Noname manuscript No.
(will be inserted by the editor)

The integrated orderline batching, batch scheduling, and picker routing problem with multiple pickers: the benefits of splitting customer orders

Received: date / Accepted: date

Abstract Fast delivery is one of the most popular services in e-commerce retail. It consists in shipping the items ordered on-line in short times. Customer orders in this segment come with deadlines, and respecting this latter is pivotal to ensure a high service quality. The most time-consuming process in the warehouse is order picking. It consists in regrouping orders into batches, assigning those batches to order pickers, sequencing the batches assigned to each order picker such that the orders deadlines are satisfied, and the picking time is minimized. To speed up the order picking operations, e-commerce warehouses implement new logistical practices. In this paper, we study the impact of splitting the orders (assigning the orderlines of an order to multiple pickers). We thus generalize the integrated orders batching, batch scheduling, and picker routing problem by allowing the orders splitting and propose a route first-schedule second heuristic to solve the problem. In the routing phase, the heuristic divides the orders into clusters and constructs the picking tours that retrieve the orderlines of each cluster using a split-based procedure. In the scheduling phase, the constructed tours are assigned to pickers such that the orders deadlines are satisfied using a constraint programming formulation. On a publicly available benchmark, we compare our results against a state-of-the-art iterated local search algorithm designed for the non-splitting version of the problem. Results show that splitting the customer orders using our algorithm reduces the picking time by 30% on average with a maximum reduction of 60%.

Keywords Order picking · Splitting customer orders · Route first-schedule second heuristic · Hard due-dates

1 Introduction

Warehousing involves four main activities: receiving, storing, picking, and shipping goods (Tompkins et al. (2010)). Among all these activities, order picking is unanimously considered as the most time-consuming and costly process. It can

Address(es) of author(s) should be given

induce up to 65% of all labor activities (Drury (1988)) and 70% of the total operating costs (Kulak et al. (2012)). Two classes of systems can be distinguished to process the picking activities in the warehouses: manual picking systems and automated picking systems (Wäscher (2004)). In this paper, we focus on the manual picking system, more precisely a picker-to-part system, which is the most common in practice (De Koster et al. (2007)). It is characterized by (human) order pickers starting from depot, walking or driving trolley through aisles to pick stock-keeping units (SKUs) from storage locations, and coming back to depot. Pickers are directed using picking lists where the orderlines to retrieve in a single tour and their respective storage locations are itemized. A picking list can be composed of orderlines of a single order (pick by order) or orderlines of multiple orders (pick by batch).

In the last decades, a new retail model known as "e-commerce" has grown up significantly throughout the world. Canadian e-commerce retailers for instance sold for almost 40 billion $ of goods in 2018, accounting for 8.1% of the Canadian total retail sales. It represents a 9.1% increase over the previous year (36.6 billion $ in 2017), and it is expected to reach 56.6 billion $ in 2023. Similar growth rates can be observed on a more global scale, where business to costumer e-commerce sales have generated almost 995 billion in 2015, and are expected to exceed 1.5 trillion $ in 2018 (Statista (accessed on December 2, 2019)).

The demand pattern in e-commerce warehouses is characterized by many but small, time critical customer orders to process ((Boysen et al. 2019a)). Shifting toward e-commerce requires combining a wide range of planning problems in integrated approaches. The recent survey of Van Gils et al. (2018) classifies the main tactical and operational planning problems that occur in e-commerce warehousing and shows the correlation between these problems. In this study, we focus on the integration of 3 planning problems: regrouping the customer orders into pick lists (order batching), designing the tour that retrieves each pick list (picker routing), and assigning the picking lists to a set of pickers and scheduling the lists assigned to each picker (picker scheduling) in order to optimize a performance criterion.

A very large part of the picking literature assumes the integrity of the customer orders when performing the picking process, *i.e.* the orderlines of customer order have to be retrieved in a single tour. It is generally argued by the fact that splitting the customer orders over several batches will increase human errors or causes unacceptable packing efforts (Scholz et al. (2017), Valle et al. (2017), Henn and Schmid (2013)). Nonetheless, new sorting technologies and logistical techniques have emerged in the e-commerce warehouses making the order accumulation and packing procedure more efficient. For instance, some warehouses use fully automated conveyors that sort the orderlines according to the belt (Boysen et al. (2018)). This mechanism, although it is efficient, requires high investment costs. Other warehouses of business-to-customer (B2C) segment (Amazon Europe, Zalando) apply a manual consolidation and packing process based on the so-called "put walls" (Boysen et al. (2019b)). Figure 1 schematizes this process. A logistical worker, named "putter" receives orderlines from the picking area and stores each orderline in a dedicated shelf of the put wall. On the other side of the put wall, another logistical worker named "packer" retrieves each completed order, packs it, and sends it to the shipping area. A put-to-light mechanism is used to guide the putter and packer in their tasks. This system is particularly relevant in the e-commerce context since the characteristics of the orders (small sizes and tight shipping times) avoid the overload of the put wall.

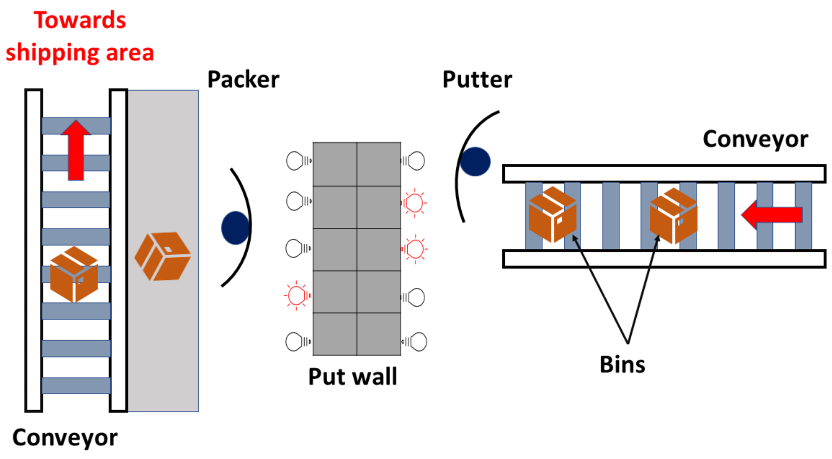

Fig. 1: Consolidation and packing schema using put walls (Boysen et al. (2019b))

We aim through this work to demonstrate the benefit of splitting the customer orders when organizing the picking process in e-commerce warehouses. We thus generalize the integrated order batching, batching routing, and picker scheduling problem of Van Gils et al. (2019) by allowing the splitting of customer orders and propose a route first-schedule second (RFSS) heuristic to solve the problem. In the computational experiments, we show that reductions up to 60% of the total order picking time are obtained by splitting the customer orders using our heuristic compared to the ILS of Van Gils et al. (2019) for the non-splitting version of the problem.

The remainder of this paper is organized as follows: in Section 2, we formally describe the orderline batching, batch routing, and picker scheduling problem. A MILP formulation of the problem is given in appendix. Section 3 comprises a literature review regarding the traditionnal and e-commerce warehouses. Section 4 presents a route first-schedule second heuristic to solve the problem. In Section 5, the numerical results of our heuristic are presented and compared with the results of Van Gils et al. (2019) to demonstrate the benefits of splitting the customer orders. Finally, section 6 provides conclusions and outlines research perspectives.

2 Problem Statement

We tackle an order picking problem in a warehouse of e-commerce segment characterized by:

- **A low-level picker-to-part system:** in which order pickers start from a central **depot**, travel through **picking** and **cross-aisles** to pick SKUs, and return to the depot. The SKUs are stored in shelves (storage locations) directly accessible to the pickers without using a fork truck. The cross-aisles divide the picking area into blocks and the part of a picking aisle located between two cross-aisles is named **subaisle**. Figure 2 sketches an example of a warehouse with a 2-block layout. The "pick locations" represent the stop-points from where the pickers retrieve the SKUs. A set of storage locations is associated with each pick location (red arrow in the figure 2). We assume that the aisles are large enough to ignore the effect of pickers blocking.
- **Time-critical picking orders:** Online retailers promise to customers short time frames between the order "click" and the "knock" on the door announcing its delivery. For instance, Amazon offers to its customers to deliver the orders requested online in the next day or even in the same day in some regions. Satisfying this promise is a main factor for the brand management of the retailer. We thus assume that deadlines are associated with the orders. The deadline of each order is assumed to be fixed in an upper stream level according to the departure time of the vehicle that delivers this order.
- **Small orders:** the orders in e-commerce segment consist of few orderlines. For instance, the average number of orderlines in the Amazon warehouse in Germany is 1.6 SKUs (Weidinger (2018)).
- **Mixed-shelves storage policy:** In e-commerce warehouses, a common and efficient practice is to break up unit-load of an SKU into small quantities that are scattered through multiple shelves in the warehouse. This storage policy is referred to as mixed-shelves storage policy. It increases the probability to find SKUs of a pick list close by irrespective of the position in the warehouse (Boysen et al. (2019a)). However, the picking location of each orderline has to be selected when assuming this storage policy. To keep the problem tractable, we assume in our problem definition that the picking location of each orderline is selected in an upperstream level (*i.e.* a fixed picking location is associated with each orderline such as storage constraints are satisfied).

Formally, we consider a complete and undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that models the warehouse layout, where $\mathcal{V} = \{0\} \cup \mathcal{L}$ is the node set and $\mathcal{E} = \{(i, j) \in \mathcal{E} : i, j \in \mathcal{V}, i < j\}$ the edge set. In $\mathcal{V}$, node 0 represents the depot while the set $\mathcal{L}$ represents the pick locations where the pickers stop to collect the SKUs around them. A non-negative weight $t_{i,j}$ is associated with each edge (i, j), and it represents the travel time between nodes i and j. We assume that the pickers travel at a constant speed, and hence the travel times are proportional to the travel distances.

Let $\mathcal{O}$ be the set of customer orders. Each order $o \in \mathcal{O}$ consists of few orderlines $\mathcal{M}_o$ that must be retrieved before a deadline d_o. An orderline m corresponds to a particular SKU and has two associated parameters: a weight q_m and a dedicated pick location $l_m \in \mathcal{L}$ from where to retrieve it.

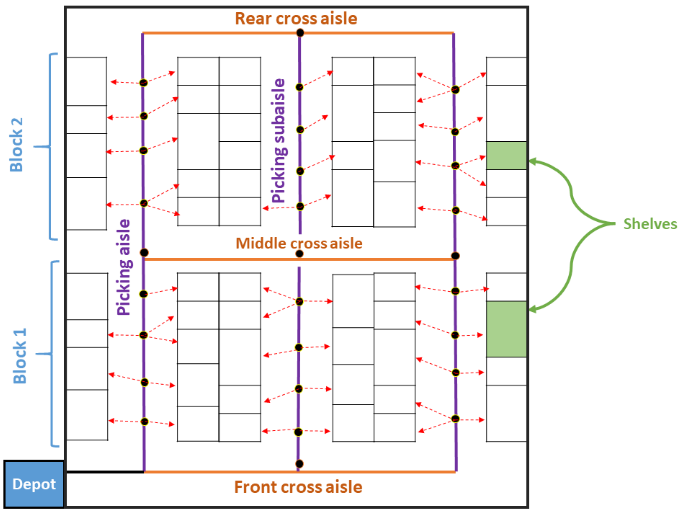

Fig. 2: Two blocks warehouse layout

The customer orders are processed by a set of homogeneous order pickers $\mathcal{K}$ equipped with carts of capacity W. Each picker performs picking tours in the warehouse starting and ending at the depot. For each tour v, a set of orderlines $\mathcal{M}_v$ is retrieved from their corresponding pick locations in a predefined sequence. The processing time p_v of a tour v is defined as the sum of:

- a fixed setup time β^s to prepare the tour;
- a fixed search and pick time β^p for each picked orderline;
- the travel time to walk between depot and the first pick location plus the travel time between the pick locations according to the sequence visit plus the travel time between the last pick location and depot.

