European Journal of Operational Research 288 (2021) 80–97

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Production, Manufacturing, Transportation and Logistics

Introducing split orders and optimizing operational policies in robotic mobile fulfillment systems

Lin Xie[a,*], Nils Thieme[a], Ruslan Krenzler[a], Hanyi Li[b]

[a] Leuphana University of Lüneburg, Universitätallee 1, 21335 Lüneburg, Germany
[b] Beijing Hanning Intelligent Tech Co., Ltd, Room 107–108, No.9 Kexing Road, Fengtai, Beijing, China

ARTICLE INFO

Article history:
Received 26 December 2019
Accepted 17 May 2020
Available online 26 May 2020

Keywords:
logistics
MIP models
Integrated operational optimization
Robotic mobile fulfillment systems
Split orders

ABSTRACT

In robotic mobile fulfillment systems, human pickers don't go to the inventory area to search for and pick the ordered items. Instead, robots carry shelves (called "pods") containing ordered items from the inventory area to picking stations. At the picking stations, pickers put ordered items into totes; then these items are transported to the packing stations. This type of warehousing system relieves the human pickers and improves the picking process. In this paper, we concentrate on decisions about the assignment of pods to stations and orders to stations to fulfill picking for each incoming customer's order. In previous research for an RMFS with multiple picking stations, these decisions are made sequentially with heuristics. Instead, we present a new MIP-model to integrate both decision problems. To improve the system performance even more, we extend our model by splitting orders. This means parts of an order are allowed to be picked at different stations. To the best of the authors' knowledge, this is the first publication on split orders in an RMFS. And we prove the computational complexity of our models. We analyze different performance metrics, such as pile-on, pod-station visits, robot moving distance and throughput. We compare the results of our models in different instances with the sequential method in our open-source simulation framework RAWSim-O. The integration of the decisions brings better performances, and allowing split orders further improves the performances (for example: increasing throughput by 46%). In order to reduce the computational time for a real-world application, we have proposed a heuristic.

1. Introduction

The most important and time-consuming task in a warehouse is the collection of items from their storage locations to fufill customer orders. The process is called *order picking*, which may constitute about 50–65% of the operating costs. Therefore, the order picking is considered as the highest-priority area for productivity improvements (see De Koster, Le-Duc, & Roodbergen, 2007). Due to the increasingly fast-paced economy, it is becoming more and more important that the orders are processed in a short time window.

In a traditional manual order picking system (also called a *picker-to-parts system*), the pickers spend 70% of their working time on the tasks of search and travel (see Tompkins, 2010; for an overview of manual order picking systems see De Koster et al., 2007). The unproductive searching and traveling times require the picker-to-parts system to have a large workforce, especially for companies which have millions of small items in large warehouses, such as e-commerce companies like Amazon or retailers like Zara and Walmart, which provide both brick-and-mortar and online shops. Dependent on the type of retailers, they are facing many diverse customer orders each day (both single-line and multi-line orders). Also, the workforce of such companies is under high pressure due to the long traveling time (see Wulfraat, 2012). Kiva Systems LLC, now Amazon Robotics LLC, came up with a unique solution to avoid the unproductive times of human pickers in picker-to-parts systems; therefore, this solution accelerates the order picking process (see Wurman, D'Andrea, & Mountz, 2008). In such system, robots are sent to carry storage units, so-called "pods," from the inventory area and bring them to human operators, who work only at picking stations. At the stations, the items are picked according to the customers' orders. After picking, the robot transports the pod back to the storage area. There are also some other suppliers of such systems, such as Scallog, Swisslog (KUKA), GreyOrange and Hitachi (see Banker, 2016). All of these systems may differ

* Corresponding author.
E-mail addresses: xie@leuphana.de (L. Xie), nils.thieme@stud.leuphana.de (N. Thieme), ruslan.krenzler@leuphana.de (R. Krenzler), hli@hanningzn.com (H. Li).

https://doi.org/10.1016/j.ejor.2020.05.032

Table 1
Literatur overview in an RMFS.

decision problems	publications
Strategic level:	
storage area dimensioning,	Lamballais, Roy, and De Koster (2017),
workstation placement	Lamballais, Roy, and De Koster (2019)
Tactical level:	
number of robots	Yuan, Dong, and Li (2016), Yuan and Gong (2017),
	Zou, Xu, De Koster et al. (2018), Otten, Krenzler, Xie, Daduna, and Kruse (2019)
Operational level:	
decisions for each incoming order about	
which robot carries	Zhang, Yang, and Weng (2019), Roy, Nigam, de Koster, Adan, and Resing (2019)
which pod	Boysen et al. (2017)
along which path	Cohen, Uras, and Koenig (2015),Cohen, Wagner, Kumar, Choset, and Koenig (2017),
	Merschformann et al. (2017)
to which station	
decisions about where to	
put the used pod back to	Merschformann (2017),
	Weidinger, Boysen, and Briskorn (2018),
	Krenzler, Xie, and Li (2018)
decision rules	Wurman et al. (2008),
	Merschformann et al. (2019)
simulation	Merschformann et al. (2018)
demonstration	Xie et al. (2019)
performance characteristics	Hanson, Medbo, and Johansson (2018)

technically in certain aspects, such as the lifting mechanism, but they share the same principle of the system (see the description of such system in Section 2). This system is called *robotic mobile fulfillment system* (RMFS).

1.1. Research background

In an RMFS environment, various optimization and allocation problems have to be solved in real time. An overview of an RMFS and other automated warehousing systems can be found in Azadeh, de Koster, and Roy (2017) and Boysen, de Koster, and Weidinger (2018). Here we give you an short literature overview of RMFSs in Table 1, which also includes new researches that are not listed in both overview papers. The classification of three levels of problems is based on Merschformann, Lamballais, de Koster, and Suhl (2019).

In this work, we concentrate on operational problems. As shown in Table 1, there are usually the following decision problems in the order picking process in an RMFS (described in Wurman et al., 2008 and Merschformann, Xie, & Li, 2018): Each time a new order arrives, we have to decide which robot carries which pod along which path to which station to fulfill picking. So the order is first assigned to a station (*pick order assignment*, in short: POA), and then one or several pods are assigned to that station to fulfill that order (*pick pod selection*, in short: PPS). Robots are assigned to deliver pods to that station (*robot task allocation*), while *path planning* plans the paths for the robots. After a pod is finished at a picking station, we have to decide where to put it back in the inventory area (*pod repositioning*). And there are two decisions for the replenishment process, namely replenishment order assignment and replenishment pod station. Table 2 describes the operational problems in an RMFS.

1.2. Contributions and paper structure

We concentrate in this paper on the picking process, especially POA and PPS. In the studies of the throughput performance of decision rules for multiple online decision problems in Merschformann et al. (2019), they concluded that POA should be paid more attention in the literature and practice, since it affects the throughput performance of RMFS the most. In Boysen et al. (2018), they mentioned that the existing research into order picking is under

Table 2
The operational decision problems in an RMFS.

Decision problem	Description
Order Assignment:	
Replenishment Order Assignment (ROA)	assignment of replenishment orders to replenishment stations
Pick Order Assignment (POA)	assignment of pick orders to picking stations
Task Creation:	
Replenishment Pod Selection (RPS)	selection of pods that will move to a replenishment station to store replenishment items
Pick Pod Selection (PPS)	selection of pods to use for picking the pick orders assigned at a picking station
Pod Repositioning (PR)	assignment of an available storage location to a pod that is not in the inventory area
Task Allocation (TA)	assignment of generated tasks for robots in RPS, PPS, PR and additional tasks such as idling
Path Planning (PP)	planning of the paths for the robots to execute

the assumption that orders have already been assigned to stations (in other words: without optimization of POA), such as in Boysen, Briskorn, and Emde (2017).

Due to the importance of POA, POA and PPS should be optimized together; however, the complexity is increased due to considering all possible pods and multiple picking stations (see examples in Section 2.2). Therefore, the authors in Wurman et al. (2008) and Merschformann et al. (2019) provided heuristics to solve POA and PPS sequentially, first POA and then PPS. We list here the contributions of our work:

- We develop a new mathematical model to solve integrated POA and PPS for multiple stations.
- We extend our integrated model to allow split orders, so not all parts of an order are picked at the same station. To the best of the authors' knowledge, this is the first publication on split orders in an RMFS. The literature review of split orders can be found in Section 2.3.
- We develop a heuristic for our new models to solve a real-world instance.

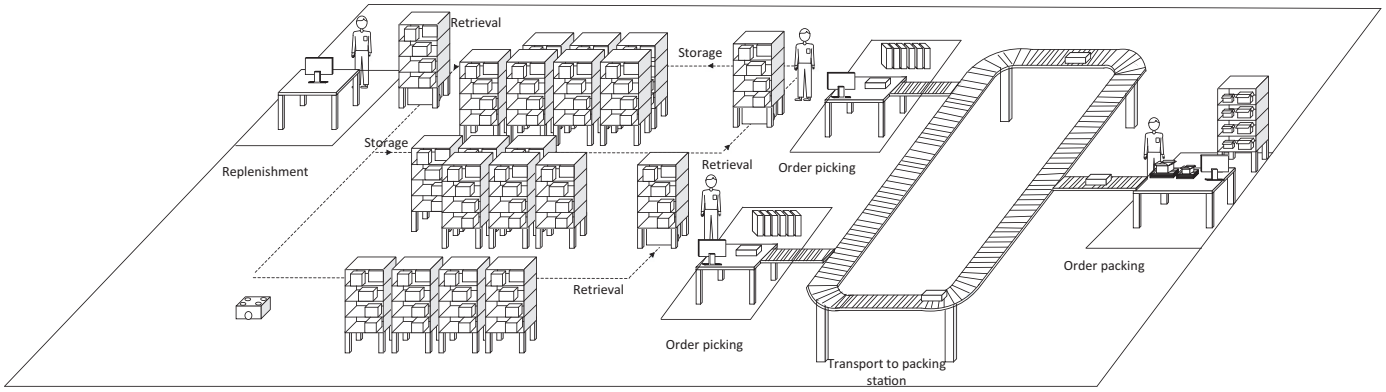


Fig. 1. The central process of an RMFS.

Table 3
Terms related to orders.

Term	Description
SKU	stock keeping unit
order line	one SKU with the ordered quantity
item	a physical unit of one SKU
pick order	a set of order lines from a customer's order
split order	a pick order that is separated into several parts
replenishment order	a number of physical units of one SKU
backlog	all unfulfilled orders

Table 4
The central components of an RMFS.

Component	Description
pods	movable shelves, on which inventory is stored
storage area	the inventory area where the pods are stored
workstations:	
picking stations	where pickers pick the order items
packing stations	where packers pack the pick order items and the split orders are consolidated
replenishment stations	where replenishment items are stored to pods
robots	moving underneath pods and carrying them to workstations
conveyors	transporting the pick orders between picking and packing stations to finish packing

- We extend our open-source simulation framework RAWSim-O to evaluate the decisions made based on our new models.

This paper is organized as follows: In the next section, we describe the RMFS and operational decision problems in detail. After that, we will describe our idea of integrated POA and PPS and split orders with examples. In Section 3, a mathematical model of integrated POA and PPS and the extensions with split orders are described. We present simulation evaluations in Section 4.2. Finally, we draw conclusions and give pointers for further research in Section 5.

2. Problem description with examples

In this section, we first describe the RMFS, and the decision problems in an RMFS. After that, we will describe our idea of integrated POA and PPS and split orders with examples.

2.1. RMFS

Firstly, we define some terms related to orders before explaining the processes in an RMFS in Table 3.

The central components of an RMFS are listed in Table 4.

