Alarcon Ortega, Emilio J.; Schilde, Michael; Doerner, Karl Franz

Article

Matheuristic search techniques for the consistent inventory routing problem with time windows and split deliveries

Operations Research Perspectives

Provided in Cooperation with:
Elsevier

Suggested Citation: Alarcon Ortega, Emilio J.; Schilde, Michael; Doerner, Karl Franz (2020) : Matheuristic search techniques for the consistent inventory routing problem with time windows and split deliveries, Operations Research Perspectives, ISSN 2214-7160, Elsevier, Amsterdam, Vol. 7, pp. 1-15,
https://doi.org/10.1016/j.orp.2020.100152

Operations Research Perspectives 7 (2020) 100152

Contents lists available at ScienceDirect

Operations Research Perspectives

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

Matheuristic search techniques for the consistent inventory routing problem with time windows and split deliveries

Emilio J. Alarcon Ortega*, Michael Schilde, Karl F. Doerner

Department of Business Decisions & Analytics, University of Vienna, Oskar-Morgenstern Platz 1, Vienna 1090, Austria

ARTICLE INFO

Keywords:
Inventory routing problem
Adaptive large neighborhood search
Consistency
Matheuristics,

ABSTRACT

This article introduces a new variant of the inventory routing problem related to real-world businesses. Specifically, in the beverage industry, business customers such as restaurants and bars, demand consistent delivery times, have different opening times and delivery time windows, and occasionally, due to special events, exhibit demands that exceed single-vehicle capacity leading to the need of splitting demands between several vehicles. We present two variants of a mathematical formulation that include all the characteristics of this inventory routing problem. In the first, we apply the maximum level policy, whereas in the second variant, we apply an order-up-to-level policy. As a solution technique, we propose a matheuristic based on an adaptive large neighborhood search algorithm for which we developed several destroy and repair operators specifically designed to address the special problem features. Extensive computational tests based on artificial and real-world instances affirm the efficiency of the solution approach. Furthermore, we analyze the solution quality, the impact of the characteristics and policies applied, and the practicability for the real world.

1. Introduction and literature review

The consistent inventory routing problem with time windows and split deliveries (CIRPTWSD) arises in the beverage industry, where transportation plans must meet multiple requests from customers and various characteristics create the need to develop innovative, efficient solution approaches for distribution and stocking decisions. Many companies already have adopted vendor-managed inventory (VMI) systems, such that the supplier manages all the replenishment and distribution plans centrally, largely because the application of VMI can substantially reduce overall logistics costs [1]. Within the VMI framework, the inventory routing problem (IRP), first introduced in the seminal paper [2], aims to deal with this situation. Beverage industries, in particular beer industry, are especially remarkable in their central planning efforts to reduce overall logistics costs. Although beer consumption per capita and year is quite stable, consumption is very sensitive to external factors such as weekends, holidays, and special events (e.g., sports events, music festivals). Moreover, it encompasses different types of customers, such as bars, restaurants and retailers many of which have different opening hours and delivery time windows. Hence, to deal with these differences between opening hours, to set a maximum driving time and due to different working shifts, customers divide each period into different subperiods (e.g. morning, afternoon, evening). Another special characteristic that customers present is the demand of consistency in the delivery times, to enable themselves to prepare for a delivery. In addition, temporary high demands and the need to deliver to every customer creates a distinct possibility of splitting deliveries across multiple trucks.

One critical characteristic, in the real world context where we describe our problem, is that the product is consumed continuously within the time periods. This characteristic is often present in articles about inventory management, however, it is a relatively new characteristic in the VRP and IRP literature, see [3]. In this paper, the authors refer to this variant of the IRP as continuous-time IRP. In this work, authors study the critical components of a dynamic discretization discovery algorithm, where the algorithm aims to discover which times are needed to obtain an optimal solution by solving a small sequence of integer programs. Authors deal with the problem of replenishing a set of customers over a finite planning horizon but, unlike the problem we describe in this work, stock out situations are not considered, the replenishment plan is performed over a single period and consistency in the deliveries and time windows are not considered. Due to this continuous consumption of product, high additional efforts must be driven to carefully plan delivery times and amounts. Customers that are visited too late in time can lose sales due to stock out situations. On the other hand, early deliveries to customers can be unprofitable, as the total

* Corresponding author.
E-mail addresses: emilio.jose.alarcon.ortega@univie.ac.at (E.J. Alarcon Ortega), michael.schilde@univie.ac.at (M. Schilde), karl.doerner@univie.ac.at (K.F. Doerner).

https://doi.org/10.1016/j.orp.2020.100152
Received 15 May 2020; Accepted 15 May 2020
Available online 21 May 2020

amount delivered to the customer can be too low and, therefore, additional deliveries can be necessary in the subsequent periods.

Our contributions in this paper are threefold: (1) We introduce and mathematically formulate a new variant as CIRPTWSD where the product is being consumed continuously at the customers and they demand consistency in the delivery times in order to anticipate and be ready to receive them. We present two different versions of the mathematical formulation using two different replenishment policies, the order-up-to-level policy (OU) and the maximum level policy (MLP). The first policy requires that every visit to a customer means the entire inventory capacity is filled. The second policy provides more flexibility to decide delivery amounts, such that for each customer visit, any amount can be delivered as long as inventory capacity is respected [4]. (2) We propose a matheuristic solution method based on an adaptive large neighborhood search algorithm (ALNS) to deal with the CIRPTWSD. In this algorithm several destroy and repair operators are applied to an initial solution to iteratively remove and rebuild parts of the solution. This algorithm also includes a mathematical subproblem where, given a fix plan of deliveries to the customers, the arrival times and delivery amounts are optimally calculated. (3) We evaluate the performance of the proposed algorithm and the impact of the characteristics considered in the problem. In order to evaluate the effectiveness of the algorithm, we developed a modified and more simple version of the algorithm to solve a benchmark set of instances for the IRP introduced in [4]. We then propose an adapted set of instances from another benchmark set and a real world based set of instances to conduce the experiments. Very preliminary results were presented at the OR2016 conference and published in the conference proceedings, Alarcon et al. [5].

The CIRPTWSD belongs to the family of vehicle routing problems (VRPs), first present in [6], and, more specifically, the group of IRPs. The family of IRPs, are NP-hard problems and, it is in particular the CIRPTWSD, as it can be considered as an extension of the VRP with time windows (VRPTW) when considering a single period and subperiod (the VRPTW is an NP-hard problem [7]). Extensive reviews of IRP literature are available in [8] and [9], and in [10] we find an overview of different industrial applications of IRPs. Several articles include different aspects of the CIRPTWSD, and the two key features that represent the focus of our research, consistency and split deliveries, are widely discussed in [11,12] and [13]. With regard to consistency, Coelho et al. [11], present different interpretations of consistency (quantity consistency, driver consistency, and others), but not consistency in delivery times. This is the interpretation of consistency that we introduce in the CIRPTWSD. Therefore, we propose defining deliveries to a customer as consistent if all deliveries arrive at the same time of day. Kovacs et al. [14] introduce consistency in delivery times in a mathematical model, by ensuring that the arrival time difference to a customer is lower than a certain parameter value. Further insights into VRPs in which consistency is an important characteristic are available in [12], as well as [15] and [16]. Turning to split deliveries, each customer can be serviced by more than one vehicle in the same period and subperiod. Recent studies ([17,18]) introduce new applications for split pick-ups and split deliveries, first with discrete commodities and then in real-world situations related to the fuel industry. Furthermore, Christiansen [19] presents a problem with both inventory management and the possibility of split deliveries. Finally, Archetti and Speranza [13] consider the possibility of split deliveries in VRP problems with a single period, while also noting the main properties and different solution approaches for dealing with several variants of the basic VRP with split deliveries.

Another key characteristic, which we introduce herein, is time management combined with inventory routing. We consider an IRP with intra-day time windows. In most of the existing literature, time windows refer to a group of periods during which a delivery can take place, instead of a time interval within time periods. But because each customer has different opening times, we seek to operationalize delivery time windows in a way more commonly adopted in research into VRP problems, such as in Azi et al. [20]. Although this approach is less common in relation to IRP, some works consider these types of time windows, [21]. As another contribution, related to time and commodity management, we address linear commodity consumption by customers. This characteristic is also present in [3], while the possibility of lost sales is not considered. Combining linear consumption and the existence of intra-day time windows enables us to account for the possibility of lost sales due to stockout situations, not just at the end of each period, but also within time periods. In our work, lost sales may occur if customers do not have enough goods to satisfy corresponding demand before the next delivery. In our formulation, we penalize the amount of lost sales in the objective function.

Despite the recent introduction of some exact solution methods for IRPs ([22–25], we seek to solve large, real-world instances, so we choose a heuristic method. Heuristic methods are often used to deal with IRP variants, such as the combination of an integer programming approach and variable neighborhood search approach to deal with blood inventory and supply problems [26]. A template-based adaptive large neighborhood search applied to deal with the consistent vehicle routing problem [14], with template routes that include frequently serviced customers, then an introduction of sporadic customers in the template routes. An ALNS [11] is presented to solve different interpretations of consistent multi-vehicle IRPs. Other solution approaches for IRP include a matheuristic solution approach [27], a variable neighborhood search [28], and a decomposition approach [29]. Notably, we find increasing research interest in developing solution approaches to deal with IRPs with stochastic demands, such as with the introduction of two simheuristic approaches for the stochastic IRP with stockouts for single and multiple period ([30,31]).

In Section 2, we present a new formulation for the CIRPTWSD that integrates inventory management, vehicle routing, and delivery scheduling decisions with the previously detailed characteristics. In Section 3, we present a matheuristic solution approach in which we solve a linear subproblem based on the problem formulation of the CIRPTWSD, using the idea of an ALNS. We apply several destroy and repair operators to an initial solution obtained using a cheapest insertion heuristic, followed by two local search operators, then explore different solution neighborhood operators while, iteratively assigning more importance to the most successful neighborhoods by increasing the probability of using them. After we describe both, the mathematical formulation and the proposed solution approach, in Section 4, we present an extensive computational study. To evaluate the effectiveness of the solution approach, we use a set of instances of different sizes, adapted from a benchmark set. We also compare the results obtained by an exact solver with the results obtained using our proposed method. Furthermore, we study the impact of different time window sizes with respect to the different cost factors. Our algorithm provides high quality solutions for small instances after short computation times, compared with the exact solver, but for instances of medium and large sizes, the matheuristic provides better solutions than the exact solver; the solver cannot find optimal solutions within a given computational time limit of ten hours. In Section 5 we summarize the findings and propose additional research questions related to the CIRPTWSD.