We aim at constructing a picking plan $\Pi = \{\Pi_k\}_{\{k\in\mathcal{K}\}}$, where Π_k corresponds to a set of sequenced picking tours that must be performed by the picker k during his/her work-shift. Π must satisfy the following constraints:

1. Each order is completely picked before its deadline. Note that since the splitting of a customer order is allowed, the completion time of an order o equals the completion time of the last tour that processes an orderline in $\mathcal{M}_o$
2. The charge of each tour v, which is equal to the cumulative weight of its orderlines $\mathcal{M}_v$, does not exceed the cart capacity
3. The tours assigned to each picker do not overlap. Let consider v and v' two consecutive tours assigned to the picker k, with et_v the completion time of v and $st_{v'}$ the start time of v', then: $st_{v'} \geq et_v$
4. Each tour starts and ends at the depot

Finally, Π is an optimal planning schedule of our problem if it minimizes the total processing time (order picking time) function Z.

$$Z(\Pi) = \sum_{k\in\mathcal{K}} \sum_{v\in\Pi_k} p_v \tag{1}$$

3 litterature review

In traditional warehouses, few number of customer orders are processed daily and each one is made of a multiple number of orderlines and high quantities (Le-Duc and De Koster 2005). To optimize the picking operations in this context, the picking literature focuses on two main problems depending on the nature of the picking lists: When assuming a pick by order system, the arising optimization problem is named the picker routing problem and can be stated as follows: given a set of storage locations to visit in a pick list, what sequence of visits minimizes the total processing time. This problem is modeled in the literature as a TSP problem in a complete and symmetric graph in which the nodes represent depot and storage locations. The arc weights of the graph are fixed by computing the shortest travel time between each pair of nodes. The picker routing problem is also modeled as a steiner-TSP problem in a rectangular warehouse (see figure 4 of Scholz and Wäscher (2017)). A collection of exact and heuristic methods exists to solve this problem. Ratliff and Rosenthal (1983) introduce a fundamental theorem that limits the number of possible configurations for traversing an aisle in an optimal solution to 6 configurations. Furthermore, they propose a dynamic programming algorithm (DP) that solves the problem for a particular warehouse layout called one block layout warehouse. Their work served as a basis for several extentions in the picker routing litterature: In Roodbergen and De Koster (2001), an extention to warehouses with two blocks layout is proposed; In Masae et al. (2020a), the algorithm is adapted to support an arbitrary starting and ending point; In Öztürkoğlu and Hoser (2019), the problem based on warehouses with two blocks, parralel but non equal-sized subaisles is solved; Finaly, in Masae et al. (2020b), the DP of Ratliff and Rosenthal (1983) is extended to the so-called chevron warehouses which were proposed by Öztürkoğlu et al. (2012) as a design option for unit-load warehouses with single-command operations. Pansart et al. (2018) propose two exact methods to solve the picker routing problem: a steiner-TSP formulation reinforced with warehouse-based valid inequalities and an adaptation of a dynamic programming algorithm for solving the rectilinear-TSP inspired from Cambazard and Catusse (2018). They showed that the dynamic programming algorithm outperforms (in CPU time) the efficient TSP solver "CONCORDE" for instances with less than 11 blocks. In practice, the solutions returned by exact formulations can be rejected by managers since they produce "illogical" picking tours and hence difficult to memorize (Scholz and Wäscher (2017)). The pickers are more familiar with the so-called "routing strategies". Routing strategies are warehouse-based heuristics that model intuitive human practices such as: traversing entirely any aisle (except the last one) that contains at least one orderline to pick (S-shape strategy); entering and exiting each aisle that contains an orderline to pick from the same point (return strategy)... A comparative study between those methods and the efficient LKH (Lin-Kernighan-Helsgaun) heuristic for TSP problems can be found in Theys et al. (2010). The picker routing problem was extended by Löffler et al. (2018) for AGV-assisted order picking systems where a set of customer orders has to be picked in a single tour. The orders have to be sequenced and all the pick locations of an order o must be visited before processing the next order $o+1$ of the sequence. They propose an adaptation of the DP of Ratliff and Rosenthal (1983) to solve the special case when orders sequence is assumed to be fixed and imbed it within a greedy heuristic to solve the general problem.

When the customer orders are medium-sized and the pickers are equipped with relatively high-capacity carts, a pick by batch system in which orders are regrouped into batches is set up to decrease the processing times. The order batching problem (OBP) seeks to regroup a given set of orders into batches that satisfy a maximum capacity and construct the tour associated to each batch in order to minimize the total processing time. This problem is proven strongly NP-hard for more than two orders by batch (Gademann and Velde (2005)). The picking literature proposes a rich body of exact and heuristic methods to solve the OBP. The constructive heuristics can be classified into 3 categories: "priority heuristics" in which the orders are sorted according to a sorting rule and then assigned greedily to empty batches according to an assignment rule; "seed heuristics" in which each batch is first initialized with seed order, then filled according to order congruency rule; and "saving heuristics" in which the Clarke & Wright heuristic of Clarke and Wright (1964) for the VRP is adapted to the OBP. Henn et al. (2012) gives a detailed review of the several heuristics and metaheuristics developed for the OBP. Most of the existing work assumes a fixed routing strategy for computing the processing time of each batch. To study the potential reduction of total processing time that can be achieved by implementing more sophisticated routing methods, Scholz and Wäscher (2017) propose an iterated local search algorithm embedded with several heuristics to compute the tours. They conclude that average savings of 25% can be obtained when the tours are computed using near-optimal method.

In e-commerce warehouses, the 3 planning problems presented in the introduction section (order batching, picker routing and picker scheduling) have to be considered in an integrated fashion. In Henn (2015), due dates are associated with the customer order and the total tardiness of customer orders is minimized. To solve the problem, two metaheuristics based on variable neighborhood descent (VND) and Variable neighborhood search (VNS) starting with the same initial solution are proposed. The neighborhood structures embedded in the methods are classified in two groups: the first group changes the position of a complete batch or swaps two batches in the current solution and the second group moves a customer order to another batch or swaps two customer orders of two distinct batches. Experiments show that even with higher computational efforts, the VNS is outperformed by the VND. Furthermore, the neighborhood structures improve significantly (40% on average) the initial solution generated by a priority-rule based heuristic. In Scholz et al. (2017), the same problem is solved with a metaheuristic based on VND. Their work can be outlined in three main contributions. First, a new and efficient heuristic able to reduce the total processing time of the initial solution of Henn (2015) by up to 63%. Second, a new neighborhood that breaks up a complete batch and assigns its orders to other batches is introduced and represents the largest proportion in the total tardiness reduction. Finally, the tour that processes each batch is recomputed in each local optimum solution using the LKH-heuristic leading to a massive reduction of the total tardiness (up to 95%). From a managerial point of view, the customers satisfaction is the main performance factor in e-commerce retail. Managers in that field prefer to increase the labor insensitivity of the picking process (number of order pickers) instead of accepting solutions that may violate the shipping deadlines of some orders. Given this observation, Van Gils et al. (2019) introduce the integrated batching, routing and scheduling problem with hard time constraints and total processing time minimization. An

Iterative Local Search heuristic (ILS) is developed to solve the problem and applied to a spare part e-commerce warehouse. Experiment results report savings of 16.9% on average by using the ILS algorithm compared to the priority-based rule usually applied in the warehouse.

A limited number of articles in the literature allows the splitting of customer orders in the picking process. Among these exceptions, Bué et al. (2019) (resp. Briant et al. (2020)) proposes a heuristic (resp. exact formulation) for solving a variant of the order batching problem in a men's ready-to-wear warehouse named HappyChic. In their problem definition, capacitated boxes are used to pick the customer orders. Since the orders are large, they generally cannot be regrouped in a single box. The problem is thus to find which customer orders to split into a limited number of boxes and regroup the boxes into batches in order to minimize the total processing time. Note that due to safety considerations related to the warehouse layout, the processing time of each batch is computed using a fixed routing strategy. Furthermore, some managerial restrictions impose to consider a minimum volume constraint on boxes. Tsai et al. (2008) study an integrated order batching, batch routing, and picker scheduling problem with orders splitting. However, since they assume a single order picker, the assignment of batches to pickers is not considered. Their objective function to minimize is defined as a linear combination of the total processing time and the orders earliness and tardiness. To solve the problem, they propose a multiple-Genetic-Algorithm. Their computational experiments were devoted to study the impact of simultaneously considering the distinct terms of the objective function instead of considering them one by one. To the best of our knowledge, no approach in the litterature studies the impact of splitting the customer order in e-commerce warehouses where a set of small and time critical customer orders has to be picked by a set of order pickers operating simultanoeusly.

4 Route first-Schedule second heuristic

We propose in this section a matheuristic to tackle the problem. Algorithm 1 describes the general structure of our approach. It is composed of two phases: routing phase (lines 1-5) and scheduling phase (line 6). The routing phase regroups the customer orders into clusters and constructs a set of picking tours for each cluster of orders. The clusters are built using the procedure `constructClusters`($\mathcal{O}$, γ). This procedure assembles greedily the orders that have close deadlines. To build the picking tours associated with each cluster, we ignore the time constraints to obtain a variant of the order batching problem in which we allow the orders splitting when we construct the picking tours (batches). This problem is solved using the procedure `orderlinesBatching-SplitBasedProcedure`($\mathcal{O}^c$). This latter starts by computing a giant tour that starts from depot, retrieves all orderlines of the current cluster of orders, and comes back to depot. It then uses an adaptation of the split procedure by Prins (2004) to optimally extract a set Π^c of tours that satisfy the capacity constraints from the giant tour. It is worth noting that our method differs from the classical version of the split algorithm in the computation of the arcs' costs in the auxiliary graph as will be explained in Subsection 4.1.2. The tours generated for each cluster are added to the solution Π. After finding the tours of all clusters, the routing phase ends and the scheduling phase starts. In this phase, we model the picking tours in Π as jobs and the pickers in $\mathcal{K}$ as machines. Each job has a processing time (the processing time of its associated tour) and a deadline (the shortest deadline of the orders processed in its associated tour). We thus assign the picking tours in Π to the order pickers and schedule the tours assigned to each order picker solving a constraint programming model. Note that since the processing times of the jobs (*i.e.*, the constructed tours) are fixed during the routing phase, the scheduling problem does not have an objective function (it is a decision problem). In the remainder of this section, we describe the main algorithmic components.

