0.09	0.23	0.19	0.00	0.00	0.00	0.00	0.00
29-39	15	355	402	1499	1667	0.17	0.21	0.20	0.52	0.46	5.48	4.32	4.32	1.00	1.00
29-40	14	466	488	2236	2284	0.06	0.10	0.09	0.19	0.22	0.00	0.00	0.00	0.00	0.00
29-41	14	304	359	1600	1705	0.06	0.08	0.06	0.19	0.16	0.00	0.00	0.00	0.00	0.00
29-42	14	359	404	1773	1872	0.06	0.12	0.10	0.21	0.20	0.00	0.00	0.00	0.00	0.00
29-43	14	322	362	1642	1711	0.06	0.07	0.06	0.15	0.15	0.00	0.00	0.00	0.00	0.00
29-44	14	248	300	1287	1415	0.07	0.37	0.27	1.20	1.25	0.19	0.19	0.19	0.19	0.19
29-45	15	254	298	1236	1363	0.06	0.09	0.08	0.13	0.14	4.20	2.60	2.60	2.60	2.60
29-46	14	472	488	2257	2288	0.06	0.13	0.10	0.28	0.22	0.00	0.00	0.00	0.00	0.00
29-47	14	458	488	2207	2287	0.07	0.09	0.09	0.19	0.20	5.08	5.08	5.08	5.08	5.08
29-48	14	340	396	1638	1802	0.05	0.08	0.09	0.50	0.50	0.00	0.00	0.00	0.00	0.00
29-49	15	301	339	1534	1611	0.05	0.07	0.07	0.15	0.15	7.71	7.71	4.65	4.65	4.65
29-50	15	453	489	2042	2165	0.07	0.18	0.17	0.97	0.95	7.01	5.91	5.91	2.00	2.00
29-51	15	113	135	334	392	0.04	0.09	0.08	0.23	0.13	7.29	0.13	0.13	0.00	0.00
29-54	15	119	150	351	439	0.04	0.03	0.04	0.09	0.08	7.62	2.64	2.64	2.64	2.64
29-56	14	101	124	284	333	0.03	0.04	0.03	0.06	0.05	6.37	0.00	0.00	0.00	0.00
29-57	15	120	149	381	470	0.02	0.03	0.04	0.07	0.05	3.75	3.75	3.75	3.75	3.75
29-59	15	103	128	303	358	0.03	0.04	0.03	0.07	0.05	4.65	4.65	4.65	4.65	4.65
29-61	14	124	154	446	560	0.02	0.04	0.03	0.09	0.05	0.00	0.00	0.00	0.00	0.00
29-63	15	105	130	314	375	0.02	0.06	0.07	0.11	0.10	0.00	0.00	0.00	0.00	0.00
29-64	14	104	125	314	361	0.09	0.05	0.05	0.07	0.08	0.00	0.00	0.00	0.00	0.00
29-66	14	135	166	486	651	0.03	0.04	0.07	0.16	0.14	0.00	0.00	0.00	0.00	0.00
29-67	15	127	157	454	567	0.16	0.08	0.07	0.09	0.11	0.00	0.00	0.00	0.00	0.00
29-68	14	110	131	333	387	0.02	0.03	0.03	0.15	0.04	0.00	0.00	0.00	0.00	0.00
29-69	15	112	142	350	441	0.03	0.05	0.07	0.13	0.11	0.00	0.00	0.00	0.00	0.00
29-70	15	128	156	450	535	0.33	0.19	0.20	0.21	0.19	0.00	0.00	0.00	0.00	0.00
29-71	15	121	150	376	469	0.03	0.04	0.06	0.16	0.15	7.19	0.13	0.13	0.00	0.00
29-74	15	124	158	379	512	0.03	0.04	0.04	0.08	0.09	6.22	1.73	0.04	0.00	0.00
29-75	14	107	128	326	374	0.09	0.06	0.05	0.07	0.09	0.00	0.00	0.00	0.00	0.00
29-77	15	103	128	303	358	0.02	0.04	0.03	0.06	0.06	4.65	4.65	4.65	4.65	4.65
29-78	14	107	128	325	373	0.10	0.06	0.05	0.07	0.08	0.00	0.00	0.00	0.00	0.00
29-79	14	191	228	816	935	0.10	0.23	0.29	0.97	0.39	0.20	0.20	0.20	0.20	0.20
29-80	15	180	216	692	839	0.10	0.16	0.11	2.67	2.41	3.95	0.00	0.00	0.00	0.00
29-81	15	143	178	524	686	0.05	0.03	0.04	0.13	0.14	4.44	4.44	4.44	1.92	1.92
29-83	15	200	234	867	970	0.03	0.05	0.04	0.09	0.15	3.39	3.39	3.39	3.39	3.39
29-84	15	191	226	830	936	0.03	0.04	0.05	0.16	0.08	0.87	0.87	0.87	0.87	0.87
29-85	14	148	182	530	698	0.04	0.07	0.05	0.14	0.12	0.00	0.00	0.00	0.00	0.00
29-87	14	192	224	831	934	0.03	0.05	0.04	0.12	0.07	0.00	0.00	0.00	0.00	0.00
29-89	15	176	217	772	897	0.03	0.12	0.17	0.79	0.41	3.60	3.60	3.60	3.60	1.31
29-90	15	146	181	477	613	0.03	0.04	0.09	0.11	0.11	6.07	6.07	0.04	0.00	0.00
29-91	14	193	226	832	936	0.03	0.05	0.04	0.08	0.08	0.00	0.00	0.00	0.00	0.00
29-93	14	163	203	614	794	0.03	0.06	0.07	0.09	0.10	0.00	0.00	0.00	0.00	0.00
29-95	15	179	219	781	902	0.04	0.13	0.25	0.70	0.40	3.60	3.60	3.60	3.60	1.31
29-96	14	108	132	316	378	0.03	0.05	0.05	0.08	0.08	0.22	0.22	0.22	0.22	0.22
29-97	14	99	122	248	307	0.17	0.08	0.08	0.10	0.11	0.00	0.00	0.00	0.00	0.00
29-98	14	203	234	873	972	0.02	0.04	0.04	0.09	0.08	0.00	0.00	0.00	0.00	0.00
Average	14.56	245.24	281.18	1146.75	1241.44	0.06	0.10	0.11	0.63	0.46	2.54	1.83	1.70	1.25	1.01