2. Problem description

In this section, we describe the different characteristics of the problem and propose a mathematical formulation that includes all of them. In Fig. 1, we present a graphic scheme that represents the inventory flow and other aspects of the problem for a typical customer. A customer with a given initial inventory and inventory capacity, over a planning horizon of three periods, faces four different issues. The first characteristic represents the nature of the demand the customer manifest; every customer experiences linear commodity consumption in every period and subperiod, but the commodity consumption of each customer may vary from period to period. According to the second

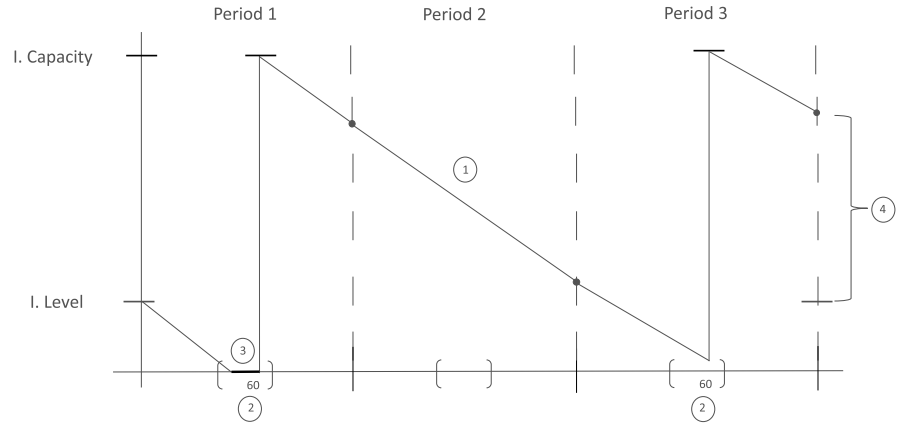

Fig. 1. Inventory flow and different characteristics of the problem for a given customer.

characteristic, each customer presents different time windows for delivery and require consistency in delivery times, we represent this element by assuming that the customer prefers to be delivered at the same time in every period or subperiod. The third characteristic denotes stockout situations, when the commodity cannot be served anymore, so we calculate the amount of commodity not served and penalize it in the objective function as lost sales. Finally, the fourth characteristic enables the creation of a pseudo-rolling horizon, such that we calculate the difference between the initial and the ending inventory level and penalize this difference in the objective function as lost sales. Excluding this characteristic would eliminate deliveries in the last periods and subperiods, to save possible routing costs, such that the ending inventory levels of the customers would tend to zero.

The CIRPTWSD is represented on a complete directed graph $G = (V, A)$, where V is the set of all nodes and A is the set of all arcs. We denote the depot as 0 and V' as the set of customers. We consider a finite planning horizon with $p \in P$ periods; each of these periods is divided into a given number of subperiods $r \in R$ with length T_r. We apply this division to deal with two issues. First, it helps us state a maximum route duration. Second, it represents a real-world situation caused by the different working shifts that a company has. We consider a finite and homogeneous fleet of vehicles, $k \in K$, with capacity Q. A cost c_{ij} and a time t_{ij}, both non-negative values, are considered for each arc $(i, j) \in A$. Furthermore, for each customer $i \in V'$ and subperiod $r \in R$, a_i^r and b_i^r represent the start and end times when the customer can be served. Initial inventory levels $I_i^{0|R|}$ are assigned to each customer $i \in V'$ and so are the inventory capacities C_i. To assign these initial inventory levels, we create a dummy period 0, and we assign the initial inventory levels to the last subperiod $|R|$ of this dummy period. This way, we can use the same notation for all the inventory information about the problem. The service time for delivering to a customer is represented as s_i. Finally, d_i^{pr} represents the commodity consumption of customer i in period p and subperiod r. We consider this consumption continuous within subperiods.

In our model, we also include different decision variables. The binary variables x_{ij}^{kpr} and y_i^{kpr} indicate if an edge (i, j) is being used by vehicle k in period p and subperiod r and if customer i is visited by vehicle k in period p and subperiod r, respectively. Moreover, we include continuous variables related to time and commodity management. The variables t_i^{kpr}, $\overline{t_i^r}$, and $\underline{t_i^r}$ relate the time and consistency management of the problem. The first type represents the arrival time of a vehicle k to a customer i in period p and subperiod r over all periods. The latter two types of variables represent the earliest and latest arrival time to each customer i for each subperiod r. Furthermore, o^{kpr} represents the loading time of each vehicle at the depot in each period and subperiod. Finally, we have other groups of decision variables associated with the commodity management. First, variables q_i^{kpr} represent the amount of commodity delivered to each customer i by each vehicle k in period p and subperiod r. Second, I_i^{pr} shows the inventory levels of each customer i at the end of each period p and subperiod r. Third, Δ_i represents the difference between the initial and the ending inventory level of each customer i, if the initial inventory is higher than the ending inventory. We use Δ_i to create a pseudo-rolling horizon and to avoid empty ending inventories. Fourth, o_i^{pr} represents the amount of lost sales at customer i in period p and subperiod r due to stockout situations. Fifth, because of the split delivery characteristic of the problem, we introduce $A_i^{kk'pr}$ to represent the amount q_i^{kpr} delivered by vehicle k to customer i if this vehicle arrives before vehicle k' in period p and subperiod r, and 0 otherwise. The notation used in the mathematical formulation is summarized in Table 1.

Using this notation, we can formulate the CIRPTWSD.

Table 1
Notation.

Data Sets	V	set of nodes
	V'	set of customers
	K	set of vehicles
	P	set of periods
	R	set of subperiods
	c_{ij}	travel cost for arc (i, j)
	t_{ij}	travel time for arc (i, j)
	a_i^r, b_i^r	time windows of customer i in subperiod r
	C_i	inventory capacity of customer i
Data and Parameters	$I_i^{0\|R\|}$	initial inventory level of customer i in dummy period 0 and subperiod $\|R\|$
	s_i	service time of customer i
	d_i^{pr}	commodity consumption of customer i in period p and subperiod r
	Q	vehicle capacity
	T_r	duration of subperiod r
	x_{ij}^{kpr}	use of arc (i, j) by vehicle k in period p and subperiod r
	y_i^{kpr}	visit to customer i in period p and subperiod r
	I_i^{pr}	inventory level of customer i at the end of period p and subperiod r
Decision Variables	t_i^{kpr}	arrival time of vehicle k to customer i in period p and subperiod r
	$\underline{t_i^r}$, $\overline{t_i^r}$	earliest and latest arrival times to customer i in subperiod r
	o^{kpr}	load time of vehicle k in period p and subperiod r
	Δ_i	final inventory decrease of customer i
	q_i^{kpr}	quantity delivered to customer i by vehicle k in period p and subperiod r
	$A_i^{kk'pr}$	previous deliveries to customer i in period p and subperiod r
	o_i^{pr}	quantity lost due to stockouts of customer i in period p and subperiod r

2.1. Objective function

$$\text{Minimize} \quad \sum_{i\in V}\sum_{j\in V}\sum_{k\in K}\sum_{p\in P}\sum_{r\in R} c_{ij}x_{ij}^{kpr} + \alpha\sum_{i\in V'}\sum_{r\in R}(\overline{t_i^r} - \underline{t_i^r}) + \\ + \sum_{i\in V'}\sum_{p\in P}\sum_{r\in R} Lo_i^{pr} + \sum_{i\in V'} L\Delta_i \tag{1}$$

The objective of the problem is to minimize total costs (1). We consider four different cost factors to minimize. The total routing cost incurred by servicing the customers, the penalty imposed according to the difference between the earliest and latest delivery to a customer, costs caused by stockout situations, such that the amount of commodity that each customer is not able to serve is weighted by a parameter L (initially set to three times the commodity cost), and the difference between initial and ending inventory levels at each customer, when this difference is positive, measured as sales lost, which enables us to create a pseudo-rolling horizon.

2.2. Time and routing flow constraints

$$\sum_{j\in V} x_{ji}^{kpr} = y_i^{kpr} \forall\ i \in V',\ \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{2}$$

$$\sum_{r\in R} y_i^{kpr} \leqslant |R| \forall\ i \in V',\ \forall\ k \in K,\ \forall\ p \in P \tag{3}$$

$$\sum_{k\in K} y_i^{kpr} \leqslant 2 \forall\ i \in V',\ \forall\ p \in P,\ \forall\ r \in R \tag{4}$$

$$\sum_{i\in V'} x_{0i}^{kpr} \leqslant 1 \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{5}$$

$$\sum_{i\in V'} x_{0i}^{kpr} - \sum_{i\in V'} x_{i(n+1)}^{kpr} = 0 \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{6}$$

$$\sum_{i\in V'} x_{ih}^{kpr} - \sum_{j\in V'} x_{hj}^{kpr} = 0 \forall\ h \in V',\ \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{7}$$

$$t_i^{kpr} + s_i + t_{ij} - M_1(1 - x_{ij}^{kpr}) \leqslant t_j^{kpr} \forall\ i, j \in V,\ \forall\ k \in K,\ \forall\ p \in P, \\ \forall\ r \in R \tag{8}$$

$$a_i^r y_i^{kpr} \leqslant t_i^{kpr} \leqslant b_i^r y_i^{kpr} \forall\ i \in V,\ \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{9}$$

$$t_0^{kp1} \geqslant \sigma^{kp1} \forall\ k \in K,\ \forall\ p \in P \tag{10}$$

$$t_0^{kpr} \geqslant t_{n+1}^{kp(r-1)} + \sigma^{kpr} \forall\ k \in K,\ \forall\ p \in P,\ r \in R\setminus\{1\} \tag{11}$$

$$t_0^{kpr} \geqslant \sum_{l=1}^{r-1} T_l \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{12}$$

$$t_i^{kpr} \leqslant \sum_{l=1}^{r} T_l \forall\ i \in V,\ \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{13}$$

$$\sigma^{kpr} = \beta \sum_{i\in V} s_i y_i^{kpr} \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{14}$$

$$\overline{t_i^r} \geqslant t_i^{kpr} - M_2(1 - y_i^{kpr}) \forall\ i \in V',\ \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{15}$$

$$\underline{t_i^r} \leqslant t_i^{kpr} + M_2(1 - y_i^{kpr}) \forall\ i \in V',\ \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{16}$$

We include two Big-M parameters (M_1 and M_2) related to the time management of the problem set, equal to the period length. Constraints (2)–(4) guarantee that every customer can be visited by at most two vehicles in each subperiod and each vehicle can visit each customer in every period. Constraints (5)–(7) are flow conservation constraints that describe each individual route. Constraints (8)–(14) ensure the feasibility of the time schedule. Constraints (9) force t_i^{kpr} to be 0 if customer i is not served by vehicle k in period p and subperiod r. Constraints (13) ensure that every vehicle must return to the depot before or at the moment the current subperiod ends. Constraints (14) define the loading time for each route as the sum of the service times over all customers in the routes multiplied by a constant factor β. Constraints (15) and (16) define the earliest and latest arrival times of each vehicle k to each customer i in subperiod r.