Algorithm 1 Matheuristic: the general structure

```
Require: $\mathcal{G}, \mathcal{O}, \mathcal{K}, \gamma$
1: $\mathcal{C} \leftarrow$ constructClusters($\mathcal{O}$, $\gamma$)
2: for $\mathcal{O}^c \in \mathcal{C}$ do
3:     $\Pi^c \leftarrow$ orderlinesBatching-splitProcedure($\mathcal{O}^c$)
4:     $\Pi \leftarrow \Pi \cup \Pi^c$
5: end for
6: $s^* \leftarrow$ toursScheduling ($\Pi, \mathcal{K}$)
7: return $s^*$
```

4.1 Routing phase

4.1.1 Order clustering

Algorithm 2 gives the pseudo-code of the procedure `constructClusters`($\mathcal{O}$, γ). The procedure is based on a maximum capacity of a cluster $W^{cluster} = \gamma \cdot K \cdot W$, with $\gamma \geq 1$ being a parameter of the algorithm. First, the customer orders in $\mathcal{O}$ are sorted in increasing order of their deadlines. Next, the clusters are constructed greedily one at a time. Starting from an empty cluster, the procedure adds orders to the cluster while there is slack on the capacity constraint. If adding an order o violates the maximum cluster capacity, we close the current cluster and open a new empty cluster. The process is repeated until all customer orders are assigned to a cluster. Note that the value of parameter γ impacts the feasibility and the quality of the final solution. Indeed, setting γ to a large value leads to a small set of large clusters (*i.e.* clusters with a large number of orderlines). From such set of clusters, the solution to the orderline batching problems would probably contain better tours in terms of the objective function but may lead to an infeasible final solution. Indeed, since this resolution ignores the deadlines of the customer orders, the orderlines that have close deadlines may be spread across too many tours, compared to the number of pickers. Therefore, they may be impossible to schedule their picking before the deadline.

4.1.2 Orderline Batching

Let $\mathcal{M}^c$ be the set of orderlines of the customer orders in cluster c. The orderline batching problem seeks to regroup $\mathcal{M}^c$ into a set Π^c of batches (tours) that satisfies the cart capacity and minimizes the total processing time. To perform this task, the

Algorithm 2 `constructClusters`($\mathcal{O}$, γ)

```
1: $\mathcal{C} = \emptyset$, $w = 0$, $c = 0$, $\mathcal{O}^c = \emptyset$
2: sort $\mathcal{O}$ according to the earliest deadline rule.
3: for $o \leftarrow 1$ to $\mathcal{O}$ do
4:     while $(w \leq \gamma \cdot K \cdot W)$ do
5:         $\mathcal{O}^c \leftarrow \mathcal{O}^c \cup \{o\}$
6:         $w \leftarrow w + w_o$
7:     end while
8:     $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{O}^c$
9:     $c \leftarrow c + 1$
10:    $\mathcal{O}^c = \emptyset$
11:    $w \leftarrow 0$
12: end for
13: return $\mathcal{C}$
```

procedure `orderlinesBatching-SplitBasedProcedure`($\mathcal{O}^c$) starts by computing a giant tour that retrieves all the orderlines in $\mathcal{M}^c$ using the LKH, an efficient heuristic for the TSP. According to a comprehensive computational experiment conducted by Theys et al. (2010), LKH produces near-optimal solutions for the picker routing problem (*i.e.*, solutions with an average 0.1% deviation with respect to the optimal solution).

LKH returns a giant tour to pick all orderlines in $\mathcal{M}^c$, we denote m_i^c the orderline index at the i^{th} position in the giant tour ($\forall i \in \{1, \ldots, |\mathcal{M}^c|\}$ $m_i^c \in \mathcal{M}^c$). The procedure `orderlinesBatching-SplitBasedProcedure`($\mathcal{O}^c$) then aims to solve a specific shortest path problem in an auxiliary direct and acyclic graph $\mathcal{G}^c = \{\mathcal{V}^c, \mathcal{A}^c\}$ where $\mathcal{V}^c = \{m_0^c, m_1^c, \ldots, m_{|\mathcal{M}^c|}^c\}$ is the node set and $\mathcal{A}^c$ the arc set. The node $m_0^c \in \mathcal{V}^c$ is a dummy node while $\{m_1^c, m_2^c, \ldots, m_{|\mathcal{M}^c|}^c\}$ represents the orderlines of the giant tour. The arc set $\mathcal{A}^c$ is defined as $\mathcal{A}^c = \{(m_i^c, m_{i'}^c)\ \forall i, i' \in \mathcal{M}^c / i < i'\}$ and as each arc $(m_i^c, m_{i'}^c) \in \mathcal{A}^c$ models a feasible picking tour $v_{(i,i')}$ that starts from the depot, retrieves the orderlines $(m_{i+1}^c, m_{i+2}^c, \ldots, m_{i'}^c)$ and comes back to the depot. A cost $\tilde{p}_{(m_i^c, m_{i'}^c)}$ is associated with each arc $(m_i^c, m_{i'}^c)$, and it represents the processing time of the tour $v_{(m_i^c, m_{i'}^c)}$. The classic version of the split algorithm computes the travel time of the tour $v_{(m_i^c, m_{i'}^c)}$ by assuming that the nodes between m_{i+1}^c and $m_{i'}^c$ are visited according to their associated sequence in the graph (direct arc computation). To improve this version, we adopt a different approach. As per the observation of Theys et al. (2010), combining a warehouse-based routing strategy and the best improvement 2-opt operator leads to high-quality solutions and negligible computational times for the picker routing problem. We decided to reevaluate the travel time of each arc by using a combined heuristic + 2-opt operator procedure. Combined heuristic is a hybrid routing strategy that combines S-shape routing strategy and return routing strategy. An algorithmic description of the combined heuristic can be found in Roodbergen (2001). Note that we observed in preliminary tests that the travel time of a tour $v_{(m_i^c, m_{i'}^c)}$ returned by the direct arc computation may outperform the reevaluated travel time in very rare cases. Therefore, the procedure that computes the travel time of each tour $v_{(m_i^c, m_{i'}^c)}$ (noted `Compute-Travel-Time`($m_i^c, m_{i'}^c$)) selects the best value between the direct arc computation version and the reevaluated version.

Algorithm 3 `orderlinesBatching-SplitBasedProcedure`$(\mathcal{O}^c)$

```
1: $\mathcal{V}^c \leftarrow$ compute giant tour using the LKH heuristic
2: $\tilde{t}_0 = 0$
3: for $i \leftarrow 1$ to $|\mathcal{V}^c|$ do
4:     $\tilde{t}_{m_i^c} = \infty$
5:     $P_{m_i^c} = -1$
6: end for
7: for ($i \leftarrow 0$ to $|\mathcal{V}^c| - 1$) do
8:     $i' \leftarrow i + 1$
9:     $load \leftarrow 0$
10:    while ($i' \leq |\mathcal{V}^c|$ and $load + w_{m_{i'}^c} \leq W$) do
11:        $load = load + w_{m_{i'}^c}$
12:        $\tilde{p}_{(m_i^c, m_{i'}^c)} \leftarrow \beta^s + \beta^p \cdot (i' - i) +$ Compute-Travel-Time$(\mathcal{G}^c, m_i^c, m_{i'}^c)$
13:        if $\tilde{t}_{m_i^c} + \tilde{p}_{(m_i^c, m_{i'}^c)} < \tilde{t}_{i'}$ then
14:            $\tilde{t}_{m_{i'}^c} = \tilde{t}_{m_i^c} + \tilde{p}_{(m_i^c, m_{i'}^c)}$
15:            $P_{m_{i'}^c} = m_i^c$
16:        end if
17:        $i' = i' + 1$
18:    end while
19: end for
20: Construct $\Pi^c$ from the labels
21: return $\Pi^c$
```

To find the optimal splitting of the giant tour into several feasible tours, the split procedure aims to find the shortest path from m_0^c to $m_{|\mathcal{M}^c|}^c$ in $\mathcal{G}^c$. Algorithm 3 describes the whole procedure based on a label setting algorithm. A label $(\tilde{t}_{m_i^c}, P_{m_i^c})$ is associated to each node m_i^c, $\tilde{t}_{m_i^c}$ is the value of a path from m_0^c to m_i^c and $P_{m_i^c}$ is the predecessor node from which $\tilde{t}_{m_i^c}$ has been determined. After computing the giant tour, the labels are initialized. Then two nested loops update the labels. The main loop traverses the nodes from m_0^c to $m_{|\mathcal{M}^c|-1}^c$. At each iteration i of the main loop, the inner loop explores all the arcs (the routes) that share the same tail node m_i^c and satisfy the cart capacity. For each feasible arc $(m_i^c, m_{i'}^c)$, $\tilde{p}_{(m_i^c, m_{i'}^c)}$ is computed and the label of the head node is updated using the bellman optimality principle. The picking tours in Π^c are finally constructed by a backward pass through the predecessor labels and starting by the label $(\tilde{t}_{m_{|\mathcal{M}^c|}^c}, P_{m_{|\mathcal{M}^c|}^c})$.

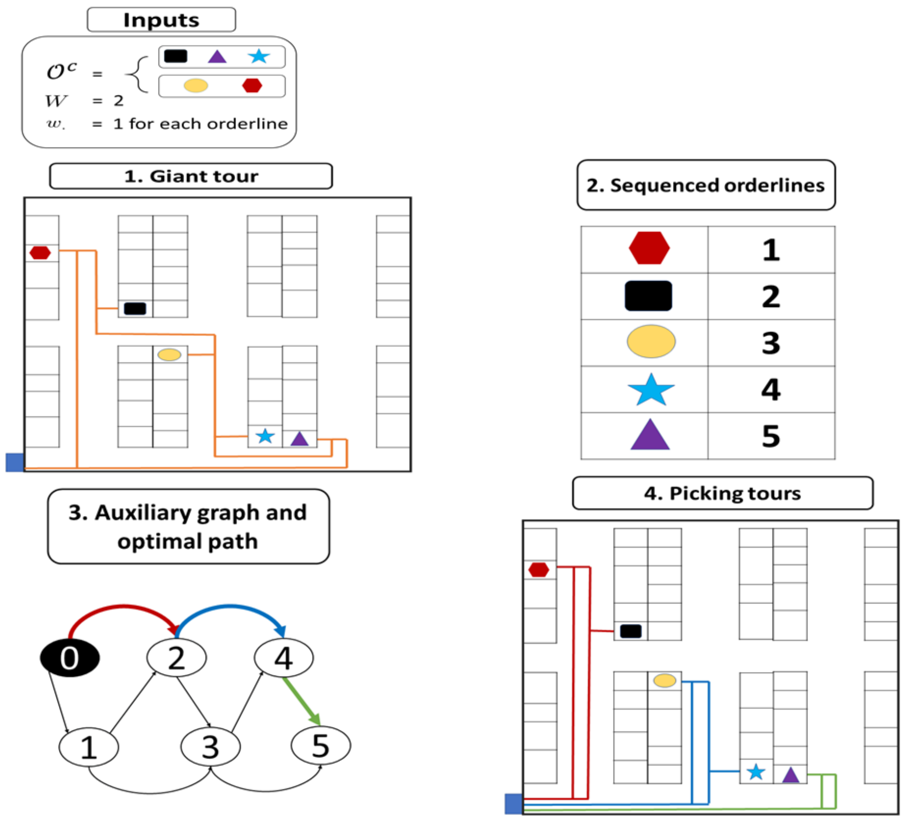

Fig. 3: Split-based procedure: graphical example

Figure (3) depicts an example of the splitting procedure for a cluster of orders composed of 2 orders, the first one consists of 3 orderlines and the second one consists of 2 orderlines. The cart capacity is 2 orderlines. The path represented by the orange lines shows the optimal TSP-like tour to retrieve all the orderlines. Furthermore, the red, blue, and green paths show the optimal picking tours associated with the red, blue, and green arcs of the optimal path in the auxiliary graph.