Table 12
Number of route changes with each initialization strategy and use of recovery procedures.

Update realization interval (0, *T*/4)							Update realization interval [*T*/4, *T*/2)						
Instance	Recovery	# of route changes					Instance	Recovery	# of route changes				
		FS	*RF*	*RR*	*AF*	*AR*			*FS*	*RF*	*AR*	*AF*	*AR*
29-1	Out-and-back route	2	2	2	1	1	29-51	Split route	2	2	2	0	0
29-2	GTSPTW	0	0	0	0	0	29-52	Artificial route	–	–	–	–	–
29-3	GTSPTW	2	0	0	0	0	29-53	Artificial route	–	–	–	–	–
29-4	Split route	9	3	4	1	1	29-54	Out-and-back route	6	1	1	1	1
29-5	Out-and-back route	1	1	1	0	0	29-55	Artificial route	–	–	–	–	–
29-6	GTSPTW	0	0	0	0	0	29-56	Node elimination	4	0	0	0	0
29-7	Out-and-back route	1	1	1	1	1	29-57	Out-and-back route	1	1	1	1	1
29-8	Out-and-back route	2	1	1	0	0	29-58	Artificial route	–	–	–	–	–
29-9	Out-and-back route	2	1	1	1	1	29-59	Out-and-back route	1	2	1	1	1
29-10	Out-and-back route	1	1	1	0	0	29-60	Artificial route	–	–	–	–	–
29-11	N/A	0	0	0	0	0	29-61	N/A	0	0	0	0	0
29-12	Node elimination	2	2	2	0	0	29-62	Artificial route	–	–	–	–	–
29-13	Split route	5	3	2	2	2	29-63	Split route	0	0	0	0	0
29-14	Out-and-back route	2	2	1	1	1	29-64	Artificial route	2	2	1	1	1
29-15	Node elimination	6	3	3	1	1	29-65	Artificial route	–	–	–	–	–
29-16	GTSPTW	0	0	0	0	0	29-66	GTSPTW	0	0	0	0	0
29-17	Out-and-back route	2	2	1	1	1	29-67	Artificial route	2	1	1	1	1
29-18	N/A	0	0	0	0	0	29-68	N/A	0	0	0	0	0
29-19	Split route	5	3	3	2	2	29-69	Split route	0	0	0	0	0
29-20	GTSPTW	2	1	1	1	1	29-70	Artificial route	1	1	1	1	1
29-21	Split route	2	2	2	0	0	29-71	Split route	2	2	2	0	0
29-22	Artificial route	–	–	–	–	–	29-72	Artificial route	–	–	–	–	–
29-23	Split route	0	0	0	0	0	29-73	Artificial route	–	–	–	–	–
29-24	Split route	2	2	2	0	0	29-74	Out-and-back route	2	1	1	0	0
29-25	Out-and-back route	2	2	1	1	1	29-75	Artificial route	2	2	1	1	1
29-26	Artificial route	–	–	–	–	–	29-76	Artificial route	–	–	–	–	–
29-27	Out-and-back route	1	1	1	1	1	29-77	Out-and-back route	1	2	1	1	1
29-28	Out-and-back route	2	2	1	1	1	29-78	Artificial route	2	2	1	1	1
29-29	GTSPTW	3	2	2	1	1	29-79	GTSPTW	2	2	2	1	1
29-30	Split route	2	1	1	0	0	29-80	Out-and-back route	1	0	0	0	0
29-31	Out-and-back route	1	1	1	1	1	29-81	Out-and-back route	1	1	1	1	1
29-32	GTSPTW	0	0	0	0	0	29-82	Artificial route	–	–	–	–	–
29-33	N/A	0	0	0	0	0	29-83	Out-and-back route	1	1	1	1	1
29-34	GTSPTW	0	0	0	0	0	29-84	Node elimination	1	1	1	1	1
29-35	GTSPTW	0	0	0	0	0	29-85	GTSPTW	0	0	0	0	0
29-36	GTSPTW	2	0	0	0	0	29-86	Artificial route	–	–	–	–	–
29-37	N/A	0	0	0	0	0	29-87	N/A	0	0	0	0	0
29-38	Artificial route	–	–	–	–	–	29-88	Artificial route	–	–	–	–	–
29-39	Split route	2	2	2	1	1	29-89	Split route	4	3	2	2	1
29-40	N/A	0	0	0	0	0	29-90	Out-and-back route	2	1	1	0	0
29-41	GTSPTW	0	0	0	0	0	29-91	N/A	0	0	0	0	0
29-42	GTSPTW	0	0	0	0	0	29-92	Artificial route	–	–	–	–	–
29-43	N/A	0	0	0	0	0	29-93	GTSPTW	0	0	0	0	0
29-44	GTSPTW	2	2	2	1	1	29-94	Artificial route	–	–	–	–	–
29-45	Out-and-back route	2	2	2	1	1	29-95	Split route	4	3	2	2	1
29-46	N/A	0	0	0	0	0	29-96	GTSPTW	2	2	2	1	1
29-47	GTSPTW	4	4	4	3	2	29-97	Artificial route	4	2	2	0	0
29-48	GTSPTW	0	0	0	0	0	29-98	N/A	0	0	0	0	0
29-49	Out-and-back route	2	2	1	1	1	29-99	Artificial route	–	–	–	–	–
29-50	Split route	8	4	5	1	1	29-100	Artificial route	–	–	–	–	–