2.3. Inventory flow constraints

$$\sum_{i\in V'} q_i^{kpr} \leqslant Q \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{17}$$

$$q_i^{kpr} \leqslant y_i^{kpr} M_3 \forall\ i \in V',\ \forall\ k \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{18}$$

$$I_i^{p,1} - I_i^{(p-1)|R|} = \sum_{k\in K} q_i^{kp,1} - d_i^{p,1} + o_i^{p,1} \forall\ i \in V',\ \forall\ p \in P \tag{19}$$

$$I_i^{pr} - I_i^{p(r-1)} = \sum_{k\in K} q_i^{kpr} - d_i^{pr} + o_i^{pr} \forall\ i \in V',\ \forall\ p \in P,\ r \in R\setminus\{1\} \tag{20}$$

$$I_i^{pr} \geqslant 0 \forall\ i \in V,\ \forall\ p \in P,\ \forall\ r \in R \tag{21}$$

$$I_i^{pr} \leqslant C_i \forall\ i \in V',\ \forall\ p \in P,\ \forall\ r \in R \tag{22}$$

$$I_i^{0|R|} - I_i^{|P||R|} \geqslant \Delta_i \forall\ i \in V' \tag{23}$$

$$\Delta_i \geqslant 0 \forall\ i \in V' \tag{24}$$

In this second group of constrains, we include commodity and inventory management constraints. Constraints (17) establish a maximum vehicle capacity. Constraints (18) indicate that a customer cannot receive any amount by a vehicle if it is not visited by that vehicle in a period and subperiod. The big-M parameter in these constraints is equal to the vehicle capacity. Constraints (19) and (20) define the inventory levels of every customer at the end of each subperiod, calculated as the sum of the previous inventory level plus the amount of commodity received in the current subperiod and the amount that the customer is not able to serve due to stockouts. This value is further decreased by the commodity consumption of the customer in the current subperiod. Constraints (21) and (22) forbid inventory levels to be negative or exceed the inventory capacity of the customer. With Constraints (23) and (24), we calculate the difference between the initial and the ending inventory of each customer when the difference is positive.

2.4. Previous deliveries constraints:

$$t_i^{kpr} < t_i^{k'pr} = > A_i^{kk'pr} = q_i^{kpr} \forall\ i \in V',\ \forall\ k, k' \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{25}$$

$$t_i^{kpr} \geqslant t_i^{k'pr} = > A_i^{kk'pr} = 0 \forall\ i \in V',\ \forall\ k, k' \in K,\ \forall\ p \in P,\ \forall\ r \in R \tag{26}$$

We introduce two groups of constraints to deal with the split deliveries characteristic. By using Constraints (25) and (26), we define the matrix of variables $A_i^{kk'prz}$. If a customer i is visited more than once in a subperiod, the amounts delivered to the customer by the previous vehicles are calculated.

2.5. Stockout and overstock constraints

$$I_i^{(p-1)|R|} - d_i^{p,1} t_i^{kp,1}/T_1 + o_i^{p,1} \geqslant \\ - \sum_{k'\in K\ :k'\neq k} A_i^{k'kp,1} \\ \forall\ i \in V',\ \forall\ k \in K,\ \forall\ p \in P \tag{27}$$

$$I_i^{p(r-1)} - d_i^{pr}(t_i^{kpr} - y_i^{kpr} \sum_{l\in R\ :l<r} T_l)/T_r + o_i^{pr} \geqslant - \sum_{k'\in K\ :k'\neq k} A_i^{k'kpr} \quad \forall\, i \in V',\ \forall\, k \in K,\ \forall\, p \in P,\ r \in R\setminus\{1\} \tag{28}$$

$$q_i^{kp,1} + I_i^{(p-1)|R|} - d_i^{p,1} t_i^{kp,1}/T_1 + o_i^{p,1} \leqslant C_i - \sum_{k'\in K\ k'\neq k} A_i^{k'kp,1} \quad \forall\, i \in V',\ \forall\, k \in K,\ \forall\, p \in P \tag{29}$$

$$q_i^{kpr} + I_i^{p(r-1)} - d_i^{pr}(t_i^{kpr} - y_i^{kpr} \sum_{l\in R\ l<r} T_l)/T_r + o_i^{pr} \leqslant C_i - \sum_{k'\in K\ k'\neq k} A_i^{k'kpr} \quad \forall\, i \in V',\ \forall\, k \in K,\ \forall\, p \in P,\ r \in R\setminus\{1\} \tag{30}$$

We create some constraints to avoid excessive stock at the customers at any time, considering a continuous commodity consumption. Furthermore, as a result of continuous commodity consumption, we have to calculate possible stockouts that occur at any time within the time periods. Constraints (27) and (28) ensure that at the arrival time of any vehicle to a customer, the inventory level is non-negative. They also account for whether any other delivery occurs before the arrival time of each vehicle. Whenever a stockout occurs, the amount out of stock is calculated. Constraints (29) and (30) act in the opposite way, to avoid excessive inventory levels while also considering possible previous deliveries.

2.6. Additional OU constraints

The model we propose satisfies one of the most commonly used policies related to the inventory management and delivery amounts in IRP literature, the MLP. It is outpacing the OU in popularity, because the MLP allows companies to deliver any amount of commodity to each customer, as long as that amount, plus the current inventory level, does not exceed the inventory capacity. Alternatively, the OU policy ensures that every time a vehicle arrives to a customer, it delivers the exact amount of commodity necessary to fill inventory completely. Adapted to our problem, the OU policy ensures that, when the last vehicle arrives to a customer in a subperiod, the inventory must be full. If a customer is visited by multiple vehicles in a subperiod, the first vehicle does not necessarily fill the inventory entirely.

$$(1 - y_i^{kpr})M_4 + \sum_{l\in K} q_i^{lp,1} - (C_i - I_i^{(p-1)|R|}) + o_i^{p,1} \geqslant d_i^{p,1} t_i^{kp,1}/T_1 \quad \forall\, i \in V',\ \forall\, k \in K,\ \forall\, p \in P \tag{31}$$

$$(1 - y_i^{kpr})M_4 + \sum_{l\in K} q_i^{lpr} - (C_i - I_i^{p(r-1)}) + o_i^{pr} \geqslant d_i^{pr}(t_i^{kpr} - y_i^{kpr} \sum_{l\in R\ l<r} T_l)/T_r \quad \forall\, i \in V',\ \forall\, k \in K,\ \forall\, p \in P,\ r \in R\setminus\{1\} \tag{32}$$

We propose Constraints (31) and (32) to deal with the OU policy in our problem. By adding these constraints to the model, we ensure that, when the last vehicle has arrived, the total amount of commodity delivered satisfies the OU policy.

2.7. Variable domain

$$x_{ij}^{kpr} \in \{0, 1\} \forall\, i, j \in V,\ \forall\, k \in K,\ \forall\, p \in P,\ \forall\, r \in R \tag{33}$$

$$y_i^{kpr} \in \{0, 1\} \forall\, i \in V',\ \forall\, k \in K,\ \forall\, p \in P,\ \forall\, r \in R \tag{34}$$

$$t_i^{kpr} \geqslant 0 \forall\, i \in V,\ \forall\, k \in K,\ \forall\, p \in P,\ \forall\, r \in R \tag{35}$$

$$q_i^{kpr} \geqslant 0 \forall\, i \in V',\ \forall\, k \in K,\ \forall\, p \in P,\ \forall\, r \in R \tag{36}$$

$$A_i^{kk'pr} \geqslant 0 \forall\, i \in V',\ \forall\, k, k' \in K,\ \forall\, p \in P,\ \forall\, r \in R \tag{37}$$

In Constraints (33) to (37), we define the domain of each variable of the problem.

3. Solution approach

To solve the CIRPTWSD, we propose a matheuristic solution approach. Our method is based on the idea of the ALNS [32], which provides an efficient algorithm for IRPs, particularly for IRPs for which consistency is an important characteristic([11,14]). Given an initial solution, the ALNS applies several destroy and repair operators to iteratively remove and rebuild parts of the solution. We present different operators related to one or more characteristics of the problem. Thus, we explore different solution neighborhoods to increase the quality of the solutions. In contrast with a generic ALNS algorithm though, we propose a hybrid version with a mathematical subproblem, based on the mathematical formulation of the CIRPTWSD. Fig. 2 shows an overview of the proposed method, including the two parts of the algorithm, the constructive heuristic and the ALNS, their main components, and where the exact subproblem is integrated in both.

3.1. Initial solution

The initial solution is generated using an adaptation of the cheapest insertion heuristic [33] for a variant of the VRP. We combine this constructive heuristic with a method to decide which customers must be visited in each period and subperiod. We then apply two local search operators to improve the quality of the initial solution and solve a mathematical optimality subproblem to determine the optimal timing and delivery quantities and to introduce waiting times when necessary. A pseudocode of the complete heuristic is presented in Algorithm 1.