The pods are transported by robots between the inventory area and workstations. Fig. 1 shows the central process in a simplified RMFS from replenishment to packing:

- *Retrieval process*: After the arrival of a replenishment order, robots carry selected pods to a replenishment station to store units in pods. Similarly, after receiving a pick order, robots carry selected pods to a picking station, where the items for the order lines are picked. Note that in order to fulfill pick orders, several pods may be needed, since orders may have multiple lines. The items in (parts of) an order are picked into a tote.
- *Storage process*: After a pod has been processed at one or more stations, it is brought back to a storage location in the storage area. The retrieval and storage processes are based on Hoffman et al. (2013).
- *Transport to packing stations*: Once a tote is filled, it is transported by a conveyor to packing stations for packing.
- *Packing process*: If all items in an order are contained in a tote, packers are prompted by computer to select the correct-sized box and pack the items. A split order has items delivered via multiple totes, since the items are picked by different pickers (picking stations). In this case, packers first sort items from a tote to a correct-sized box on the shelf so that the items from that order are grouped together. Packers are prompted by computer to put the box for the split order on one given position on the shelf, and later to find the box in the shelf to put the rest of that order into the box. We use the term *shelf* to clarify that they might be different to the pods, since they do not need to be moved. Once all the items of a split order are in a box, the packer packs the box, and a space is open for the next split order. This packing process is based on the packing process in Amazon (see Toister, 2017).

2.2. POA and PPS

We concentrate in this paper on the online decisions for POA and PPS for multiple stations. For better clarification, *online* in our case does not mean that we consider orders that have arrived over a number of time periods. In order to have enough optimization potential, we assume that we know all orders in advance (for example, we gather all orders that have come in the last hour in the backlog). But our methods can support the decisions for the new incoming orders in each period (see Section 4.3). Instead, online decisions in our case means that we have to make decisions for both problems for each new period t. Period $t-1$ is changed to t if some jobs are finished at the stations while there are unfulfilled orders in the backlog. Also, situations such as the inventory of pods and the positions of pods in the queues at stations can and will change from $t-1$ to t. They are important for the POA and PPS

decisions and are hard to calculate exactly in advance, since errors and delays in previous time periods can affect them. For these reasons, we make the decisions for the integrated POA and PPS just before the respective time period starts. This allows us to react to the current situation and take errors or delays from previous periods into account. Furthermore, we test our results in a simulation framework, which provides us with the actual information for each time period. In the literature, the online decisions for POA and PPS are usually solved sequentially for multiple stations (first POA, then PPS; see Wurman et al., 2008, Merschformann et al., 2019). The sequential approach limits the information for both decisions. Therefore, the decisions for POA and PPS should be made simultaneously to achieve optimum performance, so we can use informations about all pods and orders, including those that are assigned to stations and are unassigned. However, the complexity of solving integrated POA and PPS is increased, and the integrated problem is NP-hard (see the proof in Section 3). We will describe the difference between sequential and integrated problems with Example 1. To the best of the authors' knowledge, this is the first publication about the integrated POA and PPS for multiple stations.

Objective Before we explain POA and PPS with examples, we want to firstly explain our objective of POA and PPS. An RMFS with higher throughput is better. Based on the performance analysis of decision rules for multiple online decision problems in Merschformann et al. (2019), they found that a high pile-on (the number of picks per handled pod) and a short distance traveled by robots together is a indicator for the success of a decision rule applied in an RMFS (to achieve high throughput). In order to achieve that, we aim at minimizing the number of visits by pods to stations (in short: *pod-station visits*) for each decision of the POA and PPS problems. If fewer pod-station visits are needed for the given orders, a shorter distance traveled by the robots can be achieved, and higher pile-on can be expected to finish the given orders. Furthermore, a smaller number of pod-station visits also causes fewer changes of pods, and the waiting time of human pickers between changes of pods is reduced (see Boysen et al., 2017; they also minimize pod-station visits for the assignment of pods to orders). Another time component of human pickers is retrieving the items from their shelves, which is considered to be fixed. Due to the reduced waiting time, more orders can be handled within a minimal time. So, this matches the suggestion in Van Gils, Ramaekers, Caris, and de Koster (2018) for an efficient order picking in picker-to-parts systems.

Example 1. Fig. 2(a) illustrates a small problem to fulfill four orders 1, 2, 3 and 4. The different colors represent different SKUs (stock keeping units). For simplicity, the quantity of each SKU in the orders is one. We have in total two picking stations. There is two empty totes at each station. In this example we assume that each tote can hold three items. Pod 1 is currently at station 1 and pod 2 at station 2, while pods 3 and 4 are in the storage area.

Sequential POA and PPS In the sequential POA and PPS, we use the same decision rule, *Pod-Match*, as in Merschformann et al. (2019), which assigns the orders from the backlog to a station so that the items for the orders best match the pods that are already assigned to that station. Note that there is another more common decision rule in Merschformann et al. (2019) (called *Common-Lines*) and Wurman et al. (2008), grouping similar orders at picking stations in POA. However, the decision rule Pod-Match for POA is shown to perform better in Merschformann et al. (2019), since this rule uses information about assigned pods at stations in addition to information about orders in the backlog.

In Example 1, in the POA problem we assign orders 2 and 3 to station 1 (Fig. 2(b)), since two of their items can be picked from pod 1 – the pod that is already at station 1. For the same reason, we assign orders 1 and 4 to station 2. To fulfill the assigned orders, both pods from the storage area, pods 3 and 4, are needed at each station. After items from pods 1 and 2 are picked, they are returned to the storage area. In the PPS, pod 3 visits station 1, while pod 4 visits station 2 (Fig. 2(c)). After picking in both stations, pods 3 and 4 switch stations so that the last item of each order can be picked (Fig. 2(d)). In total, 6 pod-station visits were necessary to fulfill both orders in this example, therefore the pile-on can be calculated as 12 picks/6 pods = 2 picks/pod.

Integrated PPS and POA In the integrated PPS and POA approach, we have more information while assigning orders to stations, since pods and orders are assigned to stations at the same time. This allows us to find optimal solutions that might not be intuitive at first glance and would not be found by the sequential POA and PPS. Note that we use information about all pods, including assigned ones at stations and unassigned ones in the storage area, but this way increases the complexity to solve both problems.

Using the same initial state as in the previous explanation of the sequential POA and PPS in Example 1 (see Fig. 3(a)), we integrate these two decisions and assign orders and pods such that the number of pod-station visits is minimized. This leads to the assignment of orders 1 and 2 and pod 3 to station 1, and orders 3 and 4 and pod 4 to station 2 (see Fig. 3(b) and 3(c)). This results in a pile-on of 3 (12 picks/4 pods) compared to 2 (12 picks/6 pods) in the sequential example and only requires 4 pod-station visits to fulfill all orders instead of 6.

Based on this example, we can see the benefit of integrating POA and PPS by using information about the inventory of all pods in these decisions. Therefore, we present a mathematical model in this paper that integrates POA and PPS for multiple stations and takes information about the inventory of all pods into account.

2.3. Allowing split orders in our integrated approach

In our integrated approach mentioned above, an order is only allowed to be assigned to a single station. The second contribution of this paper is to allow split orders in our integrated approach. And we also prove that the models to allow split orders are NP-hard (see Section 3). A *split order* means that we divide an order into two or more parts for picking (perhaps at different stations). A similar term, "splitting orders," can be traced back to 1979, when it was used by Armstrong, Cook, and Saipe (1979). They used split orders to keep batch sizes constant in batch picking. In Il-Choe and Sharp (1991) and De Koster, Le-Duc, and Zaerpour (2012) split orders are used as part of the zoning in traditional picker-to-parts warehouses, in which a storage area is split into multiple parts (called zones), each with a different order picker. When an order contains several SKUs that are stored in different zones, the SKUs for the order are picked separately in each zone and merged later for shipping. To the best of the authors' knowledge, this is the first publication on split orders in an RMFS.

According to the following example, we expect allowing split orders in an RMFS provide a better solution.

Example 2. Fig. 4 illustrates the decision problem when assigning orders and pods to two picking stations. We have one empty tote at each station, while we have two identical orders 1 and 2 in the backlog (in Fig. 4(a)). We assume that each tote can hold two items. These two orders contain SKUs shown in blue and orange. These two SKUs are located in two different pods, namely pod 1 with the orange SKU and pod 2 with the blue SKU. Fig. 4(b) shows the optimal solution to the problem without split orders. We need pod 1 to visit station 1 and pod 2 to visit station 2; after that, pod 2 visits station 1 and pod 1 visits station 2. In total, we need four visits by pods to the stations to fulfill both orders. Instead, if we split orders 1 and 2 into blue and orange parts (see Fig. 4(c)), the blue ones can be picked from pod 2 at station 2, while the orange

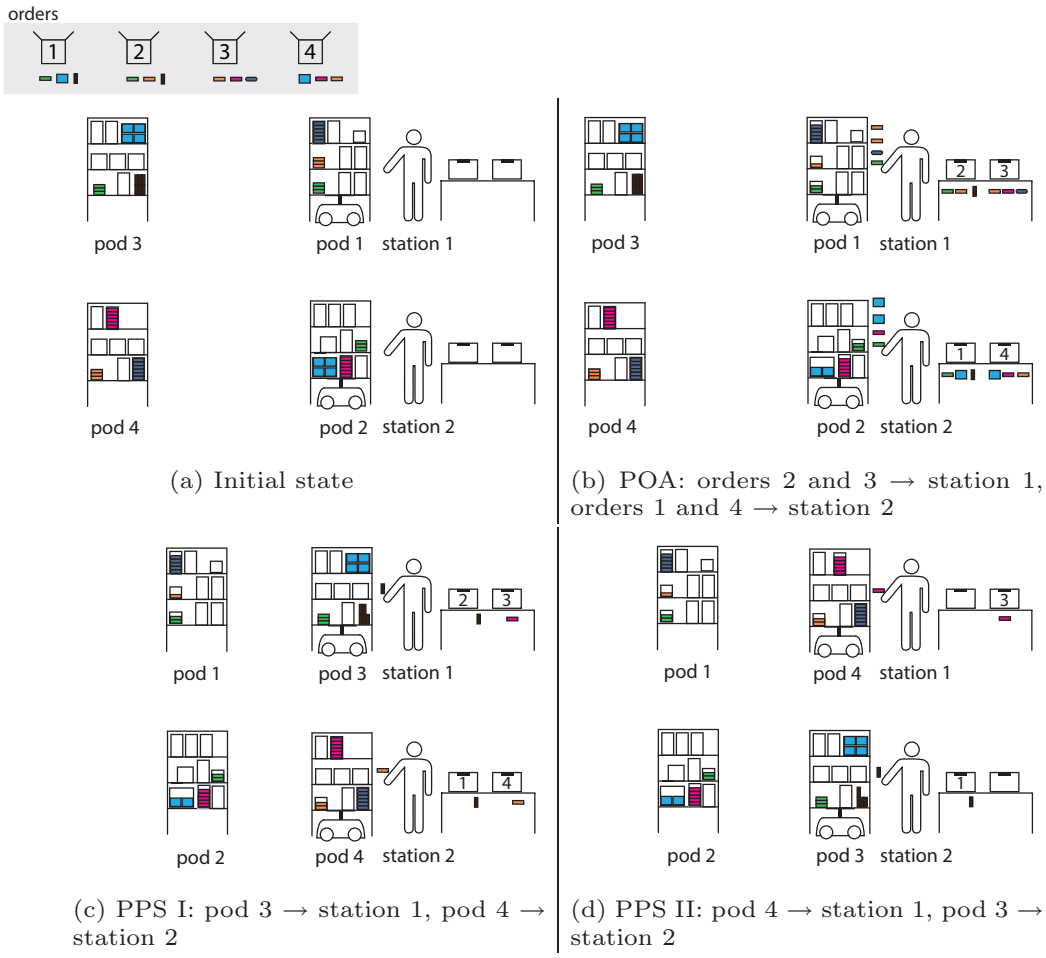


Fig. 2. An example for the sequential POA and PPS (best viewed in color).

ones can be picked from pod 1 at station 1. This allows both orders to be fulfilled with only two visits by pods to the stations instead of four. Note that in this paper we don't use one empty tote for exactly one order, but for several items, to enable comparison between the solutions with and without split orders. We will explain more about this in Section 3.1 in the paragraph *Capacity of a picking station*.

However, the split orders might need additional consolidation time in packing stations, where customers' orders are packed and ready for shipping. We ignore this time in our study, we will describe that in Section 3.1.

3. Mathematical model

In this section, we describe the assumptions in Section 3.1 before we present our mathematical model of integrated POA and PPS (we call it the *integrated model*), and extend it with two variants of allowing split orders.