References

Attanasio, A., Cordeau, J.-F., Ghiani, G., Laporte, G., 2004. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. Parallel Comput. 30 (3), 377–387.

Attanasio, A., Bregman, J., Ghiani, G., Manni, E., 2007. Real-time fleet management at ecourier ltd. In: Dynamic Fleet Management. Springer, pp. 219–238.

Azi, N., Gendreau, M., Potvin, J.-Y., 2012. A dynamic vehicle routing problem with multiple delivery routes. Ann. Oper. Res. 199 (1), 103–112.

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

Bektas, T., Repoussis, P.P., Tarantilis, C.D., 2014. Dynamic vehicle routing problems. Veh. Rout. 18, 299.

Barceló, J., Grzybowska, H., Pardo, S., 2007. Vehicle routing and scheduling models, simulation and city logistics. In: Dynamic Fleet Management. Springer, pp. 163–195.

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

Campbell, A.M., Savelsbergh, M.W., 2005. Decision support for consumer direct grocery initiatives. Transport. Sci. 39 (3), 313–327.

Chen, Z.-L., Xu, H., 2006. Dynamic column generation for dynamic vehicle routing with time windows. Transport. Sci. 40 (1), 74–88.

Chen, H.-K., Hsueh, C.-F., Chang, M.-S., 2006. The real-time-dependent vehicle routing problem. Transport. Res. Part E 42 (5), 383–408.
Dolan, E.D., More, J.J., 2002. Benchmarking optimization software with performance profiles. Math. Program. 91, 201–213.
Ferrucci, F., Bock, S., 2015. A general approach for controlling vehicle en-route diversions in dynamic vehicle routing problems. Transport. Res. Part B 77, 76–87.
Fleischmann, B., Gnutzmann, S., Sandvoß, E., 2004. Dynamic vehicle routing based on online traffic information. Transport. Sci. 38 (4), 420–433.
Gendreau, M., Guertin, F., Potvin, J.-Y., Taillard, E., 1999. Parallel tabu search for real-time vehicle routing and dispatching. Transport. Sci. 33 (4), 381–390.
Gendreau, M., Guertin, F., Potvin, J.-Y., Séguin, R., 2006. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. Transport. Res. Part C 14 (3), 157–174.
Geuss, M., 2015. Amazon, Audi and DHL Want to Turn a Car Trunk into a Delivery Locker. http://arstechnica.com/business/2015/04/amazon-audi-and-dhl-want-to-turn-a-car-trunk-into-a-delivery-locker, Accessed: 2016-08-07
Goel, A., Gruhn, V., 2008. A general vehicle routing problem. Eur. J. Oper. Res. 191 (3), 650–660.
Ichoua, S., Gendreau, M., Potvin, J.-Y., 2000. Diversion issues in real-time vehicle dispatching. Transport. Sci. 34 (4), 426–438.
Ichoua, S., Gendreau, M., Potvin, J.-Y., 2006. Exploiting knowledge about future demands for real-time vehicle dispatching. Transport. Sci. 40 (2), 211–225.
Kinable, J., Hadka, D., Lelieveld, F., 2016. Jorlib–Java Operations Research Library.
Klundert, J.v.d., Wormer, L., 2010. Asap: the after-salesman problem. Manuf. Serv. Oper. Manag. 12 (4), 627–641.