3.1.1. Preprocessing

Before we construct the routes for each period and subperiod, we decide which customers must receive a delivery in that subperiod. We create a priority list containing all customers whose inventory level at the beginning of the subperiod is not sufficient to survive without a stockout before the next possible delivery time window. We then calculate an upper and lower bound for the delivery amount for each customer. The upper bound is the difference between the current

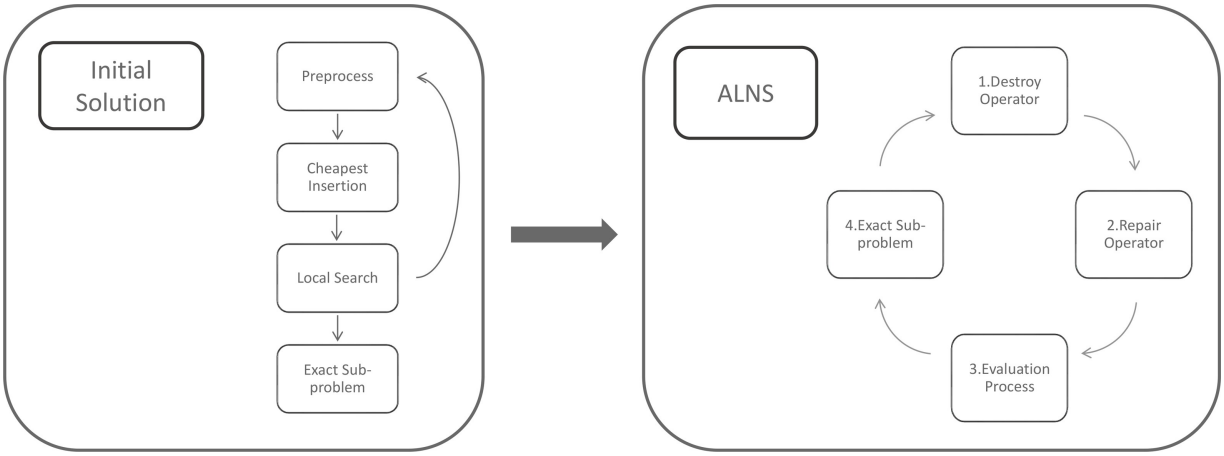

Fig. 2. Components of the method.

inventory level and the inventory capacity. The lower bound is the amount necessary to avoid a stockout before the end of the next delivery time window. In Fig. 3, we present a small example. The customer has enough inventory, so it will not suffer a stockout during Period 1. However, it runs out of stock before the next time window ends (in this case, in Period 2). Thus, the customer must receive a delivery in the current period. Then, after deciding if the customer must be visited or not, we calculate the two bounds. We repeat this process before applying the insertion heuristic for every period and subperiod.

3.1.2. Cheapest insertion

When we obtain the list of customers to be served in the current period, we insert them as follows: First, we initialize a route by inserting the customer farthest away from the depot that is included in the list of customers that we just created. Second, we insert customers according to a cheapest insertion algorithm, using the largest possible quantity, which is the upper bound, increased by the quantity consumed until the arrival of the delivery. Third, if we cannot insert any more customers in the route, due to capacity constraints of trucks or time windows of the remaining customers, we initialize a new route in the same way. We iterate until no more insertions are possible.

If all customers from the list have been inserted, the method stops. If there are still customers that need a delivery but cannot be inserted with the maximum quantity, we try to insert them with the minimum amount. We apply the cheapest insertion algorithm, as described, using the lower bound quantity increased by the amount that will be consumed until the delivery arrives.

If this second insertion step is not enough to insert all customers, a third procedure applies to balance the amount delivered to all customers and, if necessary, split deliveries to remaining customers into two different routes, assuming there is enough vehicle capacity available. If not, this customer receives a delivery in a later period. We balance the amounts delivered to customers by reducing the delivery amounts of customers inserted with the upper bound amount to reduce the total truck load and, therefore, allow for the insertion of more deliveries into the existing routes. If a customer's demand is greater than the vehicle capacity, or we have not been able to insert it, we try to split the amounts to be delivered in two vehicles.

Require: Input Data
for $p = 1 : |P|$ **do**
  for $r = 1 : |R|$ **do**
    for $i = 1 : |N|$ **do**
      if (Inventory of customer i $\leqslant$ demand until next time window) **then**
        Add customer i to the *Priority List*
        Calculate upper and lower bounds for the delivery
      end if
    end for
    Cheapest insertion with maximum quantity for all customers in *Priority List*
    if (*Priority list not empty*) **then**
      Cheapest insertion with minimum quantity for the rest of customers in *Priority List*
    end if
    if (*Priority list not empty*) **then**
      Balance quantities and split deliveries
    end if
    Local search 1: delete single customer routes
    Local search 2: 2-opt
  end for
end for
Solve exact subproblem
return Initial Solution

Algorithm 1. Initial Solution.

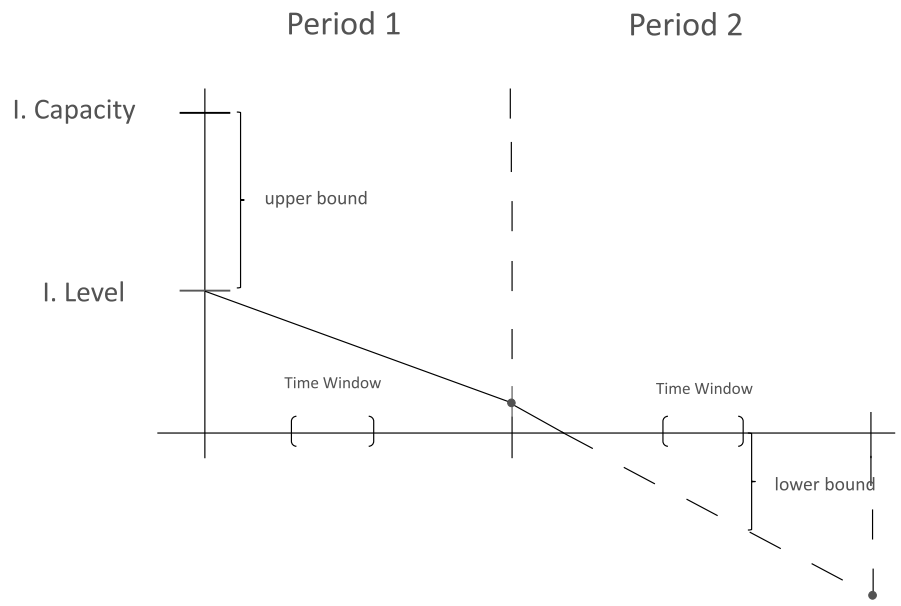

Fig. 3. Example of customer inserted in the priority list of Period 1 with the corresponding upper and lower bound amounts calculated.

3.1.3. Local search

To improve the quality of the initial solution, we apply two different local search operators that have been widely studied. The first operator aims to destroy single-customer routes and inserts these customers into other routes. The single customer route operator identifies the route and the customer to be re-allocated, then calculates the total amount of capacity left in all the remaining routes of the same period. Then, by decreasing the amounts delivered to the other customers of the route, it tries to insert the customer into an existing route. We only insert the customer in another route if the distance of the detour caused by this new insertion is lower than the total distance of the single customer route we are trying to avoid. The second operator is the well known 2-opt algorithm. In Appendix A, we list the results of the proposed algorithm using different chain lengths of the 2-opt algorithm. In the rest of the experiments, we apply the 2-opt algorithm with a maximum length of four customers. To ensure feasibility, we do not allow any changes that could violate the time windows of the customers. Thus we search for better sequences of customers, that satisfy the other characteristics of the problem.

3.1.4. Exact subproblem

We repeat the described construction heuristic for every period and subperiod to obtain a feasible initial solution. However, this first solution may not be optimal for this route sequence. To obtain the optimal delivery quantities for customers, introduce waiting times to improve possible inconsistent deliveries, and increase the final inventory levels that will create a pseudo-rolling horizon, we solve a reduced problem based on the problem formulation. The reduced problem contains all constraints of the original problem formulation, using MLP or OU when necessary, but the customer visits y_i^{kpr} and route sequences x_{ij}^{kpr} are no longer a variables and instead function as parameters.

3.2. Adaptive large neighborhood search

We use ALNS to improve the initial solution provided by the construction heuristic. The method integrates several operators to iteratively destroy and repair the current solution. Moreover, for some good solutions that we find during the ALNS, we solve the exact subproblem to obtain the optimal delivery schedule using the provided route sequences, as explained in the previous section. A pseudocode of the method is in Algorithm 2. To avoid local optima, we introduce the

Require: Initial solution and cost. $s_{initial}$ & $c_{initial}$
$s_{best}, s_{incumbent} = s_{initial}$
$c_{best}, c_{incumbent} = c_{inititial}$
$t, t_{last} = 0$ Start time and time of last improvement
while time t <MAXTIME **do**
  $s_{current} = s_{incumbent}$
  Select a pair of destroy and repair operators d and r
  Apply operator d to $s_{current}$
  Actualize inventory flows and stockouts
  Apply operator r to $s_{current}$
  2-opt local search
  Evaluate solution and get cost $c_{current}$
  if ($c_{current} < 1.5 \cdot c_{best}$) **then**
    Solve exact subproblem and get cost c_{exact}
    if ($c_{exact} < c_{best}$) **then**
      $s_{best}, s_{incumbent} = s_{current}$
      $c_{best}, c_{incumbent} = c_{exact}$
      Update weight ρ_{idr} of operators d and r
    else
      if ($c_{exact} < c_{incumbent}$) **then**
        $s_{incumbent} = s_{current}$
        $c_{incumbent} = c_{exact}$
      else
        if ($t - t_{last} > timelimit$) **then**
          $s_{incumbent} = s_{current}$
          $c_{incumbent} = c_{exact}$
        end if
      end if
    end if
  end if
end while
return s_{best} & c_{best}

Algorithm 2. ALNS.

option to accept deteriorating solutions after long periods without finding an improving solution.

In each iteration of the ALNS, we generate a new solution by applying one destroy and one repair operator. The selection of these operators relies on a roulette-wheel selection operator based on the past performance of the operators. The selection and priorization of destroy and repair operators are pairwise, rather than separate. Weights ρ_{0dr} related to each pair of destroy and repair operators are initially set to the same value of 20; when we find a new better solution, we update the weights of the operators, increasing them by $\gamma = 5$, then we resume the procedure of updating the operator weights as follows:

$$\rho_{idr} = \begin{cases} \rho_{(i-1)dr} + \gamma & \text{ifnewbettersolutionfound.} \\ \rho_{(i-1)dr} & \text{ifnobettersolutionfound.} \end{cases} \quad (38)$$

In this case, i is the current iteration of the ALNS algorithm. We propose eight destroy operators and seven repair operators, as described next.