3.1. Assumption

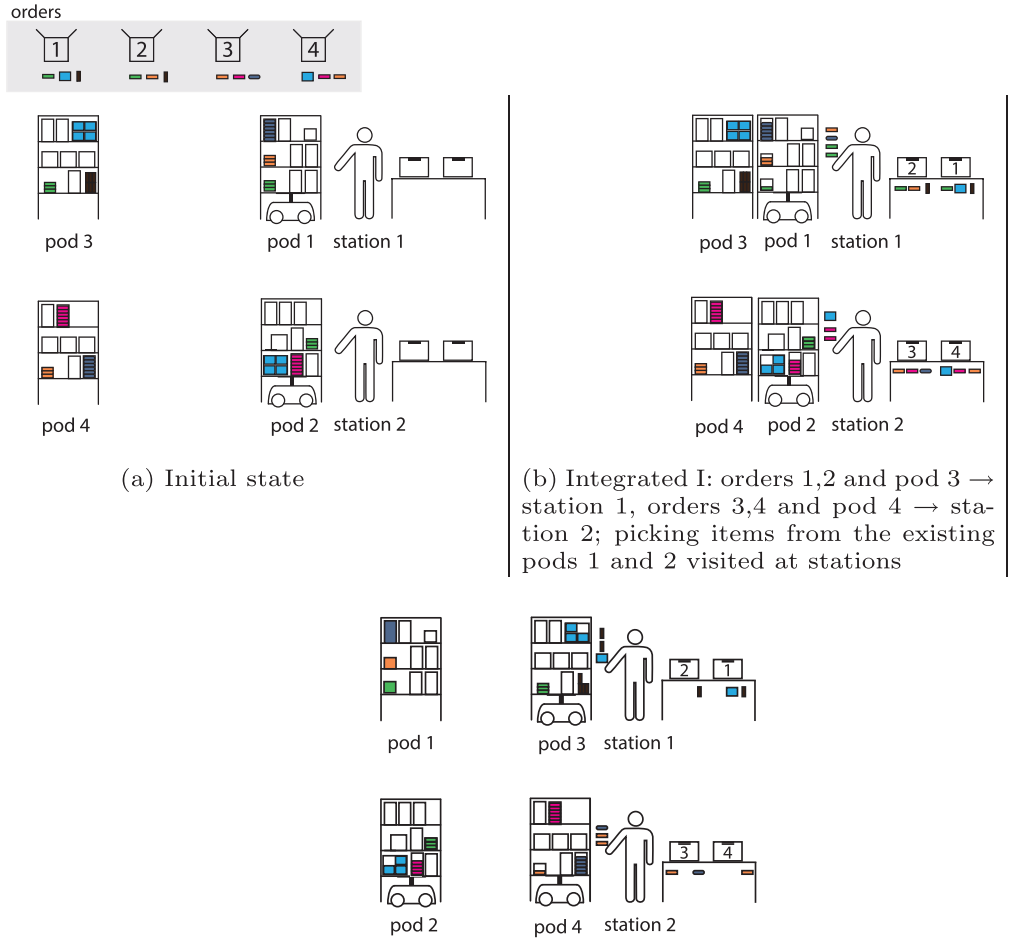
SKUs All different SKUs in orders are available in pods. We assume that the quantity of the order line for each SKU is one. This assumption is consistent with common practice, since the number of items per order line is low. If a pod contains a SKU, then we assume that there are enough items in that pod to fulfill all orders for that SKU.

Pregenerated orders We store pregenerated orders in the backlog before the beginning of optimization, so we get the same orders for testing different approaches. From time to time, no new incoming orders are stored to the backlog for the optimization. Our approaches terminate if the backlog is empty.

Split order Splitting an order means separating the original order into two or more parts (up to the number of SKUs in the order). If an order is not split, we ensure that all order lines in that order are assigned for picking at the same station (within a time period). If an order is split, this constraint is relaxed by allowing order lines for that order to be assigned to more than one picking station or more than one time period. There are two variants of a split order:

- split among stations: all order lines for a pick order are assigned in the same period but may be assigned to different picking stations (see Example 1 in Section 1)
- split over timesteps: order lines for a pick order may be assigned in different time periods and to different picking stations (see an example in Appendix A)

Capacity of a picking station Commonly, the capacity of a picking station is defined as the number of orders that can be handled at a time (*order capacity*). According to Wulfraat (2012), the typical station can support 6 to 12 orders to be picked at a time. The introductory example of split orders shows that traditional order capacity is incompatible with split orders, since simply counting the number of assigned orders does not work anymore when only



(a) Initial state

(b) Integrated I: orders 1,2 and pod 3 → station 1, orders 3,4 and pod 4 → station 2; picking items from the existing pods 1 and 2 visited at stations

(c) Integrated II: picking items from pod 3 at station 1, while picking items from pod 4 at station 2

Fig. 3. Same example as in Fig. 2 (see Fig. 3(a)), but the decision is made by the integrated POA and PPS (see Fig. 3(b) and 3(c)) (best viewed in color).

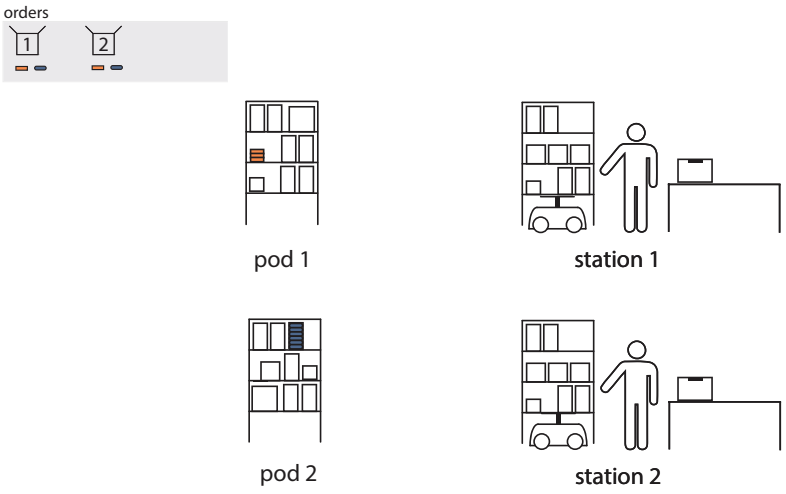
parts of an order are assigned to the station. Instead, we introduce in this paper a new way to define the capacity of a picking station – limited by the number of items to be handled at a time – that works for both, whole orders and split orders. We call this type of capacity *item capacity*. Another advantage of item capacity is a fairer distribution of workload among all stations, since the number of assigning items equals the number of picks. Note that the number of items in each order differs and reflects the number of picks.

Capacity of a packing station Orders that are not split can be packed directly by packers as soon as they arrive at the packing stations. Split orders require storage space on shelves at packing stations to wait until all parts of the order are picked. Once all parts of a split order are picked, it can be packed and one space on the shelf becomes free for the next split order. The capacity of a packing station is therefore defined as the number of shelves multiplied by the number of boxes which can be stored on a shelf. We set the total capacity of all packing stations to a parameter C, and we assume it is large enough for all necessary split orders in this paper. This assumption is supported by the calculation in Appendix B. In our calculation, up to 78 split orders can be stored on a shelf. And usually, in practice, there is more than one packing station. If more split orders are required, then additional shelves can be installed at packing stations. However, the situation might differ from one company to another. Therefore, our model can be easily extended to support a limited packing capacity, as shown in Section 4.3.

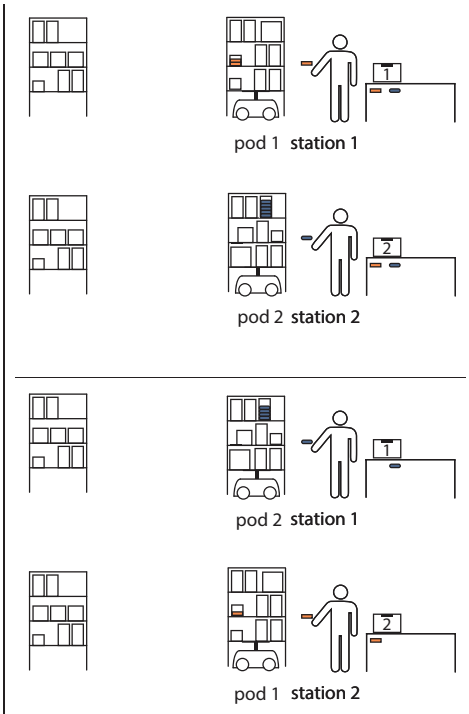
Conveyor We assume that the conveyors between picking and packing stations are long enough to temporarily store orders and parts of them. The conveyors serve as a buffer to synchronize the picking and packing stations.

Consolidation time for split orders We consider the consolidation time as the additional time for the packer to pack the split orders. Based on the packing process in Amazon (described at the end of Section 2.1, and the part of Fig. 4 on the right-hand side), shelves are used in the packing station to temporarily store split orders. Each time a part of a split order arrives, the packer puts this part into the corresponding box on the shelf. Once all items of the split order are in the box, the packer packs the box, and a space is open for the next split order. This process is almost the same as the process for a normal order, i.e. folding a new box, putting all items in the box and packing it. The additional time for packers caused by the split orders is the searching time for the correct split-order box on the shelf, which we consider to be minimal since the position of the box is stored on the computer. Therefore, we ignore this time in our study.

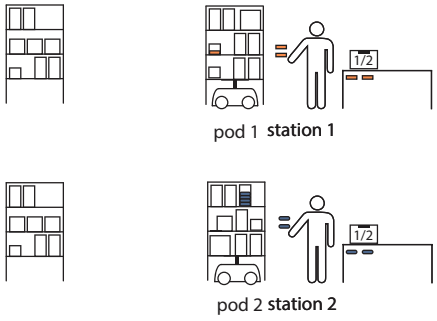
Maximal order size We assume that every order in the backlog can fit into some picking station. That means the maximal item capacity of the largest picking station is not smaller than the number of items of the largest order.



(a) Initial state



(b) Without split orders: first, pod 1 → station 1 and pod 2 → station 2 (upper part); then, pod 2 → station 1 and pod 1 → station 2 (lower part)



(c) With split orders: pod 1 → station 1 and pod 2 → station 2

Fig. 4. A solution for without and with split orders (best viewed in color).

Queue There is a queue at each station (for example, in Fig. 3(b), pod 1 and pod 3 are in the queue at station (1). The space in a queue is limited. Each time a pod leaves the queue, one pod can be added at the end of the queue. Pods leave a station once their inventory cannot be used anymore to fulfill any further assigned orders. If pod 1 leaves the queue, pod 3 moves forwards to the picker.

Period Once there is enough free item capacity at a station and there are unfilled orders in the backlog, the time period is changed from t to $t+1$ for all $t \geq 1$. The required amount of free item capacity is defined as the capacity that is needed to fit the smallest available order. In $t = 0$, no orders are assigned or picked at any picking station. All pods are in the storage area, so there are no pods waiting at picking stations. At $t = 1$ we start to assign orders from the backlog and pods to picking stations. Each time t changes, the current situation in the warehouse (such as which pods are currently in storage or on their way to stations, free capacity at stations, inventory of pods, decreasing order backlog) is updated and used to compute the next decisions. This way, we can handle errors or delays in the execution of previous decisions. The model described in this section is solved in each period t using information about the current state of the warehouse.

Shared storage policy Items of the same SKU are randomly spread over multiple pods. In Boysen et al. (2017), where this pol-

icy is called *mixed-shelves storage*, the authors showed that this policy is efficient.

Pods selection Our model computes the smallest possible set of pods to fulfill all assigned orders at each station in each period, without considering the distance between the selected pods and picking stations.

Sequencing of pods During each period t we need to know the sequence of pods at each station. As our model only calculates the optimal sets of pods for each station, we use the following policy to create a sequence of pods: Robots begin immediately to carry all assigned pods to the respective stations and the sequencing of pods is decided by the order of their arrival at the station. This ensures that station idle times are kept at a minimum.

Without replenishment We assume that there is enough inventory for all orders, so no replenishment is required.

3.2. Integrated model

Firstly, we define the notation for the following model.