4.2 Scheduling phase

The second phase problem is a variant of the identical parallel machine scheduling problem with deadlines and without objective function (decision problem). Each tour $v \in \Pi$ represents a job to schedule on one of the K machines (the pickers). To tackle the problem, we adopt a constraint programming (CP) approach. Constraint programming has been applied successfully to a variety of scheduling problems such as: manufacturing, computer and network scheduling, transportation,etc.(Laborie 2018; Ham and Cakici 2016; Baptiste et al. 2012; Hooker and van Hoeve 2018).

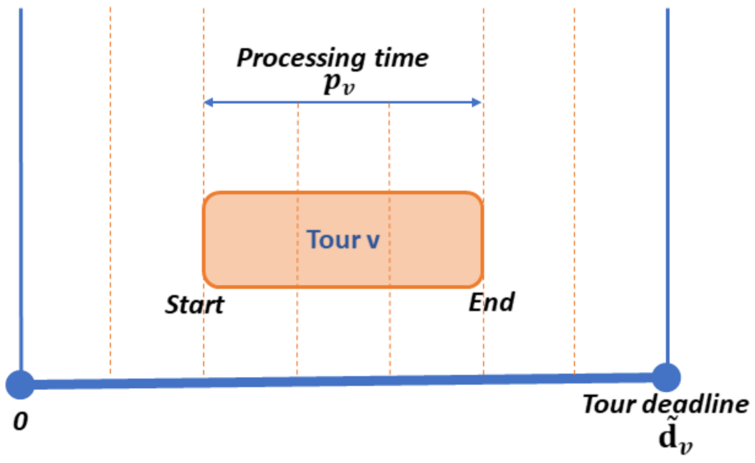

Fig. 4: Representation of interval variable x_v

For the sake of brevity we do not present our CP formulation here, but the interested reader can find it on Appendix C. To implement and solve our model, we exploit the scheduling capabilities of CP Optimizer (CPO). In particular, we use interval variables, sequence variables, and global constraints. The CP model re-written using CPO objects reads:

Parameters	
$\tilde{d}_v$	Deadline of a tour v ($\tilde{d}_v = \min_{o \in \mathcal{O}_v} d_o$)
Variables	
$y_{k,v}$	Optional interval variable when tour v is assigned to picker k with duration p_v and domain $\{[0, p_v), [1, p_v + 1), \ldots, [\tilde{d}_v - p_v, \tilde{d}_v)\}$
x_v	Interval variable associated with the tour v (not optional)
$Z_k = \{y_{k,1}, \ldots, y_{k,\lvert\Pi\rvert}\}$	Set of optional interval variables (sequence variable) that models the sequenced tours of picker k

Table 1: Notations used in the CP formulation

$$\texttt{Alternative}(x_v, \{y_{1,v}, \ldots, y_{K,v}\}) \qquad \forall v \in \Pi \qquad (2)$$

$$\texttt{NoOverlap}(Z_k) \qquad \forall k \in \mathcal{K} \qquad (3)$$

Interval variables are a powerful tool for modelling scheduling problems. They embed several attributes of a job such as start time, end time, and processing time in a single decision variable (Laborie et al. (2018)). Figure (4) depicts a representation of an interval variable x_v for a picking tour v. The processing time of tour v defines the size of its associated interval variable. The time interval (start time, completion time) of tour v is fixed by assigning values to the attributes "start" and "end" of the interval variable. Since the value "end" is bounded by the tour deadline, the domain of an interval variable associated with tour v is defined by the discrete set $\{[0, p_v), [1, p_v + 1) \ldots, [\tilde{d}_v - p_v, \tilde{d}_v)\}$. One feature of an interval variable is that it can be optional. In this case, an empty decision is added to its domain to model the case where the interval variable is absent in the schedule. We use this feature to model the assignment of picking tour v to picker k (variables $y_{k,v}$). We also use sequence variable Z_k to order the interval variables assigned to each picker k.

Constraints (2) are alternative constraints. They force each tour v to be assigned to exactly one picker. It works as follows: for a given v, if x_v is present (which is always the case since x_v is not optional) then exactly one of the elements of $\{y_{1,v}, \ldots, y_{K,v}\}$ will be present in the final schedule. Constraints (3) prevent the overlapping of interval variables in each sequence variable $\mathbf{Z_k}$. If the problem is unfeasible then no solution is returned.

5 Computational experiments

The computational experiments aim to show the benefits of splitting the customer orders when building the picking routes. We thus test our heuristic (RFSS) over the data set used in Van Gils et al. (2019) and compare our results with the results of their ILS heuristic. The instances are described in section 5.1. Then, the benefits of recomputing the arc costs of the split graph is showed in section 5.3. Next, parameter tuning and sensitivity analysis are discussed in section 5.2. Finally, the benefits of splitting the customer orders is shown in section 5.4.

All computations were performed on a 64-bit laptop equipped with an intel core i7-8550U CPU (1.80 GHz), 16 gigabytes main memory, running on Windows 10. Our RFSS heuristic is coded in C++ (visual studio 2019), using IBM ILOG CP Optimizer 12.8 to solve the constraint programming problems. After preliminary experiments, we fixed a time limit of 30 s for the constraint programming solver and assume that our algorithm returns infeasible solution if the constraint programming solver reaches the time limit without finding a feasible schedule of the picking tours. Finally, note that the computations in Van Gils et al. (2019) were performed on an Intel Xeon Processor $E5-2680$ (2.8 GHz) using a single thread.

5.1 Test problem instances

The data set generated by Van Gils et al. (2019) is referred to as `Set-VG`. The data represent a 2-block warehouse layout. Storage locations are homogeneous and assumed to be 1.3 m long and 0.9 m wide. Each picking aisle is 3 m wide. The distance between two adjacent picking aisles is thus set to 4.8 m. The depot is located in the front of the leftmost aisle and the distance between the depot and the closest picking point is 15.65 m (12 m + 3.65 m). The two blocks are separated by 6 m. Figure 5 sketches out all these metrics. Each picker travels at a constant speed of 1 $m \cdot s^{-1}$ and needs 10 s to identify and pick each orderline. Furthermore, 180 s are required to prepare each tour.

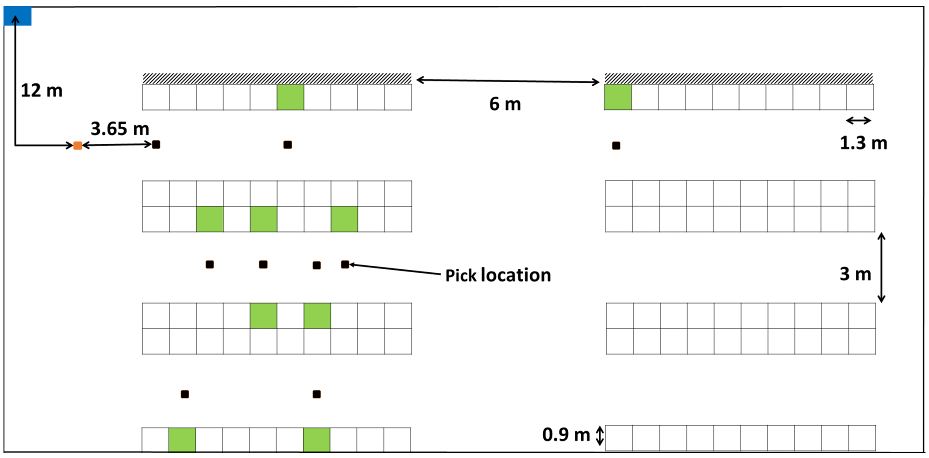

Fig. 5: Warehouse floor in `Set-VG`

In order to cover multiple warehouses scenarios, six factors (warehouse layout, storage policy, cart capacity, order structure, deadline distribution) have been varied during the instances generation process, with three levels for each factor. For the warehouse layout, the number of subaisles and storage locations per subaisle is varied resulting in small warehouses (SW) with 12 subaisles and 60 storage locations in each subaisle, medium warehouses (MW) with 24 subaisles and 120 storage locations in each subaisle, and large warehouses (LW) with 36 subaisles and 180 storage locations in each subaisle. The assignment of SKUs to storage locations is done using random (Rand), within-aisle (WA), and across-aisle storage (AA) policies. WA and AA policies are class-based storage policies where the picking area is divided into 3 classes (A,B,C). Class A includes $\frac{1}{6}$ of all SKUs with the highest order frequency (60%) whereas classes B and C include $\frac{1}{3}$ and $\frac{1}{2}$ of SKUs with order frequencies of 30% and 10% respectively. All the orderlines are assumed to have the same weight. The cart capacity is limited to 15, 30, or 45 orderlines, depending on the instance. Similarly, the number of orders is fixed to 100, 200 or 300. For each order o, the number of orderlines $|\mathcal{M}_o|$ is computed using the following formula: $min(W, \lfloor Exp(\beta) + 0.5 \rfloor)$, with $Exp(\beta)$ an exponential distribution with mean $\beta = \frac{8}{3}$ for 300 orders, $\beta = 4$ for 200 orders, and $\beta = 8$ for 100 orders. The planning horizon is set to 4 h and the deadlines are generated using a uniform distribution (Uni), a progressive distribution (Prog), and a degressive distribution (Deg). Using the progressive (resp. degressive) distribution, the highest proportion of orders has to be retrieved at the beginning (resp. at the end) of the planning horizon. A combination of the above-mentioned parameters results in 243 problem classes. 30 instances are generated for each problem class resulting in 7290 single instances. The number of pickers varies between instances and is fixed using the following regression equation: $K^{(i)} = \lceil 1.20(0.254M + 0.006OL + 0.072W + 1.383Deg) \rceil$ with M the number of aisles, OL the total number of orderlines of instance (i), W the capacity of the cart, and Deg representing the degressive distribution. The coefficients of the equation are obtained using a regression analysis on a set of 30 instances randomly selected from `test-VG`. Note that `test-VG` is a set of 243 generated test instances, each one of them corresponds to a distinct problem class. The number of pickers $K^{(i)}$ of each selected instance (i) is computed as the smallest value of K that enables the ILS heuristic of Van Gils et al. (2019) to return a feasible solution in terms of the the deadline constraints (*i.e.* for $K^{(i)} - 1$, the ILS heursitc of Van Gils et al. (2019) returns a solution with positive tardiness). Note that the number of pickers of each instance obtained by the regression equation is increased by 20% to ensure a feasible solution for each instance (*i.e.* enough pickers to pick all orderlines in time). For more details about the data set generation, please refer to Van Gils et al. (2019). The benchmark is publicly available at `https://www.uhasselt.be/Datasets-and-results`.