- Remove worst insertions: Given a solution s, we define the cost of the detour caused by the insertion of customer i as $cost = c_{ki} + c_{ij} - c_{kj}$, where k and j are the preceding and succeeding customers in the route, respectively. Then, the operator calculates the detour cost caused by every insertion, selects the p worst insertions with respect to these costs, and removes them from the solution.
- Remove random insertions: This operator selects p different insertions from random periods, subperiods, and route, and removes them from the solution.
- Remove vehicle: This operator selects a vehicle at random and removes all deliveries performed by this vehicle in every period and subperiod.
- Remove subperiod: This operator selects a random period and subperiod and removes all routes of that subperiod.
- Remove customer: This operator selects p customers at random and removes all their deliveries.
- Remove customer and closest: This operator selects p customers at random and removes all deliveries to these customers and to their closest customer in terms of distance.
- Remove furthest customer and closest: This operator identifies the p furthest customers from the depot, in terms of distance, and removes all deliveries done to these customers and their closest customers in terms of distance.
- Remove similar inventory customers: This operator selects p customers at random and, for each, selects one other customer with a similar percentage of initial inventory level. Then, it removes all deliveries to all selected customers.

Every time we apply a destroy operator, the inventory levels of the removed customers change, and new stockout situations may occur. Because we consider intra-period stockouts, it is possible that a removed customer faces a stockout before the next delivery takes place but, at the end of the subperiod in which this next delivery happens, the inventory might partially fill again. As we describe in the mathematical formulation, stockout situations can also occur inside the periods and subperiods, which requires recalculating the new inventory flow. In Fig. 4 we provide an example of how we update the inventories of removed customers. In this example, a customer with an inventory capacity of 30 units, a demand of 15 units per period, and a total period length of six hours has its first out of three deliveries removed. This delivery was performed two hours after Period 2 starts, and the quantity delivered was 30 units. The delivery arrived at the exact moment inventory ran empty. As a result of its removal, the customer does not have enough inventory to satisfy the whole period's demand and faces lost sales of $Demand - InventoryLevel = 15 - 5 = 10$ units. However, this delivery also affects the subsequent period, such that in Period 2. In Period 3, the customer does not have any delivery scheduled, and thus loses all demand, leading to the lost sales of Period 3 equal to $Demand - Inventory = 15 - 0 = 15$ units. Finally, in Period 4, the customer has a scheduled delivery two hours after the period starts and therefore loses all demand that occurs before the new delivery takes place. In this case, the new amount of lost sales is $Demand \cdot ArrivalTime/PeriodLength = 15 \cdot 2/6 = 5$ units.

After we update the inventory levels and the amount of lost sales for the removed customers, we create a list of customers that must be reinserted into the solution, because that need deliveries to avoid stockout situations caused by their removal. We then apply the corresponding repair operator to create new deliveries to the customers in the list.

- Best insertion in destroy order: This operator inserts customers in the same order in which they were removed from the previous solution. To reinsert these customers into the existing routes, we create a list of possible insertions for the period a stockout occurs and for all preceding periods. The possible insertions are sorted by the total distance of the detour caused by them. We then randomly select one of the three best possible insertion positions and perform the insertion. If, after evaluating the new inventory flow of the customer, there still are stockout situations in the later periods, we repeat the process between the new stockout period and the last insertion period. Fig. 5 shows how this repair operator works with the same customer that we used in Fig. 4. After updating the inventory and the lost sales for the customer, we find a first stockout situation in Period 2. To avoid it, we evaluate all possible insertions in Period 2 and all earlier periods. As a result, we select a possible insertion of the customer with a delivery of 20 units four hours after Period 1 starts. We then calculate the resulting inventory flow and stockouts after this new delivery. Finally, we search for other possible stockouts, and we repeat the process, calculating possible insertions between the period of the new stockout(i. e., Period 3 in Fig. 5) and the period adjacent to the previous delivery (i. e., Period 2).

- Best insertion in random order: This operator works in the same way as the previous operator, except that we randomly select the next customer to be reinserted into the solution.
- Random insertion in destroy order: This operator creates a list of possible insertions just like the *best insertion in destroy order* operator but then randomly selects one of all possible insertions from the entire list.
- Random insertion in random order: This operator creates a list of possible insertions just as the previous operator and then randomly selects one of them from the entire list. We repeat the process until a feasible solution is reached.
- Consistent insertion in destroy order: This operator inserts customers such that the difference between the earliest and the latest arrival times to this customer is minimized.
- Consistent insertion in random order: This operator inserts customers in the same way as the previous operator, but in this case, the customer to be inserted is selected randomly and not in the destroy order.
- Distance-related insertion: This operator can be considered as an extension of the *best insertion in destroy order* operator. It selects customers in the same order as they were removed from the solution, and we calculate the best possible insertions in terms of the detour they cause, as we explained for Fig. 5. After inserting this customer, we search, in the list of customers to be reinserted, for its closest customer in terms of distance. We then try to insert this new customer next to the previous customer if possible. If by inserting this customer we can reduce its lost sales, we remove not only the first customer from the list but both customers.

When we obtain a new solution after applying both, the destroy and

Inventory Capacity = 30units
Constant demand = 15units
Period Length = 6hours

Day	0	1	2	3	4	5	6
Inventory levels before removal	20	5	20	5	20	5	20
Inventory levels after removal	20	5	0	0	20	5	20
New lost sales	0	0	10	15	5	0	0

1st Delivery = 30u
Delivery time = 2h

2nd Delivery = 30u
Delivery time = 2h

3rd Delivery = 30u
Delivery time = 2h

Fig. 4. Update inventory flow after destroy operator.

Day	0	1	2	3	4	5	6
Inventory levels before insertion	20	5	0	0	20	5	20
Lost sales	0	0	10	15	5	0	0
Inventory levels after insertion	20	25	10	0	20	5	20
New lost sales	0	0	0	5	5	0	0

1st Stock Out
2nd Stock Out
2nd Insertion?
1st Insertion= 20units
Delivery time = 4hours

Fig. 5. Update inventory flow after repair operator.

the repair operators, we evaluate the total costs of the solution. If the new solution is not more than 50% worse than the best known solution, we solve a reduced variant of the mathematical problem formulation, as explained in Section 3.1. We consider a maximum 50% decrease in the quality of the solution to be able to solve the reduced exact problem for a considerable amount of new solutions, without expending unnecessary time solving solutions with high costs. For a computational justification, in Appendix A we also provide alternative results obtained by solving instances when we consider different acceptance criteria parameters in the proposed ALNS. Finally, we have a new solution with optimal costs for these route sequences. If the new solution improves the incumbent solution, we update the incumbent solution and the weights for the destroy and repair operators used in this iteration. We do the same with the best known solution if the new solution improves on it. However, if the new solution does not improve the incumbent or the best known solution, we proceed in two different ways. If the total time since the last improvement is lower than a *timelimit*, we use the previous incumbent solution as the starting solution for the next ALNS iteration. If the total time since the last improvement exceeds that time limit, we accept the new solution as the incumbent solution, and therefore, as the input for the next iteration of the ALNS. We then reset the time of the last improvement found to zero. We accept this new solution to avoid strong local optima that prevent the method from continuing to improve the quality of the solutions. The stopping criterion for the ALNS is an overall limit on runtime.

4. Computational results

To the best of our knowledge, this is the first work on the CIRPT-WSD that takes into account time windows and stockout situations within the time periods. To analyze the effectiveness of the proposed approach we first perform computational experiments using the benchmark set of instances first introduced in [4]. In a second step, to gain insights into the problem properties, we perform computational experiments on two sets of instances. The first set is adapted from a benchmark set, and the second is based on a real-life application of the problem. Other sets of instances have been proposed to solve other variants of IRPs, but the time window characteristic of our problem makes it rather difficult to adapt these instances to our problem setting. We evaluate the impact of different replenishment policies and the time windows on our solutions and compare the results to others obtained using an exact solver on our set of instances.

The algorithm we propose was coded in C++, and all computational experiments were performed on a Linux system equipped with two Intel Xeon E5-2650(2.6 GHz) processors and 64 GB RAM. IBM CPLEX 12.6.3 was used as a MIP solver with a maximum computation solving time of ten hours.

4.1. Test instances

Because previous efforts to demonstrate the computational effectiveness of the proposed methods do not include inventory information related to capacity and initial inventory at the customers for IRPs with

time windows, [21], we cannot compare this problem with any previous studies and instead create a new set of instances[1] adapted from a benchmark set[34]. This benchmark set of instances pertain to periodic vehicle routing problem with time windows (PVRPTW), for which the algorithm chooses between different visit day combinations that are already given. In our case, the visit sequences must be chosen by the algorithm based on current inventory levels.

These instances consider a planning horizon of four days and the most efficient visit pattern must be chosen between different visit combinations. We create instances of different sizes (5, 10, 15, 20, and 48 customers) and a planning horizon of four periods with one or two subperiods. We use the customer information related to coordinates and time windows from the instances of size 48 in the benchmark set. However, we increase the commodity consumptions of the benchmark set of instances to balance the inventory and stockout costs with the rest of the costs in the problem. In the proposed set of instances, we consider the commodity consumptions to be ten times the commodity consumptions of the benchmark set. For smaller conversion factors, some previous experiments show that the optimal decision would be not to perform any delivery, because the routing and consistency cost would be higher than stockout costs. Yet selecting a bigger factor would imply high stockout costs compared with the other costs considered in the problem. Finally, we created inventory information for our instances. We consider different scenarios to create inventory information and force every customer to be delivered at least once or twice, depending on its demand periodicity. Every customer with daily commodity consumption has an inventory capacity equal to two times the average demand. For customers that present consumption every second period, we set inventory capacity to 1.5 times the average consumption. For customers that present consumption in one period for instances of four periods and one subperiod, and two periods for instances with four periods and two subperiods, the inventory capacity is set to approximately 1.1 times the average consumption. Then, based on the inventory capacity of each customer, we generate initial inventory levels by randomly selecting an initial percentage of initial inventory relative to the inventory capacity (25%, 50%, 75%, or 100%). Vehicle capacity is set to 2000 units, and we fixed the fleet size to be able to deliver twice the average commodity consumption per period on a single period. Furthermore, we set the penalty cost $L = 3$ for each unit of demand that customers are not able to satisfy or unit of difference between the initial and ending inventory level and the consistency cost weight $\alpha = 1$. We select $\alpha = 1$ because the benchmark instances [34] consider time windows that depend on the routing distances, so both routing and consistency costs are measured using the same unit. We also set the penalty cost $L = 3$ to balance the possible stockout and inventory costs, relative to the rest of the costs considered in the problem.