Sets:

$\mathcal{P}$	Set of pods
$\mathcal{S}$	Set of currently available picking stations
$\mathcal{P}_s$	Set of pods $\mathcal{P}_s \subseteq \mathcal{P}$ that are currently at station s
$\mathcal{P}_i^{\text{SKU}}$	Set of pods $\mathcal{P}_i^{\text{SKU}} \subseteq \mathcal{P}$ that include SKU i
$\mathcal{O}$	Set of current orders in the backlog
$\mathcal{I}_o$	Set of SKUs $\mathcal{I}_o \subseteq \mathcal{I}$ that constitutes an order $o \in \mathcal{O}$

Parameters:

C_s	Current capacity of each picking station $s \in \mathcal{S}$

Decision variables:

$$x_{ps} \quad \begin{cases} 1, & \text{pod } p \in \mathcal{P} \text{ is assigned to station } s \in \mathcal{S} \\ 0, & \text{else} \end{cases}$$

$$y_{os} \quad \begin{cases} 1, & \text{order } o \in \mathcal{O} \text{ is assigned to station } s \in \mathcal{S} \\ 0, & \text{else} \end{cases}$$

$$y_{ios} \quad \begin{cases} 1, & \text{SKU } i \in \mathcal{I}_o \text{ of order } o \in \mathcal{O} \text{ is assigned to station } s \in \mathcal{S} \\ 0, & \text{else} \end{cases}$$

u_s Amount of unused capacity for a station $s \in \mathcal{S}$

The integrated model is invoked in the simulation each time the time period t is changed. However, for simplicity the parameter t is not used in the model. Note that all sets, parameters and decision variables may change for each time period t.

$$\text{Min} \quad \sum_{p\in\mathcal{P}} \sum_{s\in\mathcal{S}} x_{ps} + \sum_{s\in\mathcal{S}} W_u \cdot u_s \tag{1}$$

$$\text{s.t.} \quad y_{os} = y_{ios}, \ \forall i \in \mathcal{I}_o, o \in \mathcal{O}, s \in \mathcal{S} \tag{2}$$

$$\sum_{s\in\mathcal{S}} y_{os} \leq 1, \ \forall o \in \mathcal{O} \tag{3}$$

$$\sum_{o\in\mathcal{O}} \sum_{i\in\mathcal{I}_o} y_{ios} + u_s = C_s, \ \forall s \in \mathcal{S} \tag{4}$$

$$\sum_{p\in\mathcal{P}_i^{\text{SKU}}} x_{ps} \geq y_{ios}, \ \forall i \in \mathcal{I}_o, o \in \mathcal{O}, s \in \mathcal{S} \tag{5}$$

$$x_{ps} = 1, \ \forall p \in \mathcal{P}_s, s \in \mathcal{S} \tag{6}$$

$$y_{os} \in \{0,1\}, \ \forall o \in \mathcal{O}, s \in \mathcal{S} \tag{7}$$

$$y_{ios} \in \{0,1\}, \ \forall i \in \mathcal{I}_o, o \in \mathcal{O}, s \in \mathcal{S} \tag{8}$$

$$x_{ps} \in \{0,1\}, \ \forall p \in \mathcal{P}, s \in \mathcal{S} \tag{9}$$

$$u_s \in \mathbb{Z} \geq 0, \ \forall s \in \mathcal{S} \tag{10}$$

In the integrated model given above, we aim at minimizing the number of pods used in each period, while keeping the unused capacity of stations as low as possible. Constraint set (2) sets the value of y_{os} to 1 for all assigned order/station tuples and ensures that, if an order is assigned to a station, all order lines in the order are assigned to the same station as this order. And if an order is not assigned to any station, none of its order lines can be assigned to a station. Constraint set (3) ensures that each order can be assigned to at most one station. Constraint set (4) ensures that the number of assigned items equals the amount of available capacity minus the amount of unused capacity u_s at each station $s \in \mathcal{S}$. Each u_s increases the value of the cost function (assuming $W_u > 0$). Constraint set (5) ensures that for each order line of an order that is assigned to a station, at least one pod $p \in \mathcal{P}_i$ is assigned to the same station. Constraint set (6) ensures that pods that were assigned in previous periods and are currently on their way to a station or in a station's queue stay assigned to that station. Constraint sets (7)–(9) ensure that the respective variables can only have binary values while (10) ensures that u_s can only have non-negative integer values.

Proposition 1. *The integrated model always has a feasible solution which we can find in polynomial time.*

Proof. See Appendix C. □

Proposition 2. *The optimal solution of the integrated model is NP-hard.*

Proof. See Appendix C. □

3.3. Split-among-stations model

This model is an extension of the integrated model from the previous section. Now we allow splitting the SKUs in an order between two or more stations. We need the following additional decision variables.

Additional decision variables:

$$y_o \quad \begin{cases} 1, & \text{order } o \in \mathcal{O} \text{ is assigned} \\ 0, & \text{else} \end{cases}$$

e_o the number of additional assigned picking stations for an order $o \in \mathcal{O}$

$$\text{Min} \quad \sum_{p\in\mathcal{P}} \sum_{s\in\mathcal{S}} x_{ps} + \sum_{s\in\mathcal{S}} W_u \cdot u_s \tag{1}$$

$$\text{s.t.} \quad (4) - (10)$$

$$y_{os} \geq y_{ios}, \ \forall i \in \mathcal{I}_o, o \in \mathcal{O}, s \in \mathcal{S} \tag{2.1}$$

$$\sum_{s\in\mathcal{S}} y_{os} \geq y_o, \ \forall o \in \mathcal{O} \tag{3.1}$$

$$\sum_{s\in\mathcal{S}} y_{ios} = y_o, \ \forall i \in \mathcal{I}_o, o \in \mathcal{O} \tag{11}$$

$$y_o \geq y_{os}, \ \forall o \in \mathcal{O}, s \in \mathcal{S} \tag{12}$$

$$\sum_{i\in\mathcal{I}_o} y_{ios} \geq y_{os}, \ \forall o \in \mathcal{O}, s \in \mathcal{S} \tag{13}$$

$$y_o \in \{0,1\}, \ \forall o \in \mathcal{O} \tag{14}$$

The cost function (1) and constraints (4)–(10) are carried over from the previous model. Constraint set (2) is relaxed, so that the order lines for an order don't have to be assigned to the same station anymore (see constraint set (2.1)), but y_{os} is still set to 1 for all

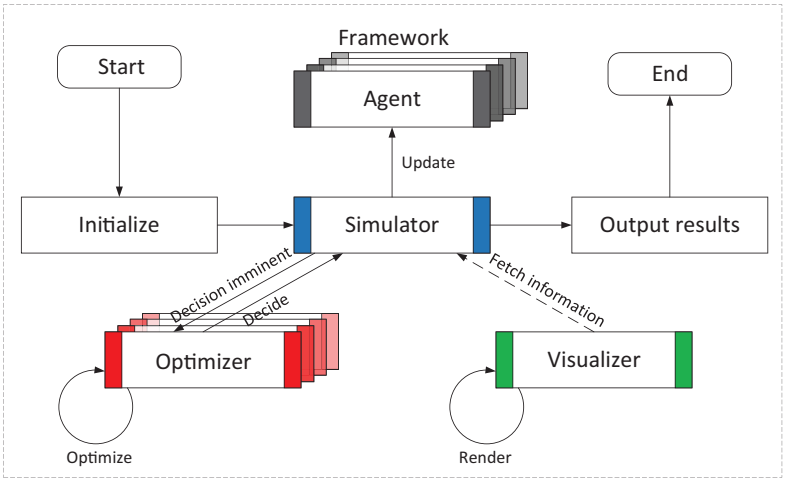


Fig. 5. Overview of the simulation process (see Merschformann et al., 2018).

stations the order is assigned to. Constraint set (3) is also relaxed, to allow for the assignment of an order to more than one station (see constraint set (3.1)). Constraint set (11) now ensures that if an order is active (at least one order line is assigned to a station), all of its order lines have to be assigned to stations. Constraint set (12) sets the value of y_o to 1 for each order that is assigned to at least to a station. Constraint set (13) ensures that an order can only be assigned to a station if at least one order line of the order is assigned to that station. Constraint set (14) ensures that y_o is a binary variable for each $o \in \mathcal{O}$.

Proposition 3. *Every solution of the integrated model also solves the split-among-stations model. There is always a feasible solution of the split-among-stations model. The split-among-stations model always provides a solution that is better than, or equally good as, the integrated model.*

Proof. See Appendix C. □

Proposition 4. *The optimal solution of the split-among-stations model is NP-hard.*

Proof. See Remark 1 in Appendix C. □

3.4. Split-over-time model

This model is an extension of the split-among-stations model from the previous section. Here we also allow every order to be split over different time periods. This means some SKUs for an order may be assigned in one period while the others will stay in the backlog to be assigned in later periods. We define one additional binary variable.

Additional decision variable:

y^b_{io}	$\begin{cases} 1, & \text{SKU } i \in \mathcal{I}_o, o \in \mathcal{O} \text{ is moved back to the backlog} \\ 0, & \text{else} \end{cases}$

$$\text{Min} \quad \sum_{p\in\mathcal{P}} \sum_{s\in\mathcal{S}} x_{ps} + \sum_{s\in\mathcal{S}} W_u \cdot u_s \qquad (1)$$

$$\text{s.t.} \quad (2.1) - (3.1), (4) - (10), (12) - (20)$$

$$\sum_{s\in\mathcal{S}} y_{ios} + y^b_{io} = y_o, \ \forall i \in \mathcal{I}_o, o \in \mathcal{O} \qquad (11.1)$$

$$y^b_{io} \in \{0,1\}, \ \forall i \in \mathcal{I}_o, o \in \mathcal{O} \qquad (15)$$

All constraints from the previous model are carried over, except for constraint set (11). Constraint set (11.1) is relaxed to allow not assigning all order lines of an order at once (see constraint set (11.1)). The new constraint set (15) ensures that the value of y^b_{io} is binary.

Proposition 5. *Every solution of the split-among-stations model also solves the split-over-time model. There is always a feasible solution of the split-over-time model. The split-over-time model always provides a solution that is better than, or equally good as, the split-among-stations model.*

Proof. See Appendix C. □

Proposition 6. *The optimal solution of the split-over-time model is NP-hard.*

Proof. See Remark 1 in Appendix C. □

4. Computational evaluation

In this section, we describe the parameters and results of the computational evaluation. We first describe the open-source simulation framework used for this paper in Section 4.1. Next, we show the results of the simulation in Section 4.2. In Section 4.3, we make some remarks regarding the assumptions that were made in our computational evaluation from a practical point of view.

4.1. Simulation framework

In the following evaluation we use RAWSim-O from Merschformann et al. (2018), an open-source, agent-based and event-driven simulation framework providing a detailed view of an RMFS. The source code is available at www.rawsim-o.de. Fig. 5 shows an overview of the simulation process, which is managed by the core *simulator* instance. The tasks of the simulator include:

- updating *agents*, which can resemble real entities, such as robots and stations
- passing decisions to *optimizers*, which can either decide immediately or buffer multiple requests and release the decision later
- exposing information to a *visualizer*, which allows optional visual feedback in 2D or 3D.