5.2 Parameter tunning and sensitivity analysis

We study in this section the impact of parameter γ on the solution feasibility, the solution quality, and the computational time of our RFSS heuristic. Recall that since the number of pickers for each instance is fixed, the maximum weight of a cluster of orders is proportional to γ (see section 4.1.2). For instance, assume that $\gamma = 3$. If we assume that the tours constructed from a cluster of orders should be executed in a relatively small time window and the workload is uniformly balanced between the pickers, then each cluster (except the last one) contains enough orders for constructing at least the three next tours of each picker. Firstly, we execute the RFSS heuristic over `Set-VG` data set by setting γ to the following values $(1.5, 1.8, 2.0, 2.5, 2.8, 3.0, 4.0, \mathit{best})$, where the value *best* corresponds to a parallel multi-start version of our heuristic in which we run our algorithm for each $\gamma \in \{1.5, 1.8, 2.0, 2.5, 2.8, 3.0, 4.0\}$ and store the best feasible solution found. Table 2 reports the total number of unsolved instances for each parameter value and Figure 6 gives the proportion of best solutions found for each $\gamma \in \{1.5, 1.8, 2.0, 2.5, 2.8, 3.0, 4.0\}$. Second, we remove all the instances for which the algorithm returns an infeasible solution for a given value of γ (1515) and study the average order picking time and CPU time for each parameter value on the remaining instances (5775). The main results are reported in Figure 7.

γ	# of unsolved instances
1.5	0
1.8	0
2.0	0
2.5	1
2.8	4
3.0	56
4.0	1514
best	0

Table 2: Number of unsolved instances per parameter value

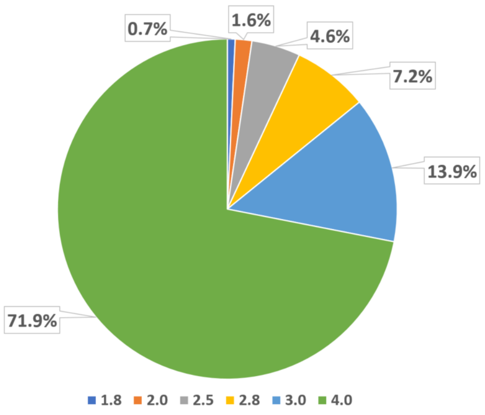

Fig. 6: Distribution of best solutions per parameter value

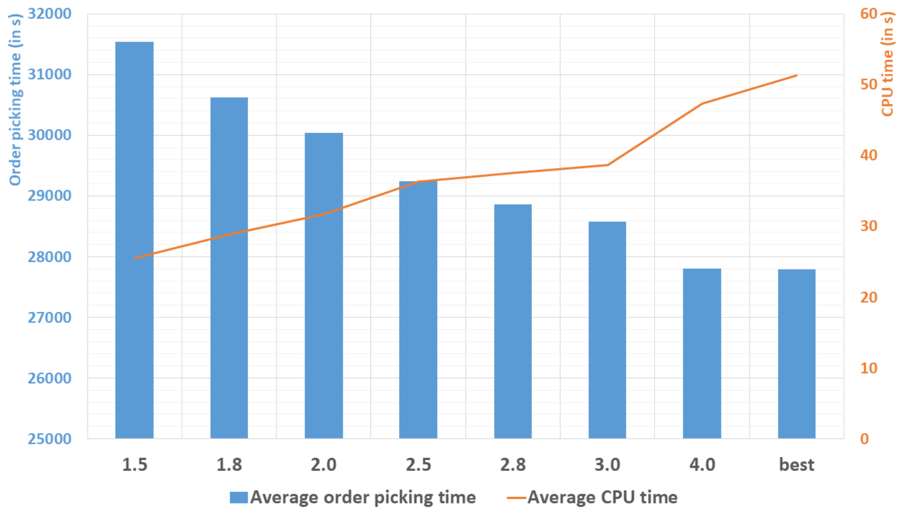

Fig. 7: Impact of parameter γ on the performance of the RFSS heuristic on the selected set of instances

From table 2, we observe that setting $\gamma \leq 2$ leads to feasible solutions for all the 7290 instances. Infeasible solutions start to appear when $\gamma = 2.5$ and the number of infeasible instances increases exponentially with the increase of γ. Those results were expected, since the batching procedure ignores the order deadlines when generating the picking tours. Thus, short-but-incoherent picking tours are produced when the clusters are too large. It follows that the constraint programming procedure is unable to produce a feasible final solution with those tours. However, we observe that our algorithm was able to find feasible solutions for almost all instances when $2 < \gamma \leq 3$ and for 79.2% of instances even when $\gamma = 4$. Hence, we conclude that even by assembling the orders into few-but-large clusters and ignoring the deadlines during the construction of the picking tours, our algorithm is able to return a feasible (and high-quality) solution for a considerable proportion of instances. This result gives an indication on how splitting the customer orders reduces the effect of the order deadline constraints during the picking process.

Figure 7 reveals that increasing γ leads to a reduction in the average order picking time. This result can be explained by the fact that the procedure `constructClusters`$(\mathcal{O}, \gamma)$ produces few-but-large clusters when γ is set to high values. Consequently, the search space of the orderline batching problem is increased, leading to the generation of more efficient picking tours. This observation is confirmed by the results in figure 6 in which we observe that 97.7% of the best solutions are found when $\gamma > 2$, 71.9% when $\gamma = 4$. Focusing on execution times, we observe a linear increase in the computational time with the increase of γ. This result is not surprising, since the highest portion of the computational time is consumed by the LKH heuristic employed to build the giant tours of each cluster. Larger clusters therefore result more complex TSP problems to solve during this phase of the algorithm.

Given the previous observations, we conclude that setting γ to higher values results in better solutions (on average) but also a higher risk of finding infeasible solutions. To extract a feasible solution with a high quality, we decided to set $\gamma = best$ (parallel multi-start version) which represents the best trade-off between the solution feasibility and the solution quality. Furthermore, we can observe from figure 7 that the increase of the average computational time of RFSS when setting $\gamma = best$ is negligible compared to the average computational time of RFSS when $\gamma = 4$. Thus, in the remainder of the manuscript, we use the parallel multi-start version of our algorithm to prove the benefits of splitting the customer orders. We refer to this version of our algorithm as $RFSS^{+}$.

5.3 Classic versus improved split

In this section, we show the impact of recomputing the cost of each arc of the split graph on the feasibility and the quality of the final solution. We thus execute our algorithm using the classic version of the split algorithm and compare it with the improved version. For the sake of brevity, we refer to the version of our algorithm that uses the classic split as $RFSS_c^{+}$ and keep $RFSS^{+}$ for the version that uses the improved split. Table 3 presents the main results. It reports the average picking time, the average CPU time for each version, and the savings indicators (mean, max.) for each factor level. Note that if we note $cost_{CV}([.])$ the order picking time returned by $RFSS_c^{+}$ for instance [.] and $cost_{IV}([.])$ the order picking time returned by $RFSS^{+}$ for the same instance, then the saving $\varDelta_{CV|IV}([.])$ is computed with the following formula:

$$\varDelta_{CV|IV} = \frac{cost_{CV}([.]) - cost_{IV}([.])}{cost_{CV}([.])} \tag{4}$$

Globally, we observe that recomputing the arc costs of the split graph improves the total picking time by 1.2% on average, with a maximal improvement of 13%. Notice that in very few cases (19/7290 instances), the total order picking time

	$RFSS_c^+$		$RFSS^+$		$\Delta_{CV\|IV}$	
	cost(s)	time(s)	cost(s)	time(s)	mean	max
Layout						
SW	21387.8	29.2	21327.5	34.8	0.3%	1.8%
MW	27303.7	55.4	26986.5	55.8	1.3%	9.0%
LW	33868.5	82.2	33231.1	77.8	2.0%	13.0%
Storage policy						
AA	26925.5	44.4	26774.4	85.5	0.6%	11.5%
Ran	28690.7	38.5	28172.2	43.6	1.8%	13.0%
WA	26943.8	83.9	26598.6	39.3	1.2%	12.0%
Cart capacity						
15	36674.0	31.3	36382.8	37.4	0.7%	3.8%
30	25104.2	55.4	24732.4	58.6	1.3%	9.9%
45	20781.8	80.1	20429.9	72.4	1.5%	13.0%
Order structure						
100	24770.0	61.5	24436.4	50.0	1.3%	12.7%
200	27999.4	56.5	27664.7	56.2	1.2%	13.0%
300	29790.6	48.8	29444.0	62.2	1.1%	12.5%
Deadline distribution						
Deg	27254.7	56.9	26952.8	59.8	1.1%	12.7%
Prog	27532.2	49.6	27202.6	56.7	1.2%	12.0%
Uni	27773.1	60.2	27389.8	52.0	1.3%	13.0%
All instances	27520.0	55.6	27181.7	56.1	1.2%	13.0%

Table 3: Comparaison between the performance of $RFSS_c^+$ and $RFSS^+$.

returned by $RFSS_c^+$ is better than the one returned by $RFSS^+$. This is due to the fact that for those instances, the solution returned by $RFSS_c^+$ and the one returned by $RFSS^+$ do not correspond to the same value of γ. Indeed, using the improved version of split results in no feasible solutions when setting γ to large value. Thus, the solution returned by the $RFSS^+$ corresponds to a medium or small value of γ. On the other hand, $RFSS_c^+$ returns feasible solutions when setting γ to large value for those instances.

If we analyze the improvement indicators (mean, max.) per factor, we observe that the impact of recomputing the cost of each arc of the split graph is more significant on the instances with a large warehouse layout, for the instances with a random storage assignment policy, and for the instances with a cart capacity of 45 orderlines, which globally represent the conditions of e-commerce warehouses. By restricting the computations for the instances of large warehouses with a random storage assignment policy and a cart capacity of 45 orderlines (270 instances), we found that the mean gap increases to 3.7%. For the order structure and the deadline, we do not observe a statistically significant correlation between the factor levels and the improvement indicators. Concerning the CPU time, we observe that recomputing the arc costs of the split graph does not generate a significant computational overhead. For some instances, we found that $RFSS^+$ outperforms $RFSS_c^+$ in term of CPU time. Indeed, for each of those instances, the constraint programming solver quickly finds a feasible solution for each value of $\gamma \in \{1.5, 1.8, 2.0, 2.5, 2.8, 3.0, 4.0\}$ when running $RFSS^+$ contrary to $RFSS_c^+$ where the constraint programming solver reaches the time limit without finding a feasible solution for some values of γ.