The characteristics of the proposed instances are summarized in Table 2. For each instance size, we create four groups of ten instances. For fair comparisons, each group includes different characteristics of the problem to evaluate the impact of these aspects on the costs. Instances in groups of type a include regular commodity consumption, so most of the customers engage in commodity consumption in every period. Instances of type b include the same customers but without any time windows. Instances of type c include customers with less regular consumption, such as customers with consumption once or twice per time horizon. The last group, instances of type d, includes information for four periods and two subperiods, unlike the other types for which we include information of four periods and one subperiod. In Table 2, we also include the number of vehicles available at the depot at the beginning of the planning horizon.

In addition to the described set of instances, we perform computational experiments using another two set of instances. In order to evaluate the performance of the algorithm and compare the effectiveness with respect of previous works present in the literature, we solve a simpler version of the problem using the benchmark set of instances introduced in [4]. This set of instances presents information about demand, inventory capacity and level and location of different sets of retailers whereas no time-related information is not included. Furthermore, different planning horizons are considered in the instances (three and six periods) and, unlike the problem we face in this work, two variants with high and low holding cost are considered. For further information about the benchmark set of instances we refer to the original paper.

In the last step, we create a set of instances based on a real-world application of the problem. We have information about average beer consumption per sit place, opening times and locations, and real-world distances and times for 400 bars, restaurants, and beer stands in the city of Vienna. We generated five groups of real world-instances (a, b, c, d and e) that include information about 92 customers located in its inner city. For each group, we generate demand using a normal distribution that reflects daily average beer consumption. Each group contains five instances with different initial inventory levels generated, in accordance with the adapted benchmark set of instances. We provide further information about these instances in Table 3. They feature a planning horizon of a week (seven days) with one or two subperiods and a fleet size of five vehicles. Furthermore, customers prefer different time windows and have varied opening days.

Table 2
Characteristics of instances for the CIRPTWSD.

Group	Size	Periods	SubPeriods	Vehicles	Consumption	Time Windows
1a	5	4	1	1	High	TW
1b	5	4	1	1	High	No TW
1c	5	4	1	1	Low	TW
1d	5	4	2	1	High	TW
2a	10	4	1	2	High	TW
2b	10	4	1	2	High	No TW
2c	10	4	1	2	Low	TW
2d	10	4	2	2	High	TW
3a	15	4	1	2	High	TW
3b	15	4	1	2	High	No TW
3c	15	4	1	2	Low	TW
3d	15	4	2	2	High	TW
4a	20	4	1	3	High	TW
4b	20	4	1	2	High	No TW
4c	20	4	1	3	Low	TW
4d	20	4	2	2	High	TW
5a	48	4	1	6	High	TW
5b	48	4	1	3	High	No TW
5c	48	4	1	6	Low	TW
5d	48	4	2	3	High	TW

4.2. Comparison to benchmark set of instances

In order to evaluate the effectiveness of the proposed method, and

Table 3
Characteristics of real-world instances for the CIRPTWSD.

Group	Size	Periods	SubPeriods	Vehicles	Consumption	Time Windows
92a	92	7	1	5	Normal	TW
92b	92	7	1	5	Normal	TW
92c	92	7	1	5	Normal	TW
92d	92	7	1	5	Normal	TW
92e	92	7	1	5	Normal	TW
92a2	92	7	2	5	Normal	TW
92b2	92	7	2	5	Normal	TW
92c2	92	7	2	5	Normal	TW
92d2	92	7	2	5	Normal	TW
92e2	92	7	2	5	Normal	TW

[1] The set instances are available at https://bda.univie.ac.at/research/data-and-instances/vehicle-routing-problems/ when published

Table 4
Comparison to benchmark set of instances. Gaps with respect to the known optimal solutions when the algorithm finds solutions without stock out situations.

	3 Periods				6 periods			
	High Holding cost		Low Holding cost		High holding cost		Low Holding cost	
Instance	Optimal	Gap	Optimal	Gap	Optimal	Gap	Optimal	Gap
abs1n5.dat	2149.80	0%	1281.68	0%	3335.24	1%	5942.82	1%
abs2n5.dat	1959.05	0%	1176.63	0%	2722.33	0%	5045.91	0%
abs3n5.dat	3265.44	0%	2020.65	0%	4776	0%	6956.28	0%
abs4n5.dat	2034.44	0%	1449.43	0%	3246.66	2%	5163.42	1%
abs5n5.dat	2362.16	0%	1165.40	0%	2419.67	0%	4581.66	0%
abs1n10.dat	4970.62	0%	2167.37	0%	4499.25	1%	8870.15	1%
abs2n10.dat	4803.17	0%	2510.13	0%	5236.98	1%	8569.73	0%
abs3n10.dat	4289.84	0%	2099.68	0%	4652.53	3%	8509.81	0%
abs4n10.dat	4347.06	0%	2188.01	0%	5104.91	4%	8792.29	1%
abs5n10.dat	5041.62	0%	2178.15	0%	4670.76	0%	9620.07	0%
abs1n15.dat	5713.84	2%	2236.53	4%	5462.68	1%	12118.83	2%
abs2n15.dat	5821.04	0%	2506.21	0%	5494.74	4%	11932.10	3%
abs3n15.dat	6711.25	4%	2841.06	9%	6060.38	11%	13554.15	5%
abs4n15.dat	5227.56	0%	2430.07	0%	5504.65	5%	10618.55	3%
abs5n15.dat	5210.85	0%	2453.50	1%	5309.48	4%	10385.548	3%
abs1n20.dat	7353.83	5%	2793.29	12%	6490.18	-	14702.95	-
abs2n20.dat	7385.03	2%	2799.90	4%	6082.54	3%	14646.96	1%
abs3n20.dat	7903.97	-	3101.60	-	6950.20	-	14532.91	-
abs4n20.dat	7050.91	-	3239.31	-	7432.78	-	14539.72	-
abs5n20.dat	8405.83	4%	3330.99	9%	7210.73	-	15896.71	-
abs1n25.dat	8657.70	11%	3309.64	-	7095.86	-	15581.47	-
abs2n25.dat	9266.87	-	3495.97	-	7484.84	-	16823.16	-
abs3n25.dat	9843.60	-	3481.45	-	7728.76	-	18098.02	-
abs4n25.dat	8677.86	-	3272.74	-	7509.02	-	16303.69	-
abs5n25.dat	10857.68	0%	3695.94	1%	7452.28	-	19047.70	-

Table 5
Results of our method vs CPLEX. * CPLEX is not able to solve all instances optimally. ** CPLEX is not able to find a feasible solution for some instances.

Instance set	CPLEX	CPLEX Gap	Matheuristic	Max Gap	Avg Gap	Min Gap	No Sol.	Opt	Imp
1a	1045.05	-%	1048.51	3.06%	0.31%	0.00%	0	10	0
2a	2108.96	-%	2154.38	11.93%	2.14%	0.00%	0	7	0
3a*	2820.83	4.87%	2914.63	14.28%	3.39%	-0.51%	0	3	1
4a*	3348.52	37.54%	3553.61	22.09%	6.42%	-0.56%	0	1	1
5a*	24693.08	89.82%	7546.64	-49.36%	-60.88%	-82.78%	0	0	10
1c	1023.23	-%	1028.96	3.58%	0.55%	0.00%	0	8	0
2c	1371.44	-%	1381.52	6.27%	0.71%	0.00%	0	8	0
3c*	1774.83	1.19%	1813.37	12.11%	2.20%	-0.80%	0	2	1
4c**	2356.80	9.41%	2624.99	23.54%	11.39%	3.19%	2	0	0
5c**	5299.79	59.77%	5153.55	11.71%	-2.59%	-20.26%	1	0	9
1d	1709.77	-%	1747.43	15.86%	2.26%	0.00%	0	5	0
2d*	3162.38	3.67%	3349.76	20.16%	5.89%	-2.52%	0	0	3
3d	3941.18	8.15%	4483.99	32.18%	13.99%	-4.39%	0	0	1
4d*	5247.19	36.35%	5692.06	31.50%	8.74%	-10.18%	0	0	4
5d*	71007.81	94.24%	13730.58	-69.51%	-72.08%	-87.58%	0	0	10

compare the obtained results with a benchmark set of instances, we have adapted the method to solve a simpler version of the problem. We use the set of instances introduced in [4]. In this work, authors present a different variant of the IRP where time-related information and the possibility of stock situations is not considered. Removing the calculations related to these characteristics from the proposed method, would drastically transform the algorithm. However, small adaptations can be performed to create a fair comparison. The proposed algorithm is adapted so that the continuous consumption of commodity is no longer considered. That is, customers experience demands at the end of the planning horizon and, therefore, it is no longer possible to incur in stock out situations within the time periods, whereas the stock out possibility remains at the end of the subperiods. Secondly, the mathematical subproblem solved within the algorithm must be also adapted. In this case, we solve a mathematical formulation related to the IRP variant presented in [4]. Using this mathematical formulation, we can also calculate the different cost considered in this version of the problem. Inventory holding cost are calculated using this mathematical subproblem, and those solutions which present an stock out situation can be discarded.

In Table 4, we show the gap obtained when solving the set of instances with respect to the known optimal solution. We report results for those instance sizes where the algorithm was able to find feasible solution within a total computational time of 30 minutes. Solutions that present stock out situations are treated as non feasible solution, as the problem presented in [4] does not consider this situation. For most of the small instances (5, 10 customers) with high and low holding cost and three and six subperiods, the proposed algorithm is able to find optimal or nearly optimal solutions for the problem, even if the original proposed method does not aim to reduce holding cost or can consider stock outs. For medium size instances (15, 20 customers), the solutions obtained by the method present relatively small gaps with respect to the optimal solution. Finally, we can see how, for large instances, the algorithm has difficulties to obtain solutions without stock outs, as this is not the main feature considered in the method.