The hierarchy of decision problems regarding the assignment of replenishment orders, pick orders and pods to station is illustrated in Fig. 6. We develop a new optimizer, called *POA & PPS*, to make integrated decisions for both POA and PPS. Furthermore,

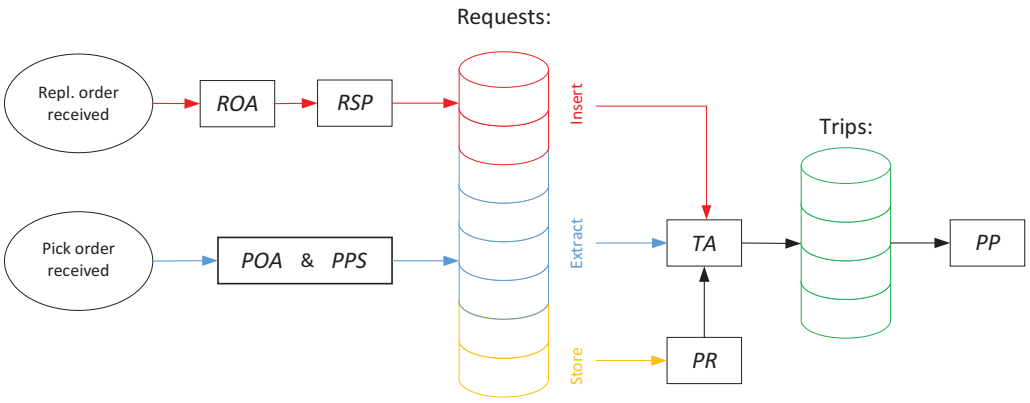


Fig. 6. Order of decisions to be made after receiving a pick or replenishment order.

we extend the new optimizer to support split orders. And we define new type of capacity for each station, called item capacity, in RAWSim-O. If a replenishment order needs to be assigned to a replenishment station, the optimizers of ROA and RPS are responsible for choosing a replenishment station and a pod. This results in an insertion request, i.e. a request for a robot to bring the selected pod to the given workstation. If a picking order needs to be assigned to a picking station, the new optimizer *POA & PPS* submits all necessary information in the simulator to our model and converts its solution into extraction requests in the simulator. Extraction requests contain both, the item that needs to be picked and the pod that it should be picked from. The model is in this case either the integrated model in Section 3.2, the split-among-stations model in Section 3.3 or the split-over-time model in Section 3.4. Our models were implemented in the python API of Gurobi Optimizer. Note that in Merschformann et al. (2018), first the POA optimizer is called and the extraction requests are generated; after that, the PPS optimizer is called. In other words, pods are only assigned to stations after orders have already been assigned. Moreover, in our new optimizer, extraction requests for an order can be assigned to different stations to support split orders. We define the item capacity as the sum of the extract requests at one station for each (part of an) order that has at least one unfulfilled request at that station. Furthermore, the system generates a store request each time a pod needs to be transported back from a station to a storage location, and the PR optimizer decides on the storage location for that pod. The TA optimizer assigns robots to these tasks. All tasks result in robot trips, which are planned by a PP optimizer.

The simulation framework conceptually consists of three different inputs:

- instance configuration: contains orders, initial inventory and available SKUs (see Section 4.1.1)
- layout configuration: determines the characteristics and dimensions of the warehouse layout (see Section 4.1.2)
- optimizer configuration: specifies the decision rules for all operational problems in an RMFS (see Section 4.1.3).

We describe these three inputs for our experiments in this paper in the following sections.

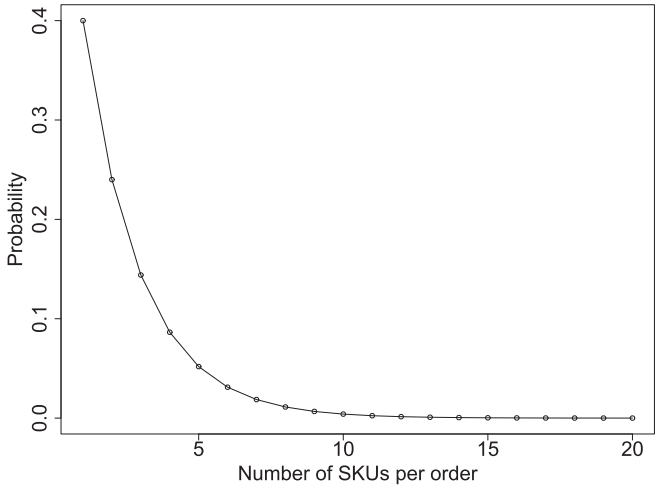


Fig. 7. Probabilities of order lengths for $|I| = 20$.

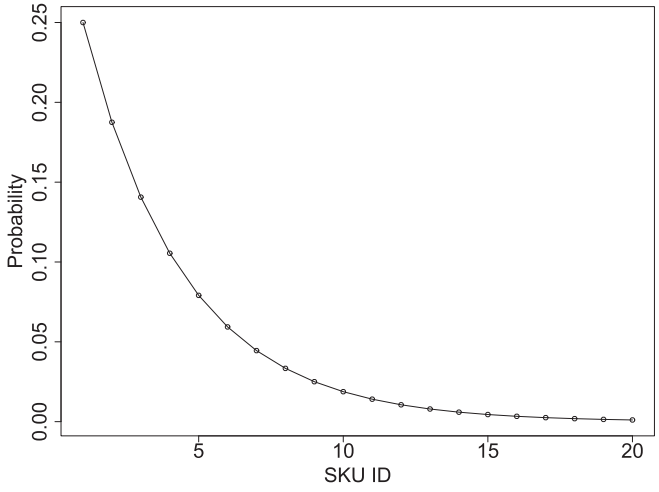


Fig. 8. Probabilities of SKUs for $|I| = 20$.

4.1.1. Instance generation

First, we describe how we generate SKUs and orders, and how to fill the pods. The set of SKUs is generated as $I = \{i_1, ..., i_{|I|}\}$. For each order $o_1, ..., o_{|O|}$ the number of different SKUs in it is determined by a truncated (1 to $|I|$) geometric distribution with $p = 0.4$ (see Fig. 7 for $|I| = 20$). And the number of items for a SKU is set to 1. It is typical in online retailing that most of the orders contain very few line items, such as in Bozer and Aldarondo (2018) and Onal, Zhang, and Das (2017). The SKUs in the order are then chosen by sampling without replacement from I using the probability mass function of a geometric distribution with $p = 5/|I|$, to account for the varying demand for different SKUs. Fig. 8 shows the probabilities of SKUs for $|I| = 20$.

A shared storage strategy (as described in Bartholdi & Hackman, 2017) is applied to fill the pods. To determine the initial inventory

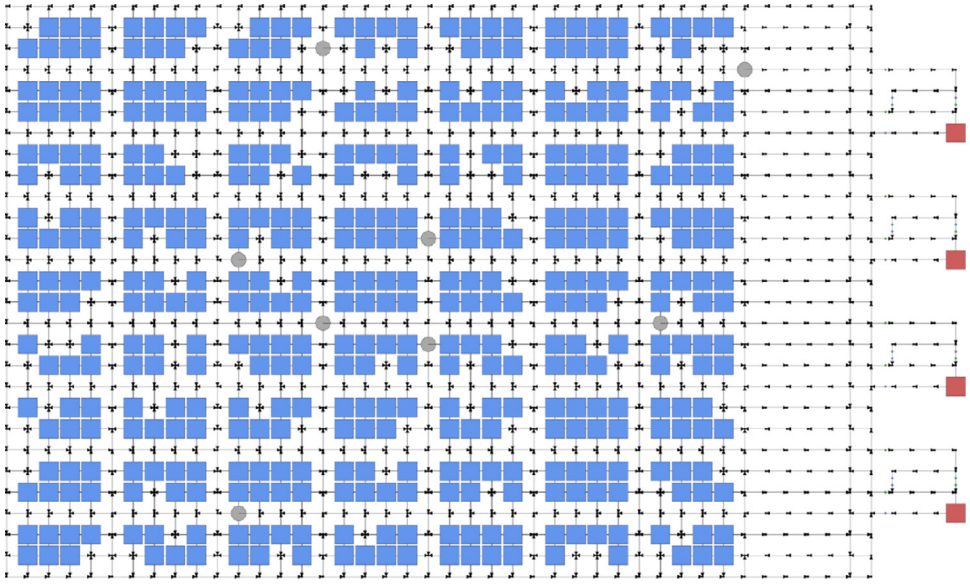


Fig. 9. Simulation layout.

Table 5
Instance parameters.

Symbol	Description	Values
$\|O\|$	Number of orders	50, 150, 250
$\|I\|$	Number of SKUs	20, 100
$\|P\|$	Number of pods	50, 100
α	SKUs per pod	2, 3

Table 6
Decision rules.

Decision Problem	Sequential	Integrated (without/with split orders)
POA	Pod-Match	Model
PPS	Demand	Model
ROA	not relevant	not relevant
RPS	not relevant	not relevant
PR	Nearest	Nearest
PP	WHCA$^*_n$	WHCA$^*_n$

of the pods, lists of all SKUs in randomized order are concatenated until the combined list includes at least $|P| \cdot \alpha$ elements. Then, for each pod p_1, ..., $p_{|P|}$, the inventory is determined by cutting off the first α items in the list.

We generate instances with different parameters from Table 5. By considering all possible combinations of the parameters, we generate 24 instances. We test all methods in this paper with the pregenerated instances to see the efficiency of the different algorithms. In a real RMFS, the orders would not be known at the start of the simulation but would instead be received during runtime. Our methods are also compatible with this type of order-generation (see Section 4.3).

4.1.2. Layout

The layout is identical for most of the test instances and is illustrated in Fig. 9: 428 pods (blue squares), 504 storage locations in 2×4 blocks, 4 picking stations (red squares) and 8 robots (gray circles). The length of station queues is 12. As we are focusing on the decisions made in the order picking process, we disregard the replenishment process. When the number of pods is higher than the value of P, only $|P|$ pods have an inventory and the empty pods are not taken into consideration. The capacity of picking stations is similar to that used in Boysen et al. (2017). Boysen et al. (2017) used an order capacity of 6 orders. Since we use item capacity instead of order capacity, we multiply 6 by 2.5, the mean of the distribution used to determine the number of items per order (described in Section 4.1.1), which leads to an item capacity of 15.

4.1.3. Decision rules

Table 6 lists the decision rules for all operational problems in the evaluation. The definitions of these can be found in Table 2.

The POA and PPS decision rules selected for the sequential approach are based on Merschformann et al. (2019), since these combinations achieved the best throughput. The rule *Pod-Match* for the POA problem selects the pick order from the backlog that best matches the pods already assigned to the station. Out of all pods that could fulfill at least part of an assigned order, the decision rule Demand for PPS chooses the pod whose inventory is in most demand when combining all unfulfilled orders. The *ROA* and *RPS* problems are concerned with the replenishment process and thus not relevant to our tests. The decision rule Nearest for *PR* assigns the closest parking space to each pod leaving a picking station. For the *PP* problem, non-volatile windowed hierarchical cooperative A* (called WHCA$^*_n$) from Merschformann, Xie, and Erdmann (2017) is used. In this algorithm, A* is used to find the optimal path without considering collisions for each robot, while a space-time reservation table is used within a limited time window to avoid collisions between robots. Non-volatile means that the existing path and reservation are stored for each robot within a limited time window. The model used for POA and PPS to test our integrated approach is the integrated model described in Section 3 and both of its extensions for split orders.

4.2. Simulation results

Each combination of method and instance is simulated ten times to lessen the effect of randomness caused by other optimizers and simulation components. Testing was done on 12 Intel Xeon X5650 Cores with 24 Gigabyte RAM. The following performance metrics are tested:

- the number of pod-station visits per order
- the distance driven by robots per order
- pile-on: the number of picks per pod-station visit
- throughput: picks per hour.

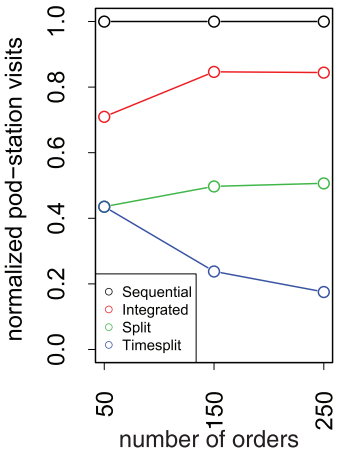


Fig. 10. Normalized pod-station visits per order.

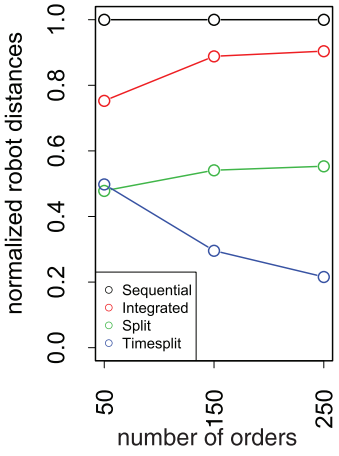


Fig. 11. Normalized robot distances per order.