5.4 The benefits of splitting the customer orders

		ILS			$RFSS^{+}$			$\Delta_{ILS\|RFSS^{+}}$		
		cost(s)	time(s)	T	cost(s)	time(s)	T	mean	min	max
Layout										
	SW	26283.1	118.4	36	21327.5	34.8	36	19%	8%	30%
	ML	39348.8	125.4	36	26986.5	55.8	37	32%	19%	50%
	LW	53683.0	127.1	37	33231.1	77.8	37	38%	23%	60%
Storage policy										
	AA	37901.6	139.5	36	26774.4	85.5	37	28%	13%	51%
	Ran	43788.9	117.0	37	28172.2	43.6	37	34%	14%	60%
	WA	37624.4	114.3	36	26598.6	39.3	37	27%	8%	50%
Cart capacity										
	15	50681.0	69.9	58	36382.8	37.4	58	27%	12%	44%
	30	37246.1	127.5	31	24732.4	58.6	31	31%	11%	54%
	45	31387.7	173.4	21	20429.9	72.4	21	32%	8%	60%
Order structure										
	100	38466.9	55.1	32	24436.4	50.0	33	34%	13%	60%
	200	40346.2	113.7	37	27664.7	56.2	37	29%	11%	54%
	300	40501.8	202.1	40	29444.0	62.2	40	26%	8%	48%
Deadline distribution										
	Deg	39891.5	116.0	37	26952.8	59.8	37	30%	12%	60%
	Prog	39471.5	140.6	36	27202.6	56.7	37	29%	8%	57%
	Uni	39951.8	114.2	37	27389.8	52.0	37	29%	9%	57%
All instances		39771.6	123.6	36	27181.7	56.1	37	30%	8%	60%

Table 4: Comparison between the performance of $RFSS^{+}$ and ILS.

To evaluate the potential benefits of splitting the customer orders during the picking process, we compare the solution returned by $RFSS^{+}$ and the solution returned by the state-of-the-art ILS of Van Gils et al. (2019) for the non-splitting version. Table 4 summarizes the main results. The first part of the table includes the average order picking time (cost), the average computational time (time), and the average number of created tours (T) for each method and factor level. Note that the ILS of Van Gils et al. (2019) is referred to simply as ILS. The second part of table 4 includes the mean savings, the min. savings, and the max. savings obtained by comparing the order picking time returned by ILS and the order picking time returned by $RFSS^{+}$. Note that if we denote $cost_{ILS}([.])$ the order picking time returned by ILS for instance [.] and $cost_{RFSS^{+}}([.])$ the order picking time returned by $RFSS^{+}$, then the saving $\Delta_{ILS|RFSS^{+}}([.])$ obtained for this instance is computed with the following formula:

$$\Delta_{ILS|RFSS^{+}} = \frac{cost_{ILS}([.]) - cost_{RFSS^{+}}([.])}{cost_{ILS}([.])} \tag{5}$$

Aditionally, non-parametric tests (test of Kruskal-Wallis, test of Dunn) are performed to analyse the effect of warehouse factors on the order picking time, on the CPU time, and on the gap $\Delta_{ILS|RFSS^{+}}$. The main results are provided in appendix B.

5.4.1 Reduction of order picking time

We observe from table 4 that splitting the customer orders during the picking process using our algorithm leads to a massive reduction of the order picking time. Indeed, $RFSS^{+}$ improves the solution without splitting of ILS by 30% (on average) with a minimal and maximal improvement of 8% and 60%. In term of workload, the order picking time reduction can reach up to 11.5 h by splitting the customer orders and it equals approximately 6 h on average. If we normalize the results by the number of pickers, we find that the picking time of each picker can be reduced by more than 1 h (on average) with a maximum reduction of 1.8 h (for a planning horizon of 4 h). In practice and compared to the scenario where the splitting order is not allowed, even if splitting orders implies an additional time to gather all the items of a same order before finalize the preparation of the order, the gain in the order picking time is large enough to justify considering it in practice.

We observe that the reduction in order picking time is significantly higher for medium and large warehouses compared to small warehouses. It is probably due to the fact that small warehouses contain fewer storage locations leading to a higher probability of finding orderlines of distinct orders stored in the same (or in a nearby) storage locations. Hence, the effect of splitting the customer orders is mitigated. For the storage policy, we observe that splitting the customer orders results on larger benefits when the SKUs are scattered in the warehouse (random assignment policy), which is the case of B2C e-commerce warehouses (Boysen et al. 2019a). This result seems to be natural. Indeed, when we prohibit the splitting of the customer orders, we are forced to retrieve all the orderlines of a subset of customer order (batch of orders) in a single tour. Due to the random storage assignment policy, those orderlines are potentially stored in remote locations leading to long picking tours. Given a set of order batches, we can considerably reduce the picking time of those orders when allowing the splitting of customer orders by decomposing the order batches and assembling the nearby located orderlines of each batch in new picking tours. For the cart capacity, we observe a small growth on average savings with the increase of the cart capacity. An opposite effect is observed for the order structure where increasing the number of orders leads to a small reduction in order picking time. For the deadline distribution, no significant correlation between each factor level and its associated indicators (min., max., mean) can be observed. Globally, we can conclude that splitting the customer orders offers the best results for the configuration that corresponds to realistic e-commerce warehouses (large warehouse, random storage assignment policy, cart capacity of 45 orderlines). By restricting the analysis to instances with this configuration, the average reduction in order picking time reaches 49%.

From table 4, we also notice that $RFSS^{+}$ does not reduce the number of picking tours with respect to ILS. Given this observation, we conclude that the savings reported do not result from a better use of the cart capacity when splitting the customer orders. Thus, the reduction in order picking time is essentially caused by the ability to split up a customer order over all the pickers and assemble closely located orderlines in a single tour leading to short picking tours.

5.4.2 Computation time analysis

We observe that although the complexity of our problem (the splitting of customer orders is allowed) is higher than the complexity of the problem introduced in Van Gils et al. (2019) (the splitting of customer orders is avoided), $RFSS^+$ is considerably faster than ILS (56 s on average versus 123 s on average). Note that this observation does not prove that our heuristic is more efficient than ILS since the problems considered are different. However, we can conclude that the significant savings in order picking time reported in table 4 do not come from higher computation efforts. Note also that in the implementation of our heuristic, we call the procedure `orderlinesBatching-SplitBasedProcedure`($\mathcal{O}^c$) for each cluster $\mathcal{O}^c$ sequentially. Since the clusters $\mathcal{O}^c \in \mathcal{C}$ are independent, significant reductions of the computational time could be obtained by calling the `orderlinesBatching-SplitBasedProcedure`($\mathcal{O}^c$) for each cluster in parallel.

The results show that the computational effort of our heuristic strongly depends on the size of the warehouse. the average CPU time increases from 34.8 s for small warehouses to 77.6 s for large warehouses. A same (but less significant) correlation is observed for the ILS algorithm (from 118.4 s to 127.1 s). For both algorithms, increasing the number of aisles and storage locations reduces the so-called "similarity of customer orders" (*i.e.* the probability of finding orderlines of multiple orders in the same pick location) which makes the total number of pick locations in the warehouse higher and hence the TSPs more complex. For the storage assignment policy, we observe that the average computation time on instances that follow the "across aisles" storage policy (85.5 s) is considerably higher than the average computation time on instances that follow the "random" (43.6 s) and the "within aisle" (39.1 s) storage policies. The difference is also less significant for the ILS algorithm (139.5 s for "across aisles" storage policy and (114.3 s, 117.0 s) for the two other storage policies). We also observe a direct and positive correlation between the cart capacity and the average computation time. The increase in execution time is caused by the computation of the routing phase. Indeed, increasing the cart capacity leads to large clusters and hence, the computation time to execute the split procedure becomes higher. Note that the same correlation between the cart capacity and the execution time of the ILS algorithm is observed.

Finally, we observe that increasing the number of orders does not cause a significant increase in the average execution time unlike the ILS algorithm where the increase of orders number leads to much higher computational efforts. This result can be explained by the fact that the complexity of our algorithm is not related to the number of customer orders rather than the total number of orderlines to pick. The number of orderlines per order decreases with the growth of the number of orders (see section 5.1) which makes the growth in the total number of orderlines stable. This is not the case for the ILS algorithm where increasing the number of customer orders leads to an increase of the search space of the algorithm as explained in Van Gils et al. (2019). For the deadline distribution, the average execution times are uniformly distributed between the factor levels.

6 Conclusion

In this paper, we study the potential benefit of splitting the customer orders into orderlines during the picking process for warehouses that process a large number of small and time-critical orders (e-commerce warehouses). For this purpose, we extend the integrated batching, routing, and picker scheduling problem by allowing the orders to be splitted over several tours. The problem consists in determining a set of picking tours by regrouping orderlines, assigning the tours to a set of order pickers, and scheduling the tours assigned to each picker such that the orders deadlines are satisfied and the total processing time is minimized.

To solve the problem, we propose a route first-schedule second heuristic. In the routing phase, we divide the set of orders into clusters and apply a modified version of the split algorithm to determine the tours to retrieve the orderlines of each cluster. In the scheduling phase, we build a feasible scheduling of the picking tours over the order pickers by solving a constraint programming problem.

To assess the benefits of splitting the customer orders, we tested our method on a data set of 2970 instances with different orders characteristics and different warehouse environments generated by Van Gils et al. (2019). We compared the total order picking time of the solutions delivered by our heuristic with that of the solutions retrieved by the ILS of Van Gils et al. (2019). We found that by allowing order splitting, our heuristic delivers a massive reduction of 30% on average and 60% in the best case. For a planning horizon of 4 h, it represents more than a 1 h reduction of the shift of each picker on average with a maximum reduction of 1.8 h.

Our model assumes that the picking operations and the packing operations (*i.e.* assemble the picked orderlines into final customer orders ready to be shipped) are planned sequentially. Since these operations appear to be correlated, further research should focus on extending our model to an integrated version that considers the picking and packing operations simultaneously. It would be worthwhile to model the several sorting systems existing in the e-commerce warehouses and compare them. Furthermore, we assume in our problem definition that the aisles are wide enought to safely neglect the effect of blocking on the performance of the picking process. This assumption may be too strong for small warehouses that are implanted in cities as a consequence of the growth of e-commerce caused by the COVID pandemic. One interessting reseach perspective would be to extend our model by considering the picker blocking as a hard constraint or a performance criterion.

Appendix A. Mathematical formulation

We propose a mixed-integer linear formulation (MILP) to model the problem. The formulation is defined on a new directed graph $\mathcal{G}' = (\mathcal{V}', \mathcal{A}')$. The node set $\mathcal{V}' = \{0, n+1\} \cup \mathcal{L}$ contains two copies of the depot ($\{0, n+1\}$) in addition to the pick locations set $\mathcal{L}$. The arc set $\mathcal{A}'$ is composed of arcs from 0 to the nodes in $\mathcal{V}' - \{0\}$, arcs from the nodes in $\mathcal{V}' - \{0, n+1\}$ to $n+1$, and two directed arcs between each pair of nodes in $\mathcal{V}' - \{0, n+1\}$. Arc $(0, n+1)$ models an empty tour with null processing time $t_{0,n+1} = 0$. Note that the travel time between pick location i and the two depot copies are equivalents ($t_{0,i} = t_{i,n+1}$). Moreover, the induced sub-graph $\mathcal{G}'[\mathcal{L}]$ is complete and symmetric ($t_{i,j} = t_{j,i} | \forall i, j \in \mathcal{L}$). The table 5 presents the sets, parameters and variables used in the following MILP formulation.