4.3. Comparison to the exact solver

In a second step, we solve the adapted set of instances and we compare the results obtained by applying our solution approach to instances of types a, c and d which include all the characteristics of the CIRPTWSD. We solve each instance of the different groups ten times with a maximum computation time of 30 minutes, where the *timelimit* without finding a new better solution is set to five minutes. In Table 5, we compare the results with those obtained by solving the instances using CPLEX with a time limit of ten hours.

In Table 5, each row reports the average objective cost of the ten instances included in every group with both CPLEX and our solution approach. Groups of instances with an asterisk (*) indicate that CPLEX was not able to prove that the best solution found was optimal within the time limit. Furthermore, groups of instances with two asterisk (**) meet that condition but also indicate that CPLEX was not able to find any feasible solution during the processing time for some instances. The column "CPLEX Gap" shows the average gap reported by CPLEX for those instances where the solver does not find the optimal solution, and the column "No Sol." reports the number of instances of each group where CPLEX is not able to find any feasible solutions. For groups 3a, 3c, 4a, and 4c, CPLEX finds optimal solutions for some instances, whereas for groups 5a and 5c, CPLEX is not able to find any optimal solution. The column labeled "Opt" reports how many of the ten instances in each group our solution approach was able to solve to optimality. The last column ("Imp") reports for how many instances, our solution approach was able to provide better results than the ones found by CPLEX. Our algorithm thus obtains good results relative to CPLEX.

For small instances with five and ten customers, we find optimal solutions for most cases. For example, for group 1a, the algorithm finds optimal solutions for all instances, with an average gap of 0.31%. For instances with 15 and 20 customers, our algorithm still finds good solutions, but we find bigger gaps compared with the solutions obtained by CPLEX when solving instances that contain 20 customers. The largest gap corresponds to the group of instances 3d, where CPLEX finds solutions that are 13.99% better on average. Furthermore, the algorithm can get stuck in local optima, such as in instances 3d and 4d, where, the maximum gaps compared with the best solution found by CPLEX reach 32.18%, though the average gap for these groups of instances is smaller than 14%. Finally, CPLEX is not able to obtain efficient solutions for bigger instances, and the proposed algorithm outperforms CPLEX when solving considerably big instances. For example, in groups 5a and 5d, the proposed algorithm finds better solutions for every instance, improving 60.88% and 72.08% on average, and for the group of instances 5c, it finds a better solution for nine out of ten instances.

Our algorithm performs better especially when customers exhibit more frequent commodity consumption, that is, for instances of group a. On the contrary, it becomes more difficult to integrate more customers with infrequent commodity consumption. However, for the biggest size of every group, the algorithm obtains better solutions than CPLEX with significantly shorter running times. Instances of type d, which present the results for a planning horizon of four periods and two subperiods, reveal similarities with instances with one subperiod, in that CPLEX is only able to provide optimal results for five customers, and some instances of 10, 15, or 20 customers, but it cannot find good solutions for bigger instances. Overall, the proposed method outperforms the alternative option for these large instances.

4.4. Time windows

A key characteristic that can drastically affect the total cost is the existence of time windows. Therefore, we examine the impact of time windows on the different cost factors. This impact has been widely studied for different variants of the VRP, yet literature related to IRP problems with time windows within the time periods is scarce.

To evaluate how time windows affect the costs, we compare the results obtained by solving data sets a and b. These customers have the same coordinates, commodity consumption, and inventory information, but type b excludes time windows. In Table 6, we list the average total cost for each instance size; the "Obj. Cost %" column reports the percentage of the decrease from the total cost, calculated using the formula $(TotalCost_{TW} - TotalCost_{NoTW})/TotalCost_{TW}$. In the last four columns, we present the difference for every cost factor using both policies.

Time windows have a strong impact on the overall costs; the total cost decreases for every instance without time windows, even if that decrease is related mainly to the routing and inventory costs. The lack of time windows makes it easier for the algorithm to obtain better routes in terms of distances and also allows the customer to receive late deliveries to increase the ending inventory level and, therefore, reduce inventory costs. In the third row, for instances of size 15, 3a vs. 3b, the presence of time windows implies an increase of 39.12% in the overall cost, whereas a smaller improvement takes place for instances of 48 customers, 5a vs. 5b, with an average of 3.32%. This difference on percentages is mainly due to a significant increase on the sales lost for bigger instances when not considering time windows. In general, stockout costs decrease for small instances but increase for larger instances. The last column shows the increase in consistency costs, which arise because the time windows force the algorithm to deliver to customers within smaller time ranges. Some of the reported percentages are quite large as in instances 2a vs. 2b, where the percentage is equal to − 894%. These big values emerge when the consistency cost for one group of instances is very big (in this case, 2b), but for the other group (2a) is relatively low, or even close to zero.

4.5. Real-world based instances

In addition to the computational experiments, we evaluate the effectiveness of the proposed algorithm using instances based on a real-world application. These instances contain information about beer consumption in different establishments in the city of Vienna. However, no previous results exist, and we cannot solve these instances using CPLEX.

There exist commercial solvers that create delivery plans for vehicle routing problem applications. Nevertheless, these solvers do not consider inventory information or multi-period problems. Therefore, companies must create delivery plans using simple heuristics. In this section we evaluate the impact of applying the ALNS and report the costs decrease compared to the solution obtained with the construction heuristic (C.Heuristic).

In Tables 7 and 8, we report costs for instances that contain information related to 92 customers located in the inner city of Vienna with a planning horizon of seven periods and one and two subperiods, respectively. We provide the percentage of difference in total costs, as well as the total difference for every individual cost factor.

In the results, we can see how the use of the ALNS algorithm with the mathematical subproblem significantly improves the quality of the solutions. Using the proposed matheuristic, we find solutions that reduce the overall costs, generally achieved by inserting additional deliveries to customers, so that it avoids customer lost sales and reduces inventory costs as well. In Table 7 for example, regarding 92a, the matheuristic reduces the costs by 46.43% compared with the initial solution. It may increase the routing cost by 79%, but it reduces inventory, stockout, and consistency costs by 59%, 49%, and 18%, respectively. For instances with a planning horizon of seven days and two subperiods, the matheuristic also drastically improves the results obtained with the insertion heuristic, in that it performs more deliveries and thus diminishes sales lost due to stockouts while also avoiding low

Table 6
Impact of time windows.

Groups	TW	No TW	Obj.Cost(%)	Rou.Cost	Inv.Cost	S.O.Cost	Cons.Cost
1a vs 1b	1048.51	770.38	26.53%	15%	71%	-46%	0%
2a vs 2b	2154.38	1412.90	34.42%	10%	80%	37%	-894%
3a vs 3b	2914.63	1774.39	39.12%	13%	76%	58%	-80%
4a vs 4b	3553.61	2444.93	31.20%	9%	71%	-67%	-239%
5a vs 5b	7546.64	7307.72	3.32%	16%	15%	-247%	-245%

inventory levels at the end of the planning horizon.

Further computational experiments where driven in order to evaluate the impact of both, MLP and OU policies. This experiments show that the difference between the MLP and OU results are caused mainly by the randomness of the algorithm. The problem we present does not include inventory holding costs, which are the ones derived from creating the pseudo-rolling horizon, so an OU policy is often considered the best replenishment policy when using the MLP approach. We present these computational experiments in Appendix B.

5. Conclusions and further research

We introduce the CIRPTWSD, which is part of the family of IRPs. The characteristics of the problem we present arise from a real-world application namely, route planning and inventory management for beer and other beverages companies. Here, customers have different opening times and time windows, so to satisfy overall demand, it may be necessary to split the deliveries across more than one vehicle. Furthermore, consistency in delivery times improves service quality and customer satisfaction, which is a key factor in a competitive market. We propose a mathematical formulation for the CIRPTWSD. We formulate the problem as a MIP with two variants. The first variant includes all characteristics of the problem and uses the maximum level replenishment policy, which gives companies the flexibility to decide on the delivery amounts and times. The second variant includes all constraints of the first variant and some additional constraints, in line with an OU policy that requires the inventory capacity of each customer to be completely filled at the moment each customer is served.

To solve the proposed mathematical formulation, in Section 3, we propose a matheuristic solution approach to solve the CIRPTWSD. The solution approach we propose is based on an ALNS that includes several destroy and repair operators applied to improve an initial solution obtained the cheapest insertion algorithm. Furthermore, during the process of obtaining good solutions, we solve an exact subproblem up to optimality using good solutions. We solve this mathematical subproblem to obtain optimal delivery amounts and times using the given routes. Successful neighborhood operators become more important as the algorithm obtains new, better solutions, until a maximum processing time is reached and the algorithm ends.

In some previous literature, authors have presented an IRP with time windows, but they rely on benchmark instances that do not include inventory information, so we cannot report any comparison with previous results. To test the efficiency of the proposed method, we present a range of computational tests with a benchmark set of instances and with new instance set adapted from a benchmark set. To perform the first computational experiments, and compare our results with previous results in the literature, we have adapted our algorithm to not account for some of the characteristics that the CIRPTWSD presents. This characteristics are related to the continuous consumption of product at the customers, and the possibility of lose sales. The algorithm finds good solutions in short processing times, whereas the exact solvers are not able to solve large instances even with long processing times. We report the results obtained from solving instances of different sizes. In these results, for small instances, the algorithm obtains optimal solutions in most occasions. However, for instances that include 20 customers, the algorithm uncovers bigger gaps compared with the results obtained using an exact solver. Finally, for the bigger instances solved, the algorithm outperforms the results obtained with CPLEX. Other experimental comparisons show the differences between the two described replenishment policies, as well as the impact of the time windows. Finally, we perform computational experiments using instances that include real-world information about beer consumption in the city of Vienna.