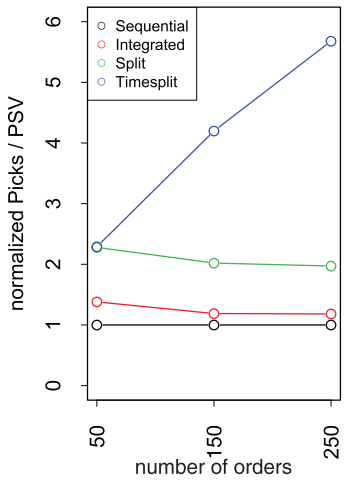


Fig. 12. Normalized pile-on (best viewed in color).

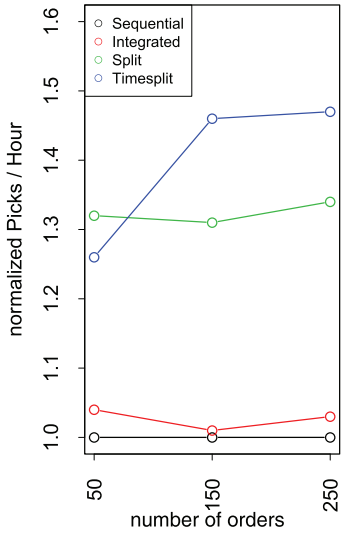


Fig. 13. Picks per hour (best viewed in color).

The parameter W_u is set to 2, since we aim to fully utilize the picking station's capacity whenever possible. The highest possible cost of assigning an additional SKU is 1 (one additional pod needed, just for this SKU), therefore any value of W_u greater than 1 ensures that as many orders as possible are assigned. Further parameters used in the simulation can be found in Appendix D, such as parameters for robot movement.

Fig. 10 shows the average number of pod-station visits per order relative to the sequential approach for each instance set of 50, 150 and 250 orders. Compared to the sequential approach for the number of pod-station visits per order: the integrated model improves this performance by 20% to 30% for different instance sets, the split-among-stations model (in short in figures and tables: split) shows improvements of about 50%, and the split-over-time model (in short in figures and tables: timesplit) improves on the sequential solution by 57% to 80% for different instance sets.

Fig. 11 shows similar improvements for the average distances driven by robots to complete an order for each instance set. The correlation between pod-station visits and distances driven by robots confirms our assumption in Section 1, that distances driven by robots can be reduced by minimizing the number of pod-station visits. As the layout shown in Fig. 9 demonstrates, the distance between the inventory area and a picking station is in most cases greater than the distance between any two picking stations. Both a pod coming from the inventory area to a station and a pod coming from another picking station count as pod-station visits. In the sequential approach, pods may have to be transported directly between stations, because the orders that share the same pods are assigned to different stations. Therefore, there are pods moving between picking stations. In the integrated approach, we try to assign orders which share the same pods to the same stations; therefore, it reduces the number of pods moving between picking stations. This explains why the distances per robot in the integrated approach do not show as much of an improvement as pod-station visits per order compared to the sequential approach.

Fig. 12 shows the average pile-on of all methods for each instance set of 50, 150 and 250 orders. Our approach with split orders causes more picks per pod-station visit (PSV), especially in the split-over-time model (up to 5.5 times as many picks per PSV) (best viewed in color).

In Fig. 13 we get the average picks per hour for each instance set of 50, 150 and 250 orders. From 30% up to 46% more picks per hours can be achieved by the split-among-stations and split-over-time.

Fig. 14 illustrates the total computing times for each instance set of 50, 150 and 250 orders. The total computing time for the integrated approaches with and without split orders are much larger than those for the sequential approach. We divide the total com-

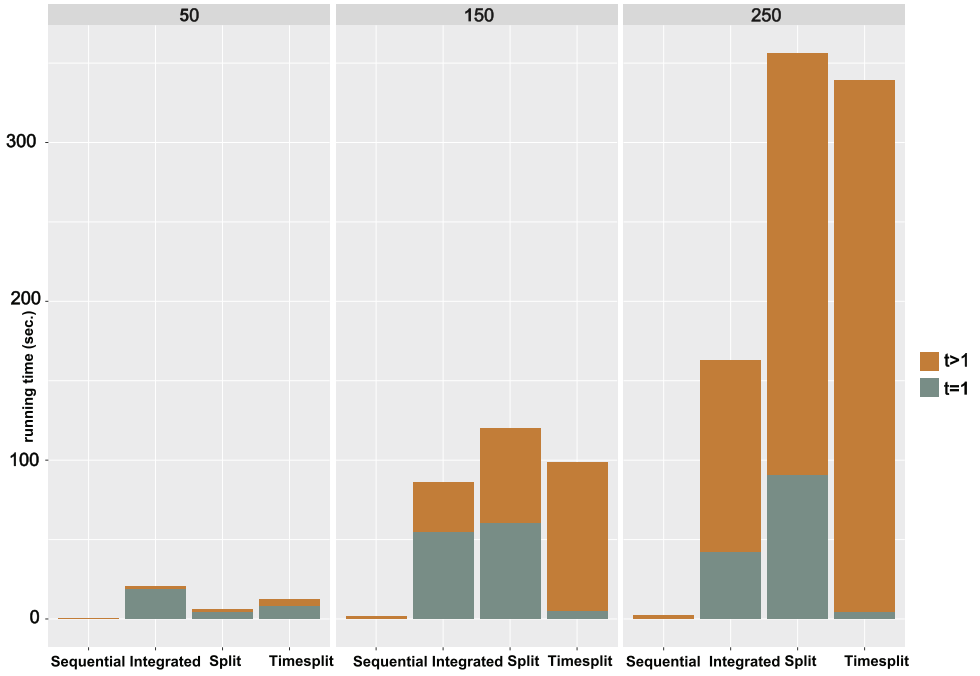


Fig. 14. Computing time for the first period and others (best viewed in color).

Table 7
Computing time per simulation time for $t > 1$.

Instance set	Sequential	Integrated	Split	Timesplit
50	0.07%	0.17%	0.24%	0.61%
150	0.07%	1.4%	3.46%	6.41%
250	0.07%	3.39%	9.67%	14.67%

Table 8
Computing time $t > 1$ per order (in seconds).

Instance set	Sequential	Integrated	Split	Timesplit
50	0.01	0.03	0.03	0.08
150	0.01	0.21	0.4	0.62
250	0.01	0.48	1.06	1.34

puting time into the time at $t = 1$ and the time of the remaining periods $t > 1$. The period at $t = 1$ takes a lot of time, since no orders or pods have previously been assigned to any stations and therefore more decisions are necessary than in the following periods. As this usually happens at most once a day (when the system is stopped and then restarted), we can consider it a warm-up. It is more interesting to see the periods $t > 1$, since most decisions are made in those periods. Note that the decisions at $t = 1$ in the split-over-time model are faster than those in the integrated and split-among-stations models. The reason is that the split-over-time model can easily find the best possible solution in the first timestep, where only one pod is needed at each station. Table 7 shows the total computing time of the decisions in $t > 1$ in relation to the simulation time. We deem computing times acceptable as long as they are lower than the total simulation time. In our tests, computing the decisions took at most around 15% of the simulation time (split-over-time, 250-order instance set). The average time for the assignment of one order is at most 1.5 seconds (see Table 8).

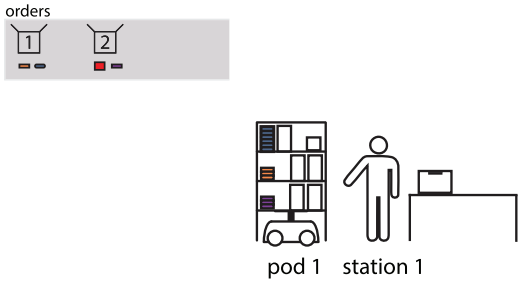


Fig. 15. Example of the heuristic method.

4.2.1. A Heuristic method for a real-world instance

The Canadian logistics company Think logistics (see ThinkLogistics, 2012) owns an RMFS, which stores 2000 SKUs in 217 pods. And there are 15 robots used in the system. We generated an instance of 2000 orders and 25 SKUs per pod, and tested it in the similar layout as in Fig. 9 (15 instead of 8 robots were used). SKUs and orders were generated as described in Section 4.1.1. The computational time of our integrated models increased extremly for such instance. Therefore, we developed a heuristic method to accelerate the computational time. The main idea is instead of considering all orders in the backlog as input for our models, we select only part of them ($n \leq |\mathcal{O}|$) in the following way: First, all orders in the backlog are sorted according to the percentage of orders lines in each respective order for which a pod containing the sku currently is in a stations queue (in short: match). Then the best n orders are selected. Fig. 15 shows an example to explain this heuristic method. We have two orders 1 and 2 in the backlog and we want to pick up $n = 1$ order. And pod 1 visits currently station 1. We have one open tote in station 1 and the match of order 1 to pod 1 is 100%, since all items of order 1 are stored in pod 1. The match of order 2 to pod 1 is 50%. So order 1 should be firstly picked from the backlog.

We test this method for different n in all instance sets (50-, 150-, 250-order) and model variants to analyze the effect n has on

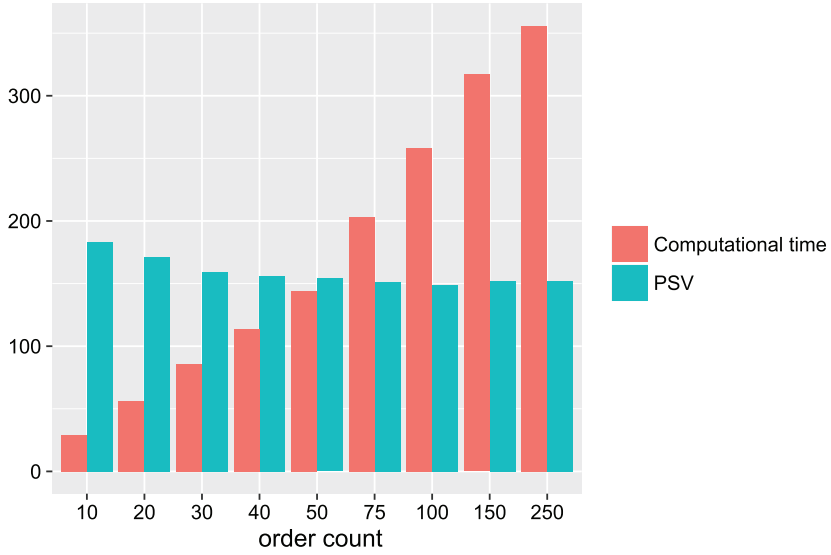


Fig. 16. Computational time vs. the number of pick-station visits for the 250-order instance set by prefiltering n orders (n is from 10 to 250) (best viewed in color).

computing time and the quality of the solution (number of pick-station visits). The results of different n between 10 and 250 for 250-order instances in the split-among-stations model are illustrated in Fig. 16. The results of other instance sets show a similar distribution for the integrated approaches. By using the heuristic method, we can save more than 55% in computing time while pod-station visits rise by 1% compared to the optimal results of the split-among-stations model (see order count 50 in Fig. 16 and order count 250 is the optimal solution).

For testing the large instance, we chose $n = 40$ and solved with split-over-time method, since this method brings the best performance, but it reqires more computational time. Compared to the sequential solution, we can reduce the number of pod-station visits by 60%, same is the reduced distances drived by robots. 47% more picks per hours can be achieved, while the average computational time of each order is 4.6 seconds, which is acceptable. We simulated this instance ten times and the testing was done on Intel Xeon E5-2670 16 Cores with 24 GB RAM.

4.3. Practical remarks

Some assumptions in Section 3.1 might differ from real-world scenarios. In this section we discuss them from a practical point of view.