Sets	
$\mathcal{O}$	Set of customer orders, where $\mathcal{O} = \{1, \ldots, o, \ldots, O\}$
$\mathcal{K}$	Set of order pickers, where $\mathcal{K} = \{1, \ldots, k, \ldots, K\}$
$\mathcal{P}$	Set of positions, where $\mathcal{P} = \{1, \ldots, p, \ldots, P\}$. Note that P represents the maximum number of tours that can be assigned to a picker.
$\mathcal{M}_o$	Set of orderlines of customer orders o
$\mathcal{L}$	Set of storage locations, where $\mathcal{L} = \{1, \ldots, i, \ldots, n\}$
$\mathcal{G}' = (\mathcal{V}', \mathcal{A}')$	Auxiliary graph with $\mathcal{V}'$ the node set and $\mathcal{A}'$ the arc set
Parameters	
W	Capacity of the cart
q_m	Weight of orderline m
d_o	Deadline of customer order o
$t_{i,j}$	Travel time between nodes i and j
β_s	Setup time of each tour
β_p	Search and pick time of each orderline
l_m	Index location of orderline m
M	A sufficiently large positive number
Variables	
$x_{m,o}^{k,p} \in \{0,1\}$	Equals 1 if orderline m of order o is picked by the picker k in the p-th tour, 0 otherwise.
$y_{i,j}^{k,p} \in \{0,1\}$	Equals 1 if the picker k uses arc (i,j) in the p-th tour, 0 otherwise.
$et_o \in \mathbb{R}^+$	Completion time of order o.
$at_i^{k,p} \in \mathbb{R}^+$	Arrival time of picker k to node i in the p-th tour.

Table 5: Notations used in the MILP formulation

$$\min \sum_{k\in\mathcal{K}} at_{n+1}^{k,P} \tag{6}$$

$$\sum_{k\in\mathcal{K}}\sum_{p\in\mathcal{P}} x_{m,o}^{k,p} = 1 \qquad \forall o\in\mathcal{O}, \forall m\in\mathcal{M}_o \tag{7}$$

$$\sum_{o\in\mathcal{O}}\sum_{m\in\mathcal{M}_o} q_m \cdot x_{m,o}^{k,p} \le W \qquad \forall k\in\mathcal{K}, \forall p\in\mathcal{P} \tag{8}$$

$$\sum_{i\in\mathcal{L}\cup\{0\}} y_{i,j}^{k,p} - \sum_{i\in\mathcal{L}\cup\{n+1\}} y_{j,i}^{k,p} = 0 \qquad \forall j\in\mathcal{L}, \forall k\in\mathcal{K}, \forall p\in\mathcal{P} \tag{9}$$

$$\sum_{i\in\mathcal{V}'-\{0\}} y_{0,i}^{k,p} = 1 \qquad \forall k\in\mathcal{K}, \forall p\in\mathcal{P} \tag{10}$$

$$\sum_{i\in\mathcal{V}'-\{n+1\}} y_{i,n+1}^{k,p} = 1 \qquad \forall k\in\mathcal{K}, \forall p\in\mathcal{P} \tag{11}$$

$$\begin{aligned} t_{i,j} - M(1-y_{i,j}^{k,p}) &\le at_j^{k,p} - at_i^{k,p} \qquad \forall (i,j)\in\mathcal{A}', \forall k\in\mathcal{K}, \forall p\in\mathcal{P} \\ &+ (\beta^s)_{\mathbb{1}_{(i=0\ \cap\ j\neq n+1)}} \\ &+ \left(\beta^p \cdot \sum_{o\in\mathcal{O}} \sum_{m\in\mathcal{M}_o | i=l_m} x_{m,o}^{k,p}\right)_{\mathbb{1}_{(i\neq 0)}} \end{aligned} \tag{12}$$

$$at_{n+1}^{k,p} \le at_0^{k,p+1} \qquad \forall k\in\mathcal{K}, \forall p\in\mathcal{P}-\{P\} \tag{13}$$

$$\sum_{j\in\mathcal{L}} y_{0,j}^{k,p} \ge \sum_{j\in\mathcal{L}} y_{0,j}^{k,p+1} \qquad \forall k\in\mathcal{K}, \forall p\in\mathcal{P}-\{P\} \tag{14}$$

$$\sum_{j\in\mathcal{L}\cup\{n+1\}} y_{l_m,j}^{k,p} \ge x_{m,o}^{k,p} \qquad \forall o\in\mathcal{O}, \forall m\in\mathcal{M}_o, \forall k\in\mathcal{K}, \forall p\in\mathcal{P} \tag{15}$$

$$et_o \ge at_{n+1}^{k,p} - M\cdot(1-x_{m,o}^{k,p}) \qquad \forall o\in\mathcal{O}, \forall m\in\mathcal{M}_o, \forall k\in\mathcal{K}, \forall p\in\mathcal{P} \tag{16}$$

$$et_o \le d_o \qquad \forall o\in\mathcal{O} \tag{17}$$

$$et_o \ge 0,\ at_i^{k,p} \in\{0,1\} \qquad \forall o\in\mathcal{O}, \forall i\in\mathcal{V}', \forall p\in\mathcal{P}, \forall k\in\mathcal{K} \tag{18}$$

$$x_{m,o}^{k,p}\in\{0,1\},\ y_{i,j}^{k,p}\in\{0,1\} \qquad \forall o\in\mathcal{O}, \forall m\in\mathcal{M}_o, \forall (i,j)\in\mathcal{A}' \quad \forall p\in\mathcal{P}, \forall k\in\mathcal{K} \tag{19}$$

The objective function (6) is the sum of pickers completion time. Assuming that the horizon start time equals 0, it is equivalent to minimize the total processing time defined in equation (1) since there is no waiting times between tours and at picking point in the optimal solution. Constraints (7) assign each orderline to exactly one position of one picker. Constraints (8) ensure that each tour satisfies the cart capacity. Constraints (9), (10),(11) are flow constraints for each picker's tour. Constraints (12) are an adaptation of the subtour elimination constraints of Miller-Tucker-Zemlin (Desrochers and Laporte (1991)) that use nodes arrival time variables. Besides the travel time between the nodes of arc (i, j) in the current tour (p, k), a setup time is added to the constraint if $(i = 0)$ and $(j \neq n + 1)$. Furthermore, the search and pick time of all orderlines retrieved from the tail of the arc (i, j) are added to the constraint if $(i \neq 0)$. Constraints (13) prevent overlaps between the consecutive tours of each picker. Constraints (14) ensure that the non-empty tours of each picker are positioned at the begenning of the sequence to avoid the exploration of some symmetric solutions (*i.e.* symmetries that result in having an empty tour at different positions between two non-empty tours). Constraints (15) link the variable $x_{m,o}^{k;p}$ and $y_{i,j}^{k;p}$: A picker k must stop at the picking locations of all orderlines that he/she retrieves during his/her tour at position p. Constraints (16) define the completion time of each order as the completion time of the last tour that retrieves one of its orderlines. Constraints (17) force the satisfaction of the deadlines. Finally, constraints (18) and (19) define the domain of the variables.

To test our MILP formulation, we conducted preliminary experiments on a benchmark of small instances generated by Van Gils et al. (2019). In these instances, the number of orders is set to $\{18, 12, 6\}$ and the batch capacity is set to $\{4, 8, 12\}$. We ran the MILP formulation on some of those instances by setting a time limit of 2 hours. We observed that the MILP is not able to return a feasible solution for most of the instances, even for the smallest ones (6 orders, $W = 15$). For the few instances where the MILP returned feasible solutions, the gaps were poor (no less than 100%). We thus conclude that those results are not exploitable in any comparative analysis.

Appendix B. Kruskall-Wallis test

Table 6 presents results of a Kruskal-Wallis H test on order picking time, on CPu time, and on gap $\Delta_{ILS|RFSS^{+}}$. Tables 7, 8, and 9 present results of a pairwise-comparison between groups of each warehouse factor on order picking time, on CPu time, and on gap $\Delta_{ILS|RFSS^{+}}$ using the test of Dunn. Note that the Test of Dunn is done when the $p\!-\!value$ returned by the Kruskall-Wallis' test is significant (*i.e.* $p - value < 0.05$).

Table 6: Kruskal-Wallis H test on order picking time, on CPU time, and on gap $\Delta_{ILS|RFSS^+}$

	N	Statistic	df	p-value
Kruskal-Wallis H test on order picking time				
Layout	7290	2340	2	0.000
Storage policy	7290	32.3	2	0.0000001
Cart capacity	7290	4110.	2	0
Order strcture	7290	318.	2	7.94e-70
Deadline distribution	7290	2.69	2	0.26
Kruskal-Wallis H test on CPU time				
Layout	7290	1607.	2	0
Storage policy	7290	2353.	2	0
Cart capacity	7290	1823.	2	0
Order strcture	7290	204.	2	5.47e-45
Deadline distribution	7290	54.5	2	1.47e-12
Kruskal-Wallis H test on $\Delta_{ILS|RFSS^+}$				
Layout	7290	5129.	2	0
Storage policy	7290	543.	2	1.35e-118
Cart capacity	7290	320.	2	2.63e-70
Order strcture	7290	780.	2	4.37e-170
Deadline distribution	7290	19.2	2	0.000067

Table 7: Dunn test on order picking time

	N_1	N_2	Statstic	p.adj	p.adj.signif
Layout					
SW - MW	2430	2430	-26.0	5.67e-149	$****$
MW - LW	2430	2430	22.3	1.50e-109	$****$
LW - SW	2430	2430	-48.3	0.	$****$
Storage policy					
AA - Ran	2430	2430	4.62	0.0000114	$****$
WA - AA	2430	2430	-0.558	1	*ns*
Ran - WA	2430	2430	-5.18	0.000000667	$****$
Cart capacity					
15 - 30	2430	2430	-39.9	0.	$****$
30 - 45	2430	2430	-23.5	9.20e-122	$****$
45 - 15	2430	2430	-63.4	0.	$****$
Order strcture					
100 - 200	2430	2430	11.2	1.74e-28	$****$
200 - 300	2430	2430	6.46	3.11e-10	$****$
300 - 100	2430	2430	17.6	4.32e-69	$****$

Table 8: Dunn test on CPU time

	N_1	N_2	Statstic	p.adj	p.adj.signif
Layout					
SW - MW	2430	2430	-26.5	1.02e-153	$****$
MW - LW	2430	2430	12.9	2.14e-37	$****$
LW - SW	2430	2430	-39.3	0.	$****$
Storage policy					
AA - Ran	2430	2430	-37.7	0.	$****$
WA - AA	2430	2430	-45.3	0.	$****$
Ran - WA	2430	2430	-7.56	1.24e-13	$****$
Cart capacity					
15 - 30	2430	2430	27.9	1.06e-170	$****$
30 - 45	2430	2430	14.1	2.18e-44	$****$
45 - 15	2430	2430	41.9	0.	$****$
Order strcture					
100 - 200	2430	2430	8.38	1.58e-16	$****$
200 - 300	2430	2430	5.82	1.77e-8	$****$
300 - 100	2430	2430	14.2	2.74e-45	$****$
Deadline distribution					
Deg - Prog	2430	2430	-3.45	1.66e-3	$**$
Deg - Uni	2430	2430	-7.38	4.85e-13	$****$
Prog - Uni	2430	2430	-3.92	2.62e-4	$***$

Table 9: Dunn test on gap $\Delta_{ILS|RFSS^{+}}$

	N_1	N_2	Statstic	p.adj	p.adj.signif
Layout					
SW - MW	2430	2430	-47.4	0.	$****$
MW - LW	2430	2430	22.8	8.65e-115	$****$
LW - SW	2430	2430	-70.2	0.	$****$
Storage policy					
AA - Ran	2430	2430	18.6	1.77e- 76	$****$
WA - AA	2430	2430	-2.90	1.10e- 2	$*$
Ran - WA	2430	2430	-21.5	8.56e-102	$****$
Cart capacity					
15 - 30	2430	2430	14.8	3.51e-49	$****$
30 - 45	2430	2430	1.29	5.88e-1	*ns*
45 - 15	2430	2430	16.1	6.69e-58	$****$
Order strcture					
100 - 200	2430	2430	-14.5	5.64e-47	$****$
200 - 300	2430	2430	-13.5	9.09e-41	$****$
300 - 100	2430	2430	-27.9	4.45e-171	$****$
Deadline distribution					
Deg - Prog	2430	2430	-4.30	0.0000521	$**$
Deg - Uni	2430	2430	-2.90	0.0111	$*$
Prog - Uni	2430	2430	1.39	0.491	*ns*

Appendix C. Cconstraint programming formulation

We propose a natural CP formulation to model the problem. The formulation uses the parameters and variables summirized in table 10.