In this field, research could take different directions. Model-specific research might study other aspects of the problem, such as the possibility of a multi-echelon distribution systems in which a central retailer supplies different intermediate storage centers, and customers are replenished from these storage centers. It also may be interesting to include stochasticity in the problem. Several studies include stochastic travel times or commodity consumption by customers. We aim to develop a two-stage stochastic formulation for the CIRPTWSD. In a first stage, we will obtain an initial delivery plan that includes the set of routes and information about the delivery times and amounts to customers over the planning horizon. In the second stage, different recourse actions can be considered to reduce the overall costs when the stochastic data are being revealed. Another research possibility would be to develop more efficient solution methods. The proposed method provides better solutions for customers with high consumption periodicity, but as the number of non-frequent customers increases, it becomes more difficult to integrate them into the solution efficiently. Novel solution approaches could seek better solutions for instances with more punctual demands, larger instance sizes, and instances with longer planning horizons.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Table 7
Real-world instances, seven periods one subperiod.

Groups	C.Heuristic	Math.	Obj. Cost	Rou. Cost	Inv. Cost	S.O. Cost	Cons. Cost
92a	5609.20	3005.10	46.43%	-79%	59%	49%	18%
92b	6093.48	5031.46	17.43%	-53%	12%	39%	12%
92c	6321.41	4507.27	28.70%	-63%	38%	44%	12%
92d	7082.99	5034.76	28.92%	-60%	33%	47%	10%
92e	6967.24	5417.25	22.25%	-67%	20%	48%	8%

Table 8
Real-world instances, seven periods two subperiods.

Groups	C.Heuristic	Math.	Obj. Cost	Rou. Cost	Inv. Cost	S.O. Cost	Cons. Cost
92a2	4435.51	3420.17	22.89%	-98%	33%	32%	-1%
92b2	4302.27	3366.31	21.76%	-80%	37%	-9%	-22%
92c2	4816.63	3393.24	29.55%	-72%	33%	-75%	33%
92d2	2947.45	1934.91	34.35%	-124%	60%	27%	-11%
92e2	6635.6	3558.50	46.37%	-120%	55%	44%	-47%

Acknowledgment

Financial support from the Austrian and German Science Funds (FWF and DFG, D-A-CH) under grant # I 2248-N32 is gratefully acknowledged. The computational results presented herein were achieved, in part, using the Vienna Scientific Cluster (VSC).

Appendix A

In Tables 9 and 10 we provide results obtained when solving instances of the group a, described in Table 2 using different values for the maximum chain length considered in the 2-Opt algorithm and different values of the acceptance criteria applied in the proposed ALNS. In the tables, we list the average costs obtained when solving the instances with the different parameters and the average gap compared with the results of the selected parameters.

Table 9
2-OPT chain length tests.

	2-OPT 4	2-OPT 3		2-OPT 5	
Group	Avg	Avg	Gap	Avg	Gap
1a	1048.51	1048.63	0.01%	1048.11	-0.04%
2a	2154.38	2212.55	2.70%	2210.99	2.63%
3a	2914.63	3325.05	14.08%	3334.46	14.40%
4a	3553.61	3850.44	8.35%	3881.62	9.23%
5a	7558.39	8853.96	17.14%	8890.04	17.62%

Table 10
ALNS acceptance parameter tests.

	ALNS 1.5	ALNS 1.25		ALNS 1.75		ALNS 2	
Group	Avg	Avg	Gap	Avg	Gap	Avg	Gap
1a	1048.51	1051.84	0.32%	1048.50	0.00%	1048.90	0.04%
2a	2154.38	2251.46	4.51%	2218.89	2.99%	2219.21	3.01%
3a	2914.63	3191.90	9.51%	3373.62	15.75%	3316.87	13.80%
4a	3553.61	3702.64	4.19%	3874.88	9.04%	3910.33	10.04%
5a	7558.39	8594.27	13.71%	8846.71	17.04%	8911.45	17.90%

Appendix B

Table 11 presents a similar structure to Table 6. We present the percentage decrease in total cost between the results obtained solving the problem with both, MLP and OU policies, along with the total difference of every cost factor.

The difference between the MLP and OU results are caused by the randomness of the algorithm. The problem we present does not include inventory holding costs, which are the ones derived from creating the pseudo-rolling horizon, so an OU policy is often considered the best replenishment policy when using the MLP approach. For example, for instances 1a, 3a, and 5a, MLP policy obtains better solutions on average than the OU policy with a 0.36% average difference in group 3a. Nevertheless, the algorithm finds better solutions for groups ‘2a and 4a when applying OU policy.

Table 11
Results of MLP vs. OU.

Instances	MLP	OU	Obj.Cost(%)	Rou.Cost	Inv.Cost	S.O.Cost	Cons.Cost
1a	1048.51	1051.38	-0.09%	0%	0%	-7%	0%
2a	2154.38	2153.19	0.06%	0%	-1%	6%	-511%
3a	2914.63	2925.03	-0.36%	-1%	-3%	22%	-22%
4a	3553.61	3548.27	0.15%	1%	0%	-6%	15%
5a	7546.64	7552.95	-0.08%	1%	0%	-9%	1%

References

[1] Lee HL, Whang S. The whose, where and how of inventory control design. Supply Chain Manag 2008;Review 12(8):22–9.
[2] Bell WJ, Dalberto LM, Fisher ML, Greenfield AJ, Jaikumar R, Kedia P, et al. Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. Interfaces (Providence) 1983;13(6):4–23.
[3] Lagos F, Boland N, Savelsbergh M. The continuous-time inventory-routing problem. Transp Sci 2020;Articles in Advance:1–25.
[4] Archetti C, Bertazzi L, Laport G, Speranza MG. A branch-and-cut algorithm for a vendor-managed inventory routing problem. Transp Sci 2007;41(3):382–91.
[5] Alarcon Emilio J, Schilde M, Doerner Karl F. Consistent inventory routing with split deliveries. Operation Research Proceedings 2001. p. 395–401.
[6] Dantzig GB, Ramser J. The truck dispatching problem. Manag Sci 1959;6(1):80–91.
[7] Savelsbergh M. Local search in routing problems with time windows. Ann Oper Res 1985;4:285–305.
[8] Bertazzi L, Speranza MG. Inventory routing. In: Raghavan R, Golden B, Wasil E, editors. The vehicle routing problem latest advances and new challenges, operations research/computer science interfaces series, Springer, Berlin. 43. 2008. p. 49–72.
[9] Coelho LC, Cordeau J-F, Laporte G. Thirty years of inventory routing. Tranp Sci 2014;48(1):1–19.
[10] Andersson H, Hoff A, Christiansen M, Hasle G, Lokketangen A. Industrial aspects and literature survey: combined inventory management and routing. Comput Oper Res 2010;37(9):1515–36.
[11] Coelho LC, Cordeau J-F, Laporte G. Consistency in multi-vehicle inventory-routing. Tranp Res Part C 2012;24:270–87.
[12] Kovacs AA, Golden BL, Hart RF, Parragh SN. Vehicle routing problems in which consistency considerations are important: a survey. Networks 2014;64(3):192–213.
[13] Archetti C, Speranza MG. Vehicle routing problems with split deliveries. Int Trans Oper Res 2012;19:3–22.
[14] Kovacs AA, Parragh SN, Hartl RF. A template-based adaptive large neighborhood search for the consistent vehicle routing problem. Networks 2014;63(1):60–81.
[15] Kovacs AA, Braekers K. A multi-period dial-a-ride problem with driver consistency. Transp Res Part B: Methodol 2016;94:355–77.
[16] Campelo P, Neves-Moreira F, Amorim P, Almada-Lobo B. Consistent vehicle routing problem with service level agreements: a case study in the pharmaceutical distribution sector. Eur J Oper Res 2019;273(1):131–45.
[17] Qiu M, Fu Z, Eglese R, Tang Q. A tabu search algorithm for the vehicle routing problem with discrete split deliveries and pick-ups. Comput Oper Res 2018;100:102–16.
[18] Hennig F, Nygreen B, Furman K, Song J. Alternative approaches to the crude oil tanker routing and scheduling problem with split pick-up and split delivery. Eur J Oper Res 2015;243(1):41–51.
[19] Christiansen M. Decomposition of a combined inventory and time constrained ship routing problem. Transp Sci 1999;33(1):3–16.
[20] Azi N, Gendreau M, Potvin J-Y. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. Eur J Oper Res 2010;202(3):756–63.
[21] Liu S-C, Lee W-T. A heuristic method for the inventory routing problem with time windows. Expert Syst Appl 2011;38(10):13223–31.
[22] Adulyasak Y, Cordeau J-F, Jans R. Formulations and branch-and-cut algorithms for multivehicle production and inventory routing problems. INFORMS J Comput 2013;26(1):103–20.
[23] Desaulniers G, Rakke JG, Coelho LC. Branch-and-price-and-cut algorithm for the inventory-routing problem. Transp Sci 2015;50(3):1060–76.
[24] Coelho LC, Laporte G. Improved solutions for inventory-routing problems through valid inequalites and input ordering. Int J Prod Econ 2014;155:391–7.
[25] Archetti C, Desaulniers G, Speranza MG. Minimizing the logistic ratio in the inventory routing problem. EURO J Transp Logist 2017;6(4):289–306.
[26] Hemmelmayr V, Doerner KF, Hartl RF, Savelsbergh MW. Delivery strategies for blood products supplies. OR Spectrum 2009;31(4):707–25.
[27] Archetti C, Boland N, Speranza MG. A matheuristic for the multivehicle inventory routing problem. INFORMS J Comput 2017;29(3):377–87.
[28] Popovic D, Vidovic M, Radivojevic G. Variable neighborhood search heuristic for the inventory routing problem in fuel delivery. Experts Syst Appl 2012;39(18):13390–8.
[29] Campbell AM, Savelsbergh MW. A descomposition approach for the inventory routing problem. Transp Sci 2004;38(4):488–502.
[30] Juan AA, Grasman SE, Caceres-Cruz J, Bektas T. A simheuristic algorithm for the single-period stochastic inventory-routing problem with stock-outs. Simul Model Pract Theory 2014;46:40–52.
[31] Gruler A, Panadero J, de Armas J, Perez JAM, Juan AA. A variable neighborhood search simheuristic for the multiperod inventory routing problem with stochastic demands. Int Trans Oper Res 2018;00:1–22.
[32] Pisinger D, Ropke S. A general heuristic for vehicle routing problems. Comput Oper Res 2007;34:2403–35.
[33] Solomon MM. Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper Res 1987;35(2):254–65.
[34] Cordeau J-F, Laporte G, Mercier A. A unified tabu search heuristic for the vehicle routing problems with time windows. J Oper Res Soc 2001;52:928–36.