- Real-world orders: in this paper we use pregenerated instances to test the performance of different approaches. Instead, in a real RMFS, new orders would constantly come in while the optimization algorithms are running. Even completely new SKUs could be stocked during the optimization. To account for this, our simulation RAWSim-O was recently extended by an interface to an ERP system to allow for its use as a robot control system in real warehouses, as described in Xie, Li, and Thieme (2019). This new feature of the simulator could also be used in conjunction with the content presented in this paper, to implement the model presented here in a real warehouse and use real instances instead of pregenerated testing instances.
- Considering orders with deadlines: in this paper we don't generate orders with deadlines. The reason for that is we want to see maximum effects of our methods. However, our heuristic can be easily extended to select the orders with tight deadlines first.
- Reduced size of queues: We set the size of the queue at each picking station to be 12 (see Section 4.1.2). Based on our observation in the simulation, the queue is rarely full; one of the main reasons for this is that we minimize the number of pod-station visits. And allowing split orders reduces the number of pod-station visits by up to 80%. In general, fewer pods are brought to queues. So, allowing split orders might cause the additional positive side-effect that the required space at picking stations is reduced.
- Capacity of packing stations: for the results shown above, we assumed the capacity of packing stations for split orders to be large enough to store all possible split orders. As this capacity might differ from one company to another, a situation where not enough capacity is available to allow for the splitting of all orders is conceivable. In order to consider this situation, we can extend, for example, the models in Sections 3.3 and 3.4 with the following parameters, variables and constraints. We need two additional parameters: C for the total capacity of packing stations (in other words: number of available split orders) and N for the number of stations. The binary variable y_o^l is activated if order $o \in O$ is split, while n^l counts the number of currently active split orders from the previous periods. Note that, this number is decreased by one if one split order is picked completely.

$$\sum_{s \in \mathcal{S}} y_{os} - e_o = y_o, \ \forall o \in \mathcal{O} \tag{3.2}$$

$$y_o^l \geq e_o/N, \ \forall o \in \mathcal{O} \tag{16}$$

$$y_o^l \leq e_o, \ \forall o \in \mathcal{O} \tag{17}$$

$$n^l + y_o^l \leq C \tag{18}$$

$$y_o^l \in \{0, 1\}, \ \forall o \in \mathcal{O} \tag{19}$$

$$e_o \in \mathbb{Z} \geq 0, \forall o \in \mathcal{O} \tag{20}$$

In constraint set (3.2), e_o is counted as the number of additional stations to finish picking order o. Constraint sets (16) and

Fig. 17. An example of using separators in totes (LocusBot).

(17) make sure that y_o^l is equal to one if $e_o \geq 0$, while constraint set (18) makes sure that the number of split orders is less than C. Constraint set (19) defines y_o^l as binary variables for each order $o \in \mathcal{O}$, while constraint set (20) ensures that e_o can only have non-negative integer values.

To see the impact that a limited capacity of packing stations has on the solution, we tested the 250-order instance set for a packing station capacity of 10. When applying this limited capacity to the split-among-stations model, pod-station visits were reduced by 35% compared to the sequential approach, instead of nearly 50% when packing capacity is not limited. Similar effect can be achieved by the split-over-time model. But the split-over-time model needs higher packing capacity, since some incomplete orders stay longer in packing stations. Note that the capacity limit of 10 is an extreme case and the solution is still better than that of the sequential approach. In the real world, higher capacities are possible.

- Item capacity: this capacity is used for comparing the sequential and integrated approaches. The capacity of a picking station in the real world is limited by the number of totes, which depends on the size of a tote. There are the following possibilities to apply our approaches without changing the layout of picking stations:
  - using different sizes of totes, with smaller ones for split orders
  - using separators in totes for split orders (an example: LocusBot in Fig. 17).
- Reliability of the simulation: as described in Xie et al. (2019), the simulation framework RAWSim-O was extended to connect to an ERP system and industry robots. So the optimizers in the simulation, including the newly developed ones in this paper, can be applied directly in real-world scenarios.

5. Conclusions

In an RMFS, the decision on the assignment of orders to stations (POA) affects the throughput of the whole system the most (see Merschformann et al., 2019). Moreover, the decision on the assignment of pods to stations (PPS) should be made together with POA to get better results (see Example 1 in Section 2.2). Therefore, we developed novel methods to solve both POA and PPS for multiple stations and make online decisions that minimize the number of visits by pods to stations (in short: pod-station visits) to fulfill all customer orders. First, we introduced a new mathematical model to integrate POA and PPS (in short: integrated). Second, we extended the integrated model to allow for split orders. An order is split when not all of its parts are assigned together to a station. Two variations of split orders are considered: split-among-stations (all order lines of a pick order have to be assigned in the same time period, but may be assigned to different picking stations) and split-over-time (order lines of a pick order may be assigned in different time periods and to different picking stations). In the instances we tested, the number of pod-station visits could be reduced by up to 30% using the integrated model, 50% using the split-among-stations model and up to 80% using the split-over-time model compared to the state-of-the-art sequential approach described in Merschformann et al. (2019).

Additionally, we analyzed the simulation results with regard to three additional performance metrics, namely robot distances, pile-on and picks per hour (throughput). According to our experiments, a reduction in pod-station visits induces a reduction in robot distances and by definition it comes with higher pile-on. The picks per hour can be increased, due to the shorter waiting time of pickers, caused by more efficient order assignments which require the robots to drive less distance and therefore better supply the stations with pods to pick from.

The running times for our test instances with 50, 150 and 250 orders were acceptable; however, we needed a heuristic method to test a real-world instance with 2000 orders. For instances of that size, it is not practicable to consider all unfulfilled orders in the models at each period; moreover, not all orders are suitable in each period. Therefore, we selected 40 best orders according to our heuristic in each period to submit to the model. Still, a 60% reduction in pod-station visits and distances driven by robots could be achieved using the split-over-time model, and 47% more picks per hours can be achieved with acceptable running times.

We additionally discussed some assumptions for the mathematical models from a practical point of view, such as the extension of the split-among-stations model to support limited capacity of packing stations.

Since an RMFS is a new type of warehousing system, the concepts specific to RMFSs have not received much scholarly attention. For example, the implementation of priority zones (see Flipse, 2011). These zones are located in the proximity of workstations to store pods which might be used in the near future. Determining the optimal size of priority zones would also be interesting.

Acknowledgments

The authors would like to thank two anonymous referees for their insightful comments and suggestions. Nils Thieme and Ruslan Krenzler are funded by the industrial project “Robotic Mobile Fulfillment System”, which is financially supported by Ecopti GmbH (Paderborn, Germany) and Beijing Hanning Tech Co., Ltd. (Beijing, China). We would like to thank the Paderborn Center for Parallel Computing for providing their clusters for our numerical experiments.

Appendix A. Example of the split-over-time approach

We have in this example one open position for station 1, while we have two identical orders 1 and 2 in the backlog (in Fig. A.18(a)). These two orders contain SKUs shown in blue and orange. These two SKUs are located in two different pods, namely pod 1 with the orange SKU and pod 2 with the blue SKU. By allowing the orders 1 and 2 to be split into blue and orange parts, the orange ones can firstly be picked from pod 1 at station 1 in period 1, while one position is open at station 2 (see Fig. A.18(b)).

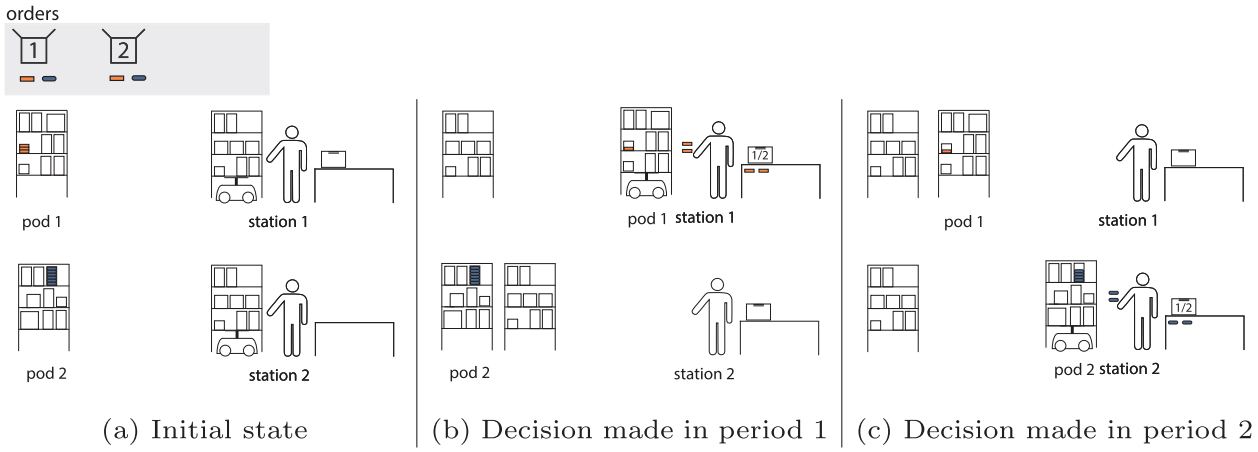


Fig. A1. An example of the split-over-time approach.

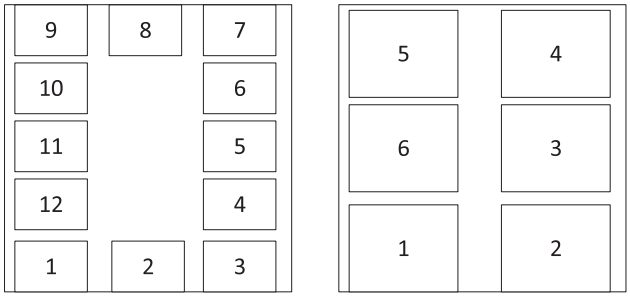


Fig. B1. Positions of the boxes in size S (left) and M (right) within a tier of a shelf.

After that, the blue ones are picked from pod 2 at station 2 in period 2 (see Fig. A.18(c)). This allows both orders to be fulfilled with two visits of pods to the stations.

Appendix B. Calculation example for the capacity of a packing station

As mentioned before, the capacity of a packing station depends on the number of parking shelves and their sizes. According to Wulfraat (2012), the typical size of a shelf is 99cm × 99cm × 244cm. And we have two common sizes of boxes, namely 25cm × 17.5cm × 10cm and 37.5cm × 30cm × 13.5cm (see DHL Packsets in sizes S and M in https://www.dhl.de/en/privatkunden/pakete-versenden/verpacken.html). As shown in Fig. B.19(a) and B.19(b), we can store 12 and 6 boxes within a tier of a shelf for the boxes in sizes S and M respectively. Then we assume we store boxes in size S in half of the shelves, and boxes in size M in the other half. By considering spaces for open boxes and usable vertical spaces of a shelf, we assume 100centimetres of vertical space is available for five tiers of boxes with sizes S, while another 100centimetres is available for three tiers of boxes with size M. Based on these assumptions, we get to store 78 boxes in total on a shelf. In other words, we can store 78 split orders on a shelf.

Appendix C. Omitted Proofs

In this section we prove that the models in Section 3 always have feasible solutions.

Proof of Proposition 1. By the assumption about *Maximal order size* in Section 3.1, there exists an order o' such that o' has no more items than capcity $C_{s'}$ at some station $s' \in \mathcal{S}$. Let us select this order and the corresponding pods for our solution.

Formally this means: Set $x_{ps} := 1$ for all $p \in \mathcal{P}_s$ as required by (6). Set $y_{o'} := 1$ and $y_o := 0$ otherwise. Set $y_{o's'} := 1$ and set $y_{os} := 0$ otherwise. Then (3) holds. Set $y_{io's'} := 1$ for all $i \in \mathcal{I}_{o'}$ and $y_{ios} := 0$ otherwise. Then (2) holds. By the choice of o' and s' constraint (4) holds too. Finally, we set $x_{ps'} := 1$ for all $p \in \mathcal{P}_i^{\text{SKU}}$ with $i \in \mathcal{I}_{o'}$, then (5) is fulfilled. Thus we have constructed a feasible solution. To find order o' and to set to set correspoding variables, we only need polynomial time. □

Proof of Proposition 2. The main idea of the proof is that we can select parameters for the integrated model with available stations set $\mathcal{S} := \{1, \dots, k\}$ from Section 3.2 in such a way that Eqs. (1)–(10) solve a given set covering problem. That means, given a set of sets $\mathcal{A} := A_1, \dots, A_n$, the universe $U := \bigcup_{i=1}^{n} A_i$, and a set $B \subset U$. Find a minimal index set $J \subset \{1, \dots, n\}$ such that corresponding sets A_i cover the set B, or formally, $B \subset \bigcup_{j \in J}^{n} A_i$.

We define an order o_1 which has the same elements as the set $B_1 := B$. We assume that the capacities of all the stations C_s, $s \in \mathcal{S}$ are equal the size of the order o_1. Furthermore, we create disjoint orders $o_2, \dots, o_k$ and define correspondsing disjoint sets $B_2, \dots, B_k$, which do not share any elements from the universe U. All the orders form the set of current orders in the backlog $\mathcal{O} := \{o_s | s \in \mathcal{S}\}$.