Parameters	
v^{+}	Artificial tour with $p_{v^{+}} = 0$ and $\tilde{d}_{v^{+}} = \max_{v \in \Pi} \tilde{d}_v$
Variables	
pos_v^k	Position number of tour v if assigned to picker k, -1 otherwise.
st_t^k	Start time of tour at position t of picker k
st_v	Start time of tour v
p_t^k	Processing time of tour at position t of picker k

Table 10: Notations used in the CP formulation

$$
\begin{aligned}
\sum_{k=1}^{K} (pos_v^k \geq 1) = 1, & \qquad \forall v \in \Pi & (20)\\
\sum_{v \in \Pi} (pos_v^k = t) \leq 1, & \qquad \forall k \in \{1,\dots,K\}, \forall t \in \{1,\dots,P\} & (21)\\
st_1^k = 0, & \qquad \forall k \in \{1,\dots,K\} & (22)\\
st_{t+1}^k = st_t^k + p_t^k, & \qquad \forall k \in \{1,\dots,K\}, \forall t \in \{1,\dots,P-1\} & (23)\\
pos_v^k = t \implies p_t^k = p_v, & \qquad \forall v \in \Pi, \forall k \in \{1,\dots,K\}, \forall t \in \{1,\dots,P\} & (24)\\
pos_v^k = t \implies st_t^k = st_v, & \qquad \forall v \in \Pi, \forall k \in \{1,\dots,K\}, \forall t \in \{1,\dots,P\} & (25)\\
st_v + p_v \leq \tilde{d}_v & \qquad \forall v \in \Pi \cup \{v^+\} & (26)\\
p_t^k, st_v, st_t^k \in \mathbb{R}^+, & \qquad \forall v \in \Pi, \forall k \in \{1,\dots,K\}, \forall t \in \{1,\dots,P\} & (27)\\
pos_v^k \in \{1,\dots,P\}, & \qquad \forall k \in \{1,\dots,K\}, \forall t \in \{1,\dots,P\} &
\end{aligned}
$$

Constraints (20) guarantee that each tour is sequenced once and only once. Constraints (21) ensure that no more then one tour is sequenced at the t^{th} position of picker k. Constraints (22) and (23) sequence the tours assigned to each picker. Constraints (24) link the variables pos_v^k and p_t^k while constraints (25) synchronize st_v and st_t^k variables. Constraints (26) bound the end time of each tour by its deadline. Finally, constraints (27) define the domain of the variables.

References

Baptiste P, Le Pape C, Nuijten W (2012) Constraint-based scheduling: applying constraint programming to scheduling problems, vol 39. Springer Science & Business Media

Boysen N, Fedtke S, Weidinger F (2018) Optimizing automated sorting in warehouses: The minimum order spread sequencing problem. European Journal of Operational Research 270(1):386–400

Boysen N, de Koster R, Weidinger F (2019a) Warehousing in the e-commerce era: A survey. European Journal of Operational Research 277(2):396–411

Boysen N, Stephan K, Weidinger F (2019b) Manual order consolidation with put walls: the batched order bin sequencing problem. EURO Journal on Transportation and Logistics 8(2):169–193

Briant O, Cambazard H, Cattaruzza D, Catusse N, Ladier AL, Ogier M (2020) An efficient and general approach for the joint order batching and picker routing problem. European Journal of Operational Research 285(2):497 – 512, DOI https://doi.org/10.1016/j.ejor.2020.01.059, URL `http://www.sciencedirect.com/science/article/pii/S0377221720300977`

Bué M, Cattaruzza D, Ogier M, Semet F (2019) A two-phase approach for an integrated order batching and picker routing problem. In: A View of Operations Research Applications in Italy, 2018, Springer, pp 3–18

Cambazard H, Catusse N (2018) Fixed-parameter algorithms for rectilinear steiner tree and rectilinear traveling salesman problem in the plane. European Journal of Operational Research 270(2):419–429

Clarke G, Wright JW (1964) Scheduling of vehicles from a central depot to a number of delivery points. Operations research 12(4):568–581

De Koster R, Le-Duc T, Roodbergen KJ (2007) Design and control of warehouse order picking: A literature review. European Journal of Operational Research 182(2):481–501

Desrochers M, Laporte G (1991) Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. Operations Research Letters 10(1):27–36

Drury J (1988) Towards more efficient order picking. IMM monograph 1

Gademann N, Velde S (2005) Order batching to minimize total travel time in a parallel-aisle warehouse. IIE transactions 37(1):63–75
Gedik R, Rainwater C, Nachtmann H, Pohl EA (2016) Analysis of a parallel machine scheduling problem with sequence dependent setup times and job availability intervals. European Journal of Operational Research 251(2):640–650
Ham AM, Cakici E (2016) Flexible job shop scheduling problem with parallel batch processing machines: Mip and cp approaches. Computers & Industrial Engineering 102:160–165
Henn S (2015) Order batching and sequencing for the minimization of the total tardiness in picker-to-part warehouses. Flexible Services and Manufacturing Journal 27(1):86–114
Henn S, Schmid V (2013) Metaheuristics for order batching and sequencing in manual order picking systems. Computers & Industrial Engineering 66(2):338–351
Henn S, Koch S, Wäscher G (2012) Order batching in order picking warehouses: a survey of solution approaches. In: Warehousing in the Global Supply Chain, Springer, pp 105–137
Hooker JN, van Hoeve WJ (2018) Constraint programming and operations research. Constraints 23(2):172–195
Kulak O, Sahin Y, Taner ME (2012) Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms. Flexible services and manufacturing journal 24(1):52–80
Laborie P (2018) An update on the comparison of mip, cp and hybrid approaches for mixed resource allocation and scheduling. In: International conference on the integration of constraint programming, artificial intelligence, and operations research, Springer, pp 403–411
Laborie P, Rogerie J, Shaw P, Vilím P (2018) Ibm ilog cp optimizer for scheduling. Constraints 23(2):210–250
Le-Duc T, De Koster RB (2005) Travel distance estimation and storage zone optimization in a 2-block class-based storage strategy warehouse. International Journal of Production Research 43(17):3561–3581
Löffler M, Boysen N, Schneider M (2018) Picker routing in agv-assisted order picking systems. Tech. rep., Working Paper, DPO-2018-01, Deutsche Post Chair-Optimization of Distribution . . .
Masae M, Glock CH, Vichitkunakorn P (2020a) Optimal order picker routing in a conventional warehouse with two blocks and arbitrary starting and ending points of a tour. International Journal of Production Research 58(17):5337–5358
Masae M, Glock CH, Vichitkunakorn P (2020b) Optimal order picker routing in the chevron warehouse. IISE Transactions 52(6):665–687
Öztürkoğlu Ö, Hoser D (2019) A discrete cross aisle design model for order-picking warehouses. European Journal of Operational Research 275(2):411–430
Öztürkoğlu Ö, Gue KR, Meller RD (2012) Optimal unit-load warehouse designs for single-command operations. IIE Transactions 44(6):459–475
Pansart L, Catusse N, Cambazard H (2018) Exact algorithms for the order picking problem. Computers & Operations Research 100:117–127
Prins C (2004) A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research 31(12):1985–2002
Ratliff HD, Rosenthal AS (1983) Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Operations Research 31(3):507–521
Roodbergen KJ (2001) Layout and routing methods for warehouses. PhD thesis, E, URL `http://hdl.handle.net/1765/861`
Roodbergen KJ, De Koster R (2001) Routing order pickers in a warehouse with a middle aisle. European Journal of Operational Research 133(1):32–43
Scholz A, Wäscher G (2017) Order batching and picker routing in manual order picking systems: the benefits of integrated routing. Central European Journal of Operations Research 25(2):491–520
Scholz A, Schubert D, Wäscher G (2017) Order picking with multiple pickers and due dates–simultaneous solution of order batching, batch assignment and sequencing, and picker routing problems. European Journal of Operational Research 263(2):461 – 478, DOI https://doi.org/10.1016/j.ejor.2017.04.038, URL `http://www.sciencedirect.com/science/article/pii/S0377221717303855`
Statista (accessed on December 2, 2019) Retail e-commerce revenue in Canada from 2017 to 2023. `https://www.statista.com/statistics/289741/canada-retail-e-commerce-sales`

Theys C, Bräysy O, Dullaert W, Raa B (2010) Using a tsp heuristic for routing order pickers in warehouses. European Journal of Operational Research 200(3):755–763

Tompkins JA, White JA, Bozer YA, Tanchoco JMA (2010) Facilities planning. John Wiley & Sons

Tsai CY, Liou JJH, Huang TM (2008) Using a multiple-ga method to solve the batch picking problem: considering travel distance and order due time. International Journal of Production Research 46(22):6533–6555, DOI 10.1080/00207540701441947, URL `https://doi.org/10.1080/00207540701441947`, `https://doi.org/10.1080/00207540701441947`

Valle CA, Beasley JE, da Cunha AS (2017) Optimally solving the joint order batching and picker routing problem. European Journal of Operational Research 262(3):817–834

Van Gils T, Ramaekers K, Caris A, de Koster RB (2018) Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. European Journal of Operational Research 267(1):1–15

Van Gils T, Caris A, Ramaekers K, Braekers K (2019) Formulating and solving the integrated batching, routing, and picker scheduling problem in a real-life spare parts warehouse. European Journal of Operational Research 277(3):814–830

Wäscher G (2004) Order picking: a survey of planning problems and methods. In: Supply chain management and reverse logistics, Springer, pp 323–347

Weidinger F (2018) Picker routing in rectangular mixed shelves warehouses. Computers & Operations Research 95:139–150