Li, J.-Q., Mirchandani, P.B., Borenstein, D., 2009. A Lagrangian heuristic for the real-time vehicle rescheduling problem. Transport. Res. Part E 45 (3), 419–433.
Li, J.-Q., Mirchandani, P.B., Borenstein, D., 2009. Real-time vehicle rerouting problems with time windows. Eur. J. Oper. Res. 194 (3), 711–727.
Mes, M., Van Der Heijden, M., Van Harten, A., 2007. Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. Eur. J. Oper. Res. 181 (1), 59–75.
Mitrović-Minić, S., Laporte, G., 2004. Waiting strategies for the dynamic pickup and delivery problem with time windows. Transport. Res. Part B 38 (7), 635–655.
Michail, D., Kinable, J., Naveh, B., Sichi, J.V., 2019. JGraphT–A Java library for graph data structures and algorithms. arXiv preprint arXiv:1904.08355.
Montemanni, R., Gambardella, L.M., Rizzoli, A.E., Donati, A.V., 2005. Ant colony system for a dynamic vehicle routing problem. J. Comb. Optim. 10 (4), 327–343.
Mu, Q., Fu, Z., Lysgaard, J., Eglese, R., 2011. Disruption management of the vehicle routing problem with vehicle breakdown. J. Oper. Res. Soc. 62 (4), 742–749.
Ozbaygin, G., Karasan, O.E., Savelsbergh, M., Yaman, H., 2017. A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations. Transport. Res. Part B 100, 115–137.
Pillac, V., Gendreau, M., Guéret, C., Medaglia, A.L., 2013. A review of dynamic vehicle routing problems. Eur. J. Oper. Res. 225 (1), 1–11.
Popken, B., 2015. Amazon Tests Delivery to Your Car Trunk. http://www.nbcnews.com/business/autos/amazon-testing-delivery-your-car-trunk-n346886, Accessed: 2016-08-07
Psaraftis, H.N., 1980. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. Transport. Sci. 14 (2), 130–154.
Psaraftis, H.N., Wen, M., Kontovas, C.A., 2016. Dynamic vehicle routing problems: three decades and counting. Networks 67 (1), 3–31.
Regan, A., Mahmassani, H., Jaillet, P., 1998. Evaluation of dynamic fleet management systems: simulation framework. Transport. Res. Rec. (1645) 176–184.
Respen, J., Zufferey, N., Potvin, J.-Y., 2014. Impact of online tracking on a vehicle routing problem with dynamic travel times. CIRRELT.
Reyes, D., Savelsbergh, M., Toriello, A., 2017. Vehicle routing with roaming delivery locations. Transport. Res. Part C 80, 71–91.
Tagmouti, M., Gendreau, M., Potvin, J.-Y., 2011. A dynamic capacitated arc routing problem with time-dependent service costs. Transport. Res. Part C 19 (1), 20–28.
Taniguchi, E., Shimamoto, H., 2004. Intelligent transportation system based dynamic vehicle routing and scheduling with variable travel times. Transport. Res. Part C 12 (3), 235–250.
Wilson, N.H., Colvin, N.J., 1977. Computer Control of the Rochester Dial-A-Ride System. Massachusetts Institute of Technology, Center for Transportation Studies.