For every order o_i, $i \in \{2, \dots, k\}$ we also define additional set $A_{n+i-1} := B_{i+1}$. We define a set of pods $\mathcal{P} = \{1, \dots, n, n+1 \dots, n+k\}$ such that every pod p in $\mathcal{P}$ has the same elements as set A_p.

Now we force the integrated model to assign all the orders $o_1, \dots, o_k$. In order to do it, we set the weight $W_u := |B| + 1$. Because every order $o \in \mathcal{O}$ needs maximaly $|o_s| = |B|$ pods, it is cheaper to use as many available capacities as possible. Because we have k orders, k stations, and the size of every order is the same as avaialble capacity of any station, the optimal solution will assign all the orders and no capacity will be left. That means, for every optimal solution, the unused capacity is $u_s = 0$ for all $s \in \mathcal{S}$. Therefore x_{ps} in (1) also solves

$$\text{Min} \quad \sum_{p \in \mathcal{P}} \sum_{s \in \mathcal{S}} x_{ps}. \tag{C. 1}$$

Let x'_{ps} be an optimal solution of the integrated problem. Because this solution minimizes (C. 1), it describes a minimal set of pods to fulfill all the orders $o_1, \dots, o_k$. Formally, we define an index

set

$$J^* := \left\{ p \in \mathcal{P} : \exists s \in \mathcal{S} : x'_{ps} = 1 \right\} \tag{C. 2}$$

then the index set J^* is a mininal set with

$$\bigcup_{s=1}^{k} B_s \subset \bigcup_{p \in J^*} A_p. \tag{C. 3}$$

Recall, $B_2, \ldots, B_k$ are the same as $A_{n+1}, \ldots, A_{n+k-1}$ with no elements from universe U. Thus, $J := J^* \setminus \{n+1, \ldots, n+k-1\}$ solves our initial set covering problem

$$B = B_1 \subset \bigcup_{p \in J^* \setminus \{n+1,\ldots,n+k-1\}} A_p = \bigcup_{p \in J} A_p. \tag{C. 4}$$

□

Remark 1. The proof of Proposition 2 also proves that the split-among-stations model is NP-hard if x'_{tp} is the optimal solution for (1), subject to constraints (2.1), (3.1), (4)-(10), and (11)-(14). This is because, after we found an optimal solution x'_{ps}, we do not require every order $o \in \mathcal{O}$ to be assigned to only one station to construct a minimal index set J.

The proof of Proposition 2 also proves that the split-over-time model is NP-hard, when we use the optimal solution of the split-over-time model for x'_{tp}. The constant W_u forces any optimal solution of the split-over-the time problem to use all available capacities. Because we have capacity for all orders in the backlog, no orders will be split in time. Thus, any optimal solution of the split-over-the time model is simultaniously an optimal solution of split-among-stations model, which is NP-hard.

Proof of Proposition 3. We show that every solution of the integrated model also solves the split-among-stations. Let $(x'_{ps})_{p\in\mathcal{P},s\in\mathcal{S}}$, $(y'_o)_{o\in\mathcal{O}}$, $(y'_{os})_{o\in\mathcal{O},s\in\mathcal{S}}$, and $(y'_{ios})_{o\in\mathcal{O},i_o\in\mathcal{I}_o,s\in\mathcal{S}}$ be a solution of the integrated model. From (2) follows (2.1). Set $e'_o := 0$ for all $o \in \mathcal{I}_o$, then from (3) follows (3.1), (12) and (20). Substitute (2) into (3), then (11) follows. If we sum (3) on both sides of the equation over $i \in \mathcal{I}_o$ we get $\sum_{i_o\in\mathcal{I}_o} y'_{ios} = \sum_{i_o\in\mathcal{I}_o} y'_{os} \geq y'_{os}$ and therefore (13) holds.

From the first part of this proof and Proposition 1, it follows that there is a feasible solution for the split-among-stations model.

Finally, we have the same objective function (1). Because the split-among-station problem is an optimization problem, its optimal solution is either

$(x'_{ps})_{p\in\mathcal{P},s\in\mathcal{S}}$, $(y'_o)_{o\in\mathcal{O}}$, $(y'_{os})_{o\in\mathcal{O},s\in\mathcal{S}}$, $(y'_{ios})_{o\in\mathcal{O},i_o\in\mathcal{I}_o,s\in\mathcal{S}}$ or better. □

Proof of Proposition 5. The proof is analogous to the proof of Proposition 3. Let $(x'_{ps})_{p\in\mathcal{P},s\in\mathcal{S}}$, $(y'_o)_{o\in\mathcal{O}}$, $(y'_{os})_{o\in\mathcal{O},s\in\mathcal{S}}$, and $(y'_{ios})_{o\in\mathcal{O},i_o\in\mathcal{I}_o,s\in\mathcal{S}}$, $(e'_o)_{o\in\mathcal{O}}$, be a solution of the split-among-stations model. Set all $y'^{b}_{i_o} := 0$, then (11.1) holds. Furthermore, we have the same objective function (1). With the same argumentation as in the proof of Proposition 3, we conclude all the statements of Proposition 5. □

Appendix D. Additional parameters in the simulation

Table D.9
Parameters of robot movement and time for picking units.

Parameter	Value
Robot acceleration/deceleration	$1\frac{m}{s^2}$
Robot maximum velocity	$1.5\frac{m}{s}$
Time needed for a full turn of a robot	$2.5s$
Time needed for lifting and storing a pod	$2.2s$
Time needed for picking a unit	$7s$
Time needed for handling a unit at picking station	$13s$

References

Armstrong, R. D., Cook, W. D., & Saipe, A. L. (1979). Optimal batching in a semi-automated order picking system. *Journal of the Operational Research Society, 30*(8), 711–720.

Azadeh, K., de Koster, M., & Roy, D. (2017). Robotized warehouse systems: Developments and research opportunities. Research paper (no. ERS-2017-009-LIS), ERIM Report Series Research in Management.

Banker, S. (2016). Robots in the warehouse: It's not just Amazon. http://www.forbes.com.

Bartholdi, J., & Hackman, S. (2017). *Warehouse & distribution science: release 0.98*. Supply Chain and Logistics Institute.

Boysen, N., Briskorn, D., & Emde, S. (2017). Parts-to-picker based order processing in a rack-moving mobile robots environment. *European Journal of Operational Research, 262*(2), 550–562.

Boysen, N., de Koster, R., & Weidinger, F. (2018). Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*.

Bozer, Y. A., & Aldarondo, F. J. (2018). A simulation-based comparison of two goods-to-person order picking systems in an online retail setting. *International Journal of Production Research, 56*(11), 3838–3858. doi:10.1080/00207543.2018.1424364.

Cohen, L., Uras, T., & Koenig, S. (2015). Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In *Proceedings of the eighth annual symposium on combinatorial search*.

Cohen, L., Wagner, G., Kumar, T. K. S., Choset, H., & Koenig, S. (2017). Rapid randomized restarts for multi-agent path finding solvers. arXiv preprint: 1706.02794

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

De Koster, R. B., Le-Duc, T., & Zaerpour, N. (2012). Determining the number of zones in a pick-and-sort order picking system. *International Journal of Production Research, 50*(3), 757–771.

Flipse, M. (2011). Altering and improving Kiva. Working Paper, Vrije Universiteit Amsterdam.

Hanson, R., Medbo, L., & Johansson, M. I. (2018). Performance characteristics of robotic mobile fulfilment systems in order picking applications. *IFAC-PapersOnLine, 51*(11), 1493–1498.

Hoffman, A. E., Mountz, M. C., Barbehenn, M. T., Allard, J. R., Kimmel, M. E., Santini, F., Decker, M. H., D'Andrea, R., & Wurman, P. R. (2013). System and method for inventory management using mobile drive units. https://www.google.com/patents/US20130103552.

Il-Choe, K., & Sharp, G. (1991). Small parts order picking: Design and operation. https://www2.isye.gatech.edu/~mgoetsch/cali/Logistics%20Tutorial/order/article.htm.

Krenzler, R., Xie, L., & Li, H. (2018). Deterministic Pod Repositioning Problem in Robotic Mobile Fulfillment Systems. ArXiv e-prints,.

Lamballais, T., Roy, D., & De Koster, M. (2017). Estimating performance in a robotic mobile fulfillment system. *European Journal of Operational Research, 256*(3), 976–990.

Lamballais, T., Roy, D., & De Koster, R. B. (2019). Inventory allocation in robotic mobile fulfillment systems. *IISE Transactions*, 1–17.

Merschformann, M. (2017). Active repositioning of storage units in robotic mobile fulfillment systems. In *Proceedings of the operations research proceedings 2017* (pp. 379–385). Springer.

Merschformann, M., Lamballais, T., de Koster, M., & Suhl, L. (2019). Decision rules for robotic mobile fulfillment systems. *Operations Research Perspectives, 6*, 100128.

Merschformann, M., Xie, L., & Erdmann, D. (2017). Multi-Agent Path Finding with Kinematic Constraints for Robotic Mobile Fulfillment Systems. ArXiv e-prints,.

Merschformann, M., Xie, L., & Li, H. (2018). RAWSim-O: A simulation framework for robotic mobile fulfillment systems. *Logistics Research, 11:8*.

Onal, S., Zhang, J., & Das, S. (2017). Modelling and performance evaluation of explosive storage policies in internet fulfilment warehouses. *International Journal of Production Research, 55*(20), 5902–5915.

Otten, S., Krenzler, R., Xie, L., Daduna, H., & Kruse, K. (2019). Lost-customers approximation of semi-open queueing networks with backordering–An application to minimise the number of robots in robotic mobile fulfilment systems. ArXiv e-prints,.

Roy, D., Nigam, S., de Koster, R., Adan, I., & Resing, J. (2019). Robot-storage zone assignment strategies in mobile fulfillment systems. *Transportation Research Part E: Logistics and Transportation Review, 122*, 119–142.

ThinkLogistics (2012). Thinking robots – canadian logistics company deploys Kiva robots in the warehouse. Materials Management & Distribution (Canada's Supply Chain Magazine)https://www.thinklogistics.com/wp-content/uploads/2013/02/MMD-ThinkLogistics.pdf.

Toister, J. (2017). An inside look at Amazon's fulfillment center operations. https://www.toistersolutions.com/blog/2017/4/10/an-inside-look-at-amazons-fulfillment-center-operations.

Tompkins, J. A. (2010). *Facilities planning* (4th). Hoboken, NJ and Chichester: John Wiley & Sons.

Van Gils, T., Ramaekers, K., Caris, A., & de Koster, R. B. (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.

Weidinger, F., Boysen, N., & Briskorn, D. (2018). Storage assignment with rack-moving mobile robots in kiva warehouses. *Transportation Science, 52*(6), 1479–1495.

Wulfraat, M. (2012). Is Kiva Systems a good fit for your distribution center? An unbiased distribution consultant evaluation.http://www.mwpvl.com/html/kiva_systems.html.

Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine, 29*. doi:10.1609/aimag.v29i1.2082.

Xie, L., Li, H., & Thieme, N. (2019). From simulation to real-world robotic mobile fulfillment systems. *Logistics Research, 12:9*.

Yuan, R., Dong, T., & Li, J. (2016). Research on the collision-free path planning of multi-agvs system based on improved a* algorithm. *American Journal of Operations Research, 06*(06), 442–449. doi:10.4236/ajor.2016.66041.

Yuan, Z., & Gong, Y. Y. (2017). Bot-in-time delivery for robotic mobile fulfillment systems. *IEEE Transactions on Engineering Management, 64*(1), 83–93. doi:10.1109/TEM.2016.2634540.

Zhang, J., Yang, F., & Weng, X. (2019). A building-block-based genetic algorithm for solving the robots allocation problem in a robotic mobile fulfilment system. *Mathematical Problems in Engineering, 2019*.

Zou, B., Xu, X., De Koster, R., et al. (2018). Evaluating battery charging and swapping strategies in a robotic mobile fulfillment system. *European Journal of Operational Research, 267*(2), 733